A Survey of Monge Properties

Swetha Sethumadhavan
University of Nevada, Las Vegas

A SURVEY OF MONGE PROPERTIES

by

Swetha Sethumadhavan

Bachelor of Science
Cochin University of Science and Technology, India
2004

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
School of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2009

UNLV
UNIVERSITY OF NEVADA LAS VEGAS

Thesis Approval
The Graduate College
University of Nevada, Las Vegas

APRIL 24TH, 2009

The Thesis prepared by

SWETHA SETHUMADHAVAN

Entitled

A SURVEY OF MONGE PROPERTIES

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

[redacted]
Examination Committee Chair

[redacted]
Dean of the Graduate College

[redacted]
Examination Committee Member

[redacted]
Examination Committee Member

[redacted]
Graduate College Faculty Representative

1017-53

ABSTRACT

A Survey of Monge Properties

by

Swetha Sethumadhavan

Dr. Wolfgang Bein, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

Monge properties play an important role in theoretical computer science. Many greedy algorithms are based on such properties, as is speedup in dynamic programming. Monge properties are simple monotonicity properties which are observed and used in various settings such as resource optimization, computational geometry, statistical sampling, computational biology and coding.

These properties occur naturally, and it was the eighteenth century French engineer Gaspard Monge who first wrote about them. The plural in "Monge properties" is used deliberately, as researchers today study a variety of different Monge-like properties: bottleneck, algebraic, higher-dimensional, joint-meet submodular on lattices, total monotonicity.

The thesis discusses various Monge properties and its application in solving various fundamental problems. It also focuses on algorithmic speed-up and the use of the SMAWK and LARSCH algorithms for exploiting total monotonicity of Monge matrices. We consider applications such as the Traveling Salesman Problem, the Assignment Problem, the Higher Dimensional Transportation Problem, various scheduling problems, specifically batching, and host of other problems.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

I would like to thank Dr. Wolfgang Bein for chairing my committee and advising this work. I am thankful for his continous guidance and help to deepen my work. Without his generous help this thesis would not have had such a rich content.

I am thankful to Dr. Ajoy K Datta for his moral support and guidance through my Masters program and help on my thesis. I would also like to specifically thank Dr.Yoohwan Kim and Dr. Venkatesan Muthukumar for serving on the committee. For this and for being generous with their time when I needed it, I am deeply indebted to them.

Special thanks go to Dr. Doina Bein for helping with my thesis report.

I would like to thank the faculty at the School of Computer Science, University of Nevada, Las Vegas for the formal education along with generous financial support.

I am grateful to my husband for his support and inspiration.

Finally and most importantly, I thank my parents and my brother, for their love and support.

CHAPTER 1

INTRODUCTION

In this chapter we present notions related to Monge matrices. This chapter goes through the history of development of Monge matrices and its properties. It discusses how the property was dormant for about a century and then gaining popularity in late 20th century.

Upon further research into the functions and uses of Monge matrices, I found immense literature on the topics detailing the various properties of Monge Matrix. For example, a Monge matrix in scheduling problems in many interesting cases improves the running time of algorithms.

Definition 1.1 *An $m \times n$ matrix C is said to be a* Monge matrix *if* $\forall\ i_1,\ j_1,\ i_2,\ j_2$ *such that*

$$1 \leq i_1 < i_2 \leq m \quad and \quad 1 \leq j_1 < j_2 \leq n$$

we have

$$C[i_1, j_1] + C[i_2, j_2] \leq C[i_1, j_2] + C[i_2, j_1] \tag{1.1}$$

A matrix is called a Monge matrix if it satisfies Monge property as shown in Figure 1.1. If we consider any two rows and any two columns of a Monge Array then the sum of upper

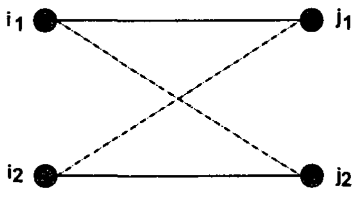

Figure 1.1: Monge Property

left and lower right element is less than or equal to the sum of the lower left and upper right element.

Matrices showing Monge properties arises often in practical applications especially in geometric settings. Monge property is named in honor of the French Mathematician Gaspard Monge who studied the property in the 18^{th} century, which was rediscovered in 1961 by A.J. Hoffman when he showed that a transportation problem can be solved by a Greedy method if the underlying cost are showing Monge properties [Hof63]. Hoffman coined the term Monge Matrix to credit Gaspard Monge for his early study and discovery.

G. Monge considered this property in connection with transporting earth, when he wanted to split two equally large volumes of earth representing point x and y. Monge looked at this problem as splitting the large volume into infinitely small particles and then associate them with each other so that sum of the product of the lengths of the paths used by the particles and the volume of the particle is minimized.

In 1961 Hoffman showed that if the cost matrix satisfies the Monge property then the Hitchcock transportation problem can be solved using a simple greedy approach [Hof63]. The research on Monge property had been silent after work of Hoffman till the paper published by Aggarwal [AKM+87] on searching in totally monotone matrices, which lead to a new interest in this area sparking a series of research and publications. Researchers interested in this field are overwhelmed by the literature's available in this area which lead to the need for a study to connect the results from these researchers. Monge properties are used in various settings such as resource optimization, computational geometry [AKM+87], statistical sampling [FP89], computational biology [EZGI90] coding theory [Yao80, AKL+89] and theory of greedy algorithms [BBH93].

Recently the Monge property has again been shown to be useful in diverse fields showing that Monge property is not a single property but a set of properties like algebraic Monge property, reverse Monge, higher dimensional, total monotonicity etc..

Thesis Overview This thesis presents an overview of different Monge properties and draws a connection between them. The thesis is organized as follows. Chapter 2 gives the definition of major terms used in the survey and it also explains some of the properties of

Monge matrices. Chapter 3 discusses Monge property and its characterization, also lists some examples of Monge matrices. Chapter 4 deals with the SMAWK algorithm for finding the minimum value of all rows in a Monge matrix in linear time. Chapter 5 discusses several applications of Monge matrices. Some of the applications discussed are Traveling Salesman Problem, Batching Problem, Assignment Problem, Scheduling Problem and d-dimensional Transportation Problem. This chapter also discusses about Gilmore-Gomory matching problem. We conclude in Chapter 6.

CHAPTER 2

DEFINITIONS

In this chapter we present some of the important properties of Monge matrices. Some of the properties which we consider are total monotonicity property, algebraic Monge property, bottleneck Monge property and higher dimensional Monge property. This chapter also discusses about the Distribution arrays and Inverse Monge matrices.

