On The Best Principal Submatrix Problem

by

Seth Charles Lewis

A thesis submitted to
University of Birmingham
for the degree of
Doctor of Philosophy (PhD)

School of Mathematics
University of Birmingham
September, 2006

Abstract

Let $A = (a_{ij})$ be an $n \times n$ matrix with entries from $\mathbb{R} \cup \{-\infty\}$ and $k \in \{1, \ldots, n\}$. The best principal submatrix problem (BPSM) is: Given matrix A and constant k, find the biggest assignment problem value from all $k \times k$ principal submatrices of A.

This is equivalent to finding the $(n-k)$'th coefficient of the max-algebraic characteristic polynomial of A. It has been shown that any coefficient can be found in polynomial time if it belongs to an essential term.

One application of BPSM is the job rotation problem: Given workers performing a total of n jobs, where a_{ij} is the benefit of the worker currently performing job i to instead perform job j, find the maximum total benefit of rotating any k jobs round.

In general, no polynomial time algorithm is known for solving BPSM (or the other two equivalent problems). BPSM and related problems will be investigated. Existing and new results will be discussed for solving special cases of BPSM in polynomial time, such as when A is a generalised permutation matrix.

Acknowledgements

I would like to thank everyone that has helped support me during this thesis. In particular I would like to thank my supervisor Dr. Butkovič for his support and guidance. Finally I would like to thank the School of Mathematics and the EPSRC for their financial assistance.

Contents

List of Figures

List of Tables

List of Notation

Page 6,	$\overline{\mathbb{R}}$
Page 6,	$a \oplus b$
Page 6,	$a \otimes b$
Page 7,	$x^{(r)}$
Page 8,	N
Page 8,	K
Page 8,	$\sum^{\oplus}$
Page 8,	$\sum^{\otimes}$
Page 8,	$\alpha \otimes A$
Page 9,	$A \oplus B$
Page 9,	$A \otimes B$
Page 10,	$A^{(r)}$
Page 10,	a_{ij}^{k}
Page 11,	$\mathrm{diag}(d)$
Page 11,	I
Page 13,	$l(p)$
Page 13,	PND
Page 14,	$w(v_i, v_j)$
Page 14,	$w(p)$
Page 14,	$\mu(p)$
Page 14,	$D(A)$
Page 14,	$D_C(A)$
Page 15,	C_n
Page 15,	C_n^k
Page 15,	P_n
Page 15,	$l(\pi)$
Page 16,	$w(A, \sigma)$
Page 16,	$\mu(A, \sigma)$
Page 16,	$w(A, \pi)$
Page 16,	$\mu(A, \pi)$
Page 16,	id
Page 21,	$\lambda(A)$
Page 22,	$\Delta(A)$
Page 22,	$\Delta_j(A)$
Page 22,	$\Delta_{ij}(A)$
Page 26,	$sp(A)$
Page 31,	$\mathrm{maper}(A)$
Page 32,	$ap(A)$
Page 33,	$\chi_A(x)$
Page 33,	$\delta_k(A)$
Page 38,	$\gamma_k(A)$
Page 40,	$A(k)$

Page 57,	$\mathbb{T}$
Page 57,	$\mathbb{T}^{n\times n}$
Page 58,	l_{min}
Page 58,	k_{min}
Page 59,	k_{max}
Page 59,	l_{max}
Page 59,	A_{max}
Page 59,	A_{min}
Page 75,	D^{-1}
Page 75,	$A \sim B$
Page 77,	$A(S)$
Page 83,	S_k matrix
Page 83,	S_k transformable
Page 92,	F
Page 96,	k_{oddmin}
Page 112,	blockdiag(A)
Page 113,	$n(j)$

Chapter 1

Introduction

1.1 Background

The assignment problem is a classical problem in discrete optimisation. It is the task of finding the maximum sum of any n elements of an $n \times n$ matrix, with no two elements in the same row or column. This is a well known problem with many practical applications. For example, it can be used to find the optimal assignment of n workers to n jobs (one job each), where it is known how well each worker will perform each job.

The assignment problem can be solved in polynomial time, for example by the Hungarian method. It has many variants, such as the bottleneck assignment problem, quadratic assignment problem and parity assignment problem. Some variants of the assignment problem do not currently have a polynomial time solution method.

One such variant is the best principal submatrix problem (BPSM): Given an $n \times n$ matrix $A = (a_{ij})$ with entries from $\mathbb{R} \cup \{-\infty\}$ and a constant $k \in \{1, \ldots, n\}$, find the biggest assignment problem value from all $k \times k$ principal submatrices of A. For this choice of A and k, we refer to this problem as BPSM(A, k).

No polynomial algorithm is known to solve BPSM, and *NP*-completeness seems not to have been proven. However, if we remove the word "principal" from the definition of BPSM, we obtain an other variant of the assignment problem called the best submatrix problem (BSM), which has been shown to be polynomially solvable.

Max-algebra is the structure that arises when we replace the operations of multiplication and addition of two numbers in conventional algebra by addition and maximum of two numbers. We are able to formulate many concepts in max-algebra, several of which are analogous to concepts in conventional algebra. One such concept is the max-algebraic characteristic polynomial (or characteristic max-polynomial) (see Section 4.3).

Given an $n \times n$ matrix $A = (a_{ij})$ and a $k \in N$, it has been shown that solving BPSM(A, k) is very closely linked to calculating the characteristic max-polynomial of A, which is the optimal assignment problem value of the following parameterised matrix:

$$\begin{pmatrix} \max(a_{11}, x) & a_{12} & \dots & a_{1n} \\ a_{21} & \max(a_{22}, x) & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & \max(a_{nn}, x) \end{pmatrix}.$$

This value will be a max-algebraic polynomial function of x. Its graph is formed by taking the upper envelope of $n+1$ linear functions. It has been shown that finding the $(n-k)$'th coefficient of the characteristic max-polynomial (i.e. where the function with slope $n-k$ will cross the vertical axis) is equivalent to solving BPSM(A, k). A polynomial time algorithm has been given to calculate any coefficient that belongs to an essential term (which is a term that appears in the graph of the characteristic max-polynomial as a function). In general, is not known how to polynomially calculate coefficients of terms that are not essential.

One application of BPSM is the job rotation problem: Given workers performing a total of n jobs, where a_{ij} is the benefit of the worker currently performing job i to instead perform job j, find the maximum total benefit of rotating any k jobs round. We may also wish to find the assignment that gives this maximum total benefit.

The job rotation problem may be useful if some jobs need to be reassigned to reduce the boredom of employees repeating monotonous tasks (which may occur in places like factory assembly lines) but for the stability of the process only want to swap a limited number of workers. More on the possible applications in Section 5.3.

1.2 Overview of chapters

This thesis will investigate the best principal submatrix problem. The work is split into several chapters:

In Chapter 2 we introduce max-algebra. Throughout the thesis we will use concepts from this, together with graph theory and permutations. We provide notations and definitions for these areas.

In Chapter 3 we summarise the max-algebraic eigenproblem of a matrix A. We will see that the eigenvalue is always unique (unlike in conventional algebra) and is equal to the maximum elementary cycle mean of the digraph of A. This result and others are used later to help provide upper bounds to the optimal solution value of BPSM(A, k).

In Chapter 4 we state the result that the max-algebraic permanent (which is defined as an analogue of the classical one) is the optimal value of the assignment problem. We define general max-polynomials and characteristic max-polynomials. We define what essential terms of a characteristic max-polynomial are, and state a result that we can calculate these in polynomial time.

In Chapter 5 we define the best principal submatrix problem. We review how it is related to the characteristic max-polynomial. We give examples of possible situations that BPSM could be applied to, one of which is the job rotation problem. We look at related problems, which are interesting in their own right, but some of which provide us with bounds or other useful results for BPSM. We investigate the complexity of these problems compared to BPSM.

In Chapter 6 we provide several bounds or other results for BPSM that can be obtained from BSM and similar problems as well as the eigenproblem. We look at existing special cases of BPSM that can be solved in polynomial time. Such cases include where the matrix is diagonally dominant, a Monge matrix, amongst some forms of Hankel matrix or a permutation matrix.

We will note that BPSM can be solved by a randomised polynomial algorithm if the entries of the matrix are polynomially bounded in the dimension of the matrix. We give some existing and new results for BPSM with a symmetric matrix having elements equal to 0 or $-\infty$.

We provide a polynomial algorithm to solve BPSM for generalised permutation matrices. We show that BPSM can be solved in polynomial time for any matrix A, if we can solve BPSM in polynomial time for all submatrices corresponding to connected components in the digraph of A.

Finally in this chapter, we give a method using simplex and branch and bound, that will solve BPSM. If terminated before completion, we can obtain upper and lower bounds for the optimal solution of BPSM from the last performed iteration of the algorithm.

In Chapter 7 we conclude the results obtained in the previous chapters and give some comments on future areas of research.

Chapter 2

Basic definitions

2.1 Max-algebra

2.1.1 Introduction

In max-algebra, we replace addition and multiplication, the binary operations in conventional linear algebra, by maximum and addition respectively.

For any problem that involves adding numbers together and taking the maximum of numbers, it may be possible to describe it in max-algebra. A problem that is non-linear when described in conventional linear algebra may convert to a max-algebraic problem that is linear with respect to max-algebra.

There are many similarities between max-algebra and conventional linear algebra, with many properties holding in both. There are also links with combinatorics and combinatorial optimisation [11]. The max-algebraic versions of many concepts in conventional linear algebra have been formulated and explored. These include the max-algebraic versions of the eigenproblem and characteristic polynomial, which we will look at in detail later.

Min-algebra and minimax-algebra have also been defined. Min-algebra is analogous to max-algebra with the operation maximum replaced by minimum. Minimax-algebra has both the operations maximum and minimum. We will not use these structures in this thesis, but for precise definitions and more details about them, see [18, 20].

There are many applications of max-algebra covering a wide range of areas. Cuninghame-Green [18, 20] has contributed greatly to the theory of max-algebra and has provided many practical applications. Max-algebra can be used to formulate (and in some cases solve) problems involving transportation networks, discrete-event dynamic systems, machine scheduling, parallel processing and others.

Much of this thesis has or can be formulated in max-algebra. The main uses of max-algebra are in the definition and use of: the max-algebraic eigenproblem (page 17), the max-algebraic permanent (page 30) and max-algebraic similar matrices (page 75). To enable us to freely use max-algebraic notation when needed, we will now state the max-algebraic definitions we will need, both for scalars and for matrices.

2.1.2 Max-algebraic operations on numbers

We will often use the set of real numbers together with $-\infty$. We will denote this by $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty\}$. On numbers from $\overline{\mathbb{R}}$, will use the binary operations $\oplus$ and $\otimes$, referred to as the max-algebraic sum and max-algebraic product. These are defined as follows:

Definition 2.1. For $a, b \in \overline{\mathbb{R}}$,

$$a \oplus b = \max(a, b)$$

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

Example 2.1.

$$4 \oplus (-7) = \max(4, -7) = 4$$

$$4 \otimes (-7) = 4 + (-7) = -3.$$

Proposition 2.2. The following properties hold for all $x, y, z \in \overline{\mathbb{R}}$.

$$
\begin{aligned}
x \oplus (y \oplus z) &= (x \oplus y) \oplus z \\
x \oplus y &= y \oplus x \\
x \otimes (y \otimes z) &= (x \otimes y) \otimes z \\
x \otimes y &= y \otimes x \\
x \otimes (y \oplus z) &= (x \otimes y) \oplus (x \otimes z) \\
x \otimes -\infty &= -\infty \\
x \oplus -\infty &= x \\
x \otimes 0 &= x \\
x \oplus x &= x.
\end{aligned}
$$

Proof. We can easily verify these properties, by substituting from the definitions of $\otimes$ and $\oplus$. $\square$

Powers use repeated multiplication:

Definition 2.3. For $x \in \overline{\mathbb{R}},\ r \in \mathbb{N}$,

$$x^{(r)} = \underbrace{x \otimes x \otimes \ldots \otimes x}_{r-\text{times}}.$$

Remark. Note that the brackets around the power indicate we are taking the max-algebraic power as opposed to the conventional power. Note also that this definition means $x^{(r)} = rx$, for $r \in \mathbb{N}$. We extend the definition of $x^{(r)} = rx$, for all $(x, r) \in (\mathbb{R} \times \mathbb{R}) \cup (\overline{\mathbb{R}} \times \mathbb{R}^+)$. For example $5^{(-2)} = -10$. Conventional inverses will not be used, so as an exception, we denote the max-algebraic inverse $-x$ by x^{-1}.

We may also note that for $x, r \in \mathbb{R}$,

$$x^{(r)} = rx = r^{(x)},$$

and

$$x \otimes x^{-1} = 0.$$

The notation $N = \{1, 2, \ldots, n\}, M = \{1, 2, \ldots, m\}$ and $K = \{1, 2, \ldots, k\}$ will be used throughout this thesis. We denote the addition of several numbers as follows:

Definition 2.4. For $x_1, x_2, \ldots, x_n \in \overline{\mathbb{R}},\ n \in \mathbb{N}$,

$$\sum_{i \in N}^{\oplus} x_i = x_1 \oplus x_2 \oplus \ldots \oplus x_n.$$

We use the following notation to denote the multiplication of several numbers:

Definition 2.5. For $x_1, x_2, \ldots, x_n \in \overline{\mathbb{R}},\ n \in \mathbb{N}$,

$$\prod_{i \in N}^{\otimes} x_i = x_1 \otimes x_2 \otimes \ldots \otimes x_n.$$

2.1.3 Max-algebraic operations on matrices

From now on, we let $A = (a_{ij}),\ B = (b_{ij})$ and $C = (c_{ij})$ unless stated otherwise.

We can multiply a matrix by a constant. This is performed elementwise as follows:

Definition 2.6. For $\alpha \in \overline{\mathbb{R}},\ A \in \overline{\mathbb{R}}^{m \times n}$,

$$\alpha \otimes A = (\alpha \otimes a_{ij}).$$

Example 2.2.

$$6 \otimes \begin{pmatrix} -5 & 3 \\ 0 & -10 \end{pmatrix} = \begin{pmatrix} 1 & 9 \\ 6 & -4 \end{pmatrix}.$$

The sum of two matrices is defined elementwise as follows:

Definition 2.7. For $A, B, C \in \overline{\mathbb{R}}^{m\times n}$, $C = A \oplus B$, if and only if

$$c_{ij} = a_{ij} \oplus b_{ij}$$

$\forall i \in M,\ \forall j \in N.$

Example 2.3.

$$\begin{pmatrix} 3 & -1 \\ 4 & 0 \end{pmatrix} \oplus \begin{pmatrix} 2 & 6 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 6 \\ 4 & 1 \end{pmatrix}.$$

Remark. Note that for $A \in \overline{\mathbb{R}}^{m\times n}$,

$$A = \underbrace{A \oplus A \oplus \ldots \oplus A}_{\text{any finite number of times}}.$$

The product of two matrices is defined elementwise as follows:

Definition 2.8. For $A \in \overline{\mathbb{R}}^{m\times l}$, $B \in \overline{\mathbb{R}}^{l\times n}$, and $C \in \overline{\mathbb{R}}^{m\times n}$, $C = A \otimes B$, if and only if

$$c_{ij} = \sum_{k=1}^{l}{}^{\oplus} (a_{ik} \otimes b_{kj})$$

for $\forall i \in M,\ \forall j \in N.$

Example 2.4.

$$\begin{pmatrix} 3 & -1 \\ 4 & 0 \end{pmatrix} \otimes \begin{pmatrix} 2 & 6 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} (3\otimes 2)\oplus(-1\otimes 3) & (3\otimes 6)\oplus(-1\otimes 1) \\ (4\otimes 2)\oplus(0\otimes 3) & (4\otimes 6)\oplus(0\otimes 1) \end{pmatrix}$$
$$= \begin{pmatrix} 5 & 9 \\ 6 & 10 \end{pmatrix}$$

For square matrices, we define the powers of matrices as follows.

Definition 2.9. For $A \in \overline{\mathbb{R}}^{n\times n}$ and $r \in \mathbb{N}$,

$$A^{(r)} = \underbrace{A \otimes A \otimes \ldots \otimes A}_{r-\text{times}}.$$

Remark. The process of matrix powering in max-algebra is similar to conventional linear algebra matrix powering: We can (max-algebraically) square matrix A to get $A^{(2)}$. We can square $A^{(2)}$ to get $A^{(4)}$, without calculating $A^{(3)}$. We can use this technique to find a power of a matrix, using less computational effort, (even with the matrix multiplication operations used to obtain the final result).

Example 2.5.

$$13 = 2^3 + 2^2 + 2^0,$$

so

$$A^{(13)} = A^{(8)} \otimes A^{(4)} \otimes A$$

can be found by calculating only $A^{(2)}, A^{(4)}$ and $A^{(8)}$.

Definition 2.10. Let a_{ij}^k denote the (i,j)'th element of $A^{(k)}$ $(k = 1, 2, \ldots)$.

Proposition 2.11. The following properties hold for all real values α and β and all matrices A, B and C, with suitable dimensions:

$$\begin{aligned}
A \oplus B &= B \oplus A \\
\alpha \otimes (A \oplus B) &= (\alpha \otimes A) \oplus (\alpha \otimes B) \\
A \otimes (B \oplus C) &= (A \otimes B) \oplus (A \otimes C) \\
A \otimes (B \otimes C) &= (A \otimes B) \otimes C \\
A \otimes (\alpha \otimes B) &= (\alpha \otimes A) \otimes B \\
\alpha \otimes (\beta \otimes A) &= \beta \otimes (\alpha \otimes A).
\end{aligned}$$

Proof. We can check the above easily from the definitions. □

Remark. Note that as in conventional algebra, $A \otimes B = B \otimes A$ does not hold in general.

Definition 2.12. If $d = (d_1, d_2, \ldots, d_n) \in \overline{\mathbb{R}}^n$, then

$$\operatorname{diag}(d) = \begin{pmatrix} d_1 & & -\infty \\ & \ddots & \\ -\infty & & d_n \end{pmatrix}.$$

If $d \in \mathbb{R}^n$, then $\operatorname{diag}(d)$ is called a *diagonal matrix*.

Definition 2.13. We define the *identity matrix* I as follows:

$$I = \operatorname{diag}(0, 0, \ldots, 0).$$

Remark. Note that for any square matrix A, and I of the same size, we have

$$A \otimes I = I \otimes A = A.$$

Definition 2.14. Any matrix that can be obtained from the identity matrix by permuting the rows and/or columns is called a *permutation matrix*.

Example 2.6. The matrix

$$\begin{pmatrix} -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty \\ 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 \end{pmatrix}$$

is an example of a permutation matrix.

Definition 2.15. Any matrix that can be obtained by multiplying a diagonal matrix and a permutation matrix is called a *generalised permutation matrix.*

Example 2.7. The matrix

$$\begin{pmatrix} -\infty & 2 & -\infty & -\infty \\ -\infty & -\infty & -7 & -\infty \\ 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 4 \end{pmatrix}$$

$$= \begin{pmatrix} 2 & -\infty & -\infty & -\infty \\ -\infty & -7 & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty \\ -\infty & -\infty & -\infty & 4 \end{pmatrix} \otimes \begin{pmatrix} -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty \\ 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 \end{pmatrix}$$

is an example of a generalised permutation matrix.

2.2 Basic concepts of graph theory

Often it is easier to work with digraphs rather than matrices. For example, a permutation matrix could be converted to a digraph with disjoint cycles. We could then use graph theory to manipulate the digraph instead of working with the matrix.

We now define digraphs (also known as directed graphs) and briefly list the definitions of other graph theoretical terms which will be used throughout this thesis.

Definition 2.16. A *digraph* is a pair (V, E), where $V \neq \emptyset$ is a finite set called the set of *nodes*, and $E \subseteq V \times V = \{(u, v) : u, v \in V\}$ is called the set of *arcs*.

Definition 2.17. If $D = (V, E)$ is a digraph, with $v_0, v_1, \ldots, v_k \in V$ and $(v_i, v_{i+1}) \in E$ (for $i = 0, 1, \ldots, k-1$), then $p = (v_0, v_1, \ldots, v_k)$ is called a *path* in D. A single node is a path.

Definition 2.18. The *length* of a path $p = (v_0, v_1, \ldots, v_k)$ is defined as $l(p) = k$.

Definition 2.19. For a path $p = (i_1, i_2, \ldots, i_k)$, we define $V(p) = \{i_1, i_2, \ldots, i_k\}$.

Definition 2.20. For a digraph D, we define paths $p_1, p_2, \ldots, p_s$ in D to be *pairwise node disjoint* (PND) if $V(p_i) \cap V(p_j) = \emptyset$ for $i, j = 1, ..., s,\ i \neq j$.

Definition 2.21. If $p = (v_0, v_1, \ldots, v_k)$ is a path in a digraph D, and $v_i \neq v_j$ (for $i, j = 0, 1, \ldots, k,\ i \neq j$), then p is called an *elementary path* in D.

Definition 2.22. If the sequence of nodes of a path q is a subsequence of the nodes of a path p, then q is called a *sub-path* of p.

Definition 2.23. If $p = (v_0, v_1, \ldots, v_k)$ is a path of a digraph D with $v_0 = v_k$, then p is called a *cycle* of D. A single node is a cycle.

Definition 2.24. Let $D = (V, E)$ be a digraph. If $(v, v) \in E$ then (v, v) is called an *elementary cycle* of D. For $k \geq 2$, if $(v_0, v_1, \ldots, v_{k-1})$ and $(v_1, v_2, \ldots, v_k)$ are elementary paths of D, with $v_0 = v_k$, then $(v_0, v_1, \ldots, v_k)$ is called an *elementary cycle* of D.

Definition 2.25. If $q = (v_r, \ldots, v_s)$ is a sub-path of p, with $v_r = v_s$, then q is called a *sub-cycle* of p.

Definition 2.26. Given a digraph (V, E) and a function $w : E \to \mathbb{R}$, then (V, E, w) is called a *weighted digraph*.

Definition 2.27. Given a weighted digraph $D = (V, E, w)$, if $(v_i, v_j) \in E$, then the *weight* of arc (v_i, v_j) is defined as $w((v_i, v_j))$ or simply $w(v_i, v_j)$.

Definition 2.28. Given a weighted digraph $D = (V, E, w)$, if $p = (v_0, v_1, \dots, v_k)$ is a path in D, then the *(total) weight* of path p is defined as $w(p) = \sum\limits_{i=0}^{k-1} w(v_i, v_{i+1})$.

Definition 2.29. Given a weighted digraph $D = (V, E, w)$, if $p = (v_0, v_1, \dots, v_k)$ is a path in D, then the *mean weight* of path p is defined as $\mu(p) = \dfrac{w(p)}{l(p)}$.

Definition 2.30. Let σ be a cycle in a weighted digraph $D = (V, E, w)$. If $w(\sigma) = 0$, then σ is a *zero cycle.* If $w(\sigma) > 0$, then σ is a *positive cycle.* If $w(\sigma) < 0$, then σ is a *negative cycle.* If $w(\sigma) \geq 0$, then σ is a *non-negative cycle.* If $w(\sigma) \leq 0$, then σ is a *non-positive cycle.*

Definition 2.31. Given a weighted digraph $D = (V, E, w)$, we say D is *strongly connected* if $\forall u, v \in V$, there exists both a path $(u, \dots, v)$ in D, and a path $(v, \dots, u)$ in D.

Definition 2.32. For $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, the *digraph associated with* A is the weighted digraph $D(A) = (V, E, w)$, where $V = \{v_1, v_2, \dots, v_n\}$, $E = \{(v_i, v_j) : a_{ij} \in \mathbb{R}\}$ and $w(v_i, v_j) = a_{ij}$, for all $(v_i, v_j) \in E$.

Definition 2.33. For $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, the *complete digraph associated with* A is a weighted digraph $D_C(A) = (V, E, w)$, where $V = \{v_1, v_2, \dots, v_n\}$, $E = \{(v_i, v_j) : v_i, v_j \in V\}$ and $w(v_i, v_j) = a_{ij}$, for all $(v_i, v_j) \in E$.

N.B.: Exceptionally, we use a function $w : E \to \overline{\mathbb{R}}$ here. (For all other weighted digraphs, we use a function $w : E \to \mathbb{R}$.)

Definition 2.34. A matrix $A \in \overline{\mathbb{R}}^{n\times n}$ is called *irreducible* if $D(A)$ is strongly connected or $n = 1$. A is called *reducible* if it is not irreducible.

Definition 2.35. An (undirected) *bipartite graph* is a triple (U, V, E), where $U, V \neq \emptyset$ are finite sets, and $E \subseteq (U \times V) = \{(x, y) : x \in U \text{ and } y \in V\}$.

Definition 2.36. Given an (undirected) bipartite graph (U, V, E) and a function $w : E \to \mathbb{R}$, then (U, V, E, w) is called an (undirected) *weighted bipartite graph*.

Definition 2.37. A *bipartite digraph* is a triple (U, V, E), where $U, V \neq \emptyset$ are finite sets, and $E \subseteq (U \times V) \cup (V \times U) = \{(x, y) : x \in U \text{ and } y \in V, \text{ or}, x \in V \text{ and } y \in U\}$.

Definition 2.38. Given a bipartite digraph (U, V, E) and a function $w : E \to \mathbb{R}$, then (U, V, E, w) is called a *weighted bipartite digraph*.

2.3 Permutations

An alternative to working with (elementary) cycles in digraphs is to use (cyclic) permutations. These are now explained, along with associated definitions such as length and weight.

Recall $N = \{1, \ldots, n\}$ and $K = \{1, \ldots, k\}$.

Definition 2.39. A *cyclic permutation* of length k on the set N is any $\sigma = (i_1, \ldots, i_k)$ that satisfies $(\forall r \in K)\ i_r \in N$ and $(\forall r \in K)(\forall s \in K)\ i_r \neq i_s$ for $r \neq s$. Let $l(\sigma) = k$ denote the length of σ. Let C_n denote the set of all cyclic permutations of any length on the set N. Let C_n^k denote the set of all cyclic permutations of length k on the set N. We define $\sigma(i_r) = i_{r+1}$ for $r = 1, \ldots, k-1$, and $\sigma(i_k) = i_1$.

Definition 2.40. If $\sigma_1 = (i_1, \ldots, i_r),\ \sigma_2 = (i_{r+1}, \ldots, i_s),\ \ldots,\ \sigma_p = (i_{t+1}, \ldots, i_n)$ satisfies $(\forall j \in N)\ i_j \in N$ and $(\forall j \in N)(\forall k \in N)\ i_j \neq i_k$ for $j \neq k$, then $\pi = (i_1, \ldots, i_r)(i_{r+1}, \ldots, i_s) \cdots (i_{t+1}, \ldots, i_n)$ or simply $\pi = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_p$ is called a *permutation* of length n. Let $l(\pi) = n$ denote the length of π. Let P_n denote the set of all permutations of length n. If i_j lies in cyclic permutation σ_l, then we define $\pi(i_j) = \sigma_l(i_j)$.

Definition 2.41. Let $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$. For a cyclic permutation $\sigma \in C_n$, we define $w(A,\sigma)$ (or simply $w(\sigma)$) as the *weight* of σ with respect to A, and $\mu(A,\sigma)$ (or simply $\mu(\sigma)$) as the *mean weight* of σ with respect to A, where

$$w(A,\sigma) = \sum_{i=1}^{l(\sigma)} a_{i,\sigma(i)} \quad \text{and} \quad \mu(A,\sigma) = \frac{w(A,\sigma)}{l(\sigma)}.$$

Definition 2.42. Let $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$. For a permutation $\pi \in P_n$, we define $w(A,\pi)$ (or simply $w(\pi)$) as the *weight* of π with respect to A, and $\mu(A,\pi)$ (or simply $\mu(\pi)$) as the *mean weight* of π with respect to A, where

$$w(A,\pi) = \sum_{i\in N} a_{i,\pi(i)} \quad \text{and} \quad \mu(A,\pi) = \frac{w(A,\pi)}{l(\pi)}.$$

Definition 2.43. Let the *identity permutation* of length n be defined as $id = (1)(2)\cdots(n)$.

Remark. A cyclic permutation $\sigma = (i_1, i_2, \ldots, i_k) \in C_n^k$, used in conjunction with a matrix A, corresponds to an elementary cycle $\sigma' = (v_{i_1}, v_{i_2}, \ldots, v_{i_k}, v_{i_1})$ in the digraph $D(A)$. In particular, $w(A,\sigma) = a_{i_1 i_2} + a_{i_2 i_3} + \cdots + a_{i_k i_1} = w(\sigma')$.

We will define or prove statements either in terms of elements of matrices, or digraphs, or in terms of permutations, depending on which seems easiest to work with. All results that are formulated in one of these forms will automatically hold when formulated in one of the two alternative forms.

Chapter 3

The max-algebraic eigenproblem

We will use the max-algebraic version of the eigenproblem as defined in [20]:

Definition 3.1. For a square matrix A, the *eigenproblem* is the task of finding x and λ, such that

$$A \otimes x = \lambda \otimes x.$$

We call λ an *eigenvalue* of A, and x an *eigenvector* of A.

We will assume that $A \in \overline{\mathbb{R}}^{n \times n}$ and has at least one finite element in each row and column, $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}$.

We will use some of the results from this chapter on the eigenproblem later in this thesis (Section 6.2 in particular).

In this chapter, we will discuss what the eigenvalues and eigenvectors are and how to find them. We initially restrict A to have all elements finite, and show that the eigenvalue always exists, is unique and equals the maximum elementary cycle mean in the digraph of A. We will present Karp's Algorithm, which is an efficient way of calculating the eigenvalue. Finally, we state that we can extend the results of this chapter to cover some matrices (irreducible ones) that have some non-finite elements. Much has been written on the eigenproblem, see for example [18], [20] and [26]

3.1 The eigenvalue

First we show that the eigenvalue (assuming it exists) is equal to the mean weight, with respect to A, of some cyclic permutation in C_n.

Lemma 3.2 ([20])**.** If $\exists\lambda$ satisfying the eigenproblem for a matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, then $\exists\sigma \in C_n$ such that $\lambda = \mu(A, \sigma)$.

Proof.

$$
\begin{aligned}
& & A \otimes x &= \lambda \otimes x \\
&\iff & \sum_{j\in N}^{\oplus} a_{ij} \otimes x_j &= \lambda \otimes x_i \quad (\forall i \in N) \\
&\iff & \max_{j\in N}(a_{ij} + x_j) &= \lambda + x_i \quad (\forall i \in N) && (3.1.1) \\
&\iff & \max(a_{i1} + x_1, a_{i2} + x_2, \dots, a_{in} + x_n) &= \lambda + x_i \quad (\forall i \in N) && (3.1.2)
\end{aligned}
$$

This implies at least one of the values we are maximising over will be equal to the RHS of (3.1.2). This is true for each i. Therefore we have:

$$(\forall i \in N)(\exists j \in N) \;\; a_{ij} + x_j = \lambda + x_i \tag{3.1.3}$$

Now (3.1.3) holds for $i = 1$, therefore:

$$(\exists j_1 \in N) \;\; a_{1j_1} + x_{j_1} = \lambda + x_1$$

where j_1 must be an integer between 1 and n, so (3.1.3) must also hold for $i = j_1$:

$$(\exists j_2 \in N) \;\; a_{j_1 j_2} + x_{j_2} = \lambda + x_{j_1}$$

Similarly for $i = j_2$ we get:

$$(\exists j_3 \in N) \;\; a_{j_2 j_3} + x_{j_3} = \lambda + x_{j_2}$$

and so on.

As there are only n values that i can take, we will eventually obtain a value of i that we have previously obtained. Let i_1 be the first such index, and if we set $i = i_1$ in (3.1.3), we obtain i_2, say, for the next value of i, and so on until we reach i_k, where the next value on from this is i_1 again. Therefore $\sigma = (i_1, i_2, \dots, i_k) \in C_n$.

Writing (3.1.3), with i set to $i_1, i_2, \dots, i_k$, we obtain:

$$\begin{aligned}
a_{i_1 i_2} + x_{i_2} &= \lambda + x_{i_1} \\
a_{i_2 i_3} + x_{i_3} &= \lambda + x_{i_2} \\
&\vdots \\
a_{i_{k-1} i_k} + x_{i_k} &= \lambda + x_{i_{k-1}} \\
a_{i_k i_1} + x_{i_1} &= \lambda + x_{i_k}
\end{aligned}$$

Summing these k equations, we obtain:

$$\sum_{r \in K} a_{i_r i_{r+1}} + \sum_{r \in K} x_{i_r} = k\lambda + \sum_{r \in K} x_{i_r}$$

where $\sum$ denotes conventional summation, and we define $i_{k+1} = i_1$.

Therefore, on cancelling and rearrangement of the above equation, we arrive at:

$$\begin{aligned}
\lambda &= \frac{1}{k} \sum_{r \in K} a_{i_r i_{r+1}} \\
&= \mu(A, \sigma)
\end{aligned}$$

which completes the proof. $\square$

Now we show that the eigenvalue (assuming it exists) is equal to the maximum mean weight, with respect to A, of all cyclic permutations in C_n.

Theorem 3.3 ([20])**.** If $\exists\lambda$ satisfying the eigenproblem for a matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, then $\lambda = \max\limits_{\sigma\in C_n} \mu(A,\sigma)$.

Proof. For an arbitrary $\sigma = (r_1, r_2, \dots, r_s) \in C_n$ of A, we have

$$a_{r_i r_{i+1}} + x_{r_{i+1}} \leq \max_{j=1,\dots,n}(a_{r_i j} + x_j) \tag{3.1.4}$$

for $1 \leq i \leq s$, where we let $r_{s+1} = r_1$. Note that (3.1.4) only holds because $s \leq n$, due to the fact that $r_1, \dots, r_s$ are all different values from N.

We also have that

$$\max_{j=1,\dots,n}(a_{r_i j} + x_j) = \lambda + x_{r_i} \tag{3.1.5}$$

for $1 \leq i \leq s$. This arises by replacing i in (3.1.1) with r_i.

By combining (3.1.4) and (3.1.5), and setting $i = 1, \dots, s$, we obtain:

$$\begin{array}{rcl} a_{r_1 r_2} + x_{r_2} & \leq & \lambda + x_{r_1} \\ a_{r_2 r_3} + x_{r_3} & \leq & \lambda + x_{r_2} \\ & \vdots & \\ a_{r_{s-1} r_s} + x_{r_s} & \leq & \lambda + x_{r_{s-1}} \\ a_{r_s r_1} + x_{r_1} & \leq & \lambda + x_{r_s} \end{array}$$

Summing these s equations, we obtain:

$$\sum_{i=1}^{s} a_{r_i r_{i+1}} + \sum_{i=1}^{s} x_{r_i} \leq s\lambda + \sum_{i=1}^{s} x_{r_i}$$

where $\sum$ denotes conventional summation, and we let $r_{s+1} = r_s$.

Therefore, cancelling, rearranging and using the fact that by Lemma 3.2, $\exists\sigma' \in$

C_n such that $\lambda = \mu(A, \sigma')$, we obtain:

$$\begin{aligned}\mu(A, \sigma') &= \lambda \\ &\geq \frac{1}{s}\sum_{i=1}^{s} a_{r_i r_{i+1}} \\ &= \mu(A, \sigma).\end{aligned}$$

Therefore, as σ was arbitrary we have $(\forall \sigma \in C_n)\ \mu(A, \sigma') \geq \mu(A, \sigma)$. Therefore $\lambda = \mu(A, \sigma') = \max\limits_{\sigma \in C_n} \mu(A, \sigma)$. $\square$

Remark. As $\max\limits_{\sigma \in C_n} \mu(A, \sigma)$ is unique, if there is an eigenvalue that satisfies the eigenproblem, then it must be also be unique.

We now formulate Theorem 3.3 in an alternative form:

Corollary 3.4. If $\exists\lambda$ satisfying the eigenproblem for a matrix $A = (a_{ij}) \in \mathbb{R}^{n\times n}$, then λ equals the greatest cycle mean of all elementary cycles in $D(A)$.

Definition 3.5. For $A \in \overline{\mathbb{R}}^{n\times n}$ with $D(A)$ containing at least one cycle, we define $\lambda(A)$ to be the greatest cycle mean of all elementary cycles in $D(A)$.