Definition 2.1 (Inverse Monge) *A matrix C is called an* inverse Monge *matrix if*

$$C[i_1, j_1] + C[i_2, j_2] \;\; \geq \;\; C[i_1, j_2] + C[i_2, j_1]$$

$\forall\ i_1, j_1, i_2, j_2$ *such that*

$$1 \leq i_1 < i_2 \leq m \;\; and \;\; 1 \leq j_1 < j_2 \leq n$$

An inverse Monge matrix can be transformed into a Monge matrix by multiplying all entries by -1. For Monge matrices it suffices to require that the Monge property holds for adjacent rows and adjacent columns. Equation (1.1) holds if and only if

$$C_{i,j} + C_{i+1,j+1} \;\; \leq \;\; C_{i+1,j} + C_{i,j+1} \tag{2.1}$$

$\forall\ 1 \leq i < m,\ 1 \leq j < n$

Lemma 2.1 *If the Monge property holds for all adjacent rows and columns of a Matrix, then that matrix is a Monge matrix.*

Proof. Since the Monge property holds for adjacent rows and columns

$$C_{i,j} + C_{i+1,j+1} \leq C_{i+1,j} + C_{i,j+1} \quad (2.2)$$

$$C_{i,j+1} + C_{i+1,j+2} \leq C_{i,j+2} + C_{i+1,j+1} \quad (2.3)$$

$$C_{i+1,j} + C_{i+2,j+1} \leq C_{i+1,j+1} + C_{i+2,j} \quad (2.4)$$

$$C_{i+1,j+1} + C_{i+2,j+2} \leq C_{i+1,j+2} + C_{i+2,j+1} \quad (2.5)$$

By adding equations (2.2),(2.3),(2.4) and (2.5) will obtain that

$$C_{i,j} + C_{i+2,j+2} \leq C_{i,j+2} + C_{i+2,j} \quad (2.6)$$

Equation (2.6) shows that Monge property holds for any rows and columns thus the Matrix C is a Monge Matrix. □

Definition 2.2 (Monotone and Totally Monotone) *Let C be a matrix with real entries and let $j(i)$ be the index of the leftmost column containing minimum value in row i of C. C is said to be* monotone *if $i_1 < i_2$ implies $j(i_1) \leq j(i_2)$.*

C is called totally monotone *if all sub matrices are monotone.*

For example the matrix $C = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 2 \end{bmatrix}$ is a monotone matrix but not a totally monotone matrix, since the sub matrix $\begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ is not a monotone matrix.

Thus a matrix C is a totally monotone matrix if

$$C_{i,j} > C_{i,j+1} \text{ then } C_{i+1,j} > C_{i+1,j+1} \ \forall \ i < i+1, \ j < j+1$$

Monge matrices are totally monotone matrices. Figure 2.1 shows that if C is a Monge matrix and $C_{i,j}$ is greater than $C_{i,j+1}$ then $C_{i+1,j}$ cannot be less than or equal to $C_{i+1,j+1}$.

Definition 2.3 (Algebraic Monge Matrices) *Let $(H, *, \preceq)$ be an ordered commutative semi-group and the operation * is compatible with the order relation $\preceq$ i.e.*

$$\forall \ a, b, c \in H \ a \prec b \textit{ if and only if } c * a \preceq c * b$$

An $m \times n$ array C has the algebraic Monge *property if $\forall \ 1 \leq i < k \leq m$ and $1 \leq j < l \leq n$,*

$$c_{i,j} * c_{kl} \preceq c_{i,l} * c_{k,j}.$$

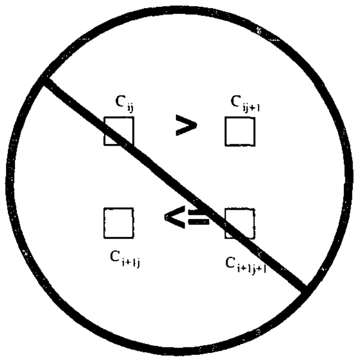

Figure 2.1: Forbidden criteria

Any matrix that satisfies the algebraic Monge property is called as an algebraic Monge *matrix.*

*If the semi-group operator * is strictly compatible with the order relation $\preceq$ then*

$$\forall\ a, b, c \in H\ \ a \prec b \textit{ if and only if } c * a \prec c * b$$

Lemma 2.2 *Let the operation * be strictly compatible with the order relation and let $C = c_{i,j}$ denote an array whose entries are drawn from the semi-group $(H, *, \preceq)$. If the matrix C has the algebraic Monge property, then C is a totally monotone matrix.*

If the operation * is compatible with the order relation but not strictly compatible, then the array whose entries are drawn from semi-group may have the algebraic Monge property without being a totally monotone matrix.

For example consider the semi-group $(R, max, \leq)$. The array $C = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ is an algebraic Monge matrix since $\max\{c_{ij}, c_{kl}\} \leq \max\{c_{il}, c_{kj}\}\ \forall\ i < k$ and $j < l$. But the above matrix is not totally monotone, whereas the matrix $C = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$ is an algebraic Monge matrix as well as it is totally monotone since the operator max is strictly compatible with the order relation $\leq$.

There are the following two special cases of Algebraic Monge property:

a) If the operation * is replaced with $+$ operator and $\leq$ as natural order then the array C has Monge property.

b) If the operation * is replaced with "*max*" operator and $\leq$ as natural order then we say that the array C has the *bottleneck Monge property.*

Definition 2.4 (Bottleneck Monge property) *A matix C has bottleneck Monge property if and only if*

$$max\{C_{i_1,j_1}, C_{i_2,j_2}\} < max\{C_{i_1,j_2}, C_{i_2,j_1}\}$$

$\forall\ i_1, j_1, i_2, j_2$ *such that* $i_1 < i_2$ *and* $j_1 < j_2$

A matrix is called a bottleneck Monge *matrix if it satisfies the bottleneck Monge property.*

Definition 2.5 (Distribution Matrix) *Let $D = (d_{i,j})$ be any nonnegative real $m \times n$ matrix. The matrix C obtained by*

$$C_{k,l} = -\sum_{k=1}^{i}\sum_{l=1}^{j} d_{kl}$$

is a Monge matrix and $-C$ is a Reverse Monge matrix. Matrix C is called the distribution matrix, *generated by the density matrix D.*

For example, let the matrix D be $\begin{bmatrix} 2 & 3 \\ 3 & 5 \end{bmatrix}$. Then the matrix C will be $\begin{bmatrix} -2 & -5 \\ -5 & -13 \end{bmatrix}$ and it is a Monge matrix.

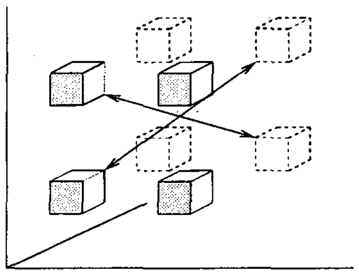

Figure 2.2: Higher-dimensional Monge Property

Definition 2.6 (Higher-Dimensional Monge Matrices) *An $n_1 \times n_2 \times \ldots n_d$ d-dimensional array $C = \{c[i_1, i_2, \ldots i_d]\}$, where $d \geq 2$ has the Monge property if, for all entries $c[i_1, i_2, \ldots i_d]$ and $c[j_1, j_2, \ldots c_d]$ the following relation holds (Figure 2.2)*

$$c[s_1, s_2, \ldots s_d] + c[t_1 + t_2, \ldots t_d] \leq c[i_1, i_2, \ldots i_d] + c[j_1, j_2, \ldots j_d]$$

$$\textit{where } 1 \leq k \leq d, s_k = \min(i_k, j_k) \textit{ and } t_k = \max(i_k, j_k)$$

An $n_1 \times n_2 \times \ldots n_d$ d-dimensional array $C = \{c[i_1, i_2, \ldots i_d]\}$ is called Monge *if every two dimensional array of C is Monge.*

The d-dimensional Monge property was first proposed by Aggarwal and Park [AP88a, AP88b].

CHAPTER 3

MONGE PROPERTY AND ITS CHARACTERIZATION

In this chapter we present examples of Monge matrices and also give the characterization of Monge matrices. Namely, any Monge matrix can be represented as sum of two unit vectors and the distribution array. We also discusses the Gilmore-Gomory matching problem.