Example 3.1.

$$\text{Let } A = \begin{pmatrix} 3 & 4 & 3 \\ 2 & 1 & -3 \\ 3 & 7 & 2 \end{pmatrix},$$

$$\begin{aligned}\text{then } \lambda(A) &= \max\left\{ \underbrace{\frac{3}{1}, \frac{1}{1}, \frac{2}{1},}_{\substack{\text{means of elementary} \\ \text{cycles of length 1}}} \underbrace{\frac{2+4}{2}, \frac{3+3}{2}, \frac{7-3}{2},}_{\substack{\text{means of elementary} \\ \text{cycles of length 2}}} \underbrace{\frac{3+7+2}{3}, \frac{4-3+3}{3}}_{\substack{\text{means of elementary} \\ \text{cycles of length 3}}} \right\} \\ &= \max\left\{3, 1, 2, 3, 2, 3, 4, \frac{4}{3}\right\} \\ &= 4.\end{aligned}$$

Therefore if the eigenproblem is solvable for A, then the eigenvalue will be $\lambda(A) = 4$.

3.2 The eigenvectors

In this section, we will show that there does exist an eigenvalue and eigenvector(s) satisfying the eigenproblem for all matrices. We will also find eigenvectors satisfying the eigenproblem.

Let $A \in \overline{\mathbb{R}}^{n\times n}$. In $D_C(A)$ we have that a_{ij} is the weight of the (only) path of length 1 from v_i to v_j. The weight of the heaviest path (heaviest meaning having largest weight) of length 2 from v_i to v_j is written

$$\max_{k=1,\ldots,n} (a_{ik} + a_{kj}).$$

Converting to max-algebraic notation and using Definition 2.10, we see that

$$\sum_{k\in N}^{\oplus} (a_{ik} \otimes a_{kj}) = a_{ij}^2.$$

Similarly, we can show that the weight of the heaviest path of length r from v_i to v_j is a_{ij}^r.

Definition 3.6. For $A \in \overline{\mathbb{R}}^{n\times n}$, we define

$$\Delta(A) = \sum_{i\in N}^{\oplus} A^{(i)} = A^{(1)} \oplus A^{(2)} \oplus \ldots \oplus A^{(n)}.$$

Let $\Delta_j(A)$ be the j'th column of $\Delta(A)$, and $\Delta_{ij}(A)$ the (i,j)'th element of $\Delta(A)$.

We observe that the weight of the heaviest path with length up to n from v_i to v_j is $\Delta_{ij}(A)$. Any path p of length greater than n in $D_C(A)$ will not be elementary. This means there will be one or more sub-cycles in p. We delete these until we are left with an elementary path p', with $l(p') \leq n$. If $\lambda(A) \leq 0$, then we will have $w(p) \leq w(p')$, as the sub-cycles we have deleted had non-positive weight. This means $a_{ij}^{l(\sigma)} \leq a_{ij}^{l(\sigma')} \leq \Delta_{ij}(A)$. So provided that $\lambda(A) \leq 0$, $\Delta_{ij}(A)$ is equal to the weight of the heaviest path of any length from v_i to v_j.

If $A = (a_{ij}) \in \mathbb{R}^{n \times n}$, and we define $B = (b_{ij}) \in \mathbb{R}^{n \times n}$ by $b_{ij} = a_{ij} - \lambda(A)$, or equivalently $B = \lambda(A)^{-1} \otimes A$ or $A = \lambda(A) \otimes B$, then $\lambda(B) = 0$.

Consider the following quantity:

$$\sum_{k \in N}^{\oplus} (b_{ik} \otimes \Delta_{kj}(B)).$$

This is simply the weight of the heaviest path of length greater than one from v_i to v_j. The weight of the (only) path of length 1 from v_i to v_j is b_{ij}, and so

$$\begin{aligned}
& (\exists j \in N)(\forall i \in N) \sum_{k \in N}^{\oplus} (b_{ik} \otimes \Delta_{kj}(B)) \geq b_{ij} \\
\iff & (\exists j \in N)(\forall i \in N) \sum_{k \in N}^{\oplus} (b_{ik} \otimes \Delta_{kj}(B)) = \Delta_{ij}(B) \\
\iff & (\exists j \in N)\ B \otimes \Delta_j(B) = \Delta_j(B) \\
\iff & (\exists j \in N)\ B \otimes \Delta_j(B) = 0 \otimes \Delta_j(B).
\end{aligned}$$

The last line states that 0 is the eigenvalue and $\Delta_j(B)$ is an eigenvector that satisfy the eigenproblem for matrix B.

Therefore we have proved the following:

Lemma 3.7 ([3]). For $B = (b_{ij}) \in \mathbb{R}^{n \times n}$ and $\lambda(B) = 0$, we have that 0 is an eigenvalue and $\Delta_j(B)$ is an eigenvector satisfying the eigenproblem for matrix B if and only if $(\exists j \in N)(\forall i \in N)\ b_{ij} \leq \sum_{k \in N}^{\oplus} (b_{ik} \otimes \Delta_{kj}(B))$.

Definition 3.8. Assume that $D(B) = (V, E, w)$ has a maximum cycle weight of 0. An *eigennode* of B is a node $v \in V$ that lies on a zero cycle of $D(B)$. The set of all eigennodes of B is denoted $E(B)$.

Remark. If $\lambda(B) = 0$, then $v_j \in E(B)$ if and only if $\Delta_{jj}(B) = 0$.

Now if we let $v_j \in E(B)$, then there is a zero cycle $(v_j, \ldots, v_j)$ in $D(B)$, and

$$\begin{aligned} b_{ij} &= w(v_i, v_j) \\ &= w(v_i, v_j, \ldots, v_j) \\ &\leq \sum_{k \in N}^{\oplus} (b_{ik} \otimes \Delta_{kj}(B)). \end{aligned}$$

Therefore, using Lemma 3.7, we have proved for any $v_j \in E(B)$,

$$B \otimes \Delta_j(B) = 0 \otimes \Delta_j(B).$$

We can pre-multiply by $\lambda(A)$ to obtain

$$\lambda(A) \otimes B \otimes \Delta_j(B) = \lambda(A) \otimes \Delta_j(B),$$

which is equivalent to

$$A \otimes \Delta_j(\lambda(A)^{-1} \otimes A)) = \lambda(A) \otimes \Delta_j(\lambda(A)^{-1} \otimes A)),$$

and so we have proved:

Theorem 3.9 ([18])**.** If $A = (a_{ij}) \in \mathbb{R}^{n \times n}$, then $\lambda = \lambda(A)$ is the unique eigenvalue of A and every column of $\Delta(\lambda^{-1} \otimes A)$ with a zero diagonal element is an eigenvector of A.

Example 3.2. Let $B = \lambda(A)^{-1} \otimes A,$ where A is as in Example 3.1.

$$\text{Then } B = \begin{pmatrix} -1 & 0 & -1 \\ -2 & -3 & -7 \\ -1 & 3 & -2 \end{pmatrix} = B^{(1)}$$

$$B^{(2)} = \begin{pmatrix} -1 & 0 & -1 \\ -2 & -3 & -7 \\ -1 & 3 & -2 \end{pmatrix} \otimes \begin{pmatrix} -1 & 0 & -1 \\ -2 & -3 & -7 \\ -1 & 3 & -2 \end{pmatrix} = \begin{pmatrix} -2 & 2 & -2 \\ -3 & -2 & -3 \\ 1 & 1 & -2 \end{pmatrix}$$

$$B^{(3)} = \begin{pmatrix} -1 & 0 & -1 \\ -2 & -3 & -7 \\ -1 & 3 & -2 \end{pmatrix} \otimes \begin{pmatrix} -2 & 2 & -2 \\ -3 & -2 & -3 \\ 1 & 1 & -2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & -3 \\ -4 & 0 & -4 \\ 0 & 1 & 0 \end{pmatrix}.$$

So $\Delta(B) = B^{(1)} \oplus B^{(2)} \oplus B^{(3)}$

$$= \begin{pmatrix} -1 & 0 & -1 \\ -2 & -3 & -7 \\ -1 & 3 & -2 \end{pmatrix} \oplus \begin{pmatrix} -2 & 2 & -2 \\ -3 & -2 & -3 \\ 1 & 1 & -2 \end{pmatrix} \oplus \begin{pmatrix} 0 & 1 & -3 \\ -4 & 0 & -4 \\ 0 & 1 & 0 \end{pmatrix}$$

$$= \begin{pmatrix} 0 & 2 & -1 \\ -2 & 0 & -3 \\ 1 & 3 & 0 \end{pmatrix}.$$

Let $\Delta_j = \Delta_j(\lambda(A)^{-1} \otimes A)$. All diagonal elements of $\Delta(\lambda(A)^{-1} \otimes A)$ are zero, so all the columns are eigenvectors. So for example, letting $x = \Delta_2$ in Definition 3.1, we have:

$$\begin{pmatrix} 3 & 4 & 3 \\ 2 & 1 & -3 \\ 3 & 7 & 2 \end{pmatrix} \otimes \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix} = \begin{pmatrix} 6 \\ 4 \\ 7 \end{pmatrix} = 4 \otimes \begin{pmatrix} 2 \\ 0 \\ 3 \end{pmatrix}$$

Note that in this example, all the eigenvectors of A are (max-algebraic) multiples of each other. i.e.

$$\Delta_2 = 2 \otimes \Delta_1$$

and

$$\Delta_3 = -1 \otimes \Delta_1.$$

Note that while every column of $\Delta(\lambda^{-1} \otimes A)$ with a zero diagonal element is an eigenvector of A, as stated in Theorem 3.9, there may be many more eigenvectors of A that do not have a zero diagonal element.

Definition 3.10. Let the set of all eigenvectors of A be defined as

$$sp(A) = \{x \in \mathbb{R}^n : A \otimes x = \lambda \otimes x\}.$$

Any (max-algebraic) linear combination of eigenvectors of A, is also an eigenvector of A:

Theorem 3.11. If $x, y \in sp(A)$, and $\alpha, \beta \in \mathbb{R}$ then $((\alpha \otimes x) \oplus (\beta \otimes y)) \in sp(A)$

Proof. We have that

$$\begin{aligned} A \otimes ((\alpha \otimes x) \oplus (\beta \otimes y)) &= (A \otimes (\alpha \otimes x)) \oplus (A \otimes (\beta \otimes y)) \\ &= (\alpha \otimes (A \otimes x)) \oplus (\beta \otimes (A \otimes y)) \\ &= (\alpha \otimes (\lambda \otimes x)) \oplus (\beta \otimes (\lambda \otimes y)) \\ &= (\lambda \otimes (\alpha \otimes x)) \oplus (\lambda \otimes (\beta \otimes y)) \\ &= \lambda \otimes ((\alpha \otimes x) \oplus (\beta \otimes y)) \end{aligned}$$

which completes the proof. □

3.3 Karp's algorithm for finding the eigenvalue

Karp's algorithm will allow us to efficiently find the eigenvalue by calculating the greatest elementary cycle mean in the digraph of A.

Before we come to Karp's algorithm, we first need to consider the following lemma.

Lemma 3.12 ([20])**.** If $B = (b_{ij}) \in \mathbb{R}^{n\times n}$, $\lambda(B) = 0$ and $D(B) = (V, E, w)$, then by setting $(\Delta_{ij}) = \Delta(B)$, we have

$$(\forall v_j \in V)(\exists v_i \in E(B))\ \Delta_{ij} = b_{ij}^{n+1}.$$

Proof. For any $v_j \in V$, choose an arbitrary eigennode $v_k \in V$. Now choose an elementary path $p = (v_k, \ldots, v_j)$ in $D(B)$ of maximal weight starting at v_k and ending at v_j and having length up to n. Node v_k lies on some cycle σ of weight zero, by Definition 3.8. We can form a new path $p' = (v_k, \ldots, v_j)$ of maximal weight, by attaching σ to the start of p enough times so that the length of p' is $\geq n + 1$.

Form a new path, p'' by removing enough nodes from the start of p', so that the length of p'' is exactly $n + 1$. As the length of p'' is greater than the length of p, we must have that some (eigen)nodes (from one or more of the σ cycles we added) are at the start of p''. In particular, the first node of p'', say v_i, must be an eigennode.

If $v_i = v_k$, then $\Delta_{ij} = w(p) = w(p'') = b_{ij}^{n+1}$. If $v_i \neq v_k$, then we split σ into two paths; $\tau = (v_i, \ldots, v_k)$ and $\tau' = (v_k, \ldots, v_i)$. Let $p''' = (v_i, \ldots, v_k, \ldots, v_j)$ be the path formed from joining τ and p. Note that $l(p''') \leq n$.

If there exists a path $q = (v_i, \ldots, v_j)$ with $w(q) > w(p'')$, then by setting $q' = (v_k, \ldots, v_i, \ldots, v_j)$ to be the path formed from joining τ' and q, we have that

$$\begin{aligned} w(q') &= w(\tau') + w(q) \\ &> w(\tau') + w(p'') \\ &= w(\tau') + w(\tau) + w(p) \\ &= w(p) \\ &\geq w(q'). \end{aligned}$$

Therefore q does not exist and we have $\Delta_{ij} = w(p''') = w(p'') = b_{ij}^{n+1}$. □

It is extremely time consuming to find all possible elementary cycle means in the digraph of a matrix. The following algorithm, developed by Karp, uses a different approach to finding the eigenvalue, which is much faster, finding λ in $\mathrm{O}(n^3)$. (Further details of this time bound can be found in [20].)

Theorem 3.13 ([28, 20])**.** For $A = (a_{ij}) \in \mathbb{R}^{n\times n}$,

$$\lambda(A) = \max_{i\in N}\left(\min_{k\in N}\left(\frac{a_{i1}^{n+1} - a_{i1}^{k}}{n+1-k}\right)\right).$$

Proof. Let B be defined by subtracting the (unknown) finite number λ from every element of A. Clearly, all cycle means are thereby also reduced by λ, so $\lambda(B) = 0$. Let b_{ij}^k denote the (i,j)'th element of $B^{(k)}$ $(k = 1, 2, \ldots)$. Then a_{i1}^k and b_{i1}^k denote the greatest weight of a path of length k from v_i to v_1 in A and B, respectively, so

$$a_{i1}^k = b_{i1}^k + k\lambda$$

and hence, for all $i \in N$,

$$(a_{i1}^{n+1} - (n+1)\lambda) - (a_{i1}^k - k\lambda) = b_{i1}^{n+1} - b_{i1}^k.$$

Thus for all $k \neq n+1$,

$$\frac{a_{i1}^{n+1} - a_{i1}^{k}}{n+1-k} - \lambda = \frac{b_{i1}^{n+1} - b_{i1}^{k}}{n+1-k}. \tag{3.3.1}$$

If $\Delta = (\Delta_{ij}) = \Delta(B)$, then by Definition 2.1 and Definition 3.6, we have that the (i,j)'th element of Δ is greater than or equal to the (i,j)'th element of $B^{(k)}$, for all k, and equal to the (i,j)'th element of $B^{(r)}$, for some $r \in N$. Hence we have:

$$\text{For each } i,\ \Delta_{i1} \geq b_{i1}^k \quad \text{for all } k \geq 1. \tag{3.3.2}$$

$$\text{For each } i,\ \Delta_{i1} = b_{i1}^r \quad \text{for some } r \in N. \tag{3.3.3}$$

If we set k in (3.3.1) to be the r of (3.3.3), then the denominator of the RHS of (3.3.1) is ≥ 0. Also we have $b_{i1}^{n+1} \leq \Delta_{i1}$ from (3.3.2), and $b_{i1}^{r} = \Delta_{i1}$ by (3.3.3). So the RHS of (3.3.1) is ≤ 0 for at least one $k \in N$. Therefore:

$$\min_{k \in N} \frac{a_{i1}^{n+1} - a_{i1}^{k}}{n+1-k} - \lambda \leq 0 \tag{3.3.4}$$

If we set $j = 1$ in Lemma 3.12 we get the following:

$$\text{For some } i \in N,\ \Delta_{i1} = b_{i1}^{n+1} \tag{3.3.5}$$

Using (3.3.2) and (3.3.5), we have that $b_{i1}^{n+1} \geq b_{i1}^{k}$, for some $i \in N$. Therefore the RHS of (3.3.1) is ≥ 0 for at least one $i \in N$. Hence we have:

$$\max_{i \in N} \min_{k \in N} \frac{a_{i1}^{n+1} - a_{i1}^{k}}{n+1-k} - \lambda = 0,$$

and the result follows. $\square$

3.4 The eigenproblem for non-finite matrices

It is straightforward to show the results of this chapter extend to irreducible matrices in $\overline{\mathbb{R}}^{n \times n}$. In particular:

Theorem 3.14 ([18, 20])**.** Let $A \in \overline{\mathbb{R}}^{n \times n}$ be an irreducible matrix. The eigenvalue exists, is finite, unique and is given by $\lambda = \lambda(A)$. Every column of $\Delta(\lambda^{-1} \otimes A)$ with a zero diagonal element is a finite eigenvector of A.

We could consider reducible matrices in the eigenproblem. However, this is slightly more complicated and we will not need to solve the eigenproblem for reducible matrices as explained later (Section 6.4.8).

Chapter 4

The max-algebraic characteristic polynomial

In this chapter we define the max-algebraic characteristic polynomial (or characteristic max-polynomial). As the name suggests, a max-polynomial, which we will introduce shortly, is a polynomial in the max-algebra setting. The characteristic max-polynomial is a specific type of max-polynomial, defined using maper, the max-algebraic permanent. We will discuss the max-algebraic permanent and its relationship to the classical assignment problem in the next section.

It is not known how to efficiently calculate all terms of the characteristic max-polynomial. We can calculate some terms (called essential terms) in polynomial time. We will describe the graph of the characteristic max-polynomial, and will show why only essential terms are needed to describe the characteristic max-polynomial as a function.

4.1 The max-algebraic permanent

Recall that $N = \{1, \ldots, n\}$ and P_n is set of all permutations of length n (Definition 2.40). Let $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, then we define the max-algebraic *permanent* of A as

$$\mathrm{maper}(A) = \sum_{\pi \in P_n}^{\oplus} \prod_{i \in N}^{\otimes} a_{i,\pi(i)},$$

which in conventional notation is

$$\mathrm{maper}(A) = \max_{\pi \in P_n} \sum_{i \in N} a_{i,\pi(i)}.$$

Note in particular, that $\prod_{i \in N}^{\otimes} a_{i,\pi(i)}$ in conventional algebra is $\sum_{i \in N} a_{i,\pi(i)}$, which we recall, for $\pi \in P_n$, was defined as the weight $w(A, \pi)$ of permutation π with respect to matrix A. Hence,

$$\mathrm{maper}(A) = \max_{\pi \in P_n} w(A, \pi)$$

is the maximum weight of permutations $\pi \in P_n$ with respect to matrix A. This is also known as the optimal assignment problem value for the matrix A. The assignment problem is to find n independent entries in an $n \times n$ matrix with maximum sum.

There are a number of efficient solution methods for solving the assignment problem. One of the best known is the Hungarian method, which will solve the assignment problem with computational complexity $O(n^3)$. Orlin and Ahuja [36] produced an algorithm that will solve the assignment problem by combining the auction algorithm and the shortest path algorithm, in $O(\sqrt{n}m \log(nA_{max}))$, where m is the number of finite entries in A, and A_{max} is the maximum entry in A. If all entries of the matrix are finite and bounded by a polynomial in n, then the complexity of their algorithm simplifies to $O(n^{2.5} \log n)$.

If $w(A, \pi) = \mathrm{maper}(A)$, then π is known as an *optimal permutation* (or *optimal solution*) for the assignment problem of matrix A. There may be more than one optimal permutation. We define $ap(A)$ to be the set of all optimal permutations:

$$ap(A) = \big\{\pi \in P_n : w(A, \pi) = \mathrm{maper}(A)\big\}.$$

4.2 Max-polynomials

A *max-polynomial* has the form

$$\begin{aligned} p(x) &= \sum_{r=0}^{n}{}^{\oplus} a_r \otimes x^{(r)} \\ &= \max_{r=0,\ldots,n} (a_r + rx), \end{aligned}$$

for coefficients $a_0, a_1, \ldots, a_n \in \overline{\mathbb{R}}$.

Obviously, the graph of a max-polynomial is a piecewise linear convex function, whose slopes are from the set $\{0, 1, \ldots, n\}$. All max-polynomials can be factorised into max-algebraic linear factors:

Proposition 4.1 ([21])**.** Any max-polynomial can be written as the product of a constant and n linear factors

$$p(x) = a \otimes (x \oplus b_1) \otimes (x \oplus b_2) \otimes \ldots \otimes (x \oplus b_n),$$

where $a, b_1, b_2, \ldots, b_n \in \overline{\mathbb{R}}$.

In conventional notation,

$$p(x) = a + \max(x, b_1) + \max(x, b_2) + \ldots + \max(x, b_n).$$

The constants b_r, for $r \in N$, are called the *corners* of $p(x)$. Some corners may be equal, but the set of corners of $p(x)$ is uniquely determined.

4.3 The characteristic max-polynomial

There are several ways to define the max-algebraic characteristic polynomial (also called the *characteristic max-polynomial*) of a square matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$. (See

[35, 25] for details of these, and also the related concepts of *characteristic equation* and *characteristic bi-polynomial* in max-algebra.)

We will use the definition given in [19], which states that the characteristic max-polynomial is

$$\chi_A(x) = \operatorname{maper}(A \oplus x \otimes I).$$

In other words, it is the max-algebraic permanent of the matrix

$$\begin{pmatrix} a_{11} \oplus x & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} \oplus x & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \oplus x \end{pmatrix}.$$

This means that

$$\begin{aligned} \chi_A(x) &= \delta_0 \oplus (\delta_1 \otimes x) \oplus \dots \oplus (\delta_{n-1} \otimes x^{(n-1)}) \oplus x^{(n)} \\ &= \sum_{i=0}^{n}{}^{\oplus} \delta_i \otimes x^{(i)} \end{aligned}$$

for some $\delta_0, \dots \delta_{n-1} \in \overline{\mathbb{R}}$, and we set $\delta_n = 0$ and $x^{(0)} = 0$. Where we use more than one matrix, we shall write $\delta_k(A)$ to refer to the max-algebraic coefficient of $x^{(k)}$ in the characteristic max-polynomial of matrix A. Written in conventional notation we have

$$\chi_A(x) = \max(\delta_0, \delta_1 + x, \dots, \delta_{n-1} + (n-1)x, nx). \tag{4.3.1}$$

Viewed as a function of x, this is obviously a piecewise linear convex function whose slopes are from the set $\{0, 1, \dots, n\}$.

If for some $k \in \{0, 1, \dots, n\}$ the inequality

$$\delta_k \otimes x^{(k)} \leq \sum_{i \in \{0,\dots,n\} - \{k\}}{}^{\oplus} \delta_i \otimes x^{(i)} \tag{4.3.2}$$

holds for every real x, then the term $\delta_k \otimes x^{(k)}$ is called *inessential*, otherwise it is called *essential*. The term $x^{(n)}$ is always essential. If $\delta_k \otimes x^{(k)}$ is inessential then

$$\chi_A(x) = \sum_{i \in \{0,\ldots,n\}-\{k\}}^{\oplus} \delta_i \otimes x^{(i)}$$

holds for every real x. Therefore inessential terms are not needed to describe $\chi_A(x)$ as a function of x.

Where the gradient of $\chi_A(x)$ changes, we define x as a breakpoint:

Definition 4.2. If $y_1(x)$ and $y_2(x)$ are two different essential terms of $\chi_A(x)$, and $\exists x'$ such that $y_1(x') = y_2(x') = \chi_A(x')$, then x' is called a *breakpoint* of $\chi_A(x)$.

Theorem 4.3. The set of breakpoints of $\chi_A(x)$ is equal to the set of corners of $\chi_A(x)$.

Proof. We have

$$\begin{aligned}\chi_A(x) &= \delta_0 \oplus (\delta_1 \otimes x) \oplus \cdots \oplus (\delta_{n-1} \otimes x^{(n-1)}) \oplus x^{(n)} \\ &= \max(\delta_0, \delta_1 + x, \ldots, \delta_{n-1} + (n-1)x, nx) \qquad (4.3.3)\end{aligned}$$

and

$$\begin{aligned}\chi_A(x) &= a \otimes (x \oplus b_1) \otimes (x \oplus b_2) \otimes \cdots \otimes (x \oplus b_n) \\ &= a + \max(x, b_1) + \max(x, b_2) + \cdots + \max(x, b_n)\end{aligned}$$

and may assume that $b_1 \leq b_2 \leq \cdots \leq b_n$. Some of these may be $-\infty$. Assume that $b_1 = \cdots = b_r = -\infty$ and $-\infty < b_{r+1} \leq \cdots \leq b_n$. This enables us to write

$$\chi_A(x) = a + rx + \max(x, b_{r+1}) + \max(x, b_{r+2}) + \ldots + \max(x, b_n) \qquad (4.3.4)$$

for $0 \leq r \leq n$.

If a linear piece of the graph has slope k, then by (4.3.3), the equation of this piece must be given by

$$y = \delta_k + kx. \tag{4.3.5}$$

Note that $\forall i \in \{r+1, \ldots, n\}$, by using (4.3.4), we see that in (4.3.5):

$$\text{If } x < b_i \text{ then } k < i$$
$$\text{If } x > b_i \text{ then } k \geq i.$$

Therefore the slope of $\chi_A(x)$ changes immediately before and after b_i. Hence each corner b_i is also a breakpoint.

Also note that $\forall i \in \{r+1, \ldots, n-1\}, \forall x \in (b_i, b_{i+1})$, k remains unchanged. Hence the slope of $\chi_A(x)$ remains unchanged between corners. This is also true for $x < b_{r+1}$ and for $x > b_n$. So each breakpoint must occur at a corner of $\chi_A(x)$. □

This tells us that where the graph of $\chi_A(x)$ changes gradient, the x coordinate at this (break)point is equal to a corner, that is value from the set of corners $\{b_{r+1}, \ldots, b_n\}$ (where $b_{r+1}, \ldots, b_n$ are as in (4.3.4)).

We shall use corners later (Section 6.2), to find bounds on the coefficient of the characteristic max-polynomial. There is no need to find a bound for a coefficient of an essential term though, as we can efficiently calculate these:

Theorem 4.4 ([7])**.** If $A \in \overline{\mathbb{R}}^{n \times n}$, then as a function, (the essential terms of) $\chi_A(x)$ can be found in $\mathrm{O}(n^2(m + n \log n))$ steps, where A has m finite entries.

The method contained within [7] is based on ideas from computational geometry combined with solving the assignment problem. It finds all essential terms but none of the inessential terms. It is not known how to efficiently calculate the coefficient values of inessential terms.

The complexity has recently been improved by a factor of n to $\mathrm{O}(n(m+n\log n))$ steps [24].

Chapter 5

The best principal submatrix problem and its variants

This chapter will include the following. We will define the best principal submatrix problem (BPSM). This problem is to find the maximum assignment problem value of principal submatrices of A. BPSM is important, as it is very closely linked to finding the characteristic max-polynomial. We know of no polynomial method to solve this problem in general.

Possible applications of BPSM will be given. One such application is the job rotation problem, which involves reassigning jobs, maximising the total benefit of the job swaps.

A more general version of BPSM is given, where submatrices need not be principal. It is called the best submatrix problem (BSM). We will show that BSM is polynomially solvable.

Other problems related to BPSM will be given. Some of these help us find bounds to the optimal solution value of BPSM. We will explore the complexity of these related problems compared to BPSM.

5.1 The best submatrix problem (BSM)

Definition 5.1. Let $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$. A matrix of the form

$$\begin{pmatrix} a_{i_1j_1} & a_{i_1j_2} & \dots & a_{i_1j_k} \\ a_{i_2j_1} & a_{i_2j_2} & \dots & a_{i_2j_k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_kj_1} & a_{i_kj_2} & \dots & a_{i_kj_k} \end{pmatrix},$$

with $1 \leq i_1 < i_2 < \dots < i_k \leq n$ and $1 \leq j_1 < j_2 < \dots < j_k \leq n$, is called a $k \times k$ *submatrix* of A.

Problem 1. Given a matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and an integer $k \in N$, find the greatest max-algebraic permanent of all $k \times k$ submatrices of A.

Problem 1 will be referred to as the *best submatrix problem (BSM)*. It is equivalent to the task of finding the maximum sum of k independent elements of A (where *independent* means in a different row and column).

Let BSM(A, k) denote the problem of solving BSM for matrix A and a particular value of k.

Definition 5.2. For $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, the optimal solution value of BSM(A, k) will be denoted as $\gamma_{n-k}(A)$, or γ_{n-k} for short.

Theorem 5.3 ([16])**.** Given $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, BSM(A, k) can be solved (i.e. we can calculate $\gamma_{n-k}(A)$) in $O(n^3)$ time.

BSM(A, k) can be solved by constructing a new matrix $B = (b_{ij}) \in \overline{\mathbb{R}}^{(2n-k)\times(2n-k)}$ defined by

$$b_{ij} = \begin{cases} a_{ij}, & \text{for } i = 1, \ldots, n \text{ and } j = 1, \ldots, n \\ 0, & \text{for } i = 1, \ldots, n \text{ and } j = n+1, \ldots, 2n-k \\ 0, & \text{for } i = n+1, \ldots, 2n-k \text{ and } j = 1, \ldots, n \\ -\infty, & \text{for } i = n+1, \ldots, 2n-k \text{ and } j = n+1, \ldots, 2n-k. \end{cases}$$

So

$$B = \left(\begin{array}{ccc|ccc} a_{11} & \ldots & a_{1n} & 0 & \ldots & 0 \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & \ldots & a_{nn} & 0 & \ldots & 0 \\ \hline 0 & \ldots & 0 & -\infty & \ldots & -\infty \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & -\infty & \ldots & -\infty \end{array} \right).$$

We can then apply the Hungarian method on B in $O((2n-k)^3) = O(n^3)$ time, to find the biggest weight, with respect to B, of all permutations in P_n. Due to the construction of B, this is equal to the greatest sum of k independent entries of A, i.e. $\gamma_{n-k}(A)$.

For a full proof, the reader is referred to [16].

It is useful not only to calculate the value of $\gamma_{n-k}(A)$, but also to know which submatrix of A the optimal assignment is in, or which elements of A add together to give $\gamma_{n-k}(A)$. Both of these can be found from the above method.

An alternative version of BSM is the following:

Problem 2. Given a matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ and an integer $k \in N$, find the lowest max-algebraic permanent of all $k \times k$ submatrices of A.

Clearly, if any element of A is $-\infty$, then the lowest sum of any k independent elements will be $-\infty$. Otherwise, define $-A = (-a_{ij})$. In this case, the optimal solution value is simply $-\gamma_{n-k}(-A)$.

As any problem of this kind can be converted to the first kind, we shall look only at problems of the first kind.

5.2 The best principal submatrix problem (BPSM)

Definition 5.4. Let $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$. A matrix of the form

$$\begin{pmatrix} a_{i_1 i_1} & a_{i_1 i_2} & \dots & a_{i_1 i_k} \\ a_{i_2 i_1} & a_{i_2 i_2} & \dots & a_{i_2 i_k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_k i_1} & a_{i_k i_2} & \dots & a_{i_k i_k} \end{pmatrix},$$

with $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, is called a $k \times k$ *principal submatrix* of A.

Definition 5.5. Let $A(k)$ be the set of all $k \times k$ principal submatrices of A.

Problem 3. Given a matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ and an integer $k \in N$, find

$$\max_{B \in A(k)} \operatorname{maper}(B).$$

Problem 3 will be referred to as the *best principal submatrix problem (BPSM).* Let BPSM(A, k) denote the problem of solving BPSM for matrix A and a particular value of k, and let BPSM(A) denote the problem of solving BPSM for matrix A and all values of $k \in N$. Unlike BSM, in general, BPSM has no known polynomial solution method.

Solving BPSM(A) and finding the characteristic max-polynomial of A are closely related, as can be seen from the following theorem and discussion.

Theorem 5.6 ([19])**.** If $A \in \overline{\mathbb{R}}^{n \times n}$, then $(\forall k \in N)\ \delta_{n-k}(A) = \max_{B \in A(k)} \operatorname{maper}(B)$.

This means that $\delta_{n-k}(A)$ is the optimal solution value of BPSM(A, k).

We will test Theorem 5.6 with the following example.

Example 5.1. Let

$$A = \begin{pmatrix} 5 & 9 & 6 \\ 1 & 4 & 3 \\ 8 & 7 & 2 \end{pmatrix}.$$

Then

$$\max_{B \in A(1)} \operatorname{maper}(B) = 5,$$

$$\max_{B \in A(2)} \operatorname{maper}(B) = 6 + 8 = 14,$$

$$\max_{B \in A(3)} \operatorname{maper}(B) = \operatorname{maper}(A) = 9 + 3 + 8 = 20.$$

Also,

$$\begin{aligned}
\chi_A(x) = \ & \operatorname{maper} \begin{pmatrix} 5 \oplus x & 9 & 6 \\ 1 & 4 \oplus x & 3 \\ 8 & 7 & 2 \oplus x \end{pmatrix} \\
&= ((5 \oplus x) \otimes (4 \oplus x) \otimes (2 \oplus x)) \oplus ((5 \oplus x) \otimes 3 \otimes 7) \\
&\qquad \oplus (6 \otimes (4 \oplus x) \otimes 8) \oplus (9 \otimes 1 \otimes (2 \oplus x)) \\
&\qquad\qquad \oplus (9 \otimes 3 \otimes 8) \oplus (6 \otimes 1 \otimes 7) \\
&= x^{(3)} \oplus 5 \otimes x^{(2)} \oplus 14 \otimes x \oplus 20.
\end{aligned}$$

Therefore,

$$\delta_2(A) = \delta_{3-1}(A) = 5 = \max_{B \in A(1)} \operatorname{maper}(B)$$

$$\delta_1(A) = \delta_{3-2}(A) = 14 = \max_{B \in A(2)} \operatorname{maper}(B)$$

$$\delta_0(A) = \delta_{3-3}(A) = 20 = \max_{B \in A(3)} \operatorname{maper}(B).$$

So Theorem 5.6 does indeed hold for this example.

Solving BPSM(A) allows us to find $\chi_A(x)$. However, we do not need to solve BPSM for all values of $k \in N$, because if for a particular k, say k' we have that $\delta_{n-k'}(A) \otimes x^{(n-k')}$ is an inessential term, it is not needed for the description of $\chi_A(x)$ as a function, so we do not need to calculate BPSM(A, k').

We have that $\delta_{n-k}(A) = -\infty$ is equivalent to finding that maper(B) $= -\infty$ for all $B \in A(k)$. If this is the case, $\delta_{n-k}(A) \otimes x^{(n-k)} = -\infty$, hence $\delta_{n-k}(A) \otimes x^{(n-k)}$ is inessential, and by convention we can remove this term if considering $\chi_A(x)$ as a function.

From Theorem 5.6 we can easily compute $\delta_0(A) = \text{maper}(A)$ in $\mathrm{O}(n^3)$ steps and $\delta_{n-1}(A) = \max(a_{11}, a_{22}, \ldots, a_{nn})$ in $\mathrm{O}(n)$ steps. In general $\delta_{n-k}(A)$ can be computed by solving and comparing the assignment problem values for every $k \times k$ principal submatrix of A. There are $\binom{n}{k}$ matrices in $A(k)$, so this can be done in $\mathrm{O}\big(k^3 \cdot \binom{n}{k}\big)$ steps. If k is a fixed constant (independent of n), then $\delta_{n-k}(A)$ can be computed within a polynomial of n steps. But in general, this does not provide us with an efficient solution method.

If we had a polynomial method for solving BPSM, then we could find *all* coefficients of the max-algebraic characteristic polynomial, not just coefficients that belong to essential terms.

However, in general, it is not known how to polynomially solve BPSM, or if it is possible to do so. As we know how to find $\chi_A(x)$ (as a function) in $\mathrm{O}(n(m+n \log n))$ steps, (where A has m finite entries), by calculating all essential terms of $\chi_A(x)$, we can polynomially solve BPSM(A, k) for all essential k. If all terms of $\chi_A(x)$ are essential, then this polynomially solves BPSM(A). For some other special cases of matrix A, it is possible to solve BPSM(A) in polynomial time, as will be shown later (in Section 6.4).

It is useful not only to calculate the value of $\delta_{n-k}(A)$, but also to know which principal submatrix of A the optimal assignment is in, or which elements of A add together to give $\delta_{n-k}(A)$. Any method that finds $\delta_{n-k}(A)$, will need to check that there do exist k elements in a principal submatrix of A with sum equal to $\delta_{n-k}(A)$. So any method for solving BPSM can easily be adapted to find these extra pieces of information.

An alternative version of BPSM is the following:

Problem 4. Given a matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and an integer $k \in N$, find

$$\min_{B\in A(k)} \text{maper}(B).$$

Clearly, if any element of A is $-\infty$, then the lowest sum of any k independent elements of any principal submatrix of A will be $-\infty$. Otherwise, Define $-A = (-a_{ij})$. In this case, the optimal solution value of this problem is simply $-\delta_{n-k}(-A)$.

Again, as any problem of this kind can be converted to one from Problem 3, we shall look only at Problem 3.

5.3 Applications of BPSM

Now we give some examples of possible applications of BPSM.

5.3.1 The job rotation problem

The first application we give is the job rotation problem. We will explain this with the help of two examples:

Example 5.2. A theme park has n staff. To avoid boredom (which may reduce concentration and therefore decrease safety levels) it wants k staff to swap jobs amongst each other. Each person should still have one job afterwards.

They choose a $k \in N$ so that it is not too high, which may cause too much disruption, or be difficult or expensive to retrain so many people at once. Also, if they set k too low, then there may not have much effect on boredom, morale and safety levels.

The benefit of person i taking over person j's job has been calculated for each person based on the estimated increase in concentration levels and retraining costs. Which jobs should be assigned to each member of staff to maximise the combined benefit of the job swaps?

We define an $n \times n$ matrix $A = (a_{ij})$, by setting a_{ij} equal to the benefit of person i taking over the job of person j. To avoid a person i swapping jobs with themselves, we set a_{ii} to $-\infty$. If we solve BPSM(A, k) and element a_{ij} is selected then person i should now do person j's old job, (with those not selected remaining with the same job). The total benefit of the job swaps will be $\delta_{n-k}(A)$.

Example 5.3. A Mathematics department wishes to swap teaching duties of some lecturers. They think it is important to keep some stability in the department, so not all courses should have a new lecturer.

If each lecturer teaches exactly one course each, then to maximise total teaching effectiveness, we could use a similar approach to Example 5.2, with each entry of A related to a particular lecturer's interest (to teach) and ability (to prepare quickly / teach) in a particular course.

In the case where some lecturers have more than one course to teach, we proceed by defining $A = (a_{ij})$ as follows: If course i and course j are taught by different lecturers, then set a_{ij} to be course i's current lecturer's interest and ability to now teach course j instead. If course i and course j are taught by the same lecturer, then set a_{ij} to $-\infty$ (to ensure a lecturer can't swap teaching a course with himself).

If we solve BPSM(A, k) and element a_{ij} is selected then lecturer of course i should now teach course j (with courses not selected remaining being taught by the old lecturer). The total effect of the re-allocation of teaching duties will be $\delta_{n-k}(A)$.

The above will cause the number of courses a lecturer teaches to be the same as before the re-allocation of teaching duties. This is not always desired. A lecturer may want the number of courses he teaches to be decreased so that he can have more time to research. So we need to adapt this process slightly to cope with this.

The way we do this is by including "dummy courses". A dummy course requires no teaching, so if a lecturer is assigned to one of these, then he can use this time for research.

These are examples of the *job rotation problem.* In general this is where we have employees doing a total of n jobs between them, and we want to know the maximum benefit of re-assigning / rotating k jobs between the employees, $(k \in N)$, given that the benefit is known of each person being assigned each of the jobs.

There are similar examples where we may want to minimise total costs from a cost matrix. In this case, we simply define a benefit matrix to be the negative of the cost matrix and try to maximise the total benefit. This idea also applies to the other applications that we will give.

5.3.2 Other applications

Swapping jobs is not the only application. In general if we have a cost / benefit matrix $A = (a_{ij})$, where a_{ij} is the cost / benefit of i performing some action associated with j, then we may be able to use BPSM to find the total cost / benefit of k such actions. Such actions could be "perform j's current job" as in the case of the job rotation problem, or may be "play a football match at home to team j" or "e-mail person j". The optimal permutation for BPSM selects exactly one element in each row and column of a principal submatrix. Therefore we must also have that there exist a j that i performs an action on if and only if there exist a k that performs an action on i. This means we will have "chains" of actions between people / objects, with each performing an action on the next. More precisely, these "chains" are actually cycles.

Example 5.4. An Art teacher has n students, and wants them to practise drawing portraits of each other. Only k of the students can draw portraits, due to constraints on equipment, time, etc. She decides that each of the k portrait takers should also have their portraits taken, so that the rest of the class can do other work. Each person should take no more than one portrait, and have no more than one portrait of themselves taken.

Due to past experience with the students, the teacher knows the strengths and weaknesses of each student, and in particular, can rate how successful each student will be at drawing each of the faces. She wants to assign k students so that the sum of the ratings of the students selected is as high as possible. She needs to know who should draw whom.

We define an $n \times n$ matrix $A = (a_{ij})$, by setting a_{ij} equal to student i's rating for drawing the portrait of student j. If self-portraits are not allowed, we may set a_{ii} to $-\infty$ for all i. If we solve BPSM(A, k), then element a_{ij} will be selected if and only if student i should draw student j's portrait. The total rating of the drawings will be given by $\delta_{n-k}(A)$.

Example 5.5. A company manager decides to hold an internal assessment of some of its staff. To avoid increasing workloads too much, it is decided to select k of its n staff to assess each other. There should be at most one assessment done by a person, and at most one assessment done on a person.

Each person is better at assessing the work of another if they are working (or have worked) in a similar area of the company, doing similar tasks. For that reason, the manager has judged how well each member of staff will be at assessing every other member of staff. He wants to choose k staff, so that the total assessment is as accurate as possible. Which staff should he choose to assess who?

We define an $n \times n$ matrix $A = (a_{ij})$, by setting a_{ij} equal to person i's ability for assessing the work of person j. If self-assessments are not allowed, we may set a_{ii} to $-\infty$ for all i. If we solve BPSM(A, k), then element a_{ij} will be selected if and only if person i should assess person j's work. The manager can have some indication to how accurate the total assessment of the k staff is by looking at how high the value of $\delta_{n-k}(A)$ is.

5.4 Similar problems to BPSM

The aim of this section is to investigate some interesting problems that are closely related to BPSM. We shall look at whether these similar problems can help us solve BPSM (or vise versa).

We have already seen the BSM problem, for which we maximise the assignment problem value over all submatrices (not just principal ones). This change causes BSM to be polynomially solvable, where as BPSM may not be polynomially solvable. BSM immediately gives us an upper bound for BPSM. Some of the similar problems will also provide us with upper bounds for BPSM.

We will consider the problem of finding the greatest cyclic permutation mean of fixed or variable length. Other problems are where we allow the size of the principal submatrix to be between 1 and k, or between k and n, or between k_{min} and k_{max}, instead of just having $k \times k$ submatrices.

We shall also look at bottleneck versions of some of these problems, where instead of maximising the total of the elements given by some permutation, we maximise the smallest of these elements.

Recall $N = \{1, 2, \ldots, n\}$ and $K = \{1, 2, \ldots, k\}$ and the section on permutations (Section 2.3, page 15).

5.4.1 The greatest cyclic permutation mean of length k

Consider the following problems:

Problem 5. Given $A \in \overline{\mathbb{R}}^{n \times n}$ and $k \in N$, find $\max\limits_{\sigma \in C_n^k} \mu(A, \sigma)$.

Problem 5 seeks to find the greatest mean weight, with respect to A, of all cyclic permutations in C_n^k (i.e. of all cyclic permutations in C_n of length k).

Problem 6. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find $\max\limits_{\sigma \in C_n^k} w(A,\sigma)$.

Problem 6 seeks to find the greatest weight, with respect to A, of all cyclic permutations in C_n^k.

Theorem 5.7. If we can solve Problem 5 then we can solve Problem 6 and vice versa.

Proof. In both problems, all cyclic permutations have a fixed length of k, so it follows that the optimal solution to Problem 6 is k times the optimal solution of Problem 5. □

Theorem 5.8. The optimal solution to Problem 6 acts as a lower bound to $\delta_{n-k}(A)$.

Proof. We have

$$\begin{aligned}\max_{\sigma \in C_n^k} w(A,\sigma) &= \max_{B \in A(k)} \max_{\sigma' \in C_k^k} w(B,\sigma') \\ &\leq \max_{B \in A(k)} \max_{\pi \in P_k} w(B,\pi) \\ &= \delta_{n-k}(A),\end{aligned}$$

as $C_k^k \subseteq P_k$. Hence result. □

Remark. If $w(B,\sigma) = \delta_{n-k}$ for some $B \in A(k)$ and some cyclic permutation $\sigma \in C_k^k$, then the optimal solution to Problem 6 is δ_{n-k} and the optimal solution to Problem 5 is δ_{n-k}/k.

Remark. Unfortunately, Problem 6 (and 5) are *NP*-complete. To see this, set k to n. We are then trying to find the maximum weight of a cyclic permutation that includes all integers from 1 to n. This is equivalent to the Travelling Salesman Problem, which is well known to be *NP*-complete.

5.4.2 The greatest cyclic permutation mean of length k or less

Now we look at a modified version of Problem 5:

Problem 7. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find $\max\limits_{i\in K} \max\limits_{\sigma\in C_n^i} \mu(A,\sigma)$.

Problem 7 seeks to find the greatest mean weight, with respect to A, of all cyclic permutations on N of length k or less.

Problem 8. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find $\max\limits_{B\in A(k)} \lambda(B)$.

Problem 8 seeks to find the greatest elementary cycle mean from all digraphs of $k \times k$ principal submatrices of A.

Theorem 5.9. Problem 7 and Problem 8 are equivalent.

Proof. We have

$$\begin{aligned}\max_{i\in K} \max_{\sigma\in C_n^i} \mu(A,\sigma) &= \max_{i\in K} \max_{B\in A(i)} \max_{\sigma'\in C_i^i} \mu(B,\sigma') \\ &= \max_{B\in A(k)} \max_{\sigma''\in C_k} \mu(B,\sigma'') \\ &= \max_{B\in A(k)} \lambda(B),\end{aligned}$$

hence the result follows. □

Lemma 5.10. If $A \in \overline{\mathbb{R}}^{n\times n}, k \in N$ and $\pi = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_s \in P_k$, then $\mu(A,\pi) \leq \max\limits_{i=1,\dots,s} \mu(A,\sigma_i)$.

Proof. Let $\mu(A,\sigma^*) = \max\limits_{i=1,\dots,s} \mu(A,\sigma_i)$. Then

$$\begin{aligned}
\mu(A,\pi) &= \mu(A,\sigma_1\circ\sigma_2\circ\cdots\circ\sigma_s)\\
&= \frac{w(A,\sigma_1\circ\sigma_2\circ\cdots\circ\sigma_s)}{k}\\
&= \frac{w(A,\sigma_1)+w(A,\sigma_2)+\cdots+w(A,\sigma_s)}{k}\\
&= \frac{\mu(A,\sigma_1)l(\sigma_1)+\mu(A,\sigma_2)l(\sigma_2)+\cdots+\mu(A,\sigma_s)l(\sigma_s)}{k}\\
&\le \frac{\mu(A,\sigma^*)l(\sigma_1)+\mu(A,\sigma^*)l(\sigma_2)+\cdots+\mu(A,\sigma^*)l(\sigma_s)}{k}\\
&= \mu(A,\sigma^*)\frac{l(\sigma_1)+l(\sigma_2)+\cdots+l(\sigma_s)}{k}\\
&= \mu(A,\sigma^*),
\end{aligned}$$

which completes the proof. □

Theorem 5.11. The optimal solution to Problem 7 (and 8) equals

$$\max_{r\in K}\frac{\delta_{n-r}(A)}{r}. \tag{5.4.1}$$

Proof. Note that $\max_{B\in A(k)}\lambda(B)$ is the greatest cyclic permutation mean, with respect to $r\times r$ principal submatrices of A (for all $r\le k$), from all cyclic permutations of length r. Also note that $\max_{r\in K}\frac{\delta_{n-r}(A)}{r}$ is the greatest permutation mean, with respect to $r\times r$ principal submatrices of A (for all $r\le k$), from all permutations of length r. Therefore, we have the following equations.

$$\begin{aligned}
\max_{B\in A(k)}\lambda(B) &= \max_{r\in K}\ \max_{B\in A(r)}\ \max_{\sigma\in C_r^r}\mu(B,\sigma),\\
\max_{r\in K}\frac{\delta_{n-r}(A)}{r} &= \max_{r\in K}\ \max_{B\in A(r)}\ \max_{\pi\in P_r}\mu(B,\pi)=\mu(B^*,\pi^*)\text{ say,}
\end{aligned}$$

where $\pi^*=\sigma_1\circ\sigma_2\circ\cdots\circ\sigma_s$.

Then by applying Lemma 5.10, we have

$$\mu(B^*,\pi^*)\le\max_{i=1,\dots,s}\mu(B^*,\sigma_i)=\mu(B^*,\sigma^*)\text{ say.}$$

However, as

$$\mu(B^*, \sigma^*) \leq \frac{\delta_{n-l(\sigma^*)}(A)}{l(\sigma^*)} \leq \max_{r\in K} \frac{\delta_{n-r}(A)}{r} = \mu(B^*, \pi^*),$$

we must have that $\mu(B^*, \pi^*) = \mu(B^*, \sigma^*)$. Also, because $C_r^r \subseteq P_r$, we have that $\mu(B^*, \sigma^*) = \max\limits_{r\in K} \max\limits_{B\in A(r)} \max\limits_{\sigma\in C_r^r} \mu(B, \sigma)$, so

$$\begin{aligned}\max_{r\in K} \frac{\delta_{n-r}(A)}{r} &= \max_{r\in K} \max_{B\in A(r)} \max_{\pi\in P_r} \mu(B, \pi) \\ &= \mu(B^*, \pi^*) \\ &= \mu(B^*, \sigma^*) \\ &= \max_{r\in K} \max_{B\in A(r)} \max_{\sigma\in C_r^r} \mu(B, \sigma) \\ &= \max_{B\in A(k)} \lambda(B),\end{aligned}$$

which completes the proof. □

Unfortunately, we do not know how to find δ_{n-r} in general, so we need to use a different approach to solve Problem 7 (and 8):

Theorem 5.12. The optimal solution to Problem 7 (and 8) equals

$$\max_{i\in N} \max_{j\in K} \frac{a_{ii}^j}{j}, \tag{5.4.2}$$

(where as before, $A^{(r)} = (a_{ij}^r)$).

Proof. Observe that $\max\limits_{i\in N} \frac{a_{ii}^j}{j}$ is the maximum mean weight of all (not necessarily elementary) cycles of length j in $D(A)$. So $\max\limits_{i\in N} \max\limits_{j\in K} \frac{a_{ii}^j}{j}$ is the maximum mean weight of all (not necessarily elementary) cycles of length no more than k in $D(A)$.

Let the maximum mean weight of all cycles of length no more than k in $D(A)$ be given by λ_k. Let $\sigma_{opt}(k) = \{\sigma : l(\sigma) \leq k, \mu(A, \sigma) = \lambda_k\}$. If there is an elementary cycle in $\sigma_{opt}(k)$, then the result holds.

Assume this is not the case. Then let σ be a cycle of smallest length in $\sigma_{opt}(k)$. We can decompose σ into elementary cycles. If any of these elementary cycles (say σ') has a cycle mean greater than λ_k, i.e. $\mu(\sigma') > \lambda_k$, then this would contradict the definition of λ_k. If any of these elementary cycles has a cycle mean equal to λ_k, then we may remove this elementary cycle from σ and have a shorter cycle σ'' with $\mu(\sigma'') = \lambda_k$, which contradicts the definition of σ.

If any of these elementary cycles (say θ) has a cycle mean less than λ_k, then removing this θ from σ forms a cycle σ''' of length less than k. Let

$$\lambda_k = \frac{w(\sigma_1) + \cdots + w(\sigma_s) + w(\theta)}{l(\sigma_1) + \cdots + l(\sigma_s) + l(\theta)}.$$

Then as $\mu(\theta) = \dfrac{w(\theta)}{l(\theta)} < \lambda_k$, we have

$$\begin{aligned}
& & \lambda_k &< \frac{w(\sigma_1) + \cdots + w(\sigma_s) + l(\theta)\lambda_k}{l(\sigma_1) + \cdots + l(\sigma_s) + l(\theta)} \\
&\iff & \lambda_k(l(\sigma_1) + \cdots + l(\sigma_s) + l(\theta)) &< w(\sigma_1) + \cdots + w(\sigma_s) + l(\theta)\lambda_k \\
&\iff & \lambda_k(l(\sigma_1) + \cdots + l(\sigma_s)) &< w(\sigma_1) + \cdots + w(\sigma_s) \\
&\iff & \lambda_k &< \frac{w(\sigma_1) + \cdots + w(\sigma_s)}{l(\sigma_1) + \cdots + l(\sigma_s)} \\
& & &= \mu(\sigma''').
\end{aligned}$$

So we have $\mu(\sigma''') > \lambda_k$, which contradicts the definition of λ_k.

Hence there is an elementary cycle in $\sigma_{opt}(k)$ and so the result holds. $\square$

Remark. Note that (5.4.1) and (5.4.2) allow us to find an upper bound for $\delta_{n-k}(A)$ of $k \max_{i \in N} \max_{j \in K} \dfrac{a_{ii}^j}{j}$, which can be found in polynomial time.

In the next two problems, we generalise Problem 7 (and 8) by removing the principality constraint:

Problem 9. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find the greatest cycle mean from all elementary cycles of digraphs of submatrices of A of length no greater than k.

Problem 10. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, maximise $\lambda(B)$ over all $k\times k$ submatrices B of A.

Theorem 5.13. Problem 9 and Problem 10 are equivalent.

Proof. Problem 10 looks for the greatest cycle mean of elementary cycles in the digraphs of all $k \times k$ submatrices of A. Note that, σ is an elementary cycle in the digraph of a $k \times k$ submatrix of A, if and only if, σ is an elementary cycle in the digraph of an $l(\sigma) \times l(\sigma)$ submatrix of A with $l(\sigma) \leq k$. Hence result. $\square$

Recall that the optimal solution value of BSM(A, r) is denoted as $\gamma_{n-r}(A)$, or γ_{n-r} for short.

Theorem 5.14. The optimal solution to Problem 9 (and 10) equals

$$\max_{r\in K}\frac{\gamma_{n-r}}{r}.$$

Proof. Note that $\max_{r\in K}\frac{\gamma_{n-r}}{r}$ is the greatest permutation mean, with respect to B, from all permutations $\pi \in P_r$, where B is an $r \times r$ submatrix of A, and $r \in K$. Let $s \in K$ be the smallest value of r such that

$$\frac{\gamma_{n-s}}{s} = \max_{r\in K}\frac{\gamma_{n-r}}{r}.$$

Then since s is smallest, we must have (by Lemma 5.10) that $\gamma_{n-s} = w(B, \sigma')$, where $\sigma' \in C_s^s$ is some cyclic permutation and B is some $s \times s$ submatrix of A, (for some $s \in K$). Therefore $\frac{\gamma_{n-s}}{s}$ is the greatest cyclic permutation mean, with respect to B, from all cyclic permutations $\sigma \in C_r^l$, where B is an $r \times r$ submatrix of A and $1 \leq l \leq r \leq k$. This is equivalent to the value Problem 9 (and therefore also to Problem 10) is looking for. Hence result. $\square$

As we can find γ_{n-r} in polynomial time (by Theorem 5.3), we can also solve Problem 9 (and 10) in polynomial time.

5.4.3 BPSM$_{\leq}(A, k)$

BPSM finds the biggest sum of exactly k independent elements of all $k \times k$ principal submatrices of A. We have already shown that the job rotation problem is an application of this, where we swap k jobs round. It may be more useful to swap up to k jobs. This is equivalent to finding the biggest sum of l independent elements of all $l \times l$ principal submatrices of A, where $1 \leq l \leq k$.

Formally, it is the following problem:

Problem 11. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find

$$\max_{l\in K} \max_{B\in A(l)} \operatorname{maper}(B).$$

We refer to this problem as BPSM$_{\leq}(A, k)$. It is not known how to polynomially solve it in general.

Remark. Note that for $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$,

$$\max_{l\in K} \max_{B\in A(l)} \operatorname{maper}(B) = \max_{l\in K} \delta_{n-l}(A).$$

So if we know $\delta_{n-l}(A)$ for all $l \leq k$, (for example when $\delta_{n-l}(A) \otimes x^{(n-l)}$ are essential terms of $\chi_A(x)$, for all $l \leq k$), then we can polynomially solve BPSM$_{\leq}(A, k)$.

We can also solve BPSM$_{\leq}(A, k)$ if A has no positive cycles in $D_C(A)$:

Theorem 5.15. If $\lambda(A) \in [-\infty, 0]$, the optimal solution to BPSM$_{\leq}(A, k)$ is given by

$$\max_{l\in K} \max_{B\in A(l)} \operatorname{maper}(B) = \max_{l\in K} \max_{i\in N} a^{l}_{ii}. \tag{5.4.3}$$

Proof. Assume the optimal solution is $\max\limits_{l\in K} \max\limits_{B\in A(l)} \operatorname{maper}(B) = w(C,\pi)$, for some $C \in A(l), (l \in K)$ and $\pi = \sigma_1 \circ \cdots \circ \sigma_t \in P_l$. As there are no positive cycles in $D_C(A)$, we have $w(C,\sigma_i) \leq 0$, for $1 \leq i \leq t$. Without loss of generality let $w(C,\sigma_1) = \max\limits_{1\leq i\leq t} w(C,\sigma_i)$, then as (for $2 \leq i \leq t$) $w(C,\sigma_i) \leq 0$, we have

$$\begin{aligned} w(C,\pi) &= w(C,\sigma_1) + \cdots + w(C,\sigma_t) \\ &\leq w(C,\sigma_1) \\ &\leq \max_{l\in K} \max_{B\in A(l)} \max_{\sigma\in C_l^l} w(B,\sigma) \\ &\leq \max_{l\in K} \max_{B\in A(l)} \max_{\pi'\in P_l} w(B,\pi') \\ &= \max_{l\in K} \max_{B\in A(l)} \operatorname{maper}(B) \\ &= w(C,\pi). \end{aligned}$$

Therefore we can change all "$\leq$" to "$=$" in the above. So,

$$\begin{aligned} \max_{l\in K} \max_{B\in A(l)} \operatorname{maper}(B) &= \max_{l\in K} \max_{B\in A(l)} \max_{\sigma\in C_l^l} w(B,\sigma) \\ &= \max_{l\in K} \max_{\sigma\in C_n^l} w(A,\sigma). \end{aligned}$$

This means that $\max\limits_{l\in K} \max\limits_{B\in A(l)} \operatorname{maper}(B) = \max\limits_{l\in K} \max\limits_{\sigma\in C_n^l} w(A,\sigma)$ equals the maximum weight, with respect to some A, of all cyclic permutations on N of length no more than k.

We have that a_{ii}^l is the maximum weight of all (not necessarily elementary) cycles of $D_C(A)$ of length l that pass through node i. This is equal to the sum of weights of elementary cycles of $D_C(A)$ that form such a cycle, say $\sigma_1^{'}, \ldots, \sigma_{t'}^{'}$. So without loss of generality let $a_{ii}^l = w(A,\sigma_1^{'}) + \cdots + w(A,\sigma_{t'}^{'})$ with $\sigma_1^{'}$ containing node i. Elementary cycle $\sigma_1^{'}$ has length no more than l and passes through node i. So

$$\begin{aligned} a^l_{ii} &= w(A,\sigma_1') + \cdots + w(A,\sigma_{t'}') \\ &\le w(A,\sigma_1') \end{aligned}$$

as all elementary cycles $\sigma_2', \ldots, \sigma_{t'}'$ have non-positive weight. We know $a^l_{ii} \nless w(A,\sigma_1')$ (by the definition of a^l_{ii}, as σ_1' is a cycle of length no more than l that passes through node i), so therefore we must have $a^l_{ii} = w(A,\sigma_1')$. This means a^l_{ii} is the maximum weight of all elementary cycles of $D_C(A)$ of length l that pass through node i.

Therefore $\max\limits_{l\in K} \max\limits_{i\in N} a^l_{ii}$ will be the maximum weight over all elementary cycles of $D_C(A)$ of length no more than k. This will give the same value as $\max\limits_{l\in K} \max\limits_{\sigma\in C^l_n} w(A,\sigma)$. Hence result. □

Remark. If $\lambda(A) \in [-\infty, 0]$ and l^* is the minimum length of all the (elementary) cycles that have the maximum weight w^*, then for $k \ge l^*$,

$$\begin{aligned} \max_{l\in K} \max_{B\in A(l)} \text{ maper}(B) &= \max_{l\in K} \max_{i\in N} a^l_{ii} \\ &= \max_{1\le l\le l^*} \max_{i\in N} a^l_{ii} \\ &= \max_{i\in N} a^{l^*}_{ii} \\ &= w^*. \end{aligned}$$

Thus if l^* and w^* are already known, $\lambda(A) \in [-\infty, 0]$ and $k \ge l^*$, then we have $\max\limits_{l\in K} \max\limits_{B\in A(l)} \text{maper}(B) = w^*$ (without needing to calculate $\max\limits_{l\in K} \max\limits_{i\in N} a^l_{ii}$). So

$$\begin{aligned} w^* &= \max_{l\in\{1,\ldots,l^*\}} \max_{B\in A(l)} \text{ maper}(B) \\ &= \max_{l\in\{1,\ldots,l^*+1\}} \max_{B\in A(l)} \text{ maper}(B) \\ &= \ldots \\ &= \max_{l\in\{1,\ldots,n\}} \max_{B\in A(l)} \text{ maper}(B) = \max_{l\in N} \delta_{n-l}(A). \end{aligned}$$

Remark. If there are no positive cycles in $D(A)$, we can find $\max\limits_{l\in N} \delta_{n-l}(A)$, the maximum value of coefficients in the characteristic max-polynomial, by calculating $\max\limits_{l\in K} \max\limits_{i\in N} a^l_{ii}$, the optimal value of BPSM$_{\leq}(A,n)$. (This also means $\max\limits_{l\in K} \max\limits_{i\in N} a^l_{ii}$ is an upper bound for any $\delta_{n-k}(A)$ if there are no positive cycles in $D(A)$.)

If instead there is at least one non-negative cycle in $D(A)$, then by (4.3.1),

$$\begin{aligned}\chi_A(0) &= \max\left(\max_{l\in N} \delta_{n-l}(A),\ 0\right)\\ &= \max_{l\in N} \delta_{n-l}(A)\end{aligned}$$

will be the maximum value of coefficients in the characteristic max-polynomial. (This also means $\chi_A(0)$ is an upper bound for any $\delta_{n-k}(A)$ if there is at least one non-negative cycle in $D(A)$.)

Definition 5.16. Let $\mathbb{T}$ be the set $\{0,-\infty\}$, and $\mathbb{T}^{n\times n}$ be the set of $n\times n$ matrices with entries from $\mathbb{T}$.

We will now look at the following problem and show how this problem is related to BPSM$_{\leq}(A,k)$.

Problem 12. Given $A\in\mathbb{T}^{n\times n}$ and $k\in N$, does there exists an (elementary) cycle in $D(A)$ of length no more than k?

Remark. If $A\in\mathbb{T}^{n\times n}$ then $\lambda(A)\in[-\infty,0]$ and we can use BPSM$_{\leq}(A,k)$ to solve Problem 12:

Corollary 5.17. The answer to Problem 12 is "yes" if the result given by (5.4.3) is 0, and "no" if it is $-\infty$.

Remark. If the words "no more than" are removed from Problem 12, then we do not know how to polynomially solve this problem. In fact, setting k to n and removing the words "no more than" would result in an equivalent problem to the Hamiltonian cycle problem, which is well known to be *NP*-complete.

Here is a more general version of Problem 12, which turns out to be useful in finding $\delta_{n-k}(A)$ for some values of k.

Problem 13. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, does there exists an (elementary) cycle in $D(A)$ of length no more than k?

Remark. If $A' = (a'_{ij}) \in \mathbb{T}^{n\times n}$ is defined by $a'_{ij} = 0$ if $a_{ij} \in \mathbb{R}$ and $a'_{ij} = -\infty$ otherwise, then we may use the solution method of Problem 12 on A' and k to solve Problem 13 on A and k.

Remark. Alternatively, using the solution method to the shortest cycle problem [31], we can calculate the shortest finite cycle in $A \in \overline{\mathbb{R}}^{n\times n}$, of, say, length l_{min}. We let k_{min} be the smallest value of k such that there exists k independent finite elements that lie within a $k \times k$ principal submatrix of A. We shall assume k_{min} exists, else it is trivial. So $k_{min} = l_{min}$.

For $k < k_{min}$, the answer to Problem 13 is "no". For $k \geq k_{min}$, the answer to Problem 13 is "yes". Hence $\delta_{n-k}(A) = -\infty$ for $1 \leq k < k_{min}$.

The value of $\delta_{n-k_{min}}(A)$ is finite, and we can easily find this by calculating the value of $\max_{i\in N} a_{ii}^{k_{min}}$.

It is easily seen that Problem 13 is equivalent to the following.

Problem 14. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, decide if there exists l finite independent entries in any $B \in A(l)$, for any $l \in \{1, \ldots, k\}$.

5.4.4 BPSM$_{\geq}(A, k)$

In a similar way to BPSM$_{\leq}(A, k)$, it may be more useful to swap at least k jobs in the job rotation problem (and less than the maximum number of jobs, which is n). This is equivalent to finding the biggest sum of l independent elements of all $l \times l$ principal submatrices of A, where $k \leq l \leq n$.

Formally, it is the following problem:

Problem 15. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find

$$\max_{k\leq l\leq n}\ \max_{B\in A(l)}\ \text{maper}(B).$$

We refer to this problem as BPSM$_{\geq}(A,k)$. It is not known how to solve it in general. It may be helpful to consider the following restricted problem (Problem 16). In particular, if $A \in \mathbb{T}^{n\times n}$, then we can use Problem 16 to solve BPSM$_{\geq}(A,k)$ for this type of matrix.

Problem 16. Given $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, decide if there exists l finite independent entries in any $B \in A(l)$, for any $l \in \{k,\ldots,n\}$.

Remark. If $A' = (a'_{ij}) \in \mathbb{T}^{n\times n}$ is defined by $a'_{ij} = 0$ if $a_{ij} \in \mathbb{R}$ and $a'_{ij} = -\infty$ otherwise, then by [13], we can find the maximum number of independent finite entries in $A' \in \mathbb{T}^{n\times n}$, say, k_{max}, that form disjoint cycle(s) in A'. This states that

$$k_{max} = n + \chi_{A'}(-1).$$

Note that l_{max}, the longest elementary cycle in $D(A)$, is not necessarily equal to k_{max} (unlike $k_{min} = l_{min}$ always being true).

For $k > k_{max}$, the answer to Problem 16 is “no”. For $k \leq k_{max}$, the answer to Problem 16 is “yes”. Hence $\delta_{n-k}(A) = -\infty$ for $k_{max} < k \leq n$, and $\delta_{n-k_{max}}(A)$ is finite, and can easily be found as follows.

Let

$$x_0 = -n\max(0, A_{max}, -A_{min}, A_{max} - A_{min}),$$

where A_{max} and A_{min} are the maximum and minimum finite entries of A respectively. Then due to a result in [16] which uses the fact that $\delta_{n-k_{max}}(A)$ is the first finite term in $\chi_A(x)$, so will always be visible, and can be found using small enough x, we have

$$\delta_{n-k_{max}}(A) = \chi_A(x_0) - k_{max}x_0.$$

5.4.5 BPSM$_{Range}(A, k_{min}, k_{max})$

BPSM finds the biggest sum of exactly k independent elements of all $k \times k$ principal submatrices of A. We have already shown that the job rotation problem is an application of this, where we swap exactly k jobs round. It may be more useful to swap k jobs round, where k is no longer a fixed number, but can lie anywhere in a range, say between k_{min} and k_{max}. This is equivalent to finding the biggest sum of k independent elements of all $k \times k$ principal submatrices of A, where $k_{min} \leq k \leq k_{max}$.

Formally, it is the following problem:

Problem 17. Given $A \in \overline{\mathbb{R}}^{n \times n}$ and $k_{min}, k_{max} \in N$, $k_{min} \leq k_{max}$, find

$$\max_{k \in [k_{min}, k_{max}]} \max_{B \in A(k)} \operatorname{maper}(B).$$

We refer to this problem as BPSM$_{Range}(A, k_{min}, k_{max})$. It is not known how to polynomially solve it in general. However, we have the following result:

Theorem 5.18. BPSM(A, k) and BPSM$_{Range}(A, k_{min}, k_{max})$ are polynomially equivalent problems.

Proof. If we can solve BPSM$_{Range}(A, k_{min}, k_{max})$ in polynomial time then by setting k_{min} and k_{max} equal to k, we can solve BPSM(A, k) in polynomial time, as BPSM$_{Range}(A, k_{min}, k_{max})$ becomes identical to BPSM(A, k).

If we can solve BPSM(A, k) in polynomial time, then solve BPSM(A, k) for $k = k_{min}, \ldots, k_{max}$. As we do $k_{max} - k_{min} + 1 \leq n$ repetitions, we can solve BPSM$_{Range}(A, k_{min}, k_{max})$ in polynomial time. □

5.4.6 Bottleneck BPSM (BBPSM)

Example 5.6. In Example 5.3 (an application of BPSM), we set a_{ij} to be lecturer i's ability to take over lecturer j's course. We had to find k elements of a $k \times k$ principal submatrix of $A = (a_{ij})$ with maximum sum. So the sum of these k elements was the total ability of the lecturers who swapped teaching duties.

If instead of this, the department decides that the minimum standard of teaching should be as high as possible, then we must look for k elements of a $k \times k$ principal submatrix of $A = (a_{ij})$ with the smallest of these k elements being as big as possible. From these elements, we would then be able to assign lecturers to courses maximising the worst lecturer's ability.

This is an example of the Bottleneck BPSM problem, which we will refer to as BBPSM(A, k). It is defined formally as follows:

Problem 18. Given $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ and $k \in N$, find

$$\max_{(b_{ij}) \in A(k)} \max_{\pi \in P_k} \min_{r \in K} b_{r,\pi(r)}.$$

No method is known to solve BBPSM in polynomial time. However, the following discussion provides us with an upper bound.

Define $A'_d = (a'_{ij}(d)) \in \overline{\mathbb{R}}^{n \times n}$ by

$$a'_{ij}(d) = \begin{cases} a_{ij}, & \text{for } a_{ij} \geq d \\ -\infty, & \text{otherwise} \end{cases}$$

If we can find a d^* such that there are k finite independent entries in a $k \times k$ principal submatrix of A'_{d^*}, and there are not k finite independent entries in a $k \times k$

principal submatrix of A'_d for any $d > d^*$, then we will have

$$d^* = \max_{(b_{ij})\in A(k)} \max_{\pi\in P_k} \min_{r\in K} b_{r,\pi(r)},$$

the optimal solution value to BBPSM(A, k).

Unfortunately, it is not known how to do this in polynomial time. However, it is possible to use Problem 14 to find an upper bound d_1 for d^* and Problem 16 to find another upper bound d_2 for d^*, so that we can say

$$\max_{(b_{ij})\in A(k)} \max_{\pi\in P_k} \min_{r\in K} b_{r,\pi(r)} \leq \min(d_1, d_2).$$

It would be nice to find a lower bound for d^*, but it may not be that easy. If we found a lower bound other than $-\infty$ in polynomial time, then this would mean there are k finite independent entries within a $k \times k$ principal submatrix of A. This would mean we could solve BPSM for matrices in $\mathbb{T}^{n\times n}$ in polynomial time. No such method is currently known to exist.

Remark. If we could solve BPSM(A, k) in polynomial time, then we could solve BPSM(A'_d, k) for various values of d, and complete the above method. Hence for $A \in \overline{\mathbb{R}}^{n\times n}$, if we can solve BPSM$(A, k)$, then we can also solve BBPSM(A, k). Later, we will go on to polynomially solve BPSM for some special types of matrix, and it follows that BBPSM is also polynomially solvable for the same types of matrix.

Also note, if we restrict A to be in $\mathbb{T}^{n\times n}$, then BBPSM(A, k) becomes equivalent to BPSM(A, k).

5.4.7 BBPSM$_\leq(A, k)$

If in Example 5.6 we wish to maximise the worst lecturer's ability, but have up to k job swaps taking place, then we would have an example of the BBPSM$_\leq(A, k)$ problem, defined as follows:

Problem 19. Given $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find

$$\max_{l \in K} \max_{(b_{ij})\in A(l)} \max_{\pi \in P_l} \min_{1\le r\le l} b_{r,\pi(r)}.$$

Again, define $A'_d = (a'_{ij}(d)) \in \overline{\mathbb{R}}^{n\times n}$ by

$$a'_{ij}(d) = \begin{cases} a_{ij}, & \text{for } a_{ij} \ge d \\ -\infty, & \text{otherwise} \end{cases}$$

If we can find a d^* such that there are l finite independent entries in a $l \times l$ principal submatrix of A'_{d^*}, for some $l = 1, \dots, k$, and there are not l finite independent entries in a $l \times l$ principal submatrix of A'_d for any $l = 1, \dots, k$ and $d > d^*$, then we will have

$$d^* = \max_{l \in K} \max_{(b_{ij})\in A(l)} \max_{\pi \in P_l} \min_{1\le r\le l} b_{r,\pi(r)},$$

the optimal solution value to BBPSM$_\le(A, k)$.