3.1 Some examples of Monge Matrices

1. We prove that $C[i,j] = a_i + b_j$ is a Monge matrix where a_i and b_j are unit vectors such that $a_1 \leq a_2 \leq ...a_m, b_1 \leq b_2 \leq ...b_n$

   Proof.

$$\begin{aligned} C[i_1, j_1] + C[i_2, j_2] &= a_{i_1} + b_{j_1} + a_{i_2} + b_{j_2}. \\ &= a_{i_1} + b_{j_2} + a_{i_2} + b_{j_1}. \\ &= C[i_1, j_2] + C[i_2, j_1]. \end{aligned}$$

   Thus $C[i,j] = a_i + b_j$ is a Monge matrix. □

2. We prove that $C[i,j] = nm - a_i * b_j$ is a Monge matrix where $A = \{a_i\}$ and $B = \{b_j\}$ are unit vectors, $a_{i_1} \leq a_{i_2} \leq \ldots a_{i_n}, b_{j_1} \leq b_{j_2} \leq \ldots b_{j_n}$; n represents the number of rows and m represents the number of columns.

   Proof. We first consider the matrix $C[i,j] = a_i * b_j$. The matrix C is a Monge matrix if and only if

$$\begin{aligned}
C[i_1, j_1] + C[i_2, j_2] &\leq C[i_1, j_2] + C[i_2, j_1]. \\
a_{i_1} * b_{j_1} + a_{i_2} * b_{j_2} &\leq a_{i_1} * b_{j_2} + a_{i_2} * b_{j_1}. \\
a_{i_1} * b_{j_1} - a_{i_1} * b_{j_2} &\leq a_{i_2} * b_{j_1} - a_{i_2} * b_{j_2}. \\
a_{i_1}(b_{j_1} - b_{j_2}) &\leq a_{i_2}(b_{j_1} - b_{j_2}). \\
a_{i_1} &\geq a_{i_2}
\end{aligned}$$

Since $b_{j_1} - b_{j_2}$ is negative, this shows that $C[i,j] = a_i * b_j$ is a reverse Monge matrix. Thus

$$C[i_1, j_1] + C[i_2, j_2] \geq C[i_1, j_2] + C[i_2, j_1].$$

And $C[i,j] = -a_i * b_j$ is a Monge matrix.

Now we consider the matrix $C[i,j] = n * m - a_i * b_j$. Since the matrix $C[i,j] = a_i * b_j$ is a reverse Monge matrix, then

$$C[i_1, j_1] + C[i_2, j_2] \geq C[i_1, j_2] + C[i_2, j_1].$$

We multiply both sides by the value -1 and we obtain:

$$\begin{aligned}
-C[i_1, j_1] - C[i_2, j_2] &\leq -C[i_1, j_2] - C[i_2, j_1]. \\
-a_{i_1} * b_{j_1} - a_{i_2} * b_{j_2} &\leq -a_{i_1} * b_{j_2} - a_{i_2} * b_{j_1}. \\
nm - a_{i_1} * b_{j_1} + nm - a_{i_2} * b_{j_2} &\leq nm - a_{i_1} b_{j_2} + nm - a_{i_2} * b_{j_1} \\
C[i_1, j_1] + C[i_2, j_2] &\leq C[i_1, j_2] + C[i_2, j_1].
\end{aligned}$$

Thus $C[i,j] = n * m - a_i * b_j$ is a Monge matrix. □

3. Generally $C[i,j] = a_i * b_j \ \forall \ a_1 \leq a_2 \leq \dots a_n, b_1 \geq b_2 \geq \dots b_n$ or $\forall \ a_1 \geq a_2 \geq \dots a_n, b_1 \leq b_2 \leq \dots b_n$ is a Monge matrix.

   Proof. If $C[i,j]$ is a Monge matrix then

$$\begin{aligned}
C[i_1, j_1] + C[i_2, j_2] &\leq C[i_1, j_2] + C[i_2, j_1] \\
a_{i_1} * b_{j_1} + a_{i_2} * b_{j_2} &\leq a_{i_1} * b_{j_2} + a_{i_2} * b_{j_1} \\
a_{i_1}(b_{j_1} - b_{j_2}) &\leq a_{i_2}(b_{j_1} - b_{j_2})
\end{aligned}$$

If $a_1 \leq a_2 \leq \ldots a_n$ and $b_1 \geq b_2 \geq \ldots b_n$ then $b_{j_1} - b_{j_2} \geq 0$. Thus $a_{i_1} \leq a_{i_2}$. This shows that $C[i,j] = a_i * b_j$ is a Monge matrix if $a_1 \leq a_2 \leq \ldots a_n, b_1 \geq b_2 \geq \ldots b_n$.

If $a_1 \geq a_2 \geq \ldots a_n$ and $b_1 \leq b_2 \leq \ldots b_n$ then $b_{j_1} - b_{j_2} \leq 0$. Therefore $a_{i_1} \geq a_{i_2}$, which is true. This shows that $C[i,j] = a_i * b_j$ is a Monge matrix if $a_1 \geq a_2 \geq \ldots a_n, b_1 \leq b_2 \leq \ldots b_n$. □

4. We prove that $C[i,j]$ is a Monge matrix where

$$C[i,j] = \max(i,j) \; \forall \; i_1 < i_2 \text{ and } j_1 < j_2$$

Proof. The matrix C is a Monge matrix if and only if

$$\begin{aligned} C[i_1,j_1] + C[i_2,j_2] &\leq C[i_1,j_2] + C[i_2,j_1] \\ \max(i_1,j_1) + \max(i_2,j_2) &\leq \max(i_1,j_2) + \max(i_2,j_1) \end{aligned} \tag{3.1}$$

There are four different cases:

Case 1: $i_1 > j_1$ and $i_2 > j_2$

Thus Equation (3.1) becomes

$$i_1 + i_2 \leq \max(i_1, j_2) + i_2 \tag{3.2}$$

Case 1.1: $i_1 > j_2$

Thus Equation (3.2) becomes

$$i_1 + i_2 = i_1 + i_2$$

This satisfies the Monge property.

Case 1.2: $i_1 < j_2$

Thus Equation (3.2) becomes

$$i_1 + i_2 < j_2 + i_2$$

This satisfies the Monge property.

Case 2: $i_1 < j_1$ and $i_2 < j_2$

Thus Equation (3.1) becomes

$$j_1 + j_2 < j_2 + \max(i_2, j_1) \tag{3.3}$$

Case 2.1 : $i_2 > j_1$

Thus Equation (3.3) becomes

$$j_1 + j_2 < j_2 + i_2$$

This satisfies the Monge property.

Case 2.2 : $i_2 < j_1$

Thus Equation (3.3) becomes

$$j_1 + j_2 = j_1 + j_2$$

This satisfies the Monge property.

Case 3: $i_1 < j_1$ and $i_2 > j_2$

Thus Equation (3.1) becomes

$$j_1 + i_2 < i_2 + j_2$$

This satisfies the Monge property.

Case 4: $i_1 > j_1$ and $i_2 < j_2$

Thus Equation (3.1) becomes

$$i_1 + j_2 < j_2 + i_2$$

This satisfies the Monge property.