Unlike in the BBPSM case, we can do this. We can use Problem 14 to test if the above holds for some value of d^*, changing it up or down depending on which of the two parts of the statement fail to hold. This means we can polynomially solve BBPSM$_\le(A, k)$.

Remark. If we restrict A to be in $\mathbb{T}^{n\times n}$, then BBPSM$_\le(A, k)$ becomes equivalent to BPSM$_\le(A, k)$. This gives us an alternative method to solve BPSM$_\le(A, k)$ for $A \in \mathbb{T}^{n\times n}$ than the one given before.

5.4.8 BBPSM$_\ge(A, k)$

We define BBPSM$_\ge(A, k)$ as follows:

Problem 20. Given $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, find

$$\max_{k \le l \le n} \max_{(b_{ij})\in A(l)} \max_{\pi \in P_l} \min_{1\le r\le l} b_{r,\pi(r)}.$$

Again, define $A'_d = (a'_{ij}(d)) \in \overline{\mathbb{R}}^{n\times n}$ by

$$a'_{ij}(d) = \begin{cases} a_{ij}, & \text{for } a_{ij} \geq d \\ -\infty, & \text{otherwise} \end{cases}$$

If we can find a d^* such that there are l finite independent entries in a $l \times l$ principal submatrix of A'_{d^*}, for any $l = k, \ldots, n$, and there are not l finite independent entries in a $l \times l$ principal submatrix of A'_d for any $l = k, \ldots, n$ and $d > d^*$, then we will have

$$d^* = \max_{k\leq l\leq n} \max_{(b_{ij})\in A(l)} \max_{\pi\in P_l} \min_{1\leq r\leq l} b_{r,\pi(r)},$$

the optimal solution value to BBPSM$_{\geq}(A, k)$.

Like in the BBPSM$_{\leq}(A, k)$ case, but unlike in the BBPSM case, we can do this. We can use Problem 16 to test if the above holds for some value of d^*, changing it up or down depending on which of the two parts of the statement fail to hold. This means we can polynomially solve BBPSM$_{\geq}(A, k)$.

Remark. If we restrict A to be in $\mathbb{T}^{n\times n}$, then Problem 20 becomes equivalent to BPSM$_{\geq}(A, k)$. This gives us an alternative method to solve BPSM$_{\geq}(A, k)$ for $A \in \mathbb{T}^{n\times n}$ than the one given before.

5.4.9 Complexity of BPSM and its variants

We will now show how BPSM, BPSM$_{\leq}$ and BPSM$_{\geq}$ are polynomially equivalent problems. We will need to use the following lemmas. Note that for any sets X and Y and constant M, that $Y = X + M$ means $y \in Y$ if and only if $y - M \in X$.

Under certain conditions we can guarantee that adding a larger amount of numbers together will result in a bigger total:

Lemma 5.19. If we have a set $X \subseteq \overline{\mathbb{R}}$, $(\exists x \in X)\ x \neq -\infty$, $|X| \geq n$, $X_{min} = \min_{\text{finite } x\in X} x$, $X_{max} = \max_{x\in X} x$, $M = n|X_{min}| + n|X_{max}|$, $Y = X + M$ and $1 \leq r < s \leq n$, then $\sum_{i=1}^{r} y_i < \sum_{i=1}^{s} y'_i$ for any $y_1, y_2, \ldots, y_r \in Y$ and any finite $y'_1, y'_2, \ldots, y'_s \in Y$.

Proof. Let $y_1, y_2, \ldots, y_r \in Y$ be arbitrary and let $y'_1, y'_2, \ldots, y'_s \in Y$ be arbitrary and finite. Then

$$\begin{aligned}
\sum_{i=1}^{r} y_i &\leq rX_{max} + rM \\
&= rX_{max} + r(n|X_{min}| + n|X_{max}|) \\
&= rX_{max} + rn|X_{min}| + rn|X_{max}| \\
&\leq rn|X_{min}| + (rn + r)|X_{max}| \\
&< (rn + n - r - 1)|X_{min}| + (rn + n)|X_{max}| \\
&= ((r+1)n - (r+1))|X_{min}| + (r+1)n|X_{max}| \\
&\leq (sn - s)|X_{min}| + sn|X_{max}| \\
&\leq sX_{min} + sn|X_{min}| + sn|X_{max}| \\
&= sX_{min} + s(n|X_{min}| + n|X_{max}|) \\
&= sX_{min} + sM \\
&\leq \sum_{i=1}^{s} y'_i.
\end{aligned}$$

□

Similarly, we have the following result that under certain conditions adding a larger amount of numbers together will result in a smaller total:

Lemma 5.20. If we have a set $X \subseteq \overline{\mathbb{R}}$, $(\exists x \in X)\ x \neq -\infty$, $|X| \geq n$, $X_{min} = \min_{\text{finite } x \in X} x$, $X_{max} = \max_{x \in X} x$, $M = n|X_{min}| + n|X_{max}|$, $Y = X - M$ and $1 \leq r < s \leq n$, then $\sum_{i=1}^{r} y_i > \sum_{i=1}^{s} y'_i$ for any finite $y_1, y_2, \ldots, y_r \in Y$ and any $y'_1, y'_2, \ldots, y'_s \in Y$.

Proof. Let $y_1, y_2, \ldots, y_r \in Y$ be arbitrary and finite and let $y'_1, y'_2, \ldots, y'_s \in Y$ be arbitrary. Then

$$\begin{aligned}
\sum_{i=1}^{s} y_i' &\leq sX_{max} - sM \\
&= sX_{max} - s(n|X_{min}| + n|X_{max}|) \\
&= sX_{max} - sn|X_{min}| - sn|X_{max}| \\
&\leq -sn|X_{min}| - (sn - s)|X_{max}| \\
&< -(sn - n + s - 1)|X_{min}| - (sn - n)|X_{max}| \\
&= -((s-1)n + (s-1))|X_{min}| - (s-1)n|X_{max}| \\
&\leq -(rn + r)|X_{min}| - rn|X_{max}| \\
&\leq rX_{min} - rn|X_{min}| - rn|X_{max}| \\
&= rX_{min} - r(n|X_{min}| + n|X_{max}|) \\
&= rX_{min} - rM \\
&\leq \sum_{i=1}^{r} y_i.
\end{aligned}$$

□

Theorem 5.21. For any $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, BPSM(A, k) and BPSM$_{\leq}(A, k)$ are polynomially equivalent problems.

Proof. The case where $k = 1$ is trivial, so assume $k > 1$.

If we assume that we can solve BPSM(A, l) in polynomial time for any $l \in N$, then we can solve BPSM$(A, 1)$, BPSM$(A, 2)$, $\ldots$, BPSM(A, k), to give us the values $\delta_{n-1}(A), \delta_{n-2}(A), \ldots, \delta_{n-k}(A)$ in polynomial time. We can then find $\max\limits_{1\leq l\leq k} \delta_{n-l}(A)$, the optimal value of BPSM$_{\leq}(A, k)$ in polynomial time.

Let $A_{min} = \min\limits_{\text{finite } a_{ij}\in A} a_{ij}$, $A_{max} = \max\limits_{a_{ij}\in A} a_{ij}$, $M = n|A_{min}|+n|A_{max}|$, and $B = (b_{ij})$, where $b_{ij} = a_{ij}+M$. To prove the other way, we assume we can solve BPSM$_{\leq}(A', k')$ in polynomial time for any $A' \in \overline{\mathbb{R}}^{n\times n}$ and $k' \in N$. In particular, we could solve

BPSM$_{\leq}(B,k)$ and BPSM$_{\leq}(B,k-1)$ in polynomial time. Let the optimal values of these be θ and θ' respectively.

We know that $\theta = \max\{\theta', \delta_{n-k}(B)\}$. So $\delta_{n-k}(B)$ is finite if $\theta' < \theta$.

We know that θ' is formed by adding together l entries from some matrix in $B(l)$, for some $l < k$. If $\delta_{n-k}(B)$ is finite, then there are k independent finite numbers in a principal submatrix of B. In this case then θ would be finite and made up from adding together l' independent finite entries from some matrix in $B(l')$, for some $l' \leq k$.

Assume that θ is formed by adding together strictly less than k entries from B. Then by Lemma 5.19 with the numbers used to form θ and $\delta_{n-k}(B)$, we have $\theta < \delta_{n-k}(B)$, which is not true as $\theta = \max\{\theta', \delta_{n-k}(B)\}$. Therefore θ is formed by adding together exactly k independent entries from some matrix in $B(k)$. Then by Lemma 5.19 with the numbers used to form θ' and θ, we have $\theta' < \theta$.

Therefore we have shown that $\delta_{n-k}(B)$ is finite if and only if $\theta' < \theta$, and as $\theta = \max\{\theta', \delta_{n-k}(B)\}$, if $\theta' < \theta$, then $\theta = \delta_{n-k}(B) = \delta_{n-k}(A) + kM = \delta_{n-k}(A) + kn|A_{min}| + kn|A_{max}|$ and $\delta_{n-k}(A) = \theta - kn|A_{min}| - kn|A_{max}|$.

Therefore we can solve BPSM$_{\leq}(B,k)$ and BPSM$_{\leq}(B,k-1)$ to find θ and θ' respectively, in polynomial time. We can then state that $\delta_{n-k}(A) = \theta - kn|A_{min}| - kn|A_{max}|$ if $\theta' < \theta$, or $\delta_{n-k}(A) = -\infty$ if $\theta' = \theta$.

Therefore both problems are polynomially equivalent. $\square$

A similar theorem and proof for BPSM$_{\geq}(A,k)$ are now presented:

Theorem 5.22. For any $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, BPSM(A,k) and BPSM$_{\geq}(A,k)$ are polynomially equivalent problems.

Proof. The case where $k = n$ is trivial, so assume $k < n$.

If we assume that we can solve BPSM(A,l) in polynomial time for any $l \in N$, then we can solve BPSM(A,k), BPSM$(A,k+1)$, ..., BPSM(A,n), to give us the values $\delta_{n-k}(A), \delta_{n-k-1}(A), \ldots, \delta_{n-n}(A)$ in polynomial time. We can then find $\max\limits_{k \leq l \leq n} \delta_{n-l}(A)$, the optimal value of BPSM$_{\geq}(A, k)$ in polynomial time.

Let $A_{min} = \min\limits_{\text{finite } a_{ij} \in A} a_{ij}$, $A_{max} = \max\limits_{a_{ij} \in A} a_{ij}$, $M = n|A_{min}| + n|A_{max}|$, and $B = (b_{ij})$, where $b_{ij} = a_{ij} - M$. To prove the other way, we assume we can solve BPSM$_{\geq}(A', k')$ in polynomial time for any $A' \in \overline{\mathbb{R}}^{n \times n}$ and $k' \in N$. In particular we can solve BPSM$_{\geq}(B, k)$ and BPSM$_{\geq}(B, k+1)$ in polynomial time. Let the optimal values of these be θ and θ' respectively.

We know that $\theta = \max\{\theta', \delta_{n-k}(B)\}$. So $\delta_{n-k}(B)$ is finite if $\theta' < \theta$.

We know that θ' is formed by adding together l entries from some matrix in $B(l)$, for some $l > k$. If $\delta_{n-k}(B)$ is finite, then there are k independent finite numbers in a principal submatrix of B. In this case then θ would be finite and made up from adding together l' independent finite entries from some matrix in $B(l')$, for some $l' \geq k$.

Assume that θ is formed by adding together strictly more than k entries from B. Then by Lemma 5.20 with the numbers used to form θ and $\delta_{n-k}(B)$, we have $\theta < \delta_{n-k}(B)$, which is not true as $\theta = \max\{\theta', \delta_{n-k}(B)\}$. Therefore θ is formed by adding together exactly k independent entries from some matrix in $B(k)$. Then by Lemma 5.20 with the numbers used to form θ' and θ, we have $\theta' < \theta$.

Therefore we have shown that $\delta_{n-k}(B)$ is finite if and only if $\theta' < \theta$, and as $\theta = \max\{\theta', \delta_{n-k}(B)\}$, if $\theta' < \theta$, then $\theta = \delta_{n-k}(B) = \delta_{n-k}(A) - kM = \delta_{n-k}(A) - kn|A_{min}| - kn|A_{max}|$ and $\delta_{n-k}(A) = \theta + kn|A_{min}| + kn|A_{max}|$.

Therefore we can solve BPSM$_{\geq}(B, k)$ and BPSM$_{\geq}(B, k+1)$ to find θ and θ' respectively, in polynomial time. We can then state that $\delta_{n-k}(A) = \theta + kn|A_{min}| + kn|A_{max}|$ if $\theta' < \theta$, or $\delta_{n-k}(A) = -\infty$ if $\theta' = \theta$.

Therefore both problems are polynomially equivalent. □

From the above Theorems and Theorem 5.18, it immediately follows that:

Corollary 5.23. For any $A \in \overline{\mathbb{R}}^{n \times n}$ and any $k, k_{min}, k_{max} \in N$, BPSM$(A, k)$, BPSM$_\leq(A,k)$, BPSM$_\geq(A,k)$ and BPSM$_{Range}(A,k_{min},k_{max})$ are polynomially equivalent problems.

Later (in Section 6.5.2), we shall look at another variant of BPSM called BPSM$_{Frac}$. We shall see that BPSM$_{Frac}$ is polynomially solvable.

Summary

The following table summarises the complexity of the problems investigated in this thesis.

A 'P' means the problem is polynomially solvable. An 'NPC' means the problem is *NP*-complete. A '?' means it is unknown if the problem is polynomially solvable or *NP*-complete. If a problem is later proven to be polynomially solvable and is labelled as a numbered question mark, then all problems with the same numbered question mark are polynomially solvable. If a problem is later proven to be *NP*-complete and is labelled as a numbered question mark, then all problems with the same numbered question mark are *NP*-complete. Also $\mathbb{T} \subseteq [-\infty,0] \subseteq \mathbb{R}$, so we can note that:

- If any $?_1 =$ NPC then all $?_1 =$ NPC.
- If any $?_2 =$ P then all $?_2 =$ P.
- If any $?_1 =$ P then all $?_1 =$ P and all $?_2 =$ P.
- If any $?_2 =$ NPC then all $?_1 =$ NPC and all $?_2 =$ NPC.

Problem	Name	Page	$a_{ij} \in \overline{\mathbb{R}}$	$a_{ij} \leq 0$	$a_{ij} \in \mathbb{T}$
1	BSM(A, k)	38	P	P	P
2		39	P	P	P
3	BPSM(A, k)	40	$?_1$	$?_1$	$?_2$
4		43	$?_1$	$?_1$	P
5		47	NPC	NPC	NPC
6		48	NPC	NPC	NPC
7		49	P	P	P
8		49	P	P	P
9		52	P	P	P
10		53	P	P	P
11	BPSM$_{\leq}(A, k)$	54	$?_1$	P	P
12		57	-	-	P
13		58	P	P	P
14		58	P	P	P
15	BPSM$_{\geq}(A, k)$	59	$?_1$	$?_1$	P
16		59	P	P	P
17	BPSM$_{Range}(A, k_{min}, k_{max})$	60	$?_1$	$?_1$	$?_2$
18	BBPSM(A, k)	61	$?_2$	$?_2$	$?_2$
19	BBPSM$_{\leq}(A, k)$	63	P	P	P
20	BBPSM$_{\geq}(A, k)$	63	P	P	P
21	BPSM$_{Frac}(A, k)$	128	P	P	P

Table 5.1: A table showing whether investigated problems are known to be polynomially solvable or are NP-complete for various types of matrix $A = (a_{ij})$.

Chapter 6

Estimating or solving BPSM

As we discussed in Section 5.2, in general there is no polynomial method known for solving BPSM(A, k) for $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$. Calculating the optimal assignment problem value for each principal submatrix of A (complete enumeration) is not an efficient method.

It may not always be necessary to find the exact value of $\delta_{n-k}(A)$, the optimal value of BPSM(A, k). We will look at how to estimate this value in polynomial time.

We will look at the BPSM related results that we have obtained from the previous section, as well as results we can deduce using the eigenproblem and BSM.

There are some special cases that can be solved in polynomial time (if encoded in the right way). We will present existing polynomially solvable cases, and some new results.

A branch and bound method will also be presented later, which will find the optimal solution value of BPSM (although maybe not in polynomial time). If it is terminated before it finds the value of $\delta_{n-k}(A)$, then it can at least give lower and upper bounds that $\delta_{n-k}(A)$ lies between.

6.1 Using similar problems to estimate BPSM

We derived a number of results for BPSM in Section 5.4. We now summarise these results here.

From Section 5.4.2 we obtained the following result:

$$(\forall k \in N) \;\; \delta_{n-k}(A) \le k \max_{i \in N} \max_{j \in K} \frac{a_{ii}^{j}}{j}.$$

From Section 5.4.3 we obtained the following results, assuming there are no positive cycles in $D(A)$:

$$\max_{k \in N} \delta_{n-k}(A) = \max_{i \in N} \max_{j \in N} a_{ii}^{j},$$

and as $\max\limits_{l \in K} \max\limits_{B \in A(l)} \operatorname{maper}(B) = \max\limits_{i \in N} \max\limits_{j \in K} a_{ii}^{j}$, we have

$$(\forall k \in N) \;\; \delta_{n-k}(A) \le \max_{i \in N} \max_{j \in K} a_{ii}^{j}. \tag{6.1.1}$$

From Section 5.4.3 we obtained the following results, assuming there is at least one non-negative cycle in $D(A)$:

$$\max_{k \in N} \delta_{n-k}(A) = \chi_A(0),$$

so

$$(\forall k \in N) \;\; \delta_{n-k}(A) \le \chi_A(0). \tag{6.1.2}$$

We may be able to improve these bounds as follows. For any $A = (a_{ij}) \in \overline{\mathbb{R}}$ and $k \in N$, let $B = (b_{ij}) = (a_{ij} - M)$ for some $M \in \mathbb{R}$. Then $\delta_{n-k}(A) = \delta_{n-k}(B) + kM$.

Let $M = M_1$ be such that B has no positive cycles. Therefore

$$\begin{aligned}(\forall k \in N) \;\; \delta_{n-k}(A) &= \delta_{n-k}(B) + kM \\ &\le \max_{i \in N} \max_{j \in K} b_{ii}^{j} + kM. \end{aligned} \tag{6.1.3}$$

Let $M = M_2$ be such that B has at least one non-negative cycle. Therefore

$$\begin{aligned}(\forall k \in N) \;\; \delta_{n-k}(A) &= \delta_{n-k}(B) + kM \\ &\leq \chi_B(0) + kM. \end{aligned} \tag{6.1.4}$$

Let $M = M_3$ be as small as possible such that B has no positive cycles. Therefore B has at least one zero cycle, both statements hold and $\chi_B(0) = 0$. Putting this all together, for $M = M_3$ we therefore have

$$(\forall k \in N) \;\; \delta_{n-k}(A) \leq \min\left(\max_{i \in N} \max_{j \in K} b_{ii}^{j}\,, 0\right) + kM.$$

As $\max\limits_{i \in N} \max\limits_{j \in K} b_{ii}^{j} \leq 0$ for $M = M_3$, we have

$$(\forall k \in N) \;\; \delta_{n-k}(A) \leq \max_{i \in N} \max_{j \in K} b_{ii}^{j} + kM. \tag{6.1.5}$$

Note that for each unit increase in M, the amount kM increases by is at least as great as the amount $\max\limits_{i \in N} \max\limits_{j \in K} b_{ii}^{j}$ decreases by. Therefore the value of M that results in the tightest bound for Equation 6.1.3 is $M = M_3$.

For $M = M_3$, we have seen from Equation 6.1.5 that Equation 6.1.3 is a better bound than Equation 6.1.4. However it is possible that Equation 6.1.4 for some value of M lower than M_3 could give a better bound than 6.1.3 for $M = M_3$. For each unit decrease of M, $\chi_B(0)$ may or may not increase more than kM decreases. For a value of M low enough, $\chi_B(0)$ will equal $\text{maper}(B)$, so further decreases in M will not change the bound. Between this value of M and M_3 some sort of bisection method could be used to find the best value of M (say M_4) to give the tightest bound for Equation 6.1.4. This new bound (with M set to M_4), plus the one given by Equation 6.1.5 (with M set to M_3) are at least as good as the bounds given by Equation 6.1.2 and Equation 6.1.1 respectively.

Also from Section 5.4.3 we let k_{min} be the lowest k such that there is k finite independent entries of any $k \times k$ principal submatrix of A. We stated that this can be found by solving the shortest cycle problem [31]. We then gave these results:

$$\delta_{n-1}(A) = \delta_{n-2}(A) = \cdots = \delta_{n-k_{min}+1}(A) = -\infty,$$

and

$$\delta_{n-k_{min}}(A) = \max_{i \in N} a_{ii}^{k_{min}}.$$

From Section 5.4.4 we let k_{max} be the maximum number of finite independent entries (of any principal submatrix) of A. We stated that this was equal to $n + \chi_{A'}(-1)$, where $A' = (a'_{ij})$ is defined by $(a'_{ij}) = 0$ if $(a_{ij}) \in \mathbb{R}$ and $(a'_{ij}) = -\infty$ otherwise. We let $x_0 = -n \max(0, A_{max}, -A_{min}, A_{max} - A_{min})$. We then gave these results:

$$\delta_0(A) = \delta_1(A) = \cdots = \delta_{n-k_{max}-1}(A) = -\infty,$$

and

$$\delta_{n-k_{max}}(A) = \chi_A(x_0) - k_{max} x_0.$$

6.2 Using the eigenproblem to estimate BPSM

There are interesting links between BPSM(A, k) and the eigenproblem. We shall investigate how the eigenproblem (in particular, the eigenvalue and eigenvectors) can help us estimate $\delta_{n-k}(A)$.

We will define what *similar* matrices are and describe BPSM related properties that such matrices have. One property is that if A and B are similar, then $\delta_{n-k}(A) = \delta_{n-k}(B)$. We will show, using the eigenvector, that every matrix (with a strongly connected digraph) is similar to a matrix in which every element is no greater than the eigenvalue. We shall also see that the maximum value of the corners of the characteristic max-polynomial is equal to the eigenvalue. We then deduce some bounds for $\delta_{n-k}(A)$.

Recall the definition of $\mathrm{diag}(d)$ (Definition 2.12, page 11). We now introduce similar matrices.

Definition 6.1. If $d = (d_1, d_2, \ldots, d_n) \in \mathbb{R}^n$ and $D = \mathrm{diag}(d)$, then let D^{-1} denote the matrix $\mathrm{diag}(d_1^{-1}, d_2^{-1}, \ldots, d_n^{-1})$.

Obviously, $D \otimes D^{-1} = I = D^{-1} \otimes D$ for any diagonal matrix $D \in \overline{\mathbb{R}}^{n\times n}$.

If $A \in \overline{\mathbb{R}}^{n\times n}$, $d = (d_1, d_2, \ldots, d_n) \in \mathbb{R}^n$ and $D = \mathrm{diag}(d)$, then $A \otimes D$ is the matrix that arises from adding constants $d_1, d_2, \ldots, d_n$ to the columns of A. Similarly, $D \otimes A$ is the matrix that arises from adding constants $d_1, d_2, \ldots, d_n$ to the rows of A. Hence in conventional notation, $D^{-1} \otimes A \otimes D = (-d_i + a_{ij} + d_j)$.

Definition 6.2. If $A, B \in \overline{\mathbb{R}}^{n\times n}$ and there exists a diagonal matrix D such that $B = D^{-1} \otimes A \otimes D$ then we say that A is *similar* to B, and denote this by $A \sim B$.

Theorem 6.3. The relation $\sim$ is an equivalence relation.

Proof. Let $A, B, C \in \overline{\mathbb{R}}^{n\times n}$. $A = I^{-1} \otimes A \otimes I$, so $\sim$ is reflexive.

If $A \sim B$, then there exists a diagonal matrix $D \in \overline{\mathbb{R}}^{n\times n}$ such that

$$B = D^{-1} \otimes A \otimes D.$$

Pre-multiplying by D and post-multiplying by D^{-1}, we have

$$D \otimes B \otimes D^{-1} = D \otimes D^{-1} \otimes A \otimes D \otimes D^{-1},$$

which simplifies to

$$D \otimes B \otimes D^{-1} = A$$

or

$$(D^{-1})^{-1} \otimes B \otimes (D^{-1}) = A,$$

so $B \sim A$ and $\sim$ is symmetric.

If $A \sim B$ and $B \sim C$, then there exists diagonal matrices $D, D' \in \overline{\mathbb{R}}^{n\times n}$ such that

$$B = D^{-1} \otimes A \otimes D$$

and

$$C = D'^{-1} \otimes B \otimes D'.$$

So

$$C = D'^{-1} \otimes D^{-1} \otimes A \otimes D \otimes D'.$$

Letting $D'' = D \otimes D'$ (which is a diagonal matrix), we have

$$C = D''^{-1} \otimes A \otimes D'',$$

so $A \sim C$ and $\sim$ is transitive.

Hence $\sim$ is an equivalence relation. $\square$

We now look at some properties of similar matrices.

Theorem 6.4. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n},\ B = (b_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $A \sim B$, then $(\forall \sigma \in C_n)$ $w(B, \sigma) = w(A, \sigma)$.

Proof. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $\sigma = (i_1, i_2, \ldots, i_k) \in C_n$ is an arbitrary cyclic permutation of length k, then

$$w(A, \sigma) = a_{i_1,i_2} + a_{i_2,i_3} + \cdots + a_{i_{k-1},i_k} + a_{i_k,i_1},$$

and as $A \sim B$, there exists a diagonal matrix $D = \mathrm{diag}(d_1, d_2, \ldots, d_n)$ such that $B = (b_{ij}) = (-d_i + a_{ij} + d_j)$. Therefore

$$\begin{aligned}
w(B,\sigma) &= b_{i_1,i_2} + b_{i_2,i_3} + \cdots + b_{i_{k-1},i_k} + b_{i_k,i_1} \\
&= (-d_{i_1} + a_{i_1,i_2} + d_{i_2}) + (-d_{i_2} + a_{i_2,i_3} + d_{i_3}) + \cdots + \\
&\qquad\qquad + (-d_{i_{k-1}} + a_{i_{k-1},i_k} + d_{i_k}) + (-d_{i_k} + a_{i_k,i_1} + d_{i_1}) \\
&= a_{i_1,i_2} + a_{i_2,i_3} + \cdots + a_{i_{k-1},i_k} + a_{i_k,i_1} \\
&= w(A,\sigma)
\end{aligned}$$

which completes the proof. □

Corollary 6.5. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, $B = (b_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $A \sim B$, then $(\forall \sigma \in C_n)$ $\mu(B,\sigma) = \mu(A,\sigma)$.

Theorem 6.6. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, $B = (b_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $A \sim B$, then $(\forall \pi \in P_n)$ $w(B,\pi) = w(A,\pi)$.

Proof. Let $\pi = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_s \in P_n$, where $\sigma_1, \sigma_2, \ldots, \sigma_s \in C_n$. By Theorem 6.4, $w(B,\sigma_i) = w(A,\sigma_i)$ for $i = 1,\ldots,s$. As $w(\sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_s) = w(\sigma_1) + w(\sigma_2) + \cdots + w(\sigma_s)$, we have $w(B,\pi) = w(A,\pi)$. □

Corollary 6.7. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, $B = (b_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $A \sim B$, then $(\forall \pi \in P_n)$ $\mu(B,\pi) = \mu(A,\pi)$.

Corollary 6.8. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, $B = (b_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $A \sim B$, then $\lambda(B) = \lambda(A)$.

Proof. Let $\sigma \in C_n$ be a cyclic permutation satisfying $\mu(A,\sigma) = \lambda(A)$. By Corollary 6.5, we see that the relation $A \sim B$ requires the mean weight of cyclic permutations to be preserved between A and B. Therefore $\mu(B,\sigma) = \mu(A,\sigma)$. Hence $\lambda(B) = \mu(B,\sigma) = \mu(A,\sigma) = \lambda(A)$. □

Definition 6.9. If $A \in \overline{\mathbb{R}}^{n\times n}$ and $S \subseteq N$, then let $A(S)$ be the matrix formed from A by deleting rows and columns of A with indices from $N - S$.

Corollary 6.10. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, $B = (b_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $A \sim B$, then $(\forall k \in N)$ $\delta_{n-k}(B) = \delta_{n-k}(A)$.

Proof. Let $C = B(S)$ and $D = A(S)$, where $S \subseteq N$ is a set with k elements. Note that $C \in B(k), D \in A(k)$ and $C \sim D$. By Theorem 6.6, $(\forall \pi \in P_k)$ $w(D, \pi) = w(C, \pi)$, so

$$\begin{aligned}\delta_{n-k}(B) &= \max_{C\in B(k)} \max_{\pi\in P_k} w(C, \pi)\\ &= \max_{D\in A(k)} \max_{\pi\in P_k} w(D, \pi)\\ &= \delta_{n-k}(A)\end{aligned}$$

which completes the proof. □

Remark. Note that if $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$, $B = (b_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $A \sim B$, then by setting $k = n$ in Corollary 6.10, we obtain that maper(B) = maper(A).

Now we show any irreducible matrix is similar to a matrix in which every element is no greater than the eigenvalue, and in each row, at least one element is equal to the eigenvalue:

Theorem 6.11. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ is an irreducible matrix, $x \in sp(A)$, $D = \text{diag}(x)$ is a diagonal matrix and $B = (b_{ij}) = D^{-1} \otimes A \otimes D$, then $(\forall i \in N)$ $(\forall j \in N)$ $b_{ij} \leq \lambda(A)$ and $(\forall i \in N)(\exists j \in N)$ $b_{ij} = \lambda(A)$.

Proof. Let $\lambda = \lambda(A)$, then we have

$$A \otimes x = \lambda \otimes x,$$

or equivalently

$$(\forall i \in N) \max_{j\in N}(a_{ij} + x_j) = \lambda + x_i,$$

therefore

$$(\forall i \in N)\ \max_{j \in N}(a_{ij} + x_j - x_i) = \lambda,$$

hence

$$(\forall i \in N)\ \max_{j \in N} b_{ij} = \lambda,$$

which completes the proof. □

For an irreducible matrix A, this enables us to find an upper bound for δ_k, the optimal value of BPSM$(A, n-k)$:

Corollary 6.12. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ is an irreducible matrix and $0 \leq k \leq n$, then

$$\delta_k(A) \leq (n-k)\lambda(A). \tag{6.2.1}$$

Proof. Let B be as in Theorem 6.11. Then the maximum element in B is $\lambda(A)$, so for $0 \leq k < n$, the sum of any $n-k$ elements of B is less than or equal to $(n-k)\lambda(B)$. As $\lambda(A) = \lambda(B)$, we have

$$\begin{aligned} \delta_k(A) &= \delta_k(B) \\ &\leq (n-k)\lambda(B) \\ &= (n-k)\lambda(A). \end{aligned}$$

By definition $\delta_n(A) = 0$, so the $k = n$ case is automatically satisfied. □

We now show that $\delta_k(A) = (n-k)\lambda(A)$, for at least one $k,\ 0 \leq k < n$.

Theorem 6.13 ([19])**.** If $A \in \overline{\mathbb{R}}^{n \times n}$ is an irreducible matrix, then $\lambda(A)$ is equal to the maximum of the corners of $\chi_A(x)$.

We now give an alternative proof to the one in [19].

Proof. Let the maximum of all corners be b^*. This is equal to the x co-ordinate of the intersection point between two essential linear pieces of $\chi_A(x)$. One of these pieces has equation $y = nx$, the other has equation, say, $y = \delta_s + sx$. The intersection point is therefore when $\delta_s + sx = nx$, i.e. at $b^* = \delta_s/(n-s)$. As we are assuming that all other terms cross $y = nx$ at $x \leq b^*$, we have $b^* = \max\limits_{k=0,\ldots,n-1}(\delta_k/(n-k))$.

Using Theorem 5.6, we see that $\delta_s = w(B,\pi)$, for some $B \in A(n-s)$ and some $\pi = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_r \in P_{n-s}$, where $\sigma_1, \sigma_2, \ldots, \sigma_r \in C_{n-s}$. Let $\mu(B,\sigma^*) = \max\limits_{i=1\ldots,r} \mu(B,\sigma_i)$, where $\sigma^* \in \{\sigma_1, \sigma_2, \ldots, \sigma_r\}$. Let $t = n - l(\sigma^*)$ and $d_t^* = w(B,\sigma^*)$.

Now $\forall k$, we define $d_k = \max\limits_{\sigma \in C_n^{n-k}} w(A,\sigma)$. Note that $d_k/(n-k) = \max\limits_{\sigma \in C_n^{n-k}} \mu(A,\sigma)$, and $d_k \leq \delta_k$. Also note that $\lambda(A) = \max\limits_{k=0,\ldots,n-1}(d_k/(n-k))$. Thus

$$\begin{aligned} d_t^*/(n-t) &\geq \delta_s/(n-s) \\ &= b^* \\ &= \max_{k=0,\ldots,n-1}(\delta_k/(n-k)) \\ &\geq \max_{k=0,\ldots,n-1}(d_k/(n-k)) \\ &\geq d_t/(n-t) \\ &\geq d_t^*/(n-t). \end{aligned}$$

with the first inequality due to Lemma 5.10.

Hence we have

$$b^* = \max_{k=0,\ldots,n-1}(\delta_k/(n-k)) = \max_{k=0,\ldots,n-1}(d_k/(n-k)) = \lambda(A),$$

which completes the proof. $\square$

We now give another upper bound for δ_k:

Theorem 6.14. For $A \in \overline{\mathbb{R}}^{n\times n}$, if $\delta_r + rx$ and $\delta_s + sx$ are essential terms of $\chi_A(x)$, where $0 \leq r < s \leq n$, and $\delta_k + kx$ is inessential for $k = r+1, r+2, \ldots, s-1$, then for $k = r+1, r+2, \ldots, s-1$,

$$\delta_k \leq \frac{(s-k)\delta_r + (k-r)\delta_s}{s-r}, \tag{6.2.2}$$

with equality holding if and only if $\chi_A(x) = \delta_k + kx$ for a unique value of x.

Proof. Let x_1 be the point where immediately prior to it, $\chi_A(x)$ has slope r, and $\chi_A(x)$ has slope s, immediately after it. Let $k \in \{r+1, r+2, \ldots, s-1\}$. As $\delta_k + kx$ is inessential, we have that

$$\begin{aligned} \delta_k + kx_1 &\leq \chi_A(x_1) \\ &= \delta_r + rx_1 \\ &= \delta_s + sx_1. \end{aligned}$$

Therefore

$$x_1 = \frac{\delta_r - \delta_s}{s-r},$$

and

$$\begin{aligned} \delta_k &\leq (s-k)x_1 + \delta_s \\ &= (s-k)\frac{\delta_r - \delta_s}{s-r} + \delta_s \\ &= \frac{(s-k)\delta_r + (k-r)\delta_s}{s-r}. \end{aligned}$$

If $\delta_k + kx = \chi_A(x)$ at some point, it must be at $x = x_1$. If this is the case then all the inequalities above that involve δ_k will turn to equalities. □

Theorem 6.15. The upper bound for δ_k given by (6.2.2) is as good (and in some cases better) than the one given by (6.2.1).

Proof. We have that

$$\begin{aligned}
\delta_k &\leq \frac{(s-k)\delta_r + (k-r)\delta_s}{(s-r)} \\
&= \frac{(s-k)(n-r)\delta_r}{(s-r)(n-r)} + \frac{(k-r)(n-s)\delta_s}{(s-r)(n-s)} \\
&\leq \frac{(s-k)(n-r)\lambda}{(s-r)} + \frac{(k-r)(n-s)\lambda}{(s-r)} \\
&= \big((s-k)(n-r) + (k-r)(n-s)\big)\frac{\lambda}{(s-r)} \\
&= (sn - sr - kn + kr + kn - ks - rn + rs)\frac{\lambda}{(s-r)} \\
&= (sn + kr - ks - rn)\frac{\lambda}{(s-r)} \\
&= (n-k)(s-r)\frac{\lambda}{(s-r)} \\
&= (n-k)\lambda,
\end{aligned}$$

which completes the proof. □

6.3 Using BSM to estimate BPSM

We know that $\delta_0(A) = \gamma_0(A) = \operatorname{maper}(A)$ and $\delta_n(A) = \gamma_n(A) = 0$ (by definition). However for $k \in \{1, \ldots, n-1\}$, it is not always true that $\delta_k(A) = \gamma_k(A)$.

Example 6.1. Let

$$A = \begin{pmatrix} 0 & -1 \\ -1 & 0 \end{pmatrix}.$$

Here we have $\delta_1(A) = \gamma_1(A) = 0$.

Example 6.2. Let

$$A = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

Here we have $\delta_1(A) = 0$, but $\gamma_1(A) = 1$.

For $k \in \{1, \dots, n-1\}$, $\gamma_k(A)$ is an upper bound for $\delta_k(A)$. However, it may not be a very good upper bound:

Remark. For $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ and $k \in \{1, \dots, n-1\}$, $\gamma_k(A)$ can be arbitrarily bigger than $\delta_k(A)$.

We can show this as follows. Let $B = (b_{ij})$, where $(\forall i)(\forall j)\ b_{ij} = 0$. Let $D = \text{diag}(0, \dots, 0, \theta)$ be an $n \times n$ diagonal matrix. Let

$$\begin{aligned} A &= D^{-1} \otimes B \otimes D \\ &= \begin{pmatrix} 0 & \dots & 0 & \theta \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \dots & 0 & \theta \\ -\theta & \dots & -\theta & 0 \end{pmatrix}. \end{aligned} \tag{6.3.1}$$

where θ is an arbitrary non-negative real number.

As $A \sim B$, by Corollary 6.10, we have that $(\forall k \in \{1, \dots, n-1\})\ \delta_k(A) = \delta_k(B) = 0$.

However, $(\forall k \in \{1, \dots, n-1\})\ \gamma_{n-k}(A) = \max(0, \theta - \theta, -\theta, \theta) = \theta$. As θ was an arbitrary non-negative real number, $\gamma_k(A)$ can be arbitrarily bigger than $\delta_k(A)$.

Definition 6.16. For $A \in \overline{\mathbb{R}}^{n \times n}, k \in \{1, \dots, n-1\}$, A will be called an S_k matrix if $\delta_k(A) = \gamma_k(A)$.

This means it is easy to find $\delta_k(A)$ if A is an S_k matrix, because we just need to calculate $\gamma_k(A)$, which we can do so in $O(n^3)$ time (see Section 5.1 for details).

Example 6.1 has an S_1 matrix. Example 6.2 and Example 6.3 have matrices that are not S_1 matrices.

Definition 6.17. For $A \in \overline{\mathbb{R}}^{n \times n}, k \in \{1, \dots, n-1\}$, A will be called S_k *transformable* if there exists an S_k matrix B such that $A \sim B$.

Example 6.2 is S_1 transformable, as $A \sim \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$.

So for all $k \in \{1, \ldots, n-1\}$ and all S_k transformable matrices, $\delta_k(A) = \gamma_k(A)$. This means that we can polynomially solve BPSM(A, k) for any S_k transformable matrix A, after finding a S_k matrix similar to A.

Example 6.3. Let

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}.$$

Here we have $\delta_1(A) = 0$, but $\gamma_1(A) = 1$. If there exists an S_1 matrix B such that $A \sim B = (b_{ij})$, then $b_{12} + b_{21} = a_{12} + a_{21} = 1$, so $\max(b_{12}, b_{21}) > 0$, therefore $\gamma_1(B) > 0 = \delta_1(B)$. This contradicts B being an S_1 matrix and so there is no S_1 matrix $B \sim$ A, and A is not S_1 transformable.

If the matrix is not S_k transformable, then we may want to obtain a γ value as low as possible. The transformation in Theorem 6.11 (page 78) is often useful in finding a similar matrix that may have a lower γ value. If we applied this transformation to the matrix in 6.3.1 then we would not only get a lower γ value, we transform it to an S_k matrix.

However this transformation may not give the lowest γ value of all matrices similar to our matrix, as in fact this transformation may make the value of γ increase, as in the following example:

Example 6.4. Let

$$A = \begin{pmatrix} 0 & 2 & 1 \\ 0 & 2 & 1 \\ 1 & -1 & 0 \end{pmatrix}.$$

Here we have $\gamma_2(A) = 3$. This is our current estimate for $\delta_2(A)$.

Let

$$B = \begin{pmatrix} 0 & 2 & 0 \\ 0 & 2 & 0 \\ 2 & 0 & 0 \end{pmatrix}.$$

Observe that $A \sim B$, with $\lambda(B) = 2$ being the highest entry appearing on each row. So the transformation in Theorem 6.11 has been applied correctly. Here $\gamma_2(B) = 4$.

The exact value of $\delta_2(B)$ is 2, so in this case $\gamma_2(A)$ is a better approximation to $\delta_2(B)$ than $\gamma_2(B)$ is.

6.4 Solving BPSM in special cases

For a general matrix $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, no polynomial time algorithm is known for solving BPSM(A, k). However, for some special types of matrix A with some (or all) values of k, we can solve BPSM(A, k) in polynomial time.

We will start by stating some existing results for polynomially solving BPSM: We will comment on the importance of essential terms. We give a result that allows us to solve BPSM with a randomised polynomial algorithm if entries of the matrix are bounded by n. Then we will state that BPSM is polynomially solvable for diagonally dominant matrices, Monge matrices and bi-diagonal matrices. We give results for Hankel matrices with strictly concave sequences, as well as for permutation matrices.

We will then give some new results. We will define pyramidal matrices (in which all elements satisfy a constraint given later), and show that we can polynomially solve BPSM(A, k) for a pyramidal matrix A.

We shall then consider symmetric matrices in $\mathbb{T}^{n\times n}$, i.e. symmetric matrices with elements equal to 0 or $-\infty$. Some results have already been obtained. We will review these and give some more results.

We give a polynomial time algorithm to solve BPSM for generalised permutation matrices (defined in Definition 2.15 on page 12). We will discuss how important the form of the encoding of the input is (for example as a matrix, or as a list of constitute cycles) in determining whether this algorithm is polynomial or pseudopolynomial.

We will give a similar polynomial time algorithm to solve BPSM for block diagonal matrices. We will explain how this helps towards solving BPSM for reducible matrices.

6.4.1 Some previously known special cases

Essential terms

For $A \in \overline{\mathbb{R}}^{n\times n}$, Theorem 4.4 states that essential terms of the $\chi_A(x)$ can be found in $\mathrm{O}(n^2(m + n\log n))$ steps, where A has m finite entries. This means that if $\delta_{n-k} \otimes x^{(n-k)}$ is an essential term, then by Theorem 5.6, $\delta_{n-k}(A) = \max\limits_{B\in A(k)} \mathrm{maper}(B)$, the optimal value of BPSM(A, k) can be found in $\mathrm{O}(n^2(m + n\log n))$ steps. Hence BPSM(A) can be completely solved in polynomial time if all terms of $\chi_A(x)$ are essential.

The complexity has recently been improved by a factor of n to $\mathrm{O}(n(m+n\log n))$ steps [24].

Bounded matrix elements

Remark ([8])**.** If the finite entries of $A \in \overline{\mathbb{R}}^{n\times n}$ are integer and polynomially bounded by n, then BPSM(A, k) can be solved by a randomised polynomial time algorithm.

Diagonally dominant matrices

Theorem 6.18. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ satisfies

$$(\forall i \in N)\ (\forall j \in N)\ a_{ii} \geq a_{ij}, \tag{6.4.1}$$

then we can solve BPSM(A, k) in polynomial time.

Proof. As a diagonal element is the greatest in its row, and all diagonal elements are in different rows and columns, $id(A) = (1)(2)\dots(n)$ is an optimal permutation to the assignment problem for A, i.e. $id(A) \in ap(A)$.

Similarly, for $B \in A(k)$, $id(B) \in ap(B)$. Therefore if $C \in A(k)$ is a principal submatrix of A such that $\text{maper}(C) = \delta_{n-k}$ then $id(C) \in ap(C)$.

If we simultaneously re-order the rows and columns of A so that the diagonal elements of A are in non-ascending order, then the optimal principal submatrix of A will be $A(K)$. Then

$$\begin{aligned}\delta_{n-k}(A) &= a_{11} + a_{22} + \cdots + a_{kk} \\ &= \prod_{i\in K}^{\otimes} a_{ii}\end{aligned}$$

which we can calculate in polynomial time (including re-ordering). $\square$

A more general matrix than one satisfying (6.4.1) is a diagonally dominant matrix.

Definition 6.19. A matrix $A \in \overline{\mathbb{R}}^{n\times n}$ is *diagonally dominant* if $id \in ap(A)$.

Theorem 6.20 ([34])**.** For a diagonally dominant matrix $A \in \overline{\mathbb{R}}^{n\times n}$, we can solve $\text{BPSM}(A, k)$ in polynomial time.

Bi-diagonal matrices

Definition 6.21. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ and $\exists c \in [1, \dots, n-1]$ such that $a_{ij} = -\infty$ for $|i-j| \neq c$, then A is called a *bi-diagonal* matrix.

Theorem 6.22 ([34])**.** For a bi-diagonal matrix $A \in \overline{\mathbb{R}}^{n\times n}$, $\text{BPSM}(A, k)$ can be solved in $O(n^3)$ time.

Monge matrices

Definition 6.23. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ satisfies $a_{ij} + a_{kl} \geq a_{il} + a_{kj}$ for all $i < k, j < l$, then A is called a *Monge* matrix.

Theorem 6.24 ([34])**.** A Monge matrix $A \in \overline{\mathbb{R}}^{n\times n}$ is diagonally dominant, so BPSM(A, k) can be solved in polynomial time.

Hankel matrices

Definition 6.25. For a given sequence $\{g_r \in \overline{\mathbb{R}} : r = 1, \ldots, 2n-1\}$, the *Hankel* matrix is the matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ where $a_{ij} = g_{i+j-1}$.

It is not known how to solve BPSM(A) efficiently for a general Hankel matrix A, however we can solve it for some special cases of Hankel matrix as follows.

Definition 6.26. A sequence $\{g_r \in \overline{\mathbb{R}} : r = 1, \ldots, n\}$, is said to be *strictly concave* if $g_{r+2} - g_{r+1} < g_{r+1} - g_r$ for $r = 1, \ldots, n-2$.

Theorem 6.27 ([34])**.** For a Hankel matrix $A \in \overline{\mathbb{R}}^{n\times n}$, formed from a strictly concave sequence, BPSM(A, k) can be solved in polynomial time.

Definition 6.28. A sequence $\{g_r \in \overline{\mathbb{R}} : r = 1, \ldots, n\}$, is said to be *convex* if $g_{r+2} - g_{r+1} \geq g_{r+1} - g_r$ for $r = 1, \ldots, n-2$.

Theorem 6.29 ([14])**.** A Hankel matrix $A \in \overline{\mathbb{R}}^{n\times n}$, formed from a convex sequence is a Monge matrix, hence BPSM(A, k) can be solved in polynomial time.

Some new results are given in Section 6.4.5 on the finiteness of $\delta_{n-k}(A)$ for any Hankel matrix A.

Permutation matrices

Recall the definition of a permutation matrix (Definition 2.14 on Page 11). The digraph associated with a permutation matrix consists only of disjoint elementary cycles, with all arc weights equal to zero.

Theorem 6.30 ([13])**.** If $A \in \overline{\mathbb{R}}^{n\times n}$ is a permutation matrix, then BPSM(A, k) can be solved in a polynomial of n time.

Remark. This means we can solve BPSM for permutation matrices if the size of input is of order n. For example if we encode the input as the sequence of nodes that form the disjoint elementary cycles in $D(A)$, and the value of k, then the input has a size of order n, so we can polynomially solve BPSM(A, k). Another encoding could be the elements of A and the value of k. This also has a input size of order n, so we can polynomially solve BPSM(A, k) for this encoding as well.

If we encoded the input as the lengths of the disjoint elementary cycles in $D(A)$ and the value of k, then the input size is of order p. As $p \neq O(n)$ in general, we cannot polynomially solve BPSM for permutation matrices with this input. In fact, with this input, the problem is *NP*-complete. This issue will be looked at in the section on generalised permutation matrices (Section 6.4.6, page 104).

6.4.2 Pyramidal matrices

Definition 6.31. Let $A = (a_{ij})$. If for all $i, j, r, s \in N$

$$\max(i, j) < \max(r, s) \Longrightarrow a_{ij} \geq a_{rs}, \tag{6.4.2}$$

then A is called *pyramidal*.

Theorem 6.32. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ is pyramidal then $A(K)$ will be the optimal principal submatrix for BPSM(A, k) with $\delta_{n-k}(A) = \text{maper}(A(K))$. Hence we can solve BPSM(A, k) in polynomial time.

Proof. Let $A(\{l_1, \dots l_k\})$ be an arbitrary principal submatrix with $1 \leq l_1 < \cdots < l_k \leq n$. Note that

$$i \leq l_i, \text{ for all } i \in K.$$

Therefore

$$\max(i, j) \leq \max(l_i, l_j), \text{ for } i, j \in K.$$

If equality does not hold for some i and j, then by (6.4.2) we have,

$$a_{ij} \geq a_{l_i l_j}.$$

If equality does hold for some i and j, then let $l_t = \max(l_i, l_j)$. Note that $i < j \Leftrightarrow l_i < l_j$. So we have $t = \max(i, j)$ and therefore $l_t = t$. This must mean $l_{t-1} = t - 1, \ \ldots, \ l_1 = 1$. In this case

$$a_{ij} = a_{l_i l_j}.$$

Either way, $a_{ij} \geq a_{l_i l_j}$ holds. Therefore

$$\begin{aligned} \text{maper}(A(\{l_1, \ldots l_k\})) &\leq \text{maper}(A(\{1, \ldots k\})) \\ &= \text{maper}(A(K)) \\ &= \delta_{n-k}(A), \end{aligned}$$

as $A(\{l_1, \ldots l_k\})$ was arbitrary. Hence result. □

Example 6.5. Consider the matrix

$$A = \left(\begin{array}{cccc} \multicolumn{1}{c|}{9} & \multicolumn{1}{c|}{8} & \multicolumn{1}{c|}{4} & 3 \\ \cline{1-1} 8 & \multicolumn{1}{c|}{6} & \multicolumn{1}{c|}{5} & 4 \\ \cline{1-2} 5 & 4 & \multicolumn{1}{c|}{4} & 3 \\ \cline{1-3} 3 & 2 & 3 & 1 \end{array} \right).$$

The indicated lines help to check that A is pyramidal, i.e. we can see that:

9 is greater or equal to $8, 6$ and 8,

which in turn are greater or equal to $5, 4, 4, 5$ and 4,

which in turn are greater or equal to $3, 2, 3, 1, 3, 4$ and 3.

Therefore we can use Theorem 6.32 to easily find:

$$\begin{aligned}
\delta_3(A) &= \delta_{4-1}(A) = \text{maper}(A(\{1\})) = 9\\
\delta_2(A) &= \delta_{4-2}(A) = \text{maper}(A(\{1,2\})) = 16\\
\delta_1(A) &= \delta_{4-3}(A) = \text{maper}(A(\{1,2,3\})) = 20\\
\delta_0(A) &= \delta_{4-4}(A) = \text{maper}(A(\{1,2,3,4\})) = \text{maper}(A) = 22.
\end{aligned}$$

Remark. Matrices that are not pyramidal, may become such after simultaneously permuting rows and columns. It follows from (6.4.2) that the diagonal entries of the matrix must be in descending order for (6.4.2) to be satisfied.

Once rows and columns have been simultaneously permuted in this way, additional simultaneous row and column permutations may be needed between rows and columns which have a diagonal entry equal to another diagonal entry. This is fairly simple, as if a matrix can be transformed to a pyramidal matrix by way of simultaneous row and column permutations, then at most two rows and columns that have the same diagonal entry can have elements in them that are not equal to the diagonal entry.

Thus it is possible to check if any matrix can or cannot be transformed to a pyramidal matrix in this way, and to carry out this transformation quickly (polynomial time).

6.4.3 Symmetric matrices in $\mathbb{T}^{n\times n}$

In this section we show that BPSM(A,k), for a symmetric matrix $A \in \mathbb{T}^{n\times n} = \{0,-\infty\}^{n\times n}$, and k even, can be solved in O(1) time, after finding k_{max}. We also describe some cases when this is true for odd values of k. These results can immediately be applied to the question of finiteness of $\delta_{n-k}(A)$ for symmetric matrices $A \in \overline{\mathbb{R}}^{n\times n}$.

Recall the definition for PND paths (Definition 2.20, Page 13). For all k, the unique finite value for $\delta_{n-k}(A)$ is 0. Also, $\delta_{n-k}(A) = 0$ if and only if there exist PND cycles in $D(A)$ covering a total of k nodes. Hence, deciding if $\delta_{n-k}(A) = 0$ for some matrix $A \in \mathbb{T}^{n\times n}$ is equivalent to deciding whether there exist PND cycles in $D(A)$ covering exactly k nodes.

Definition 6.33. Let $A \in \overline{\mathbb{R}}^{n\times n}$ be a symmetric matrix and σ be an arbitrary cycle of length p in $D(A)$. By symmetry, for each arc (i,j) in $D(A)$, (j,i) is also an arc ("counterarc"). If p is even, we define the operation of *splitting* σ as removing alternate arcs from σ, and adding counterarcs (j,i) for each (i,j) that remains from σ, resulting in a collection of $\frac{p}{2}$ PND cycles in $D(A)$ that cover all p nodes from $V(\sigma)$. If σ is a loop, we define the operation of *splitting* σ as removing the arc. If $p \geq 3$ is odd, we define the operation of *splitting* $\sigma - v$ as removing alternate arcs from σ, starting with an incident arc to node v and ending with the other incident arc to node v, and adding counterarcs (j,i) for each (i,j) that remains from σ, resulting in a collection of $\frac{p-1}{2}$ PND cycles in $D(A)$ that cover $p-1$ nodes from $V(\sigma)$, with node v not being covered. We define *splitting* a path with p arcs as deleting alternate arcs on that path starting from the second arc and adding counterarcs to the remaining arcs, to form a collection of $\frac{p}{2}$ 2-cycles if p was even, or $\frac{p+1}{2}$ 2-cycles if p was odd.

Definition 6.34. For $A \in \overline{\mathbb{R}}^{n\times n}$, we define $F = \{k \in N : \delta_{n-k}(A) \neq -\infty\}$ and so $k_{max} = \max(F)$.

The task of finding k_{max} for a general matrix can be solved in $O(n^3)$ time [13], however we can do better for symmetric matrices:

Theorem 6.35. [9] The task of finding k_{max} for a symmetric matrix $A \in \overline{\mathbb{R}}^{n\times n}$ is equivalent to the maximum cardinality matching problem in a bipartite digraph with $2n$ nodes and can therefore be solved in $O(n^{2.5}/\sqrt{\log n})$ time.

Proof. Let $B(A)$ be the bipartite digraph (U, V, E), where $U = \{u_1, ..., u_n\}, V = \{v_1, ..., v_n\}$ and $E = \{(u_i, v_j) : a_{ij} > -\infty\}$.

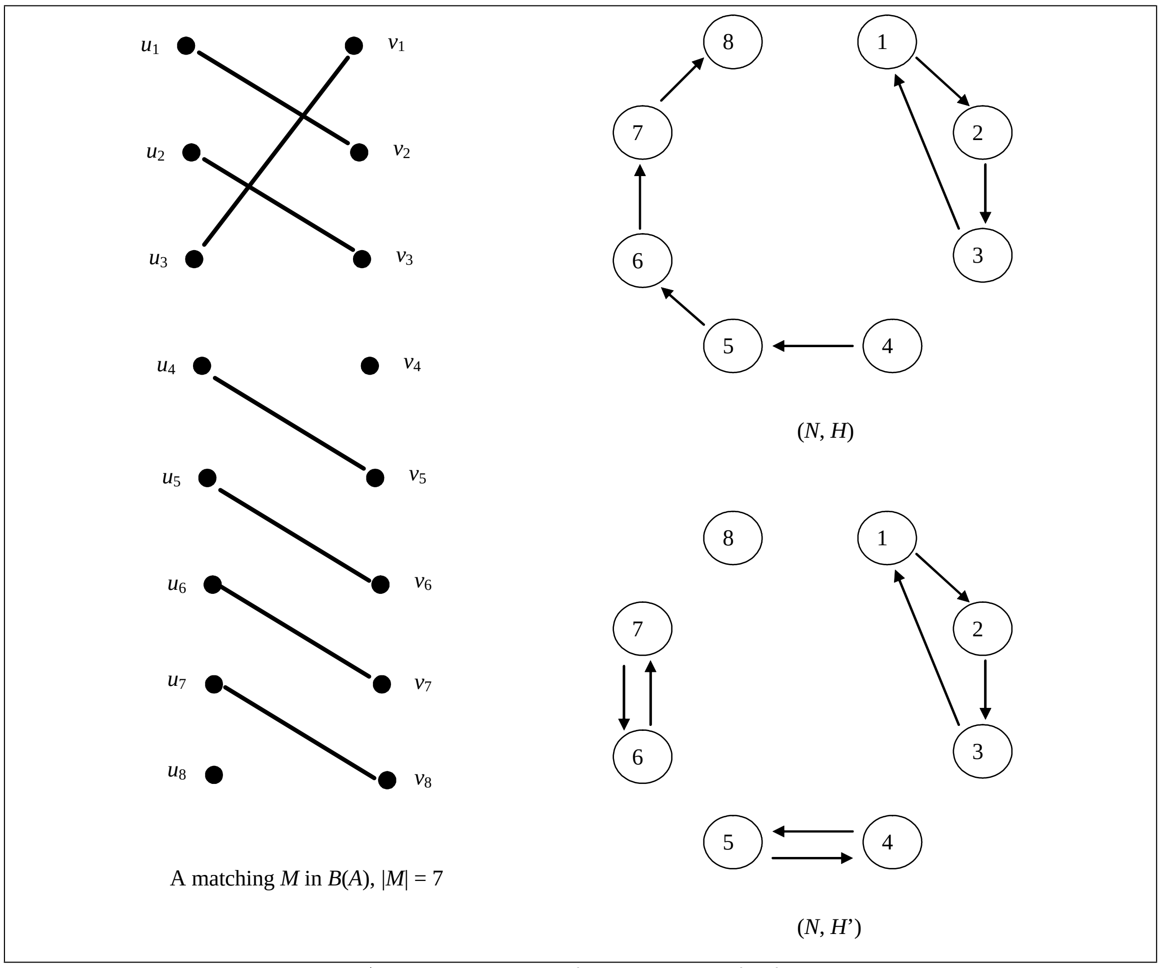

Figure 6.1: An illustration for the proof of Theorem 6.35.

Let M be a matching of maximum cardinality in $B(A)$, $|M| = m$. Obviously $k_{\max} \le m$ because if $k = k_{\max}$ then there are k finite entries in A, no two in the same row or column, say $a_{i_r \pi(i_r)}, r = 1, ..., k$, and so there is a matching of cardinality k in $B(A)$, namely, $\{(u_{i_r}, v_{\pi(i_r)}) : r = 1, ..., k\}$.

We now prove $k_{\max} \ge m$. The set of arcs $H = \{(i, j) : (u_i, v_j) \in M\}$ in $D(A)$ consists of directed PND elementary paths or cycles, since both the outdegree and indegree of each node in (N, H) is at most one. We will call a path *proper* if it is not a cycle.

Construct from H another set H' as follows (see Figure 6.1): If all paths in H are cycles then set $H' = H$. Now suppose that at least one proper path exists. Splitting every proper path in (N, H), we obtain a digraph (N, H') which consists of original cycles in (N, H) and a number of cycles of length 2. All cycles in (N, H') are PND.

Each set of PND cycles in $D(A)$ determines a matching in $B(A)$ whose cardinality is equal to the total number of arcs of these cycles. Thus none of the proper paths in (N, H) could have been of odd length, say s, as otherwise the total number of arcs on cycles constructed from this path would be $s + 1$, a contradiction with the maximality of M. Hence $|H'| = m$ and thus $k_{\max} \geq m$.

The complexity statement now follows from [5]. $\square$

Theorem 6.36. If $A \in \mathbb{T}^{n\times n}$ is a symmetric matrix and $\delta_{n-l}(A) = 0$ for some $l \in N$, then $\delta_{n-k}(A) = 0$ for all even $k \leq l$, and $\delta_{n-k}(A) = 0$ for all $k = l - r, \dots, l$, where r is the number of odd cycles in a collection of PND cycles in $D(A)$ that cover l nodes.

Proof. Let $\{\sigma_1, \dots, \sigma_t\}$ be a collection of PND cycles in $D(A)$ covering l nodes with σ_i having odd length for $i = 1, \dots, r$ and even length otherwise. By splitting the cycles σ_i for $i = r + 1, \dots, t$ if needed, we may assume that all these cycles are 2-cycles. By splitting one by one the cycles $\sigma_i - v_i$ for $v_i \in V(\sigma_i)$ and $i = 1, \dots, r$, we get that $\delta_{l-i} = 0$ for $i = 1, \dots, r$. After these splittings, all remaining cycles are 2-cycles and removing them one by one proves the result. $\square$

Corollary 6.37. Let $A \in \mathbb{T}^{n\times n}$ be a symmetric matrix. For all even $k \leq k_{max}$, $\delta_{n-k}(A) = 0$, and if $\delta_{n-k}(A) = 0$ for some odd $k \in N$ then $\delta_{n-k+1}(A) = 0$.

If A has at least one zero on the main diagonal, (or equivalently, if the digraph $D(A)$ has at least one cycle of length one (alternatively known as a *loop*)), then we can derive a number of properties:

Theorem 6.38. If $A \in \mathbb{T}^{n\times n}$ is a symmetric matrix, and there exists a collection of PND cycles in $D(A)$ covering l nodes, at least one of which is a loop, then $\delta_{n-k}(A) = 0$ for all $k \leq l$.

Proof. Let $\{\sigma_1, \dots, \sigma_t\}$ be a collection of PND cycles in $D(A)$ covering k_{max} nodes with σ_i having odd length for $i = 1, \dots, r$ and even length otherwise. Assume σ_r is a loop. By splitting the cycles σ_i for $i = r+1, \dots, t$ if needed, we may assume that all these cycles are 2-cycles. By splitting one by one the cycles $\sigma_i - v_i$ for $v_i \in V(\sigma_i)$ and $i = 1, \dots, r-1$, we get that $\delta_{n-l+i} = 0$ for $i = 1, \dots, r-1$. After these splittings, all remaining cycles except σ_r are 2-cycles. Removing the 2-cycles one by one gives us $\delta_{n-l+i} = 0$ for odd $i \in \{r+1, \dots, l-1\}$. Removing σ_r and the 2-cycles one by one gives us $\delta_{n-l+i} = 0$ for even $i \in \{r, \dots, l\}$. □

If $l \in \{k_{max}, k_{max} - 1\}$ in Theorem 6.38, then we can completely solve BPSM for this type of matrix:

Theorem 6.39. If $A \in \mathbb{T}^{n \times n}$ is a symmetric matrix, $l \in \{k_{max}, k_{max} - 1\}$ and there exists a collection of PND cycles in $D(A)$ covering l nodes, at least one of which is a loop, then $\delta_{n-k}(A) = 0$ for all $k \le k_{max}$.

Proof. The statement immediately follows from Theorem 6.38 and the fact that $\delta_{n-k_{max}}(A) = 0$. □

Theorem 6.40. If $A = (a_{ij}) \in \mathbb{T}^{n \times n}$ is a symmetric matrix and $D(A)$ contains a loop, then $\delta_{n-k}(A) = 0$ for all $k \le k_{max}$.

Proof. We assume that (j, j) is a loop in $D(A)$. As $\delta_{n-k_{max}}(A) = 0$, there exist PND cycles in $D(A)$ $\sigma_1, \sigma_2, \dots, \sigma_t$ in $D(A)$ covering k_{max} nodes. We need to show there exist PND cycles $\sigma'_1, \sigma'_2, \dots, \sigma'_{t'}$ in $D(A)$, at least one being a loop, that cover k_{max} or $k_{max} - 1$ nodes.

Clearly if $(j, j) \in \{\sigma_1, \sigma_2, \dots, \sigma_t\}$ then we can use Theorem 6.39 and we are done, so assume not. Then j is covered by these cycles, as otherwise, (j, j) together with $\sigma_1, \sigma_2, \dots, \sigma_t$ would form PND cycles in $D(A)$ covering $k_{max}+1$ nodes, which contradicts the definition of k_{max}. Hence there exists one cycle $\sigma_r = (j, i_2, i_3, \dots, i_p, j) \in \{\sigma_1, \sigma_2, \dots, \sigma_t\}$.

If p is odd, then we can split $\sigma_r - j$, and add (j,j) to the resulting cycles to form PND cycles in $D(A)$ covering k_{max} nodes. If instead p is even, then we can split σ_r, remove the 2-cycle that contains p, and add (j,j) to the remaining cycles to form PND cycles in $D(A)$ covering $k_{max}-1$ nodes. The result then follows from Theorem 6.39. □

Recall that we have previously defined $F = \{k \in N : \delta_{n-k}(A) \neq -\infty\}$.

Definition 6.41. For $A \in \overline{\mathbb{R}}^{n\times n}$, let $k_{oddmin} = \min\limits_{\text{odd } k} F$.

By Corollary 6.37, for symmetric matrices, unless $k_{max} = 1$, the smallest even $k \in F$ is 2. However, k_{oddmin}, the smallest odd value in F is more tricky.

Remark. If there exist PND cycles $\sigma_1, \sigma_2, \ldots, \sigma_t$ in $D(A)$ such that an odd number of nodes is covered, then at least one of the cycles is odd. Hence k_{oddmin} is the length of a shortest odd cycle in $D(A)$. This cycle can be found polynomially [31]. (By calculating powers of A, we can determine the value of k_{oddmin} as the smallest odd k such that $\max\limits_{i\in N} a^k_{ii}$ is finite.) Note that k_{oddmin} does not exist if there is no odd cycle in $D(A)$, and if this is the case, then $\delta_{n-k} = -\infty$ for all odd k. So for the remainder of this section, we shall assume that k_{oddmin} exists.

Theorem 6.42. Let $A \in \mathbb{T}^{n\times n}$ be a symmetric matrix, $\sigma_1, \sigma_2, \ldots, \sigma_t$ be a collection of PND cycles in $D(A)$ covering k' nodes, with at least one having odd length. Then $\delta_{n-k}(A) = 0$ for all odd $k \in \{l', \ldots, k'\}$, where l' is the minimum length of the odd cycles in this collection.

Proof. Without loss of generality, assume the length of σ_t is l'. Then the PND cycles $\sigma_1, \sigma_2, \ldots, \sigma_{t-1}$ in $D(A)$ cover $k'-l'$ nodes, hence $\delta_{n-k'+l'}(A) = 0$. By Corollary 6.37, $\delta_{n-k}(A) = 0$ for all even $k \in \{0, \ldots, k'-l'\}$. Take an arbitrary even $k \in \{0, \ldots, k'-l'\}$. So $k + l' \in \{l', \ldots, k'\}$. There exist PND cycles $\sigma'_1, \sigma'_2, \ldots, \sigma'_{t'}$ in $D(A)$ covering k nodes other than those in $V(\sigma_t)$. Therefore, $\sigma'_1, \sigma'_2, \ldots, \sigma'_{t'}$ and σ_t are PND cycles in $D(A)$ covering $k + l'$ nodes. Hence the result. □

Corollary 6.43. Let $A \in \mathbb{T}^{n\times n}$ be a symmetric matrix, $\sigma_1, \sigma_2, \ldots, \sigma_t$ be a collection of PND cycles in $D(A)$ covering k_{max} or $k_{max}-1$ nodes, with at least one having length k_{oddmin}. Then we can decide whether $\delta_{n-k}(A)$ is 0 or $-\infty$ for all k in linear time, after finding k_{max} and k_{oddmin}.

Proof. The result follows from Corollary 6.37 and Theorem 6.42. $\square$

Theorem 6.44. If $A \in \mathbb{T}^{n\times n}$ is a symmetric matrix, then $\delta_{n-k}(A) = 0$ for all odd $k \in \{k_{oddmin}, \ldots, k_{max} - k_{oddmin}\}$.

Proof. As $\delta_{n-k_{max}}(A) = 0$, there exist PND cycles $\sigma_1, \sigma_2, \ldots, \sigma_t$ in $D(A)$ that cover k_{max} nodes. There exists a cycle σ in $D(A)$ of length k_{oddmin}.

Delete all nodes in $V(\sigma)$ from $\sigma_1, \sigma_2, \ldots, \sigma_t$, as well as incident arcs. As the cycles were PND and each node was incident to precisely two arcs, up to $2k_{oddmin}$ arcs have been deleted. Therefore this leaves a total of at least $k_{max} - 2k_{oddmin}$ arcs within the remaining PND cycles and paths that have arisen from deleting the arcs from the cycles. Split any paths into 2-cycles. We now have PND cycles in $D(A)$ covering at least $k_{max} - 2k_{oddmin}$ arcs, and therefore at least $k_{max} - 2k_{oddmin}$ nodes, none of which are nodes on σ.

Therefore, by Corollary 6.37, for all even $i \le k_{max} - 2k_{oddmin}$, there exist PND cycles $\sigma'_1, \sigma'_2, \ldots, \sigma'_{t'}$ in $D(A)$ covering i nodes, but none on σ. So for all even $i \le k_{max} - 2k_{oddmin}$, we have PND cycles $\sigma'_1, \sigma'_2, \ldots, \sigma'_{t'}$ and σ in $D(A)$ which cover $i + k_{oddmin}$ nodes. Hence the result. $\square$

Remark. Note that $\{k_{oddmin}, \ldots, k_{max} - k_{oddmin}\} \neq \emptyset \iff k_{oddmin} \le \dfrac{k_{max}}{2}$.

Corollary 6.45. Let $A \in \mathbb{T}^{n\times n}$ be a symmetric matrix, with PND cycles in $D(A)$ covering k_{max} or $k_{max}-1$ nodes, one having odd length of at most $k_{max} - k_{oddmin}$. Then we can decide whether $\delta_{n-k}(A)$ is 0 or $-\infty$ for all k in linear time, after finding k_{max} and k_{oddmin}.

Proof. The result follows from Corollary 6.37, Theorem 6.42 and Theorem 6.44. $\square$

Corollary 6.46. Let $A \in \mathbb{T}^{n\times n}$ be a symmetric matrix, with PND cycles in $D(A)$ covering k_{max} or $k_{max}-1$ nodes, one having odd length of at most $\dfrac{k_{max}}{2}$. Then we can decide whether $\delta_{n-k}(A)$ is 0 or $-\infty$ for all k in linear time, after finding k_{max}.

Proof. The result follows from Corollary 6.45 as $\frac{k_{max}}{2} \le k_{max} - k_{oddmin}$. □

Corollary 6.47. Let $A \in \mathbb{T}^{n\times n}$ be a symmetric matrix, with PND cycles in $D(A)$ covering k_{max} or $k_{max}-1$ nodes, two having odd length. Then we can decide whether $\delta_k(A)$ is 0 or $-\infty$ for all k in linear time, after finding k_{max}.

Proof. The result follows from Corollary 6.46. □

Remark. Note that solving an assignment problem for $A = (a_{ij}) \in \mathbb{T}^{n\times n}$ is equivalent to deciding whether the classical permanent of the matrix $B = (b_{ij})$ is positive where B is defined by $b_{ij} = 1$ if $a_{ij} = 0$ and $b_{ij} = 0$ otherwise. Therefore the statements in Section 6.4.3 solve in special cases the question: Given $A \in \{0,1\}^{n\times n}$, and $k \le n$, is there a $k \times k$ principal submatrix of A whose classical permanent is positive?

6.4.4 Symmetric matrices with no odd cycles

This section will show that if the digraph of a symmetric matrix A has no cycles of odd length, then we can solve BPSM(A) in polynomial time.

Definition 6.48. For BPSM$(A, 2k)$, we will let a *symmetric solution* be a set S of pairs of indices of $2k$ independent entries of an $2k \times 2k$ principal submatrix of $A = (a_{ij})$ in which $(i,i) \notin S$ for any i, and if $(i,j) \in S$ then $(j,i) \in S$. Let $w(A,S) = \sum_{(i,j)\in S} a_{ij}$. We will let a *symmetric optimal solution* be one in which $w(A,S) = \delta_{n-2k}(A)$.

Note that if $\delta_{n-2k}(A)$ is finite, then a symmetric solution for BPSM$(A, 2k)$ is equivalent to a composition of k cycles of length 2 in $D(A)$. If $\delta_{n-2k}(A) = -\infty$, then a symmetric solution for BPSM$(A, 2k)$ is equivalent to a composition of k cycles of length 2 in $D_C(A)$ (see Definition 2.33 on Page 14 for the definition of $D_C(A)$).

Lemma 6.49. For even n, a symmetric matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ corresponding to a graph $D(A)$ with no odd cycles (or equally, corresponding to a bipartite digraph) and an elementary cycle $\sigma = (i_1, i_2, \ldots, i_n, i_1)$ in $D(A)$, there is a symmetric solution S for BPSM(A, n) with $w(A, S) \geq w(A, \sigma)$.