Thus $c[i, j] = \max(i, j)$ is a Monge matrix. □

5. We prove that the matrix $C[i, j]$ is a Monge matrix where

$$C[i, j] = \max(i, j) - \min(i, j) \; \forall \; i_1 < i_2 \text{ and } j_1 < j_2$$

Proof. The matrix C is a Monge matrix if and only if

$$C[i_1, j_1] + C[i_2, j_2] \leq C[i_1, j_2] + C[i_2, j_1] \tag{3.4}$$

$\min(i,j)$ can be written as

$$\min(i,j) = i + j - \max(i,j)$$

Thus Equation (3.4) becomes

$$\begin{aligned} 2\max(i_1,j_1) - i_1 - j_1 + 2\max(i_2,j_2) - i_2 - j_2 &\le 2\max(i_1,j_2) - i_1 - j_2 \\ &+ 2\max(i_2,j_1) - i_2 - j_1 \end{aligned}$$

We cancel i_1, j_1, i_2, j_2

$$2\max(i_1,j_1) + 2\max(i_2,j_2) \le 2\max(i_1,j_2) + 2\max(i_2,j_1) \tag{3.5}$$

We divide the equation by 2. Thus Equation (3.5) becomes

$$\max(i_1,j_1) + \max(i_2,j_2) \le \max(i_1,j_2) + \max(i_2,j_1)$$

Since $C[i,j] = \max(i,j)$ is a Monge matrix, $E[i,j] = \max(i,j) - \min(i,j)$ is also a Monge matrix. □

6. We prove that the matrix $C[i,j]$ is a Monge matrix where

$$C[i,j] = (i+j)^2 \ \forall\ i_1 < i_2 \text{ and } j_1 < j_2.$$

Proof.

$$(i+j)^2 = i^2 + j^2 + 2 * i * j$$

Since $C[i,j] = i * j$ is a reverse Monge matrix then

$$\begin{aligned} C[i_1,j_1] + C[i_2,j_2] &\ge C[i_1,j_2] + C[i_2,j_1] \\ i_1 * j_1 + i_2 * j_2 &\ge i_1 * j_2 + i_2 * j_1. \end{aligned} \tag{3.6}$$

We multiply both sides by 2, thus Equation (3.6) becomes

$$\begin{aligned} 2 * i_1 * j_1 + 2 * i_2 * j_2 &\geq 2 * i_1 * j_2 + 2 * i_2 * j_1 \\ i_1^2 + j_1^2 + +2 * i_1 * j_1 + i_2^2 + j_2^2 + 2 * i_2 * j_2 &\geq i_1^2 + j_1^2 + 2 * i_1 * j_2 \\ &+ i_2^2 + j_1^2 + 2 * i_2 * j_1 \\ C[i_1, j_1] + C[i_2, j_2] &\geq C[i_1, j_2] + C[i_2, j_1] \end{aligned}$$

The above equation shows that $C[i, j] = (i + j)^2$ is a reverse Monge matrix. □

7. We prove that the matrix $C[i, j]$ is a Monge matrix where

$$C[i, j] = \min(i, j) \ \forall \ i_1 < i_2 \text{ and } j_1 < j_2.$$

Proof.

$\min(i, j)$ can be written as $\min(i, j) = i + j - \max(i, j)$

We know that $C[i, j] = \max(i, j)$ is a Monge matrix

$$\begin{aligned} C[i_1, j_1] + C[i_2, j_2] &\leq C[i_1, j_2] + C[i_2, j_1] \\ \max(i_1, j_1) + \max(i_2, j_2) &\leq \max(i_1, j_2) + \max(i_2, j_1) \qquad (3.7) \end{aligned}$$

We multiply both sides by -1

Thus Equation (3.7) becomes

$$\begin{aligned} -\max(i_1, j_1) - \max(i_2, j_2) &\geq -\max(i_1, j_2) - \max(i_2, j_1) \\ i_1 + j_1 + \max(i_1, j_1) + i_2 + j_2 + \max(i_2, j_2) &\geq i_1 + j_2 \\ &+ \max(i_1, j_2) + i_2 + j_1 + \max(i_2, j_1) \\ C[i_1, j_1] + C[i_2, j_2] &\geq C[i_1, j_2] + C[i_2, j_1] \qquad (3.8) \end{aligned}$$

Equation (3.8) shows that $C[i, j] = \min(i, j)$ is a reverse Monge matrix. □

8. For permutation $\pi(i), i \in \{1, 2 \ldots n\}$, let us consider the function $f(\pi) = \sum_{i=1}^{n} i\pi(i)$. We show that the function f is maximized when π is the identity permutation.

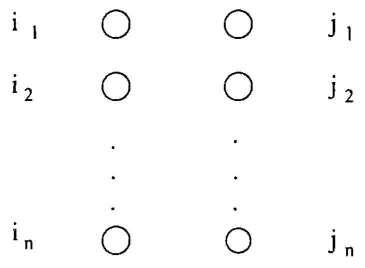

Figure 3.1: i, $\pi(i)$

Proof.

Let us consider Figure 3.1 where j_i corresponds to $\pi(i)$ $\forall$ $i \in \{1, 2, \ldots n\}$. Let $C[i_1, j_1] = i_1 * j_1 = i_1 * \pi(i_1)$ and $\pi(i)$ is identity ie $i = \pi(i), i_1 \leq i_2 \leq \ldots i_n$

$$\begin{aligned} C[i_1, j_1] + C[i_2, j_2] &\leq C[i_1, j_2] + C[i_2, j_1] \\ i_1 * j_1 + i_2 * j_2 &\leq i_1 * j_2 + i_2 * j_1 \\ i_1(j_1 - j_2) &\leq i_2(j_1 - j_2) \\ i_1 &\geq i_2 \end{aligned}$$

This shows that the matrix C is a reverse Monge matrix. The value of the function f will be maximized when $\pi(i)$ is the identity permutation. □

3.2 Characterization of Monge Matrices

Bein and Pathak [BP96] give a characterization of the Monge property. The authors show that any *mtimesn* Monge matrix $G = \{g_{ij}\}$ has a representation of the form

$$g_{i,j} = u_i + v_j + F_{i,j} \tag{3.9}$$

where u and v are vectors and F is a distribution array

$$F_{i,j} = \sum_{k \leq i, l \leq j} p_{kl}, \; p_{k,l} \leq 0$$

Thus if g is a Monge matrix then there exist the vectors u, v and the distribution array F such that Equation (3.9) holds.

Definition 3.1 *Let $\Box\, g_{i,j}$ be defined as*

$$\Box g_{i,j} = g_{i,j} - g_{i,j-1} - g_{i-1,j} + g_{i-1,j-1} \tag{3.10}$$

$g_{i,j} = 0$ *if either* $i = 0$ *or* $j = 0$.

The matrix G is said to be a Monge array *if*

$$\Box g_{i,j} \leq 0 \ \forall\ i, j \geq 2$$

The matrix G is said to be Monge *and* monotone *if*

$$\Box g_{i,j} \leq 0 \ \forall\ i \in Nm, j \in Nm, (i,j) \in Nm \times Nn - (1,1)$$

Theorem 3.1 *Given a distribution array F and the vectors u and v with*

$$g_{i,j} = u_i + v_j + F_{i,j}$$

then G is a Monge matrix.

Proof. If G is a Monge matrix then

$$\begin{aligned} g_{i,j} + g_{i+1,j+1} &\leq g_{i+1,j} + g_{i,j+1} \\ u_i + v_j + F_{i,j} + u_{i+1} + v_{j+1} + F_{i+1,j+1} &\leq u_{i+1} + v_j + F_{i+1,j} \\ &+ u_i + v_{j+1} + F_{i,j+1} \end{aligned} \tag{3.11}$$

$u_i, v_j, u_{i+1}, v_{j+1}$ get cancel out. Thus Equation (3.11) becomes

$$\begin{aligned} F_{i,j} + F_{i+1,j+1} &\leq F_{i+1,j} + F_{i,j+1} \\ F_{i+1,j+1} - F_{i+1,j} &\leq F_{i,j+1} - F_{i,j} \\ \sum_{k=i}^{i+1} p_{kl} &\leq \sum_{k=1}^{i} p_{kl} \\ p_{k,l} &\leq 0 \end{aligned}$$

This is true. Thus the matrix $G = g_{i,j}$ where $g_{i,j} = u_i + v_j + F_{i,j}$ is a Monge matrix. $\Box$

The matrix $G = \{g_{i,j}\},\ g_{i,j} = u_i + v_j - F_{i,j}$ is a reverse Monge matrix.