Proof. We can assume $n > 2$, otherwise it is trivial. Let

$$\pi_1 = (i_1 i_2) \circ (i_3 i_4) \circ \cdots \circ (i_{n-1}, i_n)$$

and

$$\pi_2 = (i_2, i_3) \circ (i_4, i_5) \circ \cdots \circ (i_n, i_1).$$

Then

$$w(A, \sigma) = a_{i_1 i_2} + a_{i_2 i_3} + \cdots + a_{i_{n-1} i_n} + a_{i_n i_1}$$

$$\begin{aligned} w(A, \pi_1) &= a_{i_1 i_2} + a_{i_2 i_1} + \cdots + a_{i_{n-1} i_n} + a_{i_n i_{n-1}} \\ &= 2(a_{i_1 i_2} + a_{i_3 i_4} + \cdots + a_{i_{n-1} i_n}) \end{aligned}$$

$$\begin{aligned} w(A, \pi_2) &= a_{i_2 i_3} + a_{i_3 i_2} + \cdots + a_{i_n i_1} + a_{i_1 i_n} \\ &= 2(a_{i_2 i_3} + a_{i_4 i_5} + \cdots + a_{i_n i_1}). \end{aligned}$$

Assume wlog that $w(A, \pi_1) \geq w(A, \pi_2)$, then

$$\begin{aligned} w(A, \sigma) &= a_{i_1 i_2} + a_{i_2 i_3} + \cdots + a_{i_{n-1} i_n} + a_{i_n i_1} \\ &= \frac{1}{2}(w(A, \pi_1) + w(A, \pi_2)) \\ &\leq \frac{1}{2}(w(A, \pi_1) + w(A, \pi_1)) \\ &= w(A, \pi_1). \end{aligned}$$

So let $S = \{(i_1, i_2), (i_2, i_1), \ldots, (i_{n-1}, i_n), (i_n, i_{n-1})\}$ and the result holds. □

Theorem 6.50. Given a symmetric matrix $A \in \overline{\mathbb{R}}^{n \times n}$, with $D(A)$ having no odd cycles (or equally, corresponding to a bipartite digraph), and given $2k \in N$, there always exists a symmetric optimal solution to BPSM$(A, 2k)$.

Proof. There is a $\pi = \sigma_1 \circ \sigma_2 \circ \cdots \circ \sigma_p$ such that $w(A, \pi) = \delta_{n-2k}(A)$. Take any of these cycles $\sigma_1, \sigma_2, \ldots, \sigma_p$, say $\sigma = (i_1, i_2, \ldots, i_r, i_1)$, for which $r > 2$. Let $B = A(i_1, i_2, \ldots, i_r, i_1)$. Then using Lemma 6.49 with B and σ, we can replace σ with cycles of length 2, and have at least as much weight. (In fact we will have the same weight, as otherwise we would have a greater weight than $\delta_{n-2k}(A)$, which isn't possible.) Repeat this for all such elementary cycles that have a length greater than 2.

We then have a composition of cycles of the form $\pi' = \sigma'_1 \circ \sigma'_2 \circ \cdots \circ \sigma'_{p'}$ such that $w(A, \pi') = \delta_{n-2k}(A)$, and each $\sigma \in \{\sigma'_1, \sigma'_2, \ldots, \sigma'_{p'}\}$ has length 2. So this corresponds to a symmetric optimal solution. □

Theorem 6.51. Given a symmetric matrix $A \in \overline{\mathbb{R}}^{n \times n}$, with $D(A)$ having no odd cycles, we can solve BPSM$(A, 2k)$ for all k in polynomial time.

Proof. We can assume there is a symmetric optimal solution by Theorem 6.50. Let $D = D(A) = (V, E, w)$ be the weighted digraph associated with $A = (a_{ij})$. As A is symmetric, if arc $(i, j) \in E$, then $(j, i) \in E$ as well, and we may form an equivalent weighted undirected graph $G = (V, E', w)$, where $(i, j) \in E'$ for all pair of pairs $(i, j), (j, i) \in E$. As there are no odd cycles in D, and hence none in G, we know that G must be a weighted (undirected) bipartite graph, say $G = (U, W, E, w)$, where $|U| = n_1$, $|W| = n_2$ and $n = n_1 + n_2 = |U \cup W|$. If $n_1 \neq n_2$, then WLOG we can assume $n_1 > n_2$, and so add $n_1 - n_2$ nodes to W without adding additional edges. Whether or not we extended the graph, we will call this undirected weighted digraph G' and its associated matrix B.

We wish to find a symmetric optimal solution set with $2k$ elements. This is equivalent to finding a set of k disjoint pairs of arcs in $D(A)$ with greatest weight. This is in turn equivalent to finding a matching in G' of k arcs with greatest weight. This problem is equivalent to BSM(B, k), which we can polynomially solve. For all r and s, if b_{rs} is an element of the optimum matrix for BSM(B, k), then the corresponding elements a_{ij} and a_{ji} are elements of the optimum matrix for BPSM$(A, 2k)$. $\square$

So which types of matrix are symmetric and have a digraph without odd cycles? Well consider an arbitrary undirected graph G of the form described in the proof above. Wlog, we may set $U = \{v_1, v_2, \ldots, v_{n_1}\}$ and $W = \{v_{n_1+1}, v_{n_1+2}, \ldots, v_n\}$. For all $(x, y) \in E$, $x \in U$ and $y \in W$ (or $x \in W$ and $y \in U$), but never $x, y \in U$ and never $x, y \in W$. Creating an equivalent digraph D as described in the above proof, and the matrix A such that $D = D(A)$, then we see A is of the following form:

$$\begin{pmatrix} -\infty & C \\ C^T & -\infty \end{pmatrix},$$

where $C = (c_{ij}) = w(i, j)$, and C^T is the transpose of C. The weight function w was arbitrary, hence we have the following result:

Theorem 6.52. For any matrix

$$A = \begin{pmatrix} -\infty & C \\ C^T & -\infty \end{pmatrix} \in \overline{\mathbb{R}}^{n \times n},$$

where $C = (c_{ij})$ and $c_{ij} \in \overline{\mathbb{R}}$, we can solve BPSM($A$) in polynomial time.

Note that C does not need to be a square matrix, but if it is, then we can simultaneously rearrange the rows and columns of A to give us a chessboard type matrix as described in the next section (page 103).

6.4.5 Hankel matrices

Recall the definition of a Hankel matrix, as given in section 6.4.1. In this section we show that finiteness of $\delta_{n-k}(H)$ can be easily decided for any Hankel matrix H. Since Hankel matrices are symmetric, we can use some of the results of Section 6.4.3.

Theorem 6.53. If $\{g_r \in \overline{\mathbb{R}} : r = 1, \ldots, 2n-1\}$ is the sequence generating Hankel matrix $H = (h_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ and $g_r \neq -\infty$ for some odd r, then $\delta_{n-k}(H) \neq -\infty$ for all $k \leq k_{max}$.

Proof. Let $C = (c_{ij})$ be defined by $c_{ij} = 0$ if $h_{ij} \neq -\infty$ and $c_{ij} = -\infty$ otherwise. Assume $g_r \neq -\infty$ for some odd r. So $(\exists i)\ c_{ii} \neq -\infty$, i.e. $(\exists i)\ c_{ii} = 0$. We now use Theorem 6.40 to give us $\delta_{n-k}(C) = 0$ for all $k \leq k_{max}$. Then as $\delta_{n-k}(C) = 0$ if and only if $\delta_{n-k}(H) \neq -\infty$, the theorem follows. □

Theorem 6.54. If a matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ is any matrix such that $a_{ij} = -\infty$ if $i + j$ is even, then $\delta_{n-k}(A) = -\infty$ for all odd k.

Proof. Assume $A = (a_{ij})$ is a matrix such that $a_{ij} = -\infty$ if $i + j$ is even. If a_{ij} is finite then $i + j$ is odd. So i and j must be of different parities.

Let $\sigma = (i_1, i_2, \ldots, i_p) \in C_n$ be any cyclic permutation of arbitrary length p such that $w(A, \sigma) \neq -\infty$.

As $w(A,\sigma) \neq -\infty$, then $a_{i_j,i_{j+1}} \neq -\infty$. So i_j and i_{j+1} must be of different parities. This means elements in the sequence $i_1, i_2, \ldots, i_p, i_1$ alternate between even and odd. This means p must be an even number, i.e. there are no cyclic permutations σ of odd length of finite weight. Hence result. $\square$

If A is symmetric, then together with Corollary 6.37, this gives us:

Theorem 6.55. If $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ is a symmetric matrix such that $a_{ij} = -\infty$ if $i+j$ is even, then $\delta_{n-k}(A) \neq -\infty$ for all even $k \leq k_{max}$, and $\delta_{n-k}(A) = -\infty$ for all odd k.

The type of matrix that satisfies Theorem 6.55 is like a chessboard. We can reformulate this for Hankel matrices:

Theorem 6.56. If $\{g_r \in \overline{\mathbb{R}} : r = 1, \ldots, 2n-1\}$ is the sequence generating Hankel matrix H and $g_r = -\infty$ for all odd r, then $\delta_{n-k}(H) \neq -\infty$ for all even $k \leq k_{max}$ and $\delta_{n-k}(H) = -\infty$ for all odd k.

Combining Theorem 6.53 and Theorem 6.56 enables us to decide whether $\delta_{n-k}(H)$ is finite or not for any Hankel matrix H.

Theorem 6.57. If $\{g_r \in \overline{\mathbb{R}} : r = 1, \ldots, 2n-1\}$ is the sequence generating Hankel matrix H then

1. $\delta_{n-k}(H) \neq -\infty$ for all even $k \leq k_{max}$,
2. $\delta_{n-k}(H) = -\infty$ for all odd k if $g_r = -\infty$ for all odd r, and
3. $\delta_{n-k}(H) \neq -\infty$ for all odd $k \leq k_{max}$ if $g_r \neq -\infty$ for some odd r.

We can use this result to solve BPSM(H) for any Hankel matrix $H \in \mathbb{T}^{n\times n}$:

Corollary 6.58. If $\{g_r \in \mathbb{T} : r = 1, \ldots, 2n-1\}$ is the sequence generating Hankel matrix H then

1. $\delta_{n-k}(H) = 0$ for all even $k \leq k_{max}$,
2. $\delta_{n-k}(H) = -\infty$ for all odd k if $g_r = -\infty$ for all odd r, and
3. $\delta_{n-k}(H) = 0$ for all odd $k \leq k_{max}$ if $g_r \neq -\infty$ for some odd r.

6.4.6 Generalised permutation matrices

Recall the definition of a generalised permutation matrix (Definition 2.15 on Page 12). The digraph associated with a generalised permutation matrix consists only of disjoint elementary cycles, with all arc weights finite (instead of zero in the case of permutation matrices).

We will give an algorithm to solve BPSM(A, k) for a generalised permutation matrix A. We will explain how this algorithm can be adapted to solve BPSM(A) more efficiently than just repeating the algorithm for BPSM(A, k) for each value of k. We will then discuss why the type of input (in particular, the length of the input) is important in determining the complexity (and polynomiality) of the algorithm. We shall explain why encoding the input in one way allows us to construct a polynomial time algorithm for solving BPSM, and encoding the input in another way means BPSM becomes NP-complete.

Solving BPSM(A, k)

If we want to solve BPSM(A, k) for a generalised permutation matrix, we can do the following (which leads to the formation of an algorithm to solve BPSM for generalised permutation matrices).

Suppose we are given a generalised permutation matrix $A \in \overline{\mathbb{R}}^{n \times n}$ and $k \in N$. From A, we can calculate the number of (disjoint) elementary cycles in $D(A)$, say p. So we have cycle 1, cycle 2, ..., cycle p. We can calculate the length and weight of cycle j, say l_j and w_j respectively, for $j = 1, \ldots, p$.

The aim is to select a set of cycles that will maximise the sum of selected cycle weights, with the total length of selected cycles equal to k. The sum of selected cycle weights will then equal $\delta_{n-k}(A)$. We can also select elements of A that correspond to arcs in selected cycles. These elements lie in a $k \times k$ principal submatrix of A and have the maximum sum (equal to $\delta_{n-k}(A)$) of any such elements. Therefore we would have solved BPSM(A, k).

It is more convenient to refer to a cycle's index rather than the cycle itself. If we do this, to keep track of which cycles, when selected, give a total weight w, and total length l, we will use a cycle index set $S \subseteq \{1, \dots, p\}$. So if we selected elements of A corresponding to arcs of cycles with cycle indices in S, and added them up, the total would be w, and we would select l elements of A.

We let the set S and values w and l form a triple (S, w, l). From all such triples, to be able to solve BPSM for generalised permutation matrices, we want S to be any set of cycle indices from $\{1, \dots, p\}$ such that w is as big as possible and $l = k$.

To find a triple that satisfies these properties, it is best to work systematically. One way to do this is by doing the following. We start by considering $(\emptyset, 0, 0)$ in stage 0. Then we consider all triples (S, w, l) with $S = \{1\}$ (in stage 1). Then we consider all triples with $S \subseteq \{1, 2\}$ (in stage 2). Then we consider all triples with $S \subseteq \{1, \dots, 3\}$ (in stage 3). Until finally we consider all triples with $S \subseteq \{1, \dots, p\}$ (in stage p).

In stage 0, $M_0 = \{(\emptyset, 0, 0)\}$. The triple $(\emptyset, 0, 0)$ means if we select no entries from any block, then we have total length 0 and total weight 0.

In stage j, $j = 1, \dots, p$, for each $l = 1, \dots, k$, we store in a set M_j, the triple (S, w, l) with $S \subseteq \{1, \dots, j\}$ that has w is as big as possible and has l as its third components (if such a triple exists). This triple is chosen as follows.

In stage j, we start by copying all previous triples from M_{j-1} by setting $M_j = M_{j-1}$. We have $S \subseteq \{1, \dots, j\}$. For each $(S, w, l) \in M_{j-1}$ with $l + l_j \leq k$, we then update M_j by doing the following. If there is no $(S', w', l + l_j) \in M_j$, then let $M_j = M_j \cup \{(S \cup \{j\}, w + w_j, l + l_j)\}$. If there is a $(S', w', l + l_j) \in M_j$ with $w' < w + w_j$, then let $M_j = M_j - \{(S', w', l + l_j)\} \cup \{(S \cup \{j\}, w + w_j, l + l_j)\}$. By updating in this way, we ensure that the second coordinate of (S, w, l) (where $S \subseteq \{1, \ldots, j\}$ and $l \in K$) is the biggest possible sum of any selection of the first j cycle weights with total length l.

By the end of stage p, we simply have to select the (S^*, w^*, k) triple from the set M_p, and we can select k independent elements (which are the elements of A given by arcs of cycles with cycle indices in S^*) that lie in a $k \times k$ principal submatrix of A with total sum of $\delta_{n-k}(A) = w^*$ (which is the maximum possible). If there was no (S^*, w^*, k) in M_p, then no finite solution exists, i.e. $\delta_{n-k}(A) = -\infty$.

In Figure 6.2 we give algorithm BPSMGENPERM for solving BPSM(A, k) for a generalised permutation matrix $A \in \overline{\mathbb{R}}^{n \times n}$ and $k \in N$. It formalises the above discussion, and was partly based on an algorithm from [38, page 422]. We then formally prove that BPSMGENPERM correctly and polynomially solves BPSM for a generalised permutation matrix.

```
Algorithm BPSMGENPERM
Input: Generalised permutation matrix $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$.
Output: Returns $\delta_{n-k}(A)$, and if $\delta_{n-k}(A)$ is finite, then also $k$ independent entries
of a $k \times k$ principal submatrix of $A$ whose total is $\delta_{n-k}(A)$.

1. Set $p$ equal to the number of distinct elementary cycles in $D(A)$.
2. Set $M_0 = \{(\emptyset, 0, 0)\}$
3. For $j = 1$ to $p$ :
   (a) Set $w_j$ equal to the $j$'th cycle weight
   (b) Set $l_j$ equal to the $j$'th cycle length
   (c) Set $M_j = M_{j-1}$
   (d) For each element $(S, w, l) \in M_{j-1}$ with $l + l_j \leq k$ :
       i. If $\nexists(S', w', l + l_j) \in M_j$, then add $(S \cup \{j\}, w + w_j, l + l_j)$ to $M_j$.
       ii. If $\exists(S', w', l + l_j) \in M_j$ and $w' < w + w_j$, then remove $(S', w', l + l_j)$
           from $M_j$ and add $(S \cup \{j\}, w + w_j, l + l_j)$ to $M_j$.
4. If $\exists(S, w, k) \in M_p$, then return $\delta_{n-k}(A) = w$ and $\forall j \in S$ select $a_{st}$ for all arcs
   $(v_s, v_t)$ in cycle $j$. Else return $\delta_{n-k}(A) = -\infty$.
```

Figure 6.2: An algorithm for solving BPSM(A, k) for a generalised permutation matrix A and integer k.

Lemma 6.59.

1. If $(S, w, l) \in M_j$ at Step 4 of the algorithm, then
   (a) $S \subseteq \{1, \dots, j\}$,
   (b) $\sum\limits_{i \in S} w_i = w$,
   (c) $\sum\limits_{i \in S} l_i = l \leq k$,
   (d) If $(S', w', l) \in M_j$, then $S' = S$ and $w' = w$,
   (e) If $S' \subseteq \{1, \dots, j\}$, $w' = \sum\limits_{i \in S'} w_i$ and $\sum\limits_{i \in S'} l_i = l$ then $w' \leq w$.

2. If $S \subseteq \{1, \ldots, j\}$, $\sum\limits_{i \in S} w_i = w$, and $\sum\limits_{i \in S} l_i = l \leq k$, then at Step 4 of the algorithm, $\exists (S', w', l) \in M_j$ where $w \leq w'$.

Proof. Statements 1(a)-1(c) are proved by induction on j, and hold automatically for $j = 0$. For $j > 0$, assume $(S, w, l) \in M_j$. We have two cases to consider:

Case 1: If $j \notin S$, then $(S, w, l) \in M_{j-1}$, so 1(a)-1(c) follow by induction.

Case 2: If $j \in S$, then $(S - \{j\}, w - w_j, l - l_j) \in M_{j-1}$. Note that $l_j \leq l \leq k$, as the third coordinate of this lies between 0 and $k - l_j$. So again 1(a)-1(c) follow by induction.

To prove 1(d), we use the fact that each element of M_j has a unique third component due to the way Step 3(d) of the algorithm was constructed.

To prove 2, we use induction on $\max\limits_{j' \in S} j'$. It holds for $S = \emptyset$, and for the inductive step, assume $t = \max\limits_{j' \in S} j'$. Note that $t \leq j$, so $t - 1 \leq j - 1$. Therefore $\exists (S', w - w_t, l - l_t) \in M_{t-1}$ by induction. Thus in Step 3(d) of the construction of M_t, $(S' \cup \{t\}, w, l)$ was added to M_t. Then either $(S' \cup \{t\}, w, l) \in M_j$ or $\exists (S'', w'', l) \in M_j$ with $w < w''$. Either way, 2 holds.

To prove 1(e), note that by 2, $\exists (S'', w'', l) \in M_j$, with $w' \leq w''$. By 1(d), we see that $S'' = S$ and $w'' = w$, therefore $w' \leq w$, and 1(e) follows. $\square$

Remark. From 2, we see that if we have a set of cycle indices $S \subseteq \{1, \ldots, p\}$ with the total lengths of these cycles equal to k (and total weight w), then $\exists (S^*, w^*, k) \in M_p$ (which gives at least as much total weight w^* as w does). Then from 1(e), we see that if $(S^*, w^*, k) \in M_p$, then no other cycle index set, with corresponding cycle lengths totaling k, will provide a bigger total weight than w^*. Selecting elements of A corresponding to arcs of cycles of $D(A)$ that have indices in S^* and adding them up will give w^*, which is the highest possible value, so $w = \delta_{n-k}(A)$.

Theorem 6.60. The algorithm BPSMGENPERM correctly solves BPSM(A, k) for any generalised permutation matrix $A \in \overline{\mathbb{R}}^{n \times n}$ and $k \in N$, in $O(n^2)$ time.

Proof. Correctness follows from Lemma 6.59: Step 4 of the algorithm chooses an element (S, w, k) from M_p (assuming $M_p \neq \emptyset$) with third component k. Thus it follows that the solution generated from S is feasible (i.e. if we select all elements of A corresponding to arcs of cycles in $D(A)$ that have cycle indices in the set S, then k finite elements of A will be selected, resulting in a finite total weight) (by 1(a) and 1(c)), its total weight is w (by 1(b)), and there is no better solution (by 1(e)). By part 2 of the lemma, it follows that if $M_p = \emptyset$ at Step 4 of the algorithm, then there is no feasible cycle index set, i.e. $\delta_{n-k}(A) = -\infty$.

For the time bound, notice that the size of each set M_{j-1} is no greater than k, because there is at most one element in M_{j-1} with the same third component (by 1(d)). Each update operation of Step 3(d) can be done in constant time, and must be repeated for all $O(k)$ elements of M_{j-1}. In Step 1, we can move from one element of a cycle to the next element, until we are back to the start of the cycle, then move on to the next cycle, counting the number of cycles as we go. This can be done in $O(n)$ time. Step 4 can be done in $O(k)$ time. Steps 2, 3(a), 3(b) and 3(c) require one operation each and so can be performed in constant time. The whole of Step 3 is repeated p times, so algorithm BPSMGENPERM runs in $O(n + pk)$ time. As p and n are $O(n)$, this becomes $O(n^2)$ time. □

Solving BPSM(A)

For a generalised permutation matrix A, the algorithm BPSMGENPERM solves BPSM(A, k) for one particular value of k only. If we want to solve BPSM(A, k) for all values of k (i.e. solve BPSM(A)), then we could repeat the algorithm n times, changing the value of k each time. This means we could solve BPSM(A) in $O(n^3)$ time.

However, we can modify BPSMGENPERM slightly so that it can solve BPSM(A) more quickly than just repeating for all k: Create a modified algorithm of BPSM-GENPERM called BPSMGENPERM2 by replacing k by n in Step 3(d) of BPSM-

```
Algorithm BPSMGENPERM2
Input: Generalised permutation matrix $A \in \overline{\mathbb{R}}^{n\times n}$.
Output: For $k = 1,\dots,n$, returns $\delta_{n-k}(A)$, and if $\delta_{n-k}(A)$ is finite, then also $k$
independent entries of a $k \times k$ principal submatrix of $A$ whose total is $\delta_{n-k}(A)$.

1. Set $p$ equal to the number of distinct elementary cycles in $D(A)$.
2. Set $M_0 = \{(\emptyset, 0, 0)\}$
3. For $j = 1$ to $p$ :
    (a) Set $w_j$ equal to the $j$'th cycle weight
    (b) Set $l_j$ equal to the $j$'th cycle length
    (c) Set $M_j = M_{j-1}$
    (d) For each element $(S, w, l) \in M_{j-1}$ with $l + l_j \leq n$ :
        i. If $\nexists(S', w', l + l_j) \in M_j$, then add $(S \cup \{j\}, w + w_j, l + l_j)$ to $M_j$.
        ii. If $\exists(S', w', l + l_j) \in M_j$ and $w' < w + w_j$, then remove $(S', w', l + l_j)$
            from $M_j$ and add $(S \cup \{j\}, w + w_j, l + l_j)$ to $M_j$.
4. For $k = 1$ to $n$ :
    If $\exists(S, w, k) \in M_p$, then return $\delta_{n-k}(A) = w$ and $\forall j \in S$ select $a_{st}$ for all arcs
    $(v_s, v_t)$ in cycle $j$. Else return $\delta_{n-k}(A) = -\infty$.
```

Figure 6.3: An algorithm for solving BPSM(A) for a generalised permutation matrix A.

GENPERM and repeating Step 4 of BPSMGENPERM for all $k \in N$. This would solve BPSM(A) in $O(n^2)$ time.

This works because in Step 4 of BPSMGENPERM, we can solve BPSM(A, k') for any $k' \leq k$ because of the way each of the M_j sets ($j = 1, \dots, p$) are created. The $l + l_j \leq k$ condition in Step 3(d) of BPSMGENPERM is simply to stop unnecessary calculations being performed. Changing this to $l + l_j \leq n$ means that in Step 4, we can solve BPSM(A, k') for any $k' \leq n$. So by repeating this step for all values of $k' \in N$, we can solve BPSM(A).

By making these changes to BPSMGENPERM, Step 3 now takes $O(n^2)$ time instead of $O(nk)$ time and Step 4 now takes $O(n^2)$ time instead of $O(k)$ time. All other steps are unchanged, meaning the modified algorithm runs in $O(n^2)$ time, which is an improvement on $O(n^3)$ time. The algorithm is shown in Figure 6.3.

Length of input

The BPSMGENPERM algorithm runs in $O(n^2)$ time. As the length of the input is $2n + 1 = O(n)$, this algorithm has a polynomial time complexity.

However, if the input for the algorithm BPSMGENPERM had been the lengths and weights of the p cycles with the value of k (i.e. $(l_1, \ldots, l_p, w_1, \ldots, w_p, k)$), then the input would have length $2p + 1 = O(p)$. Step 1 could then be found in constant time because the input would have length $2p + 1$. Steps 3(a) and 3(b) would be redundant and could be removed.

With this input, BPSMGENPERM would run in $O(pk)$ time. As in general, k is not a polynomial in p, we cannot say that BPSMGENPERM is polynomial in the new input length.

In fact we will now show that if it has this input then BPSMGENPERM is *NP*-complete. It can be formulated as the following problem:

Given an input $(l_1, \ldots, l_p, w_1, \ldots, w_p, k)$ with integer entries, maximise $\sum\limits_{j \in S} w_j$ over all subsets $S \subseteq \{1, \ldots, p\}$ satisfying $\sum\limits_{j \in S} l_j = k$, if such an S exists.

If we restrict this problem to have $w_1, \ldots, w_p = 0$ (i.e. the generalised permutation matrix is restricted to be a permutation matrix), then the answer is simply 0 whenever such an S exists. So we just have to work out if S exists, i.e. the following problem: Given an input $(l_1, \ldots, l_p, k)$ with integer entries, does there exist a subset $S \subseteq \{1, \ldots, p\}$ such that $\sum\limits_{j \in S} l_j = k$?

However, this is precisely the definition of the 0-1 KNAPSACK problem, which is well known to be *NP*-complete. Therefore with $(l_1, \ldots, l_p, w_1, \ldots, w_p, k)$ as our input, BPSM becomes *NP*-complete.

So for generalised permutation matrices, it is important to have length of the input an order of n, to have a polynomial algorithm that solves BPSM.

Let the generalised permutation matrix A have p cycles, $\sigma_1, \dots, \sigma_p$, where $\sigma_j = (i_{j_1}, \dots, i_{j_{l(\sigma_j)}})$. An alternative input encoding that would still have a polynomial algorithm that solves it would be the following:

$$(\sigma_1, \dots, \sigma_p, w(\sigma_1), \dots, w(\sigma_p), k).$$

It is straightforward to work out $l_1, \dots, l_p$ from this input. The length of this input is $n + p + 1$, which is of order n. So if BPSMGENPERM had an input of this form (instead of the whole matrix A and the value of k), then it would still solve BPSM(A, k) in polynomial time.

6.4.7 Block diagonal matrices

We will define a block diagonal matrix, provide a polynomial algorithm BPSM-BLOCKDIAG to solve BPSM(A, k) for some block diagonal matrix A, and comment on how this could be modified to polynomially solve BPSM(A) for some block diagonal matrix A more efficiently than repeating BPSMBLOCKDIAG for each value of k.

Definition 6.61. Given square matrices $A_1, A_2, \dots, A_p$, we define the *block diagonal matrix* of $(A_1, A_2, \dots, A_p)$ as

$$\operatorname{blockdiag}(A_1, A_2, \dots, A_p) = \begin{pmatrix} A_1 & & & -\infty \\ & A_2 & & \\ & & \ddots & \\ -\infty & & & A_p \end{pmatrix}.$$

Let $A = \operatorname{blockdiag}(A_1, A_2, \dots, A_p)$ be a block diagonal matrix. Any block of A (i.e. $A_1, A_2, \dots,$ or A_p) has a corresponding subgraph $D(A_i)$ of $D(A)$ for every $i = 1, \dots, p$. Every $D(A_i)$ is disjoint from any $D(A_j)$, $j \neq i$. So any cycle in $D(A)$

has nodes entirely within one of these disjoint subgraphs, and it is not possible to have a cycle in $D(A)$ with arcs corresponding to elements from more than one of the matrices $A_1, \ldots, A_p$.

Solving BPSM(A, k)

A generalised permutation matrix is a special case of block diagonal matrix. Similarly to the way we solved BPSM for a generalised permutation matrix, we can also solve BPSM(A, k) for block diagonal matrix $A = \text{blockdiag}(A_1, A_2, \ldots, A_p)$ (in polynomial time, as long as we can solve BPSM(A_j, k) in polynomial time, for $j = 1, \ldots, p$).

We will now show how to solve BPSM(A) for a block diagonal matrix $A = \text{blockdiag}(A_1, A_2, \ldots, A_p)$ in polynomial time, as long as we can solve BPSM(A_j) in polynomial time, for $j = 1, \ldots, p$. This is shown in an algorithm called BPSMBLOCKDIAG (see Figure 6.4). The BPSMBLOCKDIAG algorithm is a generalisation of the BPSMGENPERM algorithm from Figure 6.2.

Let $n(j)$ be the order of A_j, $j = 1, \ldots, p$. If a block diagonal matrix is a generalised permutation matrix (i.e. as in Section 6.4.6), then the digraph D_j of each block A_j contains exactly one elementary cycle, say of length l_j and of weight w_j (or equivalently, $\delta_{n(j)-l_j}(A_j) = w_j$ and for all $r = 1, 2, \ldots, l_{j-1}, l_{j+1}, \ldots, n(j)$, $\delta_{n(j)-r}(A_j) = -\infty$). In Section 6.4.6, the triple $(\{j\}, w_j, l_j)$ meant if we select all elements of A corresponding to arcs in cycle i, then we would select l_j elements and the total weight of these would be w_j.

For general block diagonal matrices, we want to select elements of A corresponding to some of the entries of block A_j (instead of the elements of A corresponding to all arcs of cycle j). There is not one total length and weight in a block - there are many.

Assume that we solve BPSM(A_j), for $j = 1, \ldots, p$. This may be done in polynomial time if A_j is one of the special types of matrix in this chapter. Else, it may be solved by a randomised polynomial algorithm, by complete enumeration, or by the branch and bound method presented later in Section 6.5.

So for $j = 1, \dots p$ and $r = 1, \dots, k$ we would be able to find $\delta_{n(j)-r}(A_j)$ and also a principal submatrix $B_{jr} \in A_j(r)$ and permutation $\pi_{jr} \in P_r$ such that $w(B_{jr}, \pi_{jr}) = \delta_{n(j)-r}(A_j)$. Finding this information is one step of the algorithm, so if we can't find it in polynomial time then the algorithm will not run in polynomial time.

Let $D_j = D(A_j)$. Compared to the previous section, for each block A_j (instead of cycle j) we have the following information. For $r = 1, \dots, k$ (instead of a single value of l_j), the permutation π_{jr} in D_j gives cycles of total length r (instead of just a single value of l_j) and total weight $\delta_{n(j)-r}(A_j)$ (instead of w_j). For $r = 1, \dots, k$, the elements to be selected to give $\delta_{n(j)-r}(A_j)$ are now found from B_{jr} and π_{jr} (instead of from cycle j).

Finally, instead of S telling us which cycle indices to select, we will use S, a set of pairs to tell us which submatrix to select and which elements from within it to select. We do this by assigning pairs (j, r) to S. A pair (j, r) tells us that by choosing B_{jr} and π_{jr} we select a total of r elements from B_{jr} and give a total sum of $\delta_{n(j)-r}(A_j)$.

There are p stages to the algorithm. At each stage information is collected and then stored within a set of triples called M_j. Each triple has the form (S, w, k), where S is as described above, w is the total weight of elements selected by using the information in S, and k is the total number of elements selected by using the information in S.

M_0 is set to $\{(\emptyset, 0, 0)\}$ at Stage 0. For $j = 1, \dots, p$, at Stage j, the information found from A_j (i.e. $\delta_{n(j)-1}(A_j), \delta_{n(j)-2}(A_j), \dots, \delta_0(A_j)$) and the information from Stage $j-1$ (i.e. M_{j-1}) is combined to produce M_j. We start by copying all triples from M_{j-1} to M_j. Next, if we can find a triple (S, w, k) (of the form described above) by combining the information found from A_j and M_{j-1} that is not in M_{j-1}, then we add (S, w, k) to M_j. Otherwise, if w is larger than the second coordinate of any triple in M_{j-1} having third component equal to k, then we replace that triple with (S, w, k) in M_j.

Using the above discussion, we now create the algorithm, called BPSMBLOCK-DIAG, that works in a similar way to BPSMGENPERM. The main changes are mainly notational, to deal with the extra information we have in a block diagonal matrix compared to a generalised permutation matrix. We will then discuss the correctness and complexity of this algorithm.

```
Algorithm BPSMBLOCKDIAG
Input: $A = \text{blockdiag}(A_1, A_2, \ldots, A_p) \in \overline{\mathbb{R}}^{n\times n}$, $k \in N$.
Output: Returns $\delta_{n-k}(A)$, and if $\delta_{n-k}(A)$ is finite, then also $k$ independent entries
of a $k \times k$ principal submatrix of $A$ whose total is $\delta_{n-k}(A)$.

  1. Set $M_0 = \{(\emptyset, 0, 0)\}$
  2. For $j = 1$ to $p$ :
     (a) For $r = 1$ to $\min(n(j), k)$ :
         i. Find $\delta_{n(j)-r}(A_j)$.
         ii. Find $B_{jr} \in A_j(r)$ and $\pi_{jr} \in P_r$ such that $w(B_{jr}, \pi_{jr}) = \delta_{n(j)-r}(A_j)$.
     (b) Set $M_j = M_{j-1}$
     (c) For each element $(S, w, l) \in M_{j-1}$ :
         For each $r = 1$ to $\min(\sum_{t=1}^{j} n(t) - l, k - l, n(j))$ with $\delta_{n(j)-r}(A_j)$ finite :
         i. If $\nexists(S', w', l + r) \in M_j$, then add $(S \cup \{(j, r)\}, w + \delta_{n(j)-r}(A_j), l + r)$
            to $M_j$.
         ii. If $\exists(S', w', l + r) \in M_j$ and $w' < w + \delta_{n(j)-r}(A_j)$, then remove
            $(S', w', l+r)$ from $M_j$ and add $(S \cup \{(j, r)\}, w + \delta_{n(j)-r}(A_j), l+r)$ to
            $M_j$.
  3. If $\exists(S, w, k) \in M_p$, then return $\delta_{n-k}(A) = w$, and for $i = 1, \ldots, r$ and all
     $(j, r) \in S$, return the element of $A$ that corresponds to the $(i, \pi_{jr}(i))$ entry of
     $B_{jr}$. Else return $\delta_{n-k}(A) = -\infty$.
```

Figure 6.4: An algorithm for solving BPSM(A, k) for a block diagonal matrix A and integer k.

Lemma 6.62.

1. If $(S, w, k) \in M_j$ in Step 3 of the algorithm, then

   (a) $S \subseteq \{1, \ldots, p\} \times \{1, \ldots, n\}$,

   (b) $\sum_{(i,s)\in S} \delta_{n(i)-s}(A_i) = w$,

   (c) $\sum_{(i,s)\in S} s = k$,

(d) If $(S', w', k) \in M_j$, then $S' = S$ and $w' = w$,

(e) If $S' \subseteq \{1, \ldots, p\} \times \{1, \ldots, n\}$, $w' = \sum\limits_{(i,s) \in S'} \delta_{n(i)-s}(A_i)$ and $\sum\limits_{(i,s) \in S'} s = k$ then $w' \leq w$.

2. If $S \subseteq \{1, \ldots, p\} \times \{1, \ldots, n\}$, and $\sum\limits_{(i,s) \in S} s = k \leq n$, then in Step 3 of the algorithm, $\exists (S', w', k) \in M_j$ where $w \leq w'$.

Proof. Statements 1(a)-1(c) are proved by induction on j, and hold automatically for $j = 0$. For $j > 0$, assume $(S, w, k) \in M_j$. We have two cases to consider:

Case 1: If $\nexists (j, r) \in S$, then $(S, w, k) \in M_{j-1}$, so 1(a)-1(c) follow by induction.