Lemma 3.1 *Let $g : Nm * Nn \rightarrow R$ with*

$$\Box g_{i,j} = 0 \ \forall \ Nm * Nn - (1,1)$$

Then there exist the vectors $u : Nm \rightarrow \mathbb{R}$, $v : Nn \rightarrow \mathbb{R}$ such that

$$g_{i,j} = u_i + v_j$$

Proof.

$$\begin{aligned} \Box g_{1,1} &= g_{1,1} - g_{1,0} - g_{0,1} + g_{0,0} \\ \Box g_{1,1} &= g_{1,1} \\ g_{1,1} &= \sum_{1\le k\le i, 1\le l\le j} \Box g_{k,l} \end{aligned}$$

Since $\Box g_{i,j} = 0 \ \forall \ N_m * N_n - (1,1)$

$$\begin{aligned} g_{1,1} = \sum_{1\le k\le i, 1\le l\le j} \Box g_{kl} &= g_{i,j} - g_{i,1} - g_{1,j} + g_{1,1} \\ g_{i,j} &= g_{i,1} + g_{1,j} \\ g_{i,j} &= u_i + v_j \end{aligned}$$

$\Box$

This lemma shows that if G is a Monge array then it is linearly separable *i.e.* G can be represented as $g_{i,j} = u_i + v_j$.

Lemma 3.2 *Let $G{:}N_m \times N_n \rightarrow \mathbb{R}$ then there exists the vectors u and v such that $g_{i,j} = u_i + v_j$ is monotone.*

Proof. The matrix $G = \{g_{i,j}\}$ is a Monge matrix. We need to show that

$$\begin{aligned} g_{i_1,j_1} + g_{i_2,j_2} &= u_{i_1} + v_{j_1} + u_{i_2} + v_{j_2} \\ &= u_{i_1} + v_{j_2} + u_{i_2} + v_{j_1} \\ &= g_{i_1,j_2} + g_{i_2,j_1} \end{aligned}$$

We consider the first two columns $g_{1,j}$ and $g_{2,j}$. If they are not monotone, we add the constant u_1. Then we consider the next two columns, *i.e.* the columns 2 and 3. If they are not monotone, we add the constant u_2.

$$u = u_1 + u_2 + \ldots + u_m$$

We repeat this process to obtain to obtain the vector v. If G is a Monge then $g+u+v$ is also Monge. G is indeed a Monotone matrix. $\square$

Lemma 3.3 *Let $g: N_m \times N_n$ be a Monge and monotone matrix. Let*

$$F_{i,j} = \sum_{k\le i, l\le j} p_{kl} \text{ with } p_{k,l} \;=\; \square g_{k,l}\; k \in N_m\;,\; l \in N_n$$

Then F is a distribution function and

$$\square(F-g)=0$$

Proof. Given $p_{k,l} = \square g_{k,l}$, $\square g_{k,l}$ is defined as

$$\square g_{k,l} = g_{k,l} - g_{k,l-1} - g_{k-1,l} + g_{k-1,l-1}$$

Since $G = \{g_{k,l}\}$ is a Monge matrix, then $\square g_{k,l}$ is negative. Thus $p_{k,l}$ is negative $\forall\; k,l$. Thus $F_{i,j} = \sum_{k\le i,l\le j} p_{kl}$ is a distribution function. $\square$

Theorem 3.2 *Let $g: Nm \times Nn$ be a Monge. Then there exists the distribution array $F: Nm \times Nn$ and the vectors u and v such that*

$$g_{i,j} = u_i + v_j + F_{i,j}$$

Proof. We are given the array F as a distribution array. Then, using Lemma 3 we have that

$$\begin{aligned} F_{i,j} = \sum_{k\le i,l\le j} p_{kl} &= \sum_{k\le i,l\le j} \square g_{k,l} \\ F_{i,j} &= \sum_{k\le i,l\le j} \square g_{k,l} \end{aligned}$$

$\sum_{k\le i,l\le j} \square g_{k,l}$ can be written as

$$\begin{aligned} F_{i,j} = g_{i,j} - g_{i,1} - g_{1,j} + g_{1,1} \\ g_{i,j} = u_i + v_j + F_{i,j} \end{aligned}$$

Thus we are done. $\square$

The paper [BP96] shows that the Monge characterization is a natural generalization of the results of Gilmore and Gomory. Let us consider the Monge array of the form

$$w(i,j) = \begin{cases} \int_{\alpha_i}^{\beta_j} f(y)\,dy & \alpha_i \leq \beta_j \\ \int_{\beta_j}^{\alpha_i} g(y)\,dy & \beta_j < \alpha_i \end{cases} \tag{3.12}$$

where f and g are two non-negative integrable functions, $\alpha_0 \leq \alpha_1 \leq \ldots \alpha_n$, and $\beta_0 \leq \beta_1 \leq \ldots \beta_n$. Let us consider F and G to be the primitives of the functions f and g. Then the above equation can be written as

$$\begin{aligned}
w_{i,j} &= \max(0, F(\beta_j) - F(\alpha_i)) + \max(0, G(\alpha_i) - G(\beta_j)) \\
w_{i,j} &= -G(\beta_j) - F(\alpha_i) + \max(F(\beta_j), F(\alpha_i)) + max(G(\alpha_i), G(\beta_j)) \\
\Box w_{i,j} &= w_{i,j} - w_{i,j-1} - w_{i-1,j} + w_{i-1,j-1} \\
\Box w_{i,j} &= -G(\beta_j) - F(\alpha_i) + \max(F(\beta_j), F(\alpha_i)) + \max(G(\alpha_i), G(\beta_j)) \\
&- (-G(\beta_{j-1}) - F(\alpha_i) + \max(F(\beta_{j-1}), F(\alpha_i)) + \max(G(\alpha_i), G(\beta_{j-1}))) \\
&- (-G(\beta_j) - F(\alpha_{i-1}) + \max(F(\beta_j), F(\alpha_{i-1})) + \max(G(\alpha_{i-1}), G(\beta_j)) \\
&+ (-G(\beta_{j-1}) - F(\alpha_{i-1}) + \max(F(\beta_{j-1}), F(\alpha_{i-1})) + \max(G(\alpha_{i-1}), G(\beta_{j-1})))
\end{aligned}$$

Thus

$$\Box w_{i,j} = \Box \max(F(\beta_j), F(\alpha_i)) + \Box \max(G(\alpha_i), G(\beta_j))$$

The value of $\Box w_{i,j}$ will be always positive. Thus F can be defined as

$$p_{i,j} = -\Box w_{i,j}$$

This shows that characterization of Monge matrix is a proper generalization of the Gilmore-Gomory result.

CHAPTER 4

SMAWK ALGORITHM ON A TOTALLY MONOTONE MATRIX

In this chapter we present one method of calculating the row minima of a totally monotone matrix using the SMAWK algorithm in linear time, which is an off-line algorithm. The SMAWK algorithm is one method available for calculating the row minima of a totally monotone matrix.

The paper by Aggarwal [AKM+87] gives us a linear time algorithm to compute all row minima of a totally monotone $n \times m$ matrix, when $m \geq n$. The algorithm is called the *SMAWK* algorithm. Since all Monge matrices are totally monotone, the SMAWK algorithm computes all row minima of a Monge matrix in $O(m)$ time. Let C be a Monge matrix and $j(i)$ represent the leftmost column index such that $C(i, j(i))$ contains the minimum value in row i of C. The main component of the SMAWK algorithm is the subroutine REDUCE. The subroutine REDUCE deletes the 'dead' columns of the matrix C:

> The element $C(i, j)$ is called *dead* if $j \neq j(i)$. A column is called *dead* if all its elements are dead.

Lemma 4.1 *Let C be a totally monotone $n \times m$ matrix and let $1 \leq j_1 < j_2 \leq m$. If $C(r, j_1) \leq C(r, j_2)$ then all the entries $C(i, j_2) : 1 \leq i \leq r$ are dead. If $C(r, j_1) > C(r, j_2)$ then the entries in $C(i, j_1) : r \leq i \leq n$ are dead.*

Proof. First part of the lemma, $C(r, j_1) \leq C(r, j_2)$ shows that the minimum element of the rows 1 to r has never occurred in the column j_2 since C is a totally monotone matrix. Similarly, if $C(r, j_1) > C(r, j_2)$ then $C(r+1, j_1) > C(r+1, j_2)$ $\forall$ i where $r \leq i \leq n$. □

The main ideas of the subroutine REDUCE is as follows:

If the input to the subroutine REDUCE is a totally monotone $n \times m$ matrix C then the value returned by REDUCE is a $n \times n$ sub-matrix of C, called X. The subroutine reduces the number of columns by deleting only the dead columns. Thus for $1 \leq i \leq n$, the resulting sub-matrix X contains the columns $C^{j(i)}$.

Let k be the index of the sub-matrix X.

The subroutine $REDUCE(C)$ is defined as follows:

```
$X \leftarrow C; k \leftarrow 1$
 while $X$ as more than n  columns do
   case
   $X(k,k) \leq X(k,k+1)$ and $k < n : k \leftarrow k+1$.
   $X(k,k) \leq X(k,k+1)$ and $k = n :$ Delete column $X^{k+1}$.
   $X(k,k) > X(k,k+1) :$ Delete column $X^k$; If $k > 1$ then $k \leftarrow k-1$.
   endcase
return(X)
```

Initially the value of k is set to 1. Then we can have any of the following cases:

- If $X(k,k) \leq X(k,k+1)$ then, by Lemma 4.1, all the elements of X^{k+1} in rows 1 through k are dead. If $k \neq n$, then we increase the index of k by one.

- If $k = n$ then all elements of X^{k+1} are dead, the subroutine REDUCE removes that column and the index of k is remains unchanged.

- If $X(k,k) > X(k,k+1)$ then, by Lemma 4.1, all the elements of X^k in rows k through n are dead. Since the elements of X^k in rows 1 through $k-1$ are dead, then X^k is dead. The subroutine REDUCE deletes an entire column when all the elements of that column are dead. If $k > 1$ then the index of C is decreased by one, else the index value k remains the same.

Theorem 4.1 *In $O(m)$ comparisons, the subroutine REDUCE reduces the maximum problem for an $n \times m$ totally monotone matrix to an $n \times n$ totally monotone matrix.*

Proof. Let a, b, and c denote the number of times the first, the second, and the third branches of the CASE statement are executed. The subroutine REDUCE will convert an $n \times m$ matrix to an $n \times n$ matrix. The number of columns deleted is $m - n$. In the second and third cases, the columns are deleted. Thus $b + c = m - n$.

The index value is changed in the first and third cases only. In the first case, the index value is increased by one and in the third case, it is decreased by one or is unchanged. The index value starts with value 1 and it cannot go beyond n. Thus $a - c \leq$ the net increase in the index $\leq n - 1$. Combining these two inequalities, the total time taken by the subroutine REDUCE to execute is

$$\begin{aligned} t = a + b + c &\leq a + 2b + c \\ &\leq a - c + 2b + 2c \\ &\leq 2m - n - 1 \end{aligned}$$

Thus we are done. □

The subroutine $MINCOMPUTE(C)$ is defined as follows:

```
$B \leftarrow REDUCE(C)$
if n=1 then output the minimum and return
$X \leftarrow B[2, 4, \ldots 2\lfloor n/2 \rfloor;\ 1, 2, \ldots, n]$
$MINCOMPUTE(X)$
from the known positions of the minima in the even rows of B,
find the minima in its odd rows
end
```

Theorem 4.2 *When $n \leq m$, the subroutine MINCOMPUTE solves the minimum problem on a totally monotone $n \times m$ matrix in $O(m)$.*

Proof. Let $t(n, m)$ denotes the time taken by the subroutine MINCOMPUTE to solve the minimum problem for a totally monotone matrix. We recall that the subroutine REDUCE takes $O(m)$ time. The assignment of the even number of rows of B to X takes $O(n)$ time and the recursive call to subroutine MINCOMPUTE takes $t(n/2, n)$ time. Thus the total time taken by MINCOMPUTE is

$$t(n, m) \leq t(n/2, n) + C_1 n + C_2 m$$

Since $m \geq n$, the total time taken by a MINCOMPUTE is $O(m)$.

Thus we are done. □

CHAPTER 5

APPLICATIONS OF THE MONGE MATRICES

In this chapter we present various applications of Monge matrices. Monge matrices find their use in solving fundamental combinatorial optimization problems such as traveling salesman problem, assignment problem, batching problem, a higher dimensional transportation problem, scheduling problems, etc.. We also present two special cases of the Gilmore-Gomory matching problem.

5.1 Traveling Salesman Problem

The Monge property plays an important role in solving the Traveling salesman problem (TSP). TSP can be stated as follows: Given a list of cities and pairwise distances among them, the problem is to find the shortest possible tour that visits all the cities exactly once and then returns to the starting city. TSP belongs to the set of NP-hard problems. Thus there is no efficient way of solving TSP for very large value of n where n is the number of cities. If the number of cities is n then number of different possible tours is exponential, namely $(n-1)!$. In the symmetric TSP, where the cost of traveling does not depends on the direction then the number of different possible tours is still exponential, namely $((n-1)!/2)$.

There are several special cases of TSP that can be solved in polynomial time. The paper by Gilmore [GLS85] presents a survey of special cases of TSP that can be solved efficiently. One of the solvable special cases of TSP is when the distance matrix of a traveling sales man problem fulfills the Monge property; then the traveling sales man problem can be solved in linear time and the tour on cities is pyramidal:

Let n be the number of cities. If the tour τ is pyramidal then it is of the form $(1, i_1, \ldots\ldots, i_r, n, j_1, \ldots\ldots, j_{n-r-2})$ where $i_1 < i_2 < \ldots < i_r$ and $j_1 > j_2 > \ldots > j_{n-r-2}$ or, for each city j, $1 < j < n$ either $\tau^-1(j) \leq j \leq \tau(j)$ or $\tau^-1(j) \geq j \geq \tau(j)$.

In other words, the salesman starts from the city 1, visits some cities in increasing order, reaches the city n and returns to city 1 by visiting the remaining cities in decreasing order. If the number of cities is 4 then a tour $(1,3,4,2)$ is pyramidal whereas a tour $(1,3,2,4)$ is not pyramidal.