Case 2: If $\exists (j, r) \in S$, then $(S - \{(j, r)\}, w - \delta_{n(j)-r}(A_j), k - r) \in M_{j-1}$. Note that $r \leq n$, as the third coordinate of this lies between 0 and $n - r$. So again 1(a)-1(c) follow by induction.

To prove 1(d), we use the fact that each element of M_j has a unique third component due to the way Step 2(c) of the algorithm was constructed.

To prove 2, we use induction on $\max\limits_{(h,s) \in S} h$. It holds for $S = \emptyset$, and for the inductive step, assume $i = \max\limits_{(h,s) \in S} h$, and let $(i, r) \in S$. Note that $i \leq j$, so $i - 1 \leq j - 1$. Therefore $\exists (S', w - \delta_{n(j)-r}(A_j), k - r) \in M_{i-1}$ by induction. Thus in Step 2(c) of the construction of M_i, $(S' \cup \{(i, r)\}, w, k)$ was added to M_i. Then either $(S' \cup \{(i, r)\}, w, k) \in M_j$ or $\exists (S'', w'', k) \in M_j$ with $w < w''$. Either way, 2 holds.

To prove 1(e), note that by 2, $\exists (S'', w'', k) \in M_j$, with $w' \leq w''$. By 1(d), we see that $S'' = S$ and $w'' = w$, therefore $w' \leq w$, and 1(e) follows. □

Remark. From part 2 of Lemma 6.62, we see that if we have an $S \subseteq \{1, \ldots, p\} \times \{1, \ldots, k\}$ with $\sum\limits_{(i,s) \in S} s = k$ and $\sum\limits_{(i,s) \in S} \delta_{n(i)-s}(A_i) = w$, then $\exists (S^*, w^*, k) \in M_p$ (which gives at least as much total weight w^* as w does). Then from part 1(e) of Lemma 6.62, we see that if $(S^*, w^*, k) \in M_p$, then no other first coordinate satisfying

$\sum\limits_{(i,s)\in S} s = k$ will provide a bigger total weight than w^*. Selecting elements of A that correspond to the $(i, \pi_{jr}(i))$ entry of B_{jr} for all $i = 1, \ldots, r$ and all $(j, r) \in S$ and adding them up will give w^*, which is the highest possible value, so $w = \delta_{n-k}(A)$.

Theorem 6.63. If $A = \text{blockdiag}(A_1, A_2, \ldots, A_p) \in \overline{\mathbb{R}}^{n\times n}, k \in N$ and we can solve BPSM(A_i, k') in $O(t)$ time, for all $i = 1, \ldots, p$ and $k' = 1, \ldots, k$, then we can solve BPSM(A, k) in $O(n(n+t))$ time.

Proof. Correctness follows from Lemma 6.62: Step 3 of the algorithm chooses an element (S, w, k) from M_p (assuming $M_p \neq \emptyset$) with third component k. Thus it follows (from 1(a) and 1(c) of Lemma 6.62) that the solution generated from S is feasible (i.e. if we select the elements of A that corresponds to the $(i, \pi_{jr}(i))$ entries of B_{jr} for all $i = 1, \ldots, r$ and all $(j, r) \in S$, then k finite elements of A will be selected, resulting in a finite total weight). It also follows that its total weight is w (by 1(b) of Lemma 6.62), and there is no better solution (by 1(e) of Lemma 6.62). By part 2 of Lemma 6.62, it follows that if $M_p = \emptyset$ at Step 4 of the algorithm, then $\delta_{n-k}(A) = -\infty$.

For the time bound, notice that the size of each set M_{j-1} is no greater than n, because there is at most one element in M_{j-1} with the same third component (by 1(d) of Lemma 6.62). Each update operation of Step 2(c) can be done in constant time for each r and each M_{j-1}, and must be repeated for all $O(n)$ elements of M_{j-1} and $O(n(j))$ times for the r loop. Steps 1, and 2(b) require one operation each so can be performed in constant time. Assume that for each r, Step 2(a) can be performed in $O(t)$ time. The whole of Step 2 is carried out for $j = 1, \ldots, p$. It is easily seen that Step 3 can be done in $O(k)$ time. So algorithm BPSMBLOCKDIAG runs in time $\sum_{j=1}^{p} O(n(j))t + \sum_{j=1}^{p} O(n)O(n(j)) + O(k) = O(n(n+t))$. $\square$

Corollary 6.64. If in Theorem 6.63, t is polynomial in n, that is, if we can solve BPSM(A_i, k') in polynomial time, for all $i = 1, \ldots, p$ and $k' = 1, \ldots, k$, then for a block diagonal matrix A, we can solve BPSM(A, k) in polynomial time.

Solving BPSM(A)

For a generalised permutation matrix A, the algorithm BPSMBLOCKDIAG solves BPSM(A, k) for one particular value of k only. If we want to solve BPSM(A, k) for all values of k (i.e. solve BPSM(A)), then we could repeat the algorithm n times, changing the value of k each time. This means we could solve BPSM(A) in $O(n^2(n+t))$ time.

However, we can modify BPSMBLOCKDIAG slightly so it can solve BPSM(A) more quickly than just repeating for all k: Create a modified algorithm of BPSM-BLOCKDIAG called BPSMBLOCKDIAG2 by replacing k by n in Step 2(a) and 2(c) of BPSMBLOCKDIAG and repeating Step 3 of BPSMBLOCKDIAG for all $k \in N$. This would solve BPSM(A) in $O(n(n+t))$ time.

This works because in Step 3 of BPSMBLOCKDIAG, we can solve BPSM(A, k') for any $k' \leq k$ because of the way each of the M_j sets ($j = 1, \ldots, p$) are created. The "$r = 1$ to $\min(n(j), k)$" condition in Step 2(a) and the "$r = 1$ to $\min(\sum_{t=1}^{j} n(t) - l, k - l, n(j))$" condition in Step 2(c) of BPSMBLOCKDIAG are mainly to stop unnecessary calculations being performed. Changing these to "$r = 1$ to $n(j)$" and "$r = 1$ to $\min(\sum_{t=1}^{j} n(t) - l, n(j))$" respectively means that in Step 3, we can solve BPSM(A, k') for any $k' \leq n$. So by repeating this step for all values of $k' \in N$, we can solve BPSM(A).

By making these changes to BPSMBLOCKDIAG, the complexity of the steps remains unchanged, except for Step 3 which now takes $O(n^2)$ time instead of $O(k)$ time. All other steps are unchanged, meaning the modified algorithm runs in time $\sum_{j=1}^{p} O(n(j))t + \sum_{j=1}^{p} O(n)O(n(j)) + O(n^2) = O(n(n+t))$, which is the same as the time complexity for solving BPSM for a single fixed k. The algorithm is shown in Figure 6.5.

Algorithm BPSMBLOCKDIAG2
Input: $A = \text{blockdiag}(A_1, A_2, \ldots, A_p) \in \overline{\mathbb{R}}^{n\times n}$.
Output: For $k = 1, \ldots, n$, returns $\delta_{n-k}(A)$, and if $\delta_{n-k}(A)$ is finite, then also k independent entries of a $k \times k$ principal submatrix of A whose total is $\delta_{n-k}(A)$.

```
1. Set $M_0 = \{(\emptyset, 0, 0)\}$
2. For $j = 1$ to $p$ :
    (a) For $r = 1$ to $n(j)$ :
        i. Find $\delta_{n(j)-r}(A_j)$.
        ii. Find $B_{jr} \in A_j(r)$ and $\pi_{jr} \in P_r$ such that $w(B_{jr}, \pi_{jr}) = \delta_{n(j)-r}(A_j)$.
    (b) Set $M_j = M_{j-1}$
    (c) For each element $(S, w, l) \in M_{j-1}$ :
        For each $r = 1$ to $\min(\sum_{t=1}^{j} n(t) - l, n(j))$ with $\delta_{n(j)-r}(A_j)$ finite :
        i. If $\nexists(S', w', l + r) \in M_j$, then add $(S \cup \{(j, r)\}, w + \delta_{n(j)-r}(A_j), l + r)$
           to $M_j$.
        ii. If $\exists(S', w', l + r) \in M_j$ and $w' < w + \delta_{n(j)-r}(A_j)$, then remove
            $(S', w', l+r)$ from $M_j$ and add $(S \cup \{(j, r)\}, w + \delta_{n(j)-r}(A_j), l + r)$ to
            $M_j$.
3. For $k = 1$ to $n$ :
    If $\exists(S, w, k) \in M_p$, then return $\delta_{n-k}(A) = w$, and for $i = 1, \ldots, r$ and all
    $(j, r) \in S$, return the element of $A$ that corresponds to the $(i, \pi_{jr}(i))$ entry of
    $B_{jr}$. Else return $\delta_{n-k}(A) = -\infty$.
```

Figure 6.5: An algorithm for solving BPSM(A) for a block diagonal matrix A.

6.4.8 Reducible Matrices

We will state that the digraph of any reducible matrix A consists of strongly connected components. Using this fact we can convert A to a block diagonal matrix with each block being irreducible. If we can solve BPSM for each block in polynomial time, we can show that we can use the algorithm BPSMBLOCKDIAG from Section 6.4.7 to solve BPSM(A, k) in polynomial time.

Remark. Let $A \in \overline{\mathbb{R}}^{n\times n}$. If each entry of A is $-\infty$, then obviously $\delta_{n-k}(A) = -\infty$, for all $k \in N$. So now assume at least one element of A is finite. If any row or column i of A contains only $-\infty$'s, then removing both row and column i, will form a smaller matrix $A' \in \overline{\mathbb{R}}^{n-1\times n-1}$. Applying this process iteratively will form a matrix $A'' \in \overline{\mathbb{R}}^{r\times r}$ that has no rows or columns consisting entirely of $-\infty$'s.

For $k > r$, there does not exist k independent elements of A'', therefore $\delta_{n-k}(A) = -\infty$. For $k \leq r$, there exists an optimal assignment over all $k \times k$ submatrices of A which does not include any elements in the rows or columns of A that we deleted to form A''. Hence, for $k \leq r$, we have $\delta_{n-k}(A) = \delta_{r-k}(A'')$.

So to summarise, we can determine whether $\delta_{n-k}(A) = \delta_{r-k}(A'')$ (which may or may not be finite) for some matrix A'' (which we can easily find) with no row or column consisting entirely of $-\infty$'s and $k \leq r$, or else that $\delta_{n-k}(A) = -\infty$. Hence from now on we will assume that all matrices have no rows or columns containing only $-\infty$'s.

Definition 6.65. A matrix $B \in \overline{\mathbb{R}}^{n\times n}$ is in Frobenius normal form, if

$$B = \begin{pmatrix} B_{11} & B_{12} & \dots & B_{1p} \\ & B_{22} & \dots & B_{2p} \\ & & \ddots & \vdots \\ -\infty & & & B_{pp} \end{pmatrix},$$

where all diagonal blocks B_{ii} are irreducible.

Theorem 6.66 ([39])**.** Every matrix $A \in \overline{\mathbb{R}}^{n\times n}$ can be transformed in linear time to a similar matrix $B \in \overline{\mathbb{R}}^{n\times n}$ in Frobenius normal form.

Because $A \sim B$, we have that $(\forall k \in N)\ \delta_{n-k}(B) = \delta_{n-k}(A)$, by Corollary 6.10.

Remark. For any matrix $A \in \overline{\mathbb{R}}^{n\times n}$, any arc in $D_C(A)$ that does not lie on a finite cycle may have its weight (or the corresponding element in A) set to $-\infty$ without affecting δ_{n-k} for any $k \in N$.

This means we may set all elements of off-diagonal blocks in B to $-\infty$, as arcs in $D_C(A)$ corresponding to elements in off-diagonal blocks do not belong to any finite cycle. Therefore if we define $C_i = B_{ii}$, for $i = 1, \ldots, p$, we have

$$C = \begin{pmatrix} C_1 & & & -\infty \\ & C_2 & & \\ & & \ddots & \\ -\infty & & & C_p \end{pmatrix},$$

This is a block diagonal matrix, i.e.

$$C = \text{blockdiag}(C_1, C_2, \ldots, C_p),$$

and we have $\delta_{n-k}(C) = \delta_{n-k}(A)$ for all $k \in N$. Therefore we have shown the following:

Theorem 6.67. For $A \in \overline{\mathbb{R}}^{n\times n}$ there exists $C = \text{blockdiag}(C_1, C_2, \ldots, C_p) \in \overline{\mathbb{R}}^{n\times n}$ such that each diagonal block C_i, $(i = 1, \ldots, p)$ is irreducible, and $\delta_{n-k}(C) = \delta_{n-k}(A)$ for all $k \in N$.

Corollary 6.68. For any $A \in \overline{\mathbb{R}}^{n\times n}, k \in N$, if we can solve BPSM in polynomial time for each matrix associated with a strongly connected component of $D(A)$, then we can solve BPSM(A, k) in polynomial time (by converting A to a block diagonal matrices and using the BPSMBLOCKDIAG algorithm of Figure 6.4).

6.5 A branch and bound method to solve BPSM

If A and k do not form one of the special cases in Section 6.4 for solving BPSM(A, k), then in general we do not know how to solve BPSM(A, k) in polynomial time. We could use a branch and bound method (presented later in this section) to solve

BPSM(A, k). This may not run in polynomial time, but will at least provide an alternative to complete enumeration. If the algorithm is terminated before completion, we can also obtain upper and lower bounds to the optimal solution to BPSM(A, k) from the most recent iteration of the algorithm.

6.5.1 BPSM as an integer program

Linear programming and integer linear programming have been well documented, see for example [27]. The *assignment problem* for a matrix $A = (a_{ij}) \in \overline{\mathbb{R}}^{n \times n}$ can be described as an integer linear program (ILP) as follows:

$$\max \quad \sum_{i \in N} \sum_{j \in N} a_{ij} x_{ij} \tag{6.5.1a}$$

$$\text{s.t.} \quad \sum_{i \in N} x_{ij} = 1 \qquad \text{for all } j \in N \tag{6.5.1b}$$

$$\sum_{j \in N} x_{ij} = 1 \qquad \text{for all } i \in N \tag{6.5.1c}$$

$$x_{ij} \in \{0, 1\} \qquad \text{for all } i, j \in N \tag{6.5.1d}$$

The n variables x_{ij} that equal 1 correspond to n elements of the matrix A that belong to an optimal solution to the assignment problem.

We can adapt (6.5.1) so that BPSM(A, k) can be formulated as an ILP by adding

n binary variables $y_i, i \in N$ as follows:

$$\begin{aligned}
\max \quad & \sum_{i\in N}\sum_{j\in N} a_{ij}x_{ij} && && (6.5.2a)\\
\text{s.t.} \quad & y_j + \sum_{i\in N} x_{ij} = 1 && \text{for all } j \in N && (6.5.2b)\\
& y_i + \sum_{j\in N} x_{ij} = 1 && \text{for all } i \in N && (6.5.2c)\\
& \sum_{i\in N} y_i = n - k && && (6.5.2d)\\
& x_{ij} \in \{0,1\} && \text{for all } i,j \in N && (6.5.2e)\\
& y_i \in \{0,1\} && \text{for all } i \in N && (6.5.2f)
\end{aligned}$$

This ILP ensures that exactly k variables x_{ij} equal 1. It also ensures that there is at most one element selected from each row and each column. Also, the inclusion of y_i ensures that an element from a row i is chosen if and only if an element from the column i is chosen, because

$$x_{ij} = 1 \implies (\forall l \neq i)\ x_{lj} = 0$$

$$x_{ij} = 1 \implies (\forall l \neq j)\ x_{il} = 0$$

$$x_{ij} = 1 \implies (\exists l)\ x_{jl} = 1$$

hold for all $i, j \in N$. This ensures principality. If $x_{ij} = 1$, then $a_{ij} = w(i,j)$ is the weight of an arc on a cycle in $D(A)$. Together these arcs will complete cycles in $D(A)$ that are disjoint from each other. All corresponding elements a_{ij} will thus be contained within a principal submatrix of A size k, which has a set of row and column indices equal to $\{i \in N : x_{ij} = 1\}$.

We can relax the BPSM(A,k) ILP (6.5.2) by removing the integrality constraints in (6.5.2e) and (6.5.2f) that x_{ij} and y_i be 0 or 1, and replace them with closed interval constraints given by (6.5.3e) and (6.5.3f) in the following LP:

$$\max \quad \sum_{i \in N} \sum_{j \in N} a_{ij} x_{ij} \tag{6.5.3a}$$

$$\text{s.t.} \quad y_j + \sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \tag{6.5.3b}$$

$$y_i + \sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \tag{6.5.3c}$$

$$\sum_{i \in N} y_i = n - k \tag{6.5.3d}$$

$$x_{ij} \in [0, 1] \quad \text{for all } i, j \in N \tag{6.5.3e}$$

$$y_i \in [0, 1] \quad \text{for all } i \in N \tag{6.5.3f}$$

In fact we can relax the constraints on the variables even further:

$$\max \quad \sum_{i \in N} \sum_{j \in N} a_{ij} x_{ij} \tag{6.5.4a}$$

$$\text{s.t.} \quad y_j + \sum_{i \in N} x_{ij} = 1 \quad \text{for all } j \in N \tag{6.5.4b}$$

$$y_i + \sum_{j \in N} x_{ij} = 1 \quad \text{for all } i \in N \tag{6.5.4c}$$

$$\sum_{i \in N} y_i = n - k \tag{6.5.4d}$$

$$x_{ij} \geq 0 \quad \text{for all } i, j \in N \tag{6.5.4e}$$

$$y_i \geq 0 \quad \text{for all } i \in N \tag{6.5.4f}$$

As x_{ij} and y_i are non negative, if any of the variables are greater than 1, then constraints (6.5.3b) and (6.5.3c) will not be met, so this does not extend the range of the variables, merely remove the redundant constraints $x_{ij} \leq 1$ and $y_i \leq 1$, for all $i, j \in N$.

This relaxation (6.5.4) is now not an ILP, but a linear program (LP), and can be solved, for example by the simplex method. (The simplex method is technically not polynomial in complexity, but is usually fairly quick to run in practice. Other methods exist that do polynomially solve LPs.)

If by doing this, (6.5.4) results in an optimal solution with integer variables, then this is also an optimal solution to the original integer problem (6.5.2), i.e. solves BPSM for those choices of A and k. However for alternative choices of A and k, the optimal solution to (6.5.4) may not provide integer variables. Sometimes the solution has variable(s) with fractional value(s). This solution would not be feasible in (6.5.2). However it does provide us with a useful upper bound (in the case of maximisation) to the optimal objective function value (6.5.2a), and a solution for which (6.5.2) may be "close" to optimal in some way in.

If an integral solution exists that maximises the objective function (6.5.4a), will the simplex method applied to (6.5.4) return an integral solution? Consider the following example: Let

$$A = (a_{ij}) = \begin{pmatrix} 6 & 9 & 9 & 3 & 9 & 5 \\ 9 & 0 & 1 & 1 & 1 & 5 \\ 9 & 1 & 8 & 2 & 0 & 9 \\ 3 & 1 & 2 & 9 & 8 & 7 \\ 9 & 1 & 0 & 8 & 7 & 9 \\ 5 & 5 & 9 & 7 & 9 & 7 \end{pmatrix}, k = 4.$$

The simplex method can be used to solve (6.5.4). The built in simplex method of Maple 8 was applied to this matrix. The optimal solution obtained was (with y_i variables omitted):

$$x_{4,4} = 1$$
$$x_{5,1} = 1$$
$$x_{3,6} = 1/2$$
$$x_{1,3} = 1/2$$
$$x_{6,5} = 1/2$$
$$x_{1,5} = 1/2$$
$$\text{Other } x_{ij} = 0$$

This is indeed a feasible solution to (6.5.4). The sum of x_{ij}'s is 4, and row and column sums are between 0 and 1. However, because some variables are not integer, this is not an optimal solution to (6.5.2). In this case the objective function value is $9 + 9 + 1/2 * 9 + 1/2 * 9 + 1/2 * 9 + 1/2 * 9 = 36$. Thus this is an upper bound for what any integer optimal solution to (6.5.2) could give us. Is it possible to match this bound, with integer variables? The answer to this is yes. For example, consider the following solution which has integer variables:

$$x_{1,2} = 1$$
$$x_{2,1} = 1$$
$$x_{5,6} = 1$$
$$x_{6,5} = 1$$
$$\text{Other } x_{ij} = 0$$

This is feasible, and has objective function value of $9 + 9 + 9 + 9 = 36$, and therefore is also optimal. So even though one simplex iteration did not find an integral solution, there may still be one.

I believe Maple's Simplex pivot rule is partially randomised, so repeating the above process may generate slightly different results, (in particular, may yield an integer solution).

6.5.2 BPSM$_{Frac}$

We have shown in the Section 6.5.1 how to formulate BPSM as an integer program, and how to adapt this to a linear program, that will solve BPSM if the x and y variables happen to be integer. We now define a variant of the BPSM problem that will allow fractional values:

Problem 21. For $A \in \overline{\mathbb{R}}^{n\times n}$ and real $k \in [0, n]$, solve LP 6.5.4.

We will refer to this problem as BPSM$_{Frac}(A, k)$. Note that we have allowed k to be a real number, as there isn't any real need to restrict it to be an integer here.

Example 6.6.

$$A = (a_{ij}) = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 1 & 3 \\ 0 & 1 & 0 \end{pmatrix}, k = 2.$$

For this example, it is easily seen that the optimal solution for BPSM$_{Frac}(A, k)$ is

$$x_{1,1} = 1$$
$$x_{2,3} = 1/2$$
$$x_{3,2} = 1/2$$
$$\text{Objective function value} = 7$$

The optimal solution for BSM(A, k) is

$$x_{1,1} = 1$$
$$x_{2,3} = 1$$
$$\text{Objective function value} = 8 > 7$$

The optimal solution for BPSM(A, k) is

$$x_{1,1} = 1$$
$$x_{2,2} = 1$$
$$\text{Objective function value} = 6 < 7 < 8$$

It is clear that the optimal solution value for BPSM$_{Frac}(A, k)$ is at least the optimal solution value for BPSM(A, k), for all $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$. And when the solution is integer for BPSM$_{Frac}(A, k)$, the same solution will solve BPSM(A, k), thus BPSM(A, k) and BPSM$_{Frac}(A, k)$ would have the same optimal solution value.

It is also true that the optimal solution value for BSM(A, k) is at least the optimal solution value for BPSM$_{Frac}(A, k)$, for all $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$, as the solution set for BPSM$_{Frac}(A, k)$ is a subset of the solution set for BSM(A, k). (Note here that it can be shown that there will always be an optimal integer solution for BSM however, due to total unimodularity.) Also, when the solution for BSM(A, k) is principal, the same solution will solve BPSM(A, k) and BPSM$_{Frac}(A, k)$, thus BPSM(A, k), BPSM$_{Frac}(A, k)$ and BSM(A, k), problems would have the same optimal solution value.

This means it is better to use BPSM$_{Frac}(A, k)$ rather than BSM(A, k) to find an upper bound to BPSM(A, k), as the optimal solution value of BSM(A, k) will be no closer to δ_{n-k} than the optimal solution value of BPSM$_{Frac}(A, k)$. Example 6.6 illustrates the fact that it is possible that BPSM$_{Frac}(A, k)$ can give a better upper bound to BPSM(A, k) than BSM(A, k) can.

BPSM$_{Frac}$ is polynomially solvable, as linear programs (and in particular, LP 6.5.4) can be solved in polynomial time.

Now we give an example of a possible application of BPSM$_{Frac}$.

Example 6.7. At a beer festival, the organiser wants to let some of the n contributors of the various beers to sample each others produce for free before the general public arrive. Based on previous years experience, he knows how likely they are to like each of the types of beer. He wants to fix the total amount sampled by the contributors to be T litres so that there is plenty to be bought by other people later on. He doesn't want any one contributor drinking more than D litres. Also, he feels it only fair, that if the total amount sampled by a contributor is to be x litres, then the total amount to be sampled of that particular contributor's beer should be x litres. The amount sampled by each contributor need not be in whole litres.

This can be solved using $\text{BPSM}_{Frac}(A,k)$ as follows. Let $A = (a_{ij}) \in \overline{\mathbb{R}}^{n\times n}$ be a matrix where a_{ij} is a rating of how much contributor i will enjoy contributor j's beer. He can set $a_{ij} = -\infty$ if he does not want contributor i to sample contributor j's beer. Let $k = T/D$. We can then be solve $\text{BPSM}_{Frac}(A,k)$ using the simplex method (or an alternative method) on the LP 6.5.4. The quantity Dx_{ij} will correspond to how many litres of beer contributor i should drink from contributor j.

This example can be extended to other examples with continuous items that can be split into any fraction, like liquids and gases can or fine solids such as flour or sugar for example.

6.5.3 Branch and bound: worked example

Consider the following 10×10 matrix, which has entries randomly generated between 0 and 99:

$$A = \begin{pmatrix} 21 & 21 & 98 & 72 & 88 & 13 & 53 & 24 & 97 & 65 \\ 0 & 15 & 18 & 64 & 5 & 98 & 11 & 73 & 55 & 40 \\ 7 & 11 & 10 & 23 & 59 & 90 & 69 & 22 & 30 & 45 \\ 84 & 57 & 78 & 33 & 35 & 70 & 67 & 41 & 27 & 43 \\ 88 & 76 & 73 & 53 & 32 & 81 & 32 & 15 & 84 & 3 \\ 54 & 4 & 13 & 77 & 13 & 45 & 55 & 92 & 13 & 49 \\ 57 & 51 & 83 & 30 & 49 & 89 & 30 & 67 & 23 & 31 \\ 13 & 15 & 94 & 39 & 3 & 17 & 76 & 1 & 88 & 53 \\ 93 & 23 & 44 & 54 & 92 & 82 & 71 & 4 & 89 & 98 \\ 57 & 45 & 94 & 74 & 34 & 53 & 85 & 69 & 59 & 12 \end{pmatrix}$$

We now try to solve BPSM(A, k), for $k = 4$, using a branch and bound technique. We summarise this example at the end (see Figure 6.6 on page 140).

Solving the problem (P) - BPSM$_{Frac}(A, k)$, the relaxation of BPSM(A, k), (i.e. LP 6.5.4 applied to matrix A with $k = 4$), using simplex, we obtain the following solution. (For simplicity, we omit the y_i variables and any x_{ij} variables that are equal to zero.)

$$x_{1,9} = 1$$
$$x_{9,1} = 1$$
$$x_{8,3} = 2/3$$
$$x_{3,6} = 2/3$$
$$x_{6,8} = 2/3$$

Objective function value = 374.

Unfortunately we do not have an integer solution, and so this solution is not feasible for the BPSM ILP (6.5.2). However, we do now have an upper bound of 374 to its objective function value (6.5.2a). Adding together the four highest diagonal elements of A would give a lower estimate, that can be calculated in $O(n \log n)$. In this case the lower estimate is $33 + 32 + 45 + 89 = 199$. So the optimal objective function value lies in $[199, 374]$.

We now *branch* our problem by setting one of the non-integer variables to zero (for problem (P0)) and to one (for problem (P1)) to create two new problems. It is not clear which variable we should choose (from $x_{8,3}, x_{3,6}$ and $x_{6,8}$) in order to terminate the algorithm fastest, so we randomly choose $x_{3,6}$.

$$(P0) = (P) \cup \{x_{3,6} = 0\}$$
$$(P1) = (P) \cup \{x_{3,6} = 1\}$$

We call (P0) and (P1) the *child* problems of *parent* (P). Thus the objective function values of both child problems will be no bigger than the parent.

For (P0), we solve using the simplex method, and obtain this solution:

$$x_{7,6} = 2/5$$
$$x_{5,1} = 2/5$$
$$x_{9,10} = 2/5$$
$$x_{1,9} = 3/5$$
$$x_{8,9} = 2/5$$
$$x_{6,8} = 2/5$$
$$x_{1,5} = 2/5$$
$$x_{10,7} = 2/5$$
$$x_{9,1} = 3/5$$

Objective function value $= 365.2$.

For (P1), we solve using the simplex method, and obtain this solution:

$$x_{3,6} = 1$$
$$x_{1,9} = 1/2$$
$$x_{9,1} = 1/2$$
$$x_{8,3} = 1$$
$$x_{6,8} = 1$$
Objective function value = 371.

Problem (P) is of no further use, so we remove it from consideration.

We remove any child problems from consideration if their objective function values are less the lower bound, as the objective function values of future children cannot rise back above the lower bound. Remaining child problems will have objective function values that are between the lower bound and upper bound.

If one child problem has an integer optimal solution, then its objective function value now becomes the new lower bound, and we can remove this problem from consideration. If both child problems have integer optimal solutions, then the greatest of the two child problems' objective function values now becomes the new lower bound, and we can remove both problems from consideration.

Otherwise, we keep remaining child problems for later consideration.

If there are no more problems to consider, the lower bound we have will be the optimal objective function value, so we end the process.

Otherwise, we then take a new parent problem with the biggest objective function value, from all the problems we are considering, and set our new upper bound to be its objective function value. Remaining problems will have objective function values between the lower and upper bounds (inclusive). We then branch this parent problem on one of its non-integer variables, and remove it from consideration.

In this example, both (P0) and (P1) have non-integer optimal solutions, so the lower bound remains the same. The objective function values of 365.2 and 371 are both above the lower bound of 199, so both (P0) and (P1) remain for further consideration.

We set the new upper bound to $\max(365.2, 371) = 371$, given by (P1). The optimal objective function value now lies in $[199, 371]$. We remove (P1) from consideration after branching on, say $x_{9,1}$, to form (P10) and (P11).

$$\begin{aligned}(P10) &= (P1) \cup \{x_{9,1} = 0\} \\ (P11) &= (P1) \cup \{x_{9,1} = 1\}\end{aligned}$$

For (P10), we solve using the simplex method, and obtain this solution:

$$\begin{gathered}x_{3,6} = 1 \\ x_{10,3} = 1/2 \\ x_{8,3} = 1/2 \\ x_{9,10} = 1/2 \\ x_{8,9} = 1/2 \\ x_{6,8} = 1 \\ \text{Objective function value} = 369.\end{gathered}$$

For (P11), we solve using the simplex method, and obtain this solution:

$$\begin{gathered}x_{3,6} = 1 \\ x_{9,1} = 1 \\ x_{1,3} = 1 \\ x_{6,9} = 1 \\ \text{Objective function value} = 281.\end{gathered}$$

Note that (P11) has an integer optimal solution. Its objective function value of 281 is greater than the current lower bound of 199, so the new lower bound is set to 281, and we remove (P11) from consideration.

We now consider (P0) and (P10).

Both problems currently in consideration have objective function values no less than the lower bound, so they both remain in consideration.

We have that (P0) and (P10) have non-integer solutions, so there may still be a better integer solution that can be found from branching one of these. (Even if there is no better integer solution, branching from these would allow us to be sure.)

The greatest objective function value is given by (P10). The new upper bound is thus set to 369. The optimal objective function value now lies in $[281, 369]$. We now branch (P10) on $x_{8,9}$, and remove (P10) from consideration.

$$(P100) = (P10) \cup \{x_{8,9} = 0\}$$
$$(P101) = (P10) \cup \{x_{8,9} = 1\}$$

For (P100), we solve using the simplex method, and obtain this solution:

$$\begin{gathered} x_{3,6} = 1 \\ x_{1,9} = 1/3 \\ x_{5,1} = 1/3 \\ x_{9,5} = 1/3 \\ x_{8,3} = 1 \\ x_{6,8} = 1 \\ \text{Objective function value} = 368.\dot{3}. \end{gathered}$$

For (P101), we solve using the simplex method, and obtain this solution:

$$x_{3,6} = 1$$
$$x_{8,9} = 1$$
$$x_{6,8} = 1$$
$$x_{9,3} = 1$$
$$\text{Objective function value} = 314$$

We have that (P101) has an integer optimal solution with objective function value of 314. Thus the lower bound becomes 314, and we remove (P101) from consideration.

We now consider (P0) and (P100).

The upper bound decreases to $368.\dot{3}$ given by (P100), as it has a bigger objective function value than (P0). The optimal objective function value now lies in $[314, 368.\dot{3}]$.

We now branch (P100) on $x_{1,9}$, and remove (P100) from consideration.

$$(P1000) = (P100) \cup \{x_{1,9} = 0\}$$
$$(P1001) = (P100) \cup \{x_{1,9} = 1\}$$

For (P1000), we solve using the simplex method, and obtain this solution:

$$x_{3,6} = 1$$
$$x_{9,9} = 1$$
$$x_{8,3} = 1$$
$$x_{6,8} = 1$$
$$\text{Objective function value} = 365$$

For (P1001), we solve using the simplex method, and obtain this solution:

$$x_{3,6} = 1$$
$$x_{1,9} = 1$$
$$x_{6,1} = 1$$
$$x_{9,3} = 1$$
$$\text{Objective function value} = 285$$

Both these child problems have integer optimal solutions, with the greatest objective function value of 365 coming from (P1000). Hence the lower bound becomes 365, and we remove both (P1000) and (P1001) from consideration.

We only have (P0) to consider now.

The new upper bound becomes 365.2 from (P0), and so the optimal objective function value lies in $[365, 365.2]$.

We now branch (P0) on $x_{5,1}$, and remove (P0) from consideration.

$$(P00) = (P0) \cup \{x_{5,1} = 0\}$$
$$(P01) = (P0) \cup \{x_{5,1} = 1\}$$

For (P00), we solve using the simplex method, and obtain this solution:

$$x_{6,8} = 2/3$$
$$x_{8,9} = 2/3$$
$$x_{1,9} = 1/3$$
$$x_{9,1} = 1/3$$
$$x_{9,10} = 2/3$$

$$x_{10,7} = 2/3$$
$$x_{7,6} = 2/3$$
$$\text{Objective function value} = 364.\dot{6}$$

For (P01), we solve using the simplex method, and obtain this solution:

$$x_{5,1} = 1$$
$$x_{9,5} = 3/4$$
$$x_{1,5} = 1/4$$
$$x_{1,9} = 3/4$$
$$x_{8,9} = 1/4$$
$$x_{7,6} = 1/4$$
$$x_{10,7} = 1/4$$
$$x_{6,8} = 1/4$$
$$x_{9,10} = 1/4$$
$$\text{Objective function value} = 364.75$$

Both (P00) and (P01) have objective function values less than the lower bound and are removed from consideration.

We have no more problems to consider, so (P1000) resulted in an optimal (integer) solution with an objective function value of 365.

Therefore $a_{3,6}, a_{9,9}, a_{8,3}$ and $a_{6,8}$ should be selected from matrix A, adding to a total of $90 + 89 + 94 + 92 = 365$. This means BPSM($A, 4$)=365.

See Figure 6.6 for a summary of this example.

Remark. In this particular example, all entries were integer values. This means that the optimal objective function value must also be an integer. At one stage we knew that the optimal objective function value lay in the range $[365, 365.2]$. As only one integer value lies in that range, namely 365, we could have concluded at that point, with no further calculations, that BPSM$(A, 4)$=365.