When a distance matrix is given, the shortest pyramidal tour can be computed using the dynamic programming scheme. Let $E(i,j)$ denote the cost of a minimum cost pyramidal path from the vertex i to j that visits all the cities from $1,2\ldots\max(i,j)$. If the path P is pyramidal then it can be decomposed into two sub-paths P_1 and P_2 such that one is monotonically increasing and the other is monotonically decreasing. $E(i,j)$ is defined as

$$E(i,j)=\begin{cases} E(i,j-1)+c_{j-1,j} & \text{for } i\le j-1\\ \min_{k<i}\{E(i,k)+c_{k,j}\} & \text{for } i=j-1\\ E(i-1,j)+c_{i,i-1} & \text{for } i\ge j+1\\ \min_{k<j}\{E(k,j)+c_{i,k}\} & \text{for } i=j+1\end{cases} \tag{5.1}$$

The length of the shortest pyramidal tour is given by

$$\min\{E(n-1,n)+c_{n,n-1},E(n,n-1)+c_{n-1,n}\}$$

Using the above equation, the shortest pyramidal tour can be computed in $O(n^2)$ time starting from the initial condition $E(1,2)=c_{1,2}$ and $E(2,1)=c_{2,1}$

If C is a Monge matrix then

$$c_{i_1,j_1}+c_{i_2,j_2} \le c_{i_1,j_2}+c_{i_2,j_1} \tag{5.2}$$

$\forall\ i_1<i_2,\ j_1<j_2$

The following theorem relates pyramidal tours and Monge matrices.

Theorem 5.1 *If C is a Monge Matrix then there exists an optimal tour that is pyramidal.*

Proof. The induction method is used to prove the above result.

We have the following cases:

a) If $n=2$, the number of possible tours is 1. Thus the above theorem is trivially true for $n=2$.

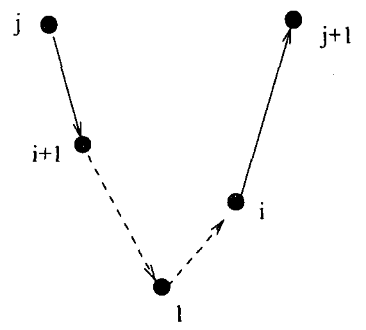

Figure 5.1: Shortest pyramidal path

b) Assume that the theorem is true for $n-1$, *i.e.* we can have an optimal tour of $n-1$ cities that is pyramidal.

c) There are n cities to travel. Let τ be the optimal tour and τ contains a city j_1, where $j_1 \neq n$ and $\tau^-1(j_1) < j_1 > \tau(j_1)$. Thus τ is not a pyramidal tour.

Let $i_1 = j_1$, $i_2 = tau^-1(j_1)$ and $j_2 = \tau(j_1)$. Since C is a Monge array, we can apply Equation (5.2) and the result is $c(\tau) \geq c(\tau')$. The new permutation τ' contains two subtours, one containing j_1 only and the other containing the other $n-1$ tours. From Case (b) we know that there is an optimal tour of $n-1$ cities that is pyramidal. Now we patch j_1 into τ'.

There is an i_1 such that $i_1 < j_1 < \tau(i_1)$. When combining Equation (5.2) with $i_2 = j_1$ and $j_2 = \tau(i_1)$, we will obtain another permutation τ'' where $c(\tau'') \leq c(\tau)$. The tour τ'' is a pyramidal tour for n cities. □

The paper [Par91] presents an $O(n)$ time algorithm for any n-vertex TSP whose cost array satisfies the Monge property. The paper has improved the running time of the earlier mentioned dynamic programming scheme to $O(n)$. To show this, let $F(j)$ denote $E(j, j+1)$, the cost of a minimum cost pyramidal path from vertex j to $j+1$. Let $G(j)$ denote $E(j+1, j)$, the cost of a minimum cost pyramidal path from vertex $j+1$ to j. Thus $F(1) = E(1,2) = c[1,2]$ and $G(1) = E(2,1) = c[2,1]$. The cost of a minimum cost pyramidal tour is

$$\min\{F(n-1,n) + c[n,n-1], G(n,n-1) + c[n-1,n]\}$$

Let P be the shortest pyramidal path from j to $j+1$ as shown in Figure 5.1. Since this is a shortest pyramidal path, it achieves $F(j)$. Let $(i, j+1)$ is the last arc traversed by P. Since P is a pyramidal path, it can be decomposed into two paths, where the labels on one path are monotonically increasing and labels on other path are monotonically decreasing. While considering $F(j)$, the labels on the path from j to 1 are monotonically decreasing thus $i+1 < j$. If $i+1 < j$ then the first $j-(i+1)$ arcs traversed by P must be $(j, j-1), (j-1, j-2), \ldots, (i+2, i+1)$ Thus the minimum cost pyramidal path from j to $j+1$ is

$$F(j) = \min_{1 \le i < j} \ \{ \ G(i) + c[i, j+1] + \sum_{l=i+1}^{j-1} c[l+1, l] \ \} \tag{5.3}$$

Similarly, the minimum cost pyramidal path from $j+1$ to j is

$$G(j) = \min_{1 \le i < j} \ \{ \ F(i) + c[j+1, i] + \sum_{l=i+1}^{j-1} c[l, l+1] \ \} \tag{5.4}$$

If the algorithm computes all the values for $F(j)$ and $G(j)$ from 1 to n then the computation takes $O(n^2)$ time. The paper [Par91] uses an online array searching technique to reduce the running time to $O(n)$. To improve the running time of the algorithm, we consider two $(n-1) \times (n-1)$ arrays, $A = \{a[i,j]\}$ and $B = \{b[i,j]\}$ where

$$a[i,j] = \begin{cases} G(i) + c[i, j+1] + \sum_{l=i+1}^{j-1} c[l+1, l] \text{ if } i < j \\ +\infty \text{ if } i \ge j \end{cases} \tag{5.5}$$

$$b[i,j] = \begin{cases} F(i) + c[j+1, i] + \sum_{l=i+1}^{j-1} c[l, l+1] \text{ if } i < j \\ +\infty \text{ if } i \ge j \end{cases} \tag{5.6}$$

From Equation (5.3), it is obvious that

$$F(j) = \min_{1 \le i < j} a[i,j]$$

This shows that minimum cost pyramidal path from j to $j+1$ is the minimum value in the j^{th} column of A. Similarly, from Equation (5.4)

$$G(j) = \min_{1 \le i < j} b[i,j]$$

The above equation shows that $G(j)$ is the j^{th} column minimum of B.

Lemma 5.1 *Both A and B are Monge matrices if and only if C is a Monge matrix.*

Proof. We show fist that A is a Monge matrix if and only if C is a Monge matrix.

A is Monge if and only if

$$a[i,j] + a[i+1,j+1] \leq a[i,j+1] + a[i+1,j]$$

$\forall 1 \leq i < n-1, 1 \leq j < n-1$. If $i+1 \geq j$ then $a[i+1,j] = +\infty$, the above Monge property is satisfied. By substituting the value of $a[i,j]$ we obtain that

$$\begin{aligned}
G(i) + c[i,j+1] + \sum_{l=i+1}^{j-1} c[l+1,l] &\;+ \\
G(i+1) + c[i+1,j+2] + \sum_{l=i+2}^{j} c[l+1,l] &\;\leq\; G(i) + c[i,j+2] + \sum_{l=i+1}^{j} c[l+1,l] \\
&\;+\; G(i+1) + c[i+1,j+1] + \sum_{l=i+2}^{j-1} c[l+1,l]
\end{aligned}$$

We cancel the values $G(i)$ and $G(i+1)$ and we obtain that

$$\begin{aligned}
c[i,j+1] + \sum_{l=i+1}^{j-1} c[l+1,l] &\;+ \\
c[i+1,j+2] + \sum_{l=i+2}^{j} c[l+1,l] &\;\leq \\
& c[i,j+2] + \sum_{l=i+1}^{j} c[l+1,l] \\
& +c[i+1,j+1] + \sum_{l=i+2}^{j-1} c[l+1,l] \\
c[i,j+1] + c[i+1,j+2] + c[j+1,j] &\;\leq\; c[i,j+2] + c[i+1,j+1] + c[j+1,j] \\
c[i,j+1] + c[i+1,j+2] &\;\leq\; c[i,j+2] + c[i+1,j+1]
\end{aligned}$$

This shows that A is Monge if and only if C is Monge.

We can show that B is a Monge matrix if and only if C is a Monge matrix in a similar way.

□

We pre-compute the value of $\sum_{l=1}^{j-1} c[l+1,l], \sum_{l=1}^{j-1} c[l,l+1] \; \forall \; 2 \leq j < n$. This computation takes $O(n)$ time. Once the preprocessing is done, any entry $a[i,j]$ of A can be computed in constant time from $G(i)$, the i^{th} column minimum of B and any entry of B can be computed in constant time from $F(i)$ which is the i^{th} column minimum of A. Thus by interleaving the computation of column minima of A with the computation of column minima of B and by using online array searching algorithm of either Larmore and Schieber [LS91] or Eppstein [D.E90] the values $F(2), F(3), \ldots, F(n-1)$ and $G(2), G(3), \ldots G(n-1)$ can be computed in $O(n)$ time. Minimum cost pyramidal tour can be found once the values of $F(n-1)$ and $G(n-1)$ are calculated. Thus the minimum cost traveling salesman tour through a graph G can be computed in $O(n)$ time when the cost matrix satisfies the Monge property.

5.2 Assignment Problem

Monge matrices play an important role in solving the assignment problem. The first algorithm for the assignment problem was given by Kuhn [Kuh04] and it solves the assignment problem in $O(n^2m)$ steps. Brucker [Bru04] gives a special case of an assignment problem that has a very simple solution. He considered the case where cost array satisfies the Monge property. He shows that if the cost array satisfies the Monge property then optimal solution for the assignment problem will have a fixed structure.

The assignment problem can be stated as follows: Given a bipartite graph $G = (V_1 \cup V_2, V_1 \times V_2)$ with $V_1 = (v_1, v_2, \ldots v_n)$, $V_2 = (w_1, w_2 \ldots w_m), n \leq m$, and $c_{i,j}$ that represent the cost associated with arc $v_i w_j$, the objective is to find the matrix $X = (x_{i,j})$ so as to

$$\text{minimize} \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

s.t

$$\begin{aligned} \sum_{j=1}^{m} x_{i,j} &= 1 \\ \sum_{i=1}^{n} x_{i,j} &\leq 1. \text{ since } n \leq m \\ x_{i,j} &\in \{0,1\}, \; i = \{1,2,\ldots n\}, j = \{1,2,\ldots m\} \end{aligned}$$

$x_{i,j} = 1$ if v_i is assigned to w_j else the value of $x_{i,j}$ is zero. Each element in V_1 is assigned to a unique element in V_2.

Brucker [Bru04] gives the following theorem which relates the Monge array and the assignment problem.

Theorem 5.2 *Given an assignment problem with $n \leq m$. Let $c_{i,j}$ be*

a) a Monge Array

$$c_{i,k} + c_{j,l} \leq c_{i,l} + c_{j,k}$$

b) monotone

$$c_{i,j} \leq c_{i,l} \ \forall \ j \leq l$$

then

$$x(i,j) = \begin{cases} 1 & i = j \\ 0 & \textit{otherwise} \end{cases} \tag{5.7}$$

is an optimal solution.

Proof.

Let y be an optimal solution with $y_{vv} = 1 \ \forall \ v = 1, 2 \ldots i$ where i is as large as possible, $i < n$. Thus $y_{i+1,i+1} = 0$ and there exists an l, $l > i + 1$ and $y_{i+1,l} = 1$.

We have the following cases:

Case 1) There exists an index j such that $y_{j,i+1} = 1$ (*i.e.* $i + 1$ is matched).

Since C is a Monge array then

$$c_{i+1,i+1} + c_{j,l} \leq c_{i+1,l} + c_{j,i+1}$$

Thus switching the two assignments will also give an optimal solution, contradicting the maximality of i.

Case 2) $y_{l,i+1} = 0 \ \forall \ l > i + 1$

Since $c_{i,j} \leq c_{i,k} \ \forall \ j < k$, changing the assignment from $y_{i+1,l}$ *to* $y_{i+1,i+1}$ will give an optimal solution, contradicting the maximality of i.

Thus we are done. □

If $n = m$ then only Case 1 will exist since all $w_j \in V_2$ have to be matched.

5.3 Special Case of Gilmore-Gomory Matching Problem

Let $G = (S, T, S \times T)$ be a complete bipartite graph with $|S| = |T| = n$. Each node $i \in S$ has associated with a real number α_i and each node $j \in T$ has associated with a real number β_j. For all i,j the weight of the arc (i, j) is given as

$$w(i,j) = \begin{cases} \int_{\alpha_i}^{\beta_j} f(y)\, dy & \alpha_i \leq \beta_j \\ \int_{\beta_j}^{\alpha_i} g(y)\, dy & \beta_j < \alpha_i \end{cases} \tag{5.8}$$

where $f(y)$ and $g(y)$ are given integrable functions. A matching problem with arc weights given as an integral of some function is referred to as Gilmore-Gomory matching problem.

Consider following matching problems:

a) Skies and Skiers:

A ski instructor has n pairs of skies to be assigned to n skiers. The objective of the problem is to assign skies to skiers such that the sum of the difference between ski length and height of the skier is minimized.

The solution to this problem is to assign the shortest pair of skies to the shortest skier. This problem can be represented as a special case of Gilmore-Gomory matching problem. Let α_i denote the length of the i^{th} pair skies, β_j denotes the height of j^{th} skier. We also have that $\alpha_1 \leq \alpha_2 \ldots \leq \alpha_n$, $\beta_1 \leq \beta_2 \ldots \leq \beta_n$. The integrable functions $f(y)$ and $g(y)$ are set as 1, and the weight of the arc (i, j) is defined as

$$w_{i,j} = |\alpha_i - \beta_j|$$

In this case a direct assignment will give the minimum value for sum of difference between ski length and height of skier, *i.e.* $X = \{(i, i)|i = \{1, 2 \ldots n\}\}$ is a minimum weight complete matching problem.

b) School Busing

A bus company has n morning runs and n afternoon runs to assign to n drivers. If a driver is given a morning runs and an afternoon run whose total duration exceeds T hours then he/she has to be paid a premium on the overtime hours. The objective from the management point of view is to match the morning runs with afternoon runs so as to minimize the total number of overtime hours.

The solution for this problem is to assign the k^{th} smallest run in the morning with the k^{th} longest run in the evening. This problem can be represented as a special case of Gilmore-Gomory matching problem. Let a_i and b_j denote the length of the i^{th} morning run and j^{th} evening run. Let $\alpha_i = T - a_i$ and $\beta_j = b_j$. The integrable functions $f(y) = 1$ and $g(y) = 0$. The weight of the arc (i, j) is defined as

$$w_{i,j} = \max\{0, \alpha_i + \beta_j\}$$

which represents the overtime hours