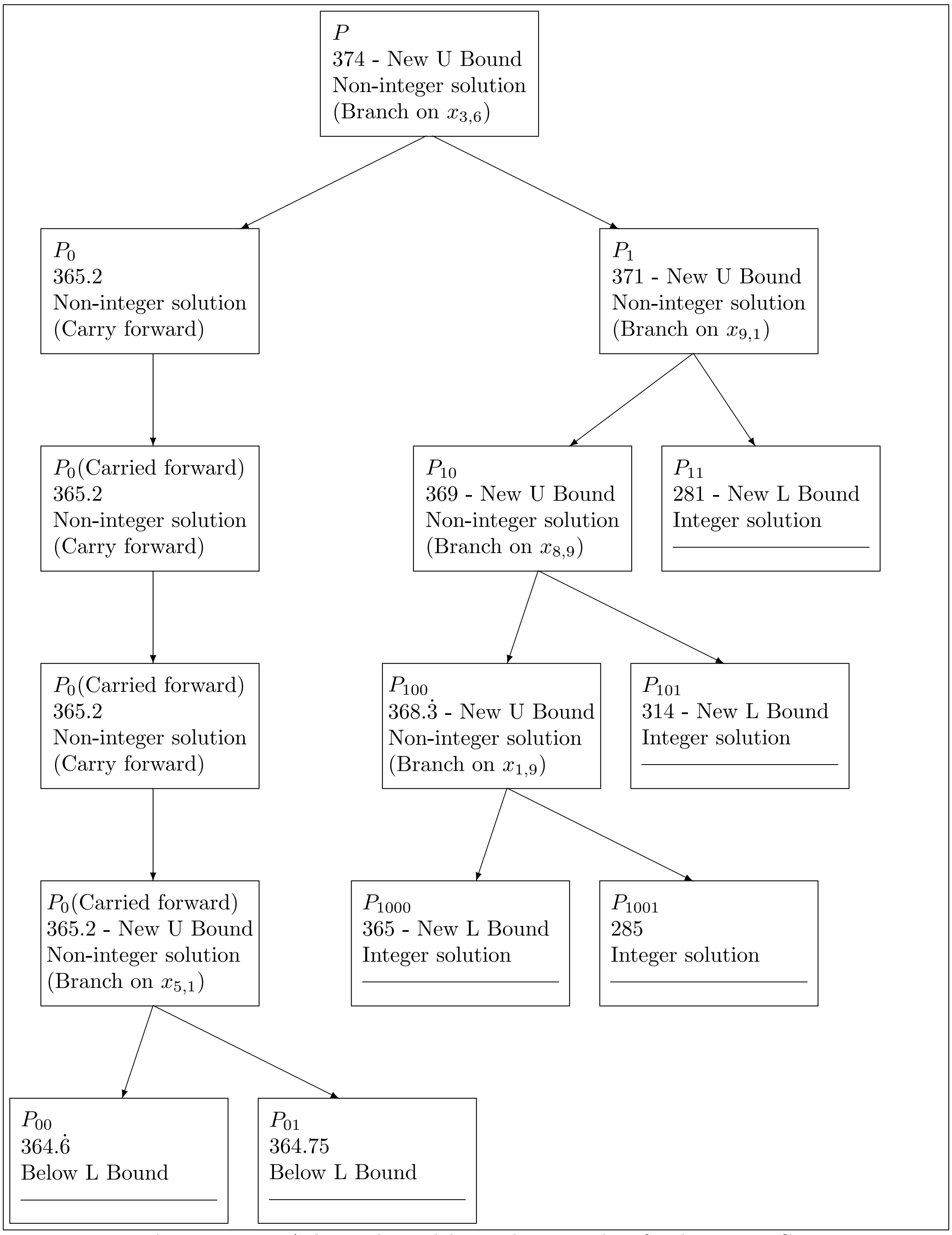

Figure 6.6: A branch and bound example of solving BPSM

6.5.4 Formalisation of the branch and bound method

We now formalise the branch and bound method for solving BPSM based on the example given in the last section. We give an algorithm BPSMBAB (see Figure 6.7) for solving BPSM by the branch and bound method.

```
Algorithm BPSMBAB
Input: $A \in \overline{\mathbb{R}}^{n\times n}$ and $k \in N$.
Output: Returns the optimal value $\delta_{n-k}(A)$ for BPSM$(A,k)$, and selects $k$ optimal
elements of $A$.

  1. Set currentbest := the set of $k$ biggest diagonal elements of $A$.
  2. Set $L$ := sum of elements in currentbest.
  3. Set $P$ to be the LP of (6.5.4) for $A$ and $k$.
  4. Set activeset := $\{P\}$.
  5. Generate $U$, the optimal objective function value of $P$.
  6. If activeset is not empty :
     (a) Remove from activeset: a $P_s \in$ activeset which has the greatest $U_s$ value.
     (b) If $U_s < L$ then goto 7.
     (c) Set $U := U_s$.
     (d) Generate from problem $P_s$: the children $P_{s0}$ and $P_{s1}$, and optimal objec-
         tive function values $U_{s0}$ and $U_{s1}$.
     (e) For each $i = 0, 1$ with $L \le U_{si}$ :
             If $P_{si}$ has an integer optimal solution then :
                 Set $L := U_{si}$.
                 Generate currentbest from $P_{si}$.
             Else :
                 Add $P_{si}$ to activeset.
  7. Return $\delta_{n-k}(A) = L$, and select elements of $A$ in currentbest.
```

Figure 6.7: A branch and bound algorithm for solving BPSM

We now comment on each of the steps of the algorithm:

Step 1 *currentbest* is a set of k independent elements that lie in a $k \times k$ principal submatrix of A (and add to L). *currentbest* is initially set to be the set of k biggest diagonal elements of A. We could initially set *currentbest* to a different set of k elements of A (especially if we knew they had a bigger sum). When *currentbest* is updated, it will be obtained from an optimal integer solution to an LP (i.e. where all x_{ij} are 0 or 1).

Step 2 L is the sum of elements in *currentbest*, which is the current lower bound for $\delta_{n-k}(A)$. It is initially set to be the sum of the k biggest diagonal elements (which may be $-\infty$).

Step 3 The initial problem P is $\text{BPSM}_{Frac}(A, k)$, the relaxation of $\text{BPSM}(A, k)$, i.e. (6.5.4) applied to the matrix $A \in \overline{\mathbb{R}}^{n\times n}$ and constant k.

Step 4 *activeset* is a set of LP problems for which we may obtain a higher lower bound than L. It is initially set to $\{P\}$.

Steps 5 and 6(d) optimal objective function values U_s, of P_s, can be generated by solving P_s with the simplex method.

Steps 6(a) and 6(b) By removing the P_s from *activeset* that has the greatest U_s value, any other problem P_r in *activeset* will have $U_r \leq U_s$. So if $U_s \leq L$, then $U_r \leq L$. In other words if it is not possible for P_s to produce a higher lower bound L for $\delta_{n-k}(A)$ then it is not possible for any of the remaining problems in *activeset* to produce a higher lower bound L for $\delta_{n-k}(A)$. This is why in step 6(b), if this happens we go to the last step of the algorithm.

Step 6(c) The maximum value $\delta_{n-k}(A)$ could be is U_s, as P_s was chosen from the remaining problems to be the one with the highest U_s value. Therefore we set U, the current upper bound for $\delta_{n-k}(A)$, to U_s. U will always be obtained from a non-integer optimal solution to an LP (i.e. where not all x_{ij} are 0 or 1).

Step 6(d) After solving P_s (by the simplex method), we generate the children of P_s by selecting a non integer variable x_{ij} from the optimal solution, and adding the constraints $\{x_{ij} = 0\}$ to P_s to form P_{s0}, and $\{x_{ij} = 1\}$ to P_s to form P_{s1}.

If there is an optimal integer solution satisfying P_s with a bigger lower bound than the current value of L that, then there will be one satisfying P_{s0} or P_{s1}.

Note that the "s" in P_s is a string of 0's and 1's. So the "si" from P_{si} is the same string of 0's and 1's, with an added digit (equal to i) at the end. We also allow s to be the empty string, for the initial LP P.

Step 6(e) Assuming it is possible to obtain from P_{si} a higher lower bound for $\delta_{n-k}(A)$ than the current value of L (i.e. $L \leq U_{si}$), we check if P_{si} has an integer solution. If P_{si} does have optimal integer solution, then U_{si} becomes the new lower bound, so we set $L = U_{si}$, and generate *currentbest* from P_s. We generate *currentbest* from P_s by setting $\{a_{ij}\} \in$ *currentbest* if and only if $x_{ij} = 1$ is in the solution of P_s. If P_{si} does not have optimal integer solution, then we may be able to get a higher lower bound by branching from P_{si}, so we add P_{si} to *activeset*.

Step 7 At the end of the algorithm, we are given $\delta_{n-k}(A) = L$ (the optimal solution value to BPSM) and elements in *currentbest* are selected (the sum of these elements totals $\delta_{n-k}(A)$).

We can terminate the algorithm before it finishes and have L as a lower bound to $\delta_{n-k}(A)$ and U as an upper bound to $\delta_{n-k}(A)$. This is useful if we don't need the exact value of $\delta_{n-k}(A)$, but need to know when it is within a certain percentage of optimality. After each iteration of the algorithm, we can check L and U and when they are close together enough, we can terminate the algorithm.

This is useful as the branch and bound method is unlikely to be polynomial in its complexity. Sometimes it may terminate very quickly, possibly with $\text{BPSM}_{Frac}(A, k)$ immediately giving an integer solution without the need to branch. But at other times, it may not terminate quickly, and we will not know how much longer it will continue for. However, the closeness of the lower and upper bounds may indicate approximately how much longer we need to terminate the algorithm.

This branch and bound algorithm has been translated into Maple. The code for this can be found in Appendix A. The algorithm and code is not intended to be especially efficient. As no method was known other than complete enumeration, the intention was to help in the formulating and initial testing of conjectures. Sample output and explanation can be found in Appendix B.

Chapter 7

Conclusions and future research

We have obtained a number of useful results for BPSM. However, we still do not know if BPSM is polynomially solvable or not. There are also several questions that remain unanswered. Trying to solve these questions is left for future research.

We showed that if an algorithm polynomially solves BPSM for any irreducible matrix, then we can solve BPSM for all matrices. So whilst trying solve BPSM, we can restrict our attention to irreducible matrices. However, questions remain: Can we find a polynomial algorithm to solve BPSM? Will it ever be possible to do so? - Is BPSM *NP*-complete?

If it is possible to polynomially solve BPSM, then we can polynomially solve the job rotation problem, and can find the characteristic max-polynomial in polynomial time.

We gave several cases where we could solve BPSM in polynomial time. Can we find more special cases, or extend the current results? Can we use these results to help solve BPSM in general?

We looked at similar problems to BPSM. These included BSM and BBPSM. We polynomially solved some in polynomial time or stated they were *NP*-complete. We solved some of the others in special cases. We showed how the problems are related, specifically that unresolved problems identified seem to be as hard as either the BPSM problem, or the existence version of BPSM (i.e. BPSM(A, k) for $A \in \mathbb{T}^{n\times n}$).

Table 5.1 on page 70 summarised the results. The problems are very much linked. Proving any of the $?_1$'s are *NP*-complete would mean all $?_1$ problems (including BPSM) are *NP*-complete. Proving any of the $?_2$'s are polynomially solvable would mean all $?_2$ problems including are polynomially solvable. Proving any of the $?_2$'s are *NP*-complete would mean all remaining unresolved problems (including BPSM) are *NP*-complete. Proving any of the $?_1$'s are polynomially solvable would mean all remaining problems (including BPSM) are polynomially solvable. Can we make further progress with any of these problems? Are $?_1$ problems and $?_2$ problems as hard as each other?

We gave several bounds for the optimal value of BPSM, indicating how they compared against each other. When are these bounds met? It is hard to find a lower bound without also finding a $k \times k$ principal submatrix with k independent entries as well. For the latter we have no known efficient solution method in general. Is it ever possible to squeeze the upper and lower bounds together giving the same value?

We gave a branch and bound algorithm to solve BPSM. How efficient is this algorithm? Can we improve it? Instead of branching on a single element, would it be more efficient to branch on whether a particular row (and column) belongs in an optimal submatrix?

BPSM$_{Frac}$, the relaxation of the BPSM linear program given by (6.5.4) where we allow fractional values was studied and shown to be polynomially solvable. Is there an alternative (perhaps combinatorial) method to the use instead of the simplex method or ellipsoid method to solve the problem and to find the complexity? Can we determine a class of matrix for which BPSM$_{Frac}$ solves BPSM?

We defined an S_k matrix. This is a matrix A for which the optimal solution to BSM(A, k) is equal to the optimal solution of BPSM(A, k). (Recall the optimal solution of BSM is always greater or equal to the optimal solution of BPSM(A, k).) Can we provide a condition for a matrix to be an S_k matrix? Can we determine when a matrix A is S_k transformable? (i.e. can we find when there exists an S_k matrix similar to A?) For any A and k, can we find a matrix B similar to A, where the optimal solution to BSM(B, k) is as low as possible? Similarly, can we find a matrix B similar to A, where the optimal solution to BSM(B, k) is equal to the optimal solution for BPSM$_{Frac}(B, k)$ for all A and k? If so, we would have an alternative method for solving BPSM$_{Frac}(A, k)$.

Is it possible to find inessential terms that satisfy (4.3.2) with equality for some isolated value of x. If so, we would be able to solve BPSM for matrices with entries from $\{0, -\infty\}$, as all but the first and last terms are inessential. If so, this would then mean that all $?_2$ problems are polynomially solvable.

As we discussed, a method that found δ_{n-k} can probably be adapted to also give an optimal principal submatrix as well. However, if we are just given the value of δ_{n-k}, can we find an optimal principal submatrix from this? If we could, then again we could solve BPSM for matrices with entries from $\{0, -\infty\}$, as we could "guess" $\delta_{n-k} = 0$ and try to find an optimal principal submatrix. If we can, then we confirm that $\delta_{n-k} = 0$, if we can't, then $\delta_{n-k} = -\infty$. If we could do this, then again this would then mean that all $?_2$ problems are polynomially solvable.

Parameterisations may be useful in answering some of these questions. We could set an element of a matrix to a parameter and see what effect changing the parameter had on the optimal solution to BPSM, or when inessential terms become essential, etc. It may be possible to identify elements that will play no part in the solution to BPSM, and so for such elements, we could set them to $-\infty$. Or if some elements of a matrix are $-\infty$, we could set them to an extremely negative (finite) number. Would either of these approaches help simplify the problem, or give us any new results?

So there are many more areas to research and questions to answer. In particular, can we find a polynomial algorithm to solve BPSM, or is it NP-complete?

Chapter A

Program Code

Here is the Maple code to solve BPSM(A, k) by using a branch and bound approach:

```
> ####################################################################################
> # Seth Lewis 30/05/06                                                              #
> # Solves BPSM(A,k) using simplex and branch an bound.                             #
> # Given a matrix A and constant k, bpsm(A,k,cyclelist,optval) will return         #
> # cyclelist:  an optimal list of cycles, and optval:  equal to delta_{n-k}.        #
> # Given integers n, lo, percfin, randmat(n,lo,hi,percfin,A) will generate          #
> # a random n*n matrix A with approximately percfin% finite integer elements       #
> # with values between lo and hi, and all other entries will be -infinity.         #
> ####################################################################################

> with(simplex):with(LinearAlgebra):with(ListTools):
> interface(rtablesize=infinity):Seed:=randomize():

> comat := proc(n,InMat,M1) local i,j,M:
> M := Matrix(2*n+1,n^2+n):  for i from 1 to n
> do M[2*n+1,n^2+i]:=1:  M[i,n^2+i]:=1:  M[i+n,n^2+i]:=1:
> for j from 1 to n do if InMat[i,j]=-infinity then
> M[i,j+(i-1)*n]:=0:
> else M[i,j+(i-1)*n]:=1:  fi:
> if InMat[j,i]=-infinity then M[i+n,i+(j-1)*n]:=0:  else
> M[i+n,i+(j-1)*n]:=1:  fi:  od:  od:  M1:=M:
> end proc:

> vars := proc(n,X1) local i,j,X:
> X := Vector(n^2+n):
> for i from 1 to n do X[n^2+i]:=y[i]; for j from 1 to n do
> X[j+(i-1)*n]:=x[i,j] od od:  X1:=X:
> end proc:
```

```
> Combine := proc(n,k,MX,Combined1) local i,Combined:
> Combined := Vector(2*n+1): for i from 1 to n do
> Combined[i]:=(MX)[i]=1: Combined[i+n]:=(MX)[i+n]=1: od:
> Combined[2*n+1]:=(MX)[2*n+1]=n-k: Combined1:=Combined:
> end proc:

> dispcosts:=proc(n,Cvec,Cmat) local i,j:
> Cmat:=Matrix(n,n): for i from 1 to n do for j from 1 to n do
> Cmat[i,j]:=Cvec[j+(i-1)*n]: od od end proc:

> costs:= proc(n,lono,hino,percfin,costs) local r, r2, costscoeff, i:
> r:=rand(lono..hino): r2:=rand(1..100): costscoeff := Vector[row](n^2+n):
> for i from 1 to n^2 do if r2()<=percfin then
> costscoeff[i]:=r() else costscoeff[i]:=-infinity: fi: od:
> for i from n^2+1 to n^2+n do costscoeff[i]:=0: od:
> costs:=costscoeff: end proc:

> settocycles := proc(n,inset,outlist)
> local varset,cycleset,cyclelist,tempcycle,ordtempcycle,i,j,xx,xxii,xxi,xxj,tempcyclecomplete:
> varset:=inset: userinfo(3,seth,lprint('varset'),print(varset)); #---
> cycleset:={}: userinfo(3,seth,lprint('cycleset'),print(cycleset)); #---
> while not varset={} do
> xx:=varset[1]: userinfo(4,seth,lprint('xx'),print(xx)); #----
> varset:=varset minus {xx};
> tempcycle:=[]: userinfo(4,seth,lprint('tempcycle'),print(tempcycle)); #----
> for i from 1 to n do
> for j from 1 to n do
> if xx=(x[i,j]=1) then xxii:=i; xxi:=i; xxj:=j;
> fi; od; od; userinfo(4,seth,lprint('xxii'),print(xxii)); #----
> userinfo(5,seth,lprint('xxi'),print(xxi)); #-----
> userinfo(5,seth,lprint('xxj'),print(xxj)); #-----

> tempcyclecomplete:=false;
> userinfo(4,seth,lprint('tempcyclecomplete'),print(tempcyclecomplete)); #----
> while not tempcyclecomplete do
> tempcycle:= [op(tempcycle),xxi]; userinfo(5,seth,lprint('tempcycle'),print(tempcycle)); #-----
> if xxj=xxi or xxj=xxii then tempcyclecomplete:=true;
> userinfo(4,seth,lprint('tempcyclecomplete'),print(tempcyclecomplete)); #----
> userinfo(4,seth,lprint('tempcycle'),print(tempcycle)); #---
> unassign('ordtempcycle'):orderlist(tempcycle,ordtempcycle);
> userinfo(3,seth,lprint('ordtempcycle'),print(ordtempcycle)); #---
> userinfo(3,seth,lprint('varset'),print(varset)); #---
> else xxi:=xxj; userinfo(5,seth,lprint('xxi'),print(xxi)); #-----
>  for j from 1 to n do
>  if member((x[xxi,j]=1), inset) then xxj:=j; fi; od;
>  userinfo(5,seth,lprint('xxj'),print(xxj)); #-----
>  varset:=varset minus {(x[xxi,xxj]=1)}; userinfo(5,seth,lprint('varset'),print(varset)); #-----
>  fi; od:
>  cycleset:=cycleset union {ordtempcycle};
>  userinfo(3,seth,lprint('cycleset'),print(cycleset)); #---
>  od:  userinfo(3,seth,lprint('cycleset'),print(cycleset)); #---
>  cyclelist:=convert(cycleset,list);
>  orderlistlist(cyclelist,outlist);
>  end proc:

>  orderlist:=proc(Lin,Lout)
>  local l,i,j,endi:
>  l:=nops(Lin);
>  endi:=1;for i from 1 to l while not Lin[i]=min(seq( Lin[j], j=1..l )) do endi:=i+1 od;
>  Lout:=Rotate(Lin,endi-1);
>  end proc:

>  orderlistlist:=proc(Lin,Lout)
>  local L,x,l,M:
>  L:=Lin:  M:=[L[1]];
>  L:=subsop(1=NULL,L);
>  while not L=[] do x:=L[1];
>  L:=subsop(1=NULL,L);
>  l:=BinaryPlace(M,x, proc(x,y) op(1,x)<op(1,y) end proc);
>  M:=[op( 1..l, M ), x, op( l+1..-1, M )]; od;
>  Lout:=M:
>  end proc:

>  ## This is an ILP algorithm by Anu Pathria, pathria@arpa.berkeley.edu, Jan/90
>  ## for maximizing a linear objective function "obj" over linear constraints
>  ## "const", with integer variables, using a branch and bound method,
>  ## branching on the greatest objective value.  An optional third argument
>  ## "sgn" specifies NONNEGATIVE or UNRESTRICTED variables.  Output :  An optimal
>  ## assignment, or NULL if unbounded, or {} if infeasible.

>  ilp := proc(obj,const)
>  local dummy,sgn,val,lower,incumb,sol,t,i,lp,c,lps,bestbound;
>  if nargs = 2 then sgn := UNRESTRICTED
>  elif nargs = 3 then sgn := args[3]
>  else ERROR(`Wrong number of arguments`) fi;
>  incumb := {};#incumbent solution
>  lower := -infinity;#lower bound on optimal value
> #with(simplex,[]); # Put here or before algorithm
> sol := simplex[maximize](obj,const,sgn);
> #If LP is unbounded, then ILP is infeasible or unbounded.
> #Because simplex[maximize] doesn't recognize a constant as a linear
> #objective function, the test is as shown here.
> if sol = NULL then sol := ilp(dummy,{dummy=0} union const,sgn);
> if sol = {} then RETURN({}) else RETURN(NULL) fi
> elif sol = {} or `ilp/intsol`(sol) then RETURN(sol) fi;

> # Boolean function for best-bound branching rule.
> bestbound := proc(x,y) evalb(x[3] <= y[3]) end:
> lps := heap[new](bestbound,[const,sol,subs(sol,obj)]);# branch prique
> while not(heap[empty](lps)) do
> lp := heap[extract](lps);#lp to branch from
> if lower = -infinity or lp[3] > lower then
> t := `ilp/intsol`(lp[2]);
> c[1] := lp[1] union {op(1,t) <= trunc(op(2,t))};
> c[2] := lp[1] union {op(1,t) >= trunc(op(2,t))+1};

> # Consider 2 branches
> for i from 1 to 2 do
> sol := simplex[maximize](obj,c[i],sgn);
> if sol <> {} then
> t := `ilp/intsol`(sol);
> val := subs(sol,obj);
> if lower = -infinity or val > lower
> then if t then incumb := sol ; lower := val;
> else heap[insert]([c[i],sol,val],lps) fi; fi; fi; od; fi; od;
> incumb;#final incumbent solution is optimal
> end proc:

> # Returns assignment S if any variables non-integer, else true.
> `ilp/intsol` := proc(S)
> local i;
> for i in S do if not(type(op(2,i),integer)) then RETURN(i) fi od;
> true; end proc:
```

```
> bpsm :=proc(InMat,k,cyclelist,optval)
> local n,MX,cnsts,C,Ctrue,i,j,obj,solset,allone,opt,smallsolset,cyclelist1:
> n:=RowDimension(InMat):
> unassign('M'):comat(n,InMat,M): userinfo(5,seth,lprint('M'),print(M)); #-----
> unassign('X'):vars(n,X): userinfo(5,seth,lprint('X'),print(X)); #-----
> MX:=M.X: userinfo(5,seth,lprint('MX'),print(MX)); #-----
> unassign('cnstsmat'):Combine(n,k,MX,cnstsmat):
> userinfo(4,seth,lprint('cnstsmat'),print(cnstsmat)); #----
> cnsts:=convert(cnstsmat,set):  userinfo(3,seth,lprint('cnsts'),print(cnsts)); #---
> unassign('C'):C := Vector[row](n^2+n):
> unassign('Ctrue'):Ctrue := Vector[row](n^2+n):
> for j from 1 to n do
> for i from 1 to n do
> Ctrue[j+n*(i-1)]:=InMat[i,j];
> if InMat[i,j]=-infinity then C[j+n*(i-1)]=0
> else C[j+n*(i-1)]:=InMat[i,j]; fi;
> od od; userinfo(5,seth,lprint('Ctrue'),print(Ctrue)); #-----
> userinfo(5,seth,lprint('C'),print(C)); #-----
> obj:=C.X: userinfo(3,seth,lprint('obj'),print(obj)); #---
> unassign('Costs'):dispcosts(n,Ctrue,Costs):  userinfo(5,seth,lprint('Costs'),print(Costs)); #-----

> solset:=ilp(obj,cnsts,NONNEGATIVE): userinfo(5,seth,lprint('solset'),print(solset)); #-----
> if solset={} then smallsolset:='id';
> cyclelist:=smallsolset;
> opt:=-infinity;
> else
> allone:={}:
> for i from 1 to n do
> for j from 1 to n do
> allone:=allone union {x[i,j]=1}; od; od; userinfo(5,seth,lprint('allone'),print(allone)); #-----
> smallsolset:=solset intersect allone;
> opt:=subs(solset,obj):

> unassign('cyclelist1'):settocycles(n,smallsolset,cyclelist1):
> cyclelist:=cyclelist1;
> fi;
> userinfo(4,seth,lprint('smallsolset'),print(smallsolset)); #----
> userinfo(2,seth,lprint('cyclelist1'),print(cyclelist1)); #----
> userinfo(2,seth,lprint('opt'),print(opt)); #--
> optval:=opt:
> end proc:

> randmat := proc(n,lono,hino,percfin,A)
> unassign('C'):costs(n,lono,hino,percfin,C): userinfo(5,seth,lprint('C'),print(C)); #-----
> unassign('Costs'):dispcosts(n,C,Costs):  userinfo(2,seth,lprint('Costs'),print(Costs)); #--
> A:=Costs:
> end proc:
> ####################################################################################
```

Chapter B

Sample Output

Here is some sample output of the program code:

Example 1

```
>  A:=Matrix(7, 7,
>  [[-infinity,0,-infinity,-infinity,-infinity,-infinity,-infinity],
>  [-infinity,-infinity,0,-infinity,-infinity,-infinity,-infinity],
>  [-infinity,-infinity,-infinity,0,-infinity,-infinity,-infinity],
>  [0,-infinity,-infinity,-infinity,0,-infinity,-infinity],
>  [0,-infinity,-infinity,-infinity,0,-infinity,-infinity],
>  [-infinity,-infinity,-infinity,0,-infinity,-infinity,0],
>  [-infinity,-infinity,-infinity,-infinity,0,-infinity,-infinity]]);
>  n:=RowDimension(A):
>  for k from 1 to n do unassign('cyclelist'):unassign('optval'):bpsm(A,k,cyclelist,optval):
>  print('k'=k,'cyclelist'=cyclelist,'optval'=optval); od:
```

$$A := \begin{bmatrix} -\infty & 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty & -\infty \\ 0 & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ 0 & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty & 0 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \end{bmatrix}$$

$$k = 1,\ \mathit{cyclelist} = [[5]],\ \mathit{optval} = 0$$

$$k = 2,\ \mathit{cyclelist} = \mathit{id},\ \mathit{optval} = -\infty$$

$$k = 3,\ \mathit{cyclelist} = \mathit{id},\ \mathit{optval} = -\infty$$

$$k = 4,\ \mathit{cyclelist} = [[1,\ 2,\ 3,\ 4]],\ \mathit{optval} = 0$$

$$k = 5,\ \mathit{cyclelist} = [[1,\ 2,\ 3,\ 4,\ 5]],\ \mathit{optval} = 0$$

$$k = 6,\ \mathit{cyclelist} = \mathit{id},\ \mathit{optval} = -\infty$$

$$k = 7,\ cyclelist = id,\ optval = -\infty$$

The output here solves BPSM(A) for a preset matrix

$$A = \begin{pmatrix} -\infty & 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & 0 & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty & -\infty \\ 0 & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ 0 & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty & 0 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \end{pmatrix}.$$

It tells us for instance that for $k = 1$, selecting the loop (v_5, v_5) or element a_{55} will give us the optimal solution value of 0.

Note that here *id* means the identity of length k, i.e. the first k diagonal entries, so for $k = 2$ we could select a_{11} and a_{22} to get an optimal solution value of $-\infty$. In fact if $\delta_{n-k}(A) = -\infty$, then any assignment within a principal submatrix would be optimal, giving $-\infty$ as the optimal solution value.

For $k = 4$, we see that cycle $(v_1, v_2, v_3, v_4, v_1)$ is optimal leading to an optimal solution value of 0. The optimal principal submatrix would be $A(\{1, 2, 3, 4\})$.

Example 2

```
> n:=7;lo:=0;hi:=0;percfin:=30; #rand no.s between lo and hi,
> #percfin = approx percentage (between 0 and 100) of finite values
> unassign('A'):randmat(n,lo,hi,percfin,A): print('A'=A); for k from 1 to n do
> unassign('cyclelist'):unassign('optval'):bpsm(A,k,cyclelist,optval):
> print('k'=k,'cyclelist'=cyclelist,'optval'=optval); od:
```

$$n := 7$$

$$lo := 0$$

$$hi := 0$$

$$percfin := 30$$

$$A = \begin{bmatrix} -\infty & 0 & -\infty & -\infty & -\infty & -\infty & -\infty \\ 0 & -\infty & -\infty & -\infty & -\infty & -\infty & 0 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty & 0 \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & 0 & -\infty & -\infty & 0 \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty & -\infty \end{bmatrix}$$

$$k = 1,\ cyclelist = id,\ optval = -\infty$$

$$k = 2,\ cyclelist = [[1,\ 2]],\ optval = 0$$

$$k = 3,\ cyclelist = [[4,\ 7,\ 5]],\ optval = 0$$

$$k = 4,\ cyclelist = id,\ optval = -\infty$$

$$k = 5,\ cyclelist = [[1,\ 2],\ [4,\ 7,\ 5]],\ optval = 0$$

$$k = 6,\ cyclelist = id,\ optval = -\infty$$

$$k = 7,\ cyclelist = id,\ optval = -\infty$$

Here we create a random matrix of dimension 7×7 and with approximately 30% of entries having finite entries between 0 and 0 (i.e. equal to 0), and the rest being $-\infty$.

Here we see that for $k = 5$ for instance, the optimal principal submatrix is $A(\{1, 2, 4, 7, 5\})$, the optimal solution value being 0, arising from the cycle weights from (v_1, v_2, v_1) and (v_4, v_7, v_5, v_4), or elements $a_{12}, a_{21}, a_{47}, a_{75}$ and a_{54}.

Example 3

```
> n:=7;lo:=-9;hi:=9;percfin:=80; #rand no.s between lo and hi,
> #percfin = approx percentage (between 0 and 100) of finite values
> unassign('A'):randmat(n,lo,hi,percfin,A): print('A'=A); for k from 1 to n do
> unassign('cyclelist'):unassign('optval'):bpsm(A,k,cyclelist,optval):
> print('k'=k,'cyclelist'=cyclelist,'optval'=optval); od:
```

$$n := 7$$

$$lo := -9$$

$$hi := 9$$

$$percfin := 80$$

$$A = \begin{bmatrix} 0 & -8 & 0 & 1 & 6 & -3 & 2 \\ 4 & -\infty & -7 & 2 & 4 & -\infty & 8 \\ -\infty & -4 & -5 & -5 & 8 & 5 & 5 \\ 7 & -2 & -8 & 7 & 7 & -6 & 6 \\ 4 & -8 & 7 & -2 & -\infty & -4 & -6 \\ -\infty & -\infty & -2 & 4 & -8 & 0 & 3 \\ -3 & -2 & 2 & 0 & 4 & 8 & 9 \end{bmatrix}$$

$$k = 1,\ cyclelist = [[7]],\ optval = 9$$

$$k = 2,\ cyclelist = [[4], [7]],\ optval = 16$$

$$k = 3,\ cyclelist = [[3, 5], [7]],\ optval = 24$$

$$k = 4,\ cyclelist = [[3, 5], [4], [7]],\ optval = 31$$

$$k = 5,\ cyclelist = [[3, 5], [4, 7, 6]],\ optval = 33$$

$$k = 6,\ cyclelist = [[1, 5, 3, 6, 4], [7]],\ optval = 38$$

$$k = 7,\ cyclelist = [[1, 5, 3, 2, 7, 6, 4]],\ optval = 36$$

The past two examples considered $0, -\infty$ matrices. Here we generate a more general matrix in $\overline{\mathbb{R}}^{n\times n}$, with approximately 80% of the entries having a finite value between -9 and 9, and the rest being $-\infty$.

Here we can see that the optimal solution value tends to increase as k increase (probably due to adding more numbers together as k increases). However this is a general trend rather than a rule (comparing for $k = 6$ and 7 we see the optimal solution value decreases).

Using this maple code on several examples can help to check if conjectures appear to be true or not, before attempting to prove them.

Chapter C
Paper: On the Job Rotation Problem

Here is a pre-print of a paper co-written by myself and Dr. P Butkovič called *On the Job Rotation Problem* [15], which at the time of writing is unpublished, but has been accepted for publication in Discrete Optimization.

List of References

[1] M. Akian, R. Bapat, and S. Gaubert. *Handbook of Linear Algebra*, volume 39 of *Discrete Mathematics and Its Applications*, chapter Max-plus algebras. Chapman and Hall, 2006, in press.

[2] F. L. Baccelli, G. Cohen, G.-J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity.* John Wiley, Chichester, New York, 1992.

[3] R. B. Bapat and T. E. S. Raghavan. Nonnegative matrices and applications. In G.-C. Rota, editor, *Encyclopedia of Mathematics and its Applications*, volume 64. Cambridge University Press, 1997.

[4] D. P. Bertsekas. *Encyclopedia of Optimization*, volume I-VI, chapter Auction Algorithms. Kluwer Academic Publishers, Dordrecht, 2001.

[5] R. E. Burkard. Selected topics in assignment problems. *Discrete Applied Mathematics*, (123):257–302, 2002.

[6] R. E. Burkard and P. Butkovič. On the best principal submatrix assignment problem with a symmetric cost matrix. In preparation.

[7] R. E. Burkard and P. Butkovič. Finding all essential terms of a characteristic maxpolynomial. *Discrete Applied Mathematics*, 130:367–380, 2003.

[8] R. E. Burkard and P. Butkovič. Max algebra and the linear assignment problem. *Math. Program. Ser. B*, 98:417–429, 2003.

[9] R. E. Burkard and R. A. Cuninghame-Green. Private communication.

[10] P. Butkovič. Max-algebra and matrix scaling. University of Birmingham - Preprint 2003/25, 2003.

[11] P. Butkovič. Max-algebra: The linear algebra of combinatorics? *Linear Algebra and its Applications*, 367:313–335, 2003.

[12] P. Butkovič. Max-algebraic eigenvalues: The reducible case. Preprint 2003/24, 2003.

[13] P. Butkovič. On the coefficients of the max-algebraic characteristic polynomial and equation. *Kybernetika*, 39(2):129–136, 2003.

[14] P. Butkovič and R.A. Cuninghame-Green. On the linear assignment problem for special matrices. *IMA Journal of Management Mathematics*, 15:112, 2004.

[15] P. Butkovič and S. Lewis. On the job rotation problem. *Discrete Optimization*, 2006. doi:10.1016/j.disopt.2006.11.003.

[16] P. Butkovič and L. Murfitt. Calculating essential terms of a characteristic maxpolynomial. *CEJOR*, 8:237–246, 2000.

[17] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, Cambridge, Massachusetts, second edition, 2001.

[18] R. A. Cuninghame-Green. Minimax algebra. *Lecture Notes in Economics and Mathematical Systems*, 166, 1979.

[19] R. A. Cuninghame-Green. The characteristic maxpolynomial of a matrix. *J. Math. Analysis and Applications*, 95:110–116, 1983.

[20] R. A. Cuninghame-Green. Minimax algebra and applications. *Advances in Imaging and Electron Physics*, 90:1–121, 1995.

[21] R. A. Cuninghame-Green and P. F. J. Meijer. An algebra for piecewise-linear minimax problems. *Discrete Appl. Math*, 2:267–294, 1980.

[22] I. Curiel. *Cooperative Game Theory and Applications.* Kluwer Academic Publishers, The Netherlands, 1997.

[23] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18(5):1013–1036, 1989.

[24] E. Gassner. *Variants of the assignment and of the transportation problem.* PhD thesis, Graz, 2004.

[25] M. Gondran and M. Minoux. L'indépendance linéaire dans les dioïdes. *Bull. Direction Études Rech. Sér. C Math. Inform.*, 1:67–90, 1978.

[26] B. Heidergott, G. J. Olsder, J. Woude, and J. Van Der Woude. *Modeling and Analysis of Synchronized Systems: A Course on Max-Plus Algebra and Its Applications.* Princeton University Press, November 2005.

[27] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research*, volume Sixth. McGraw-Hill, Inc., 1995.

[28] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[29] A. V. Karzanov. Maximum matching of given weight in complete and complete bipartite graphs. *Cybernetics*, 23(1):8–13, 1987.

[30] K. H. Kim. *Boolean Matrix Theory and Applications.* Marcel Dekker, Inc., 1982.

[31] J. Van Leeuwen. *Graph Algorithms*, volume A: Algorithms and Complexity of *Handbook of Theoretical Computer Science*, chapter 10, pages 525–631. Elsevier Science Publishers B.V., 1990.

[32] W. McEneaney. *Max-plus Methods for Non-linear Control and Estimation.* Birkhauser Verlag AG, 2004.

[33] K. Mulmuley, U. V. Vazirani, and V. V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–112, 1987.

[34] L. Murfitt. *Discrete Event Dynamic Systems in Max-Algebra: Realisation and Related Combinatorial Problems.* PhD thesis, School of Mathematics and Statistics, University of Birmingham, July 2000.

[35] G. J. Olsder and C. Roos. Cramér and cayley-hamilton in the max algebra. *Linear Algebra Appl.*, 101:87–108, 1988.

[36] J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum mean cycle problems. *Mathematical Programming*, 54:41–56, 1992.

[37] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley Publishing Company, 1994.

[38] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity.* Dover Publications, 1998.

[39] K. H. Rosen. *Handbook of discrete and combinatorial mathematics, Boca Raton.* CRC Press, London, 2000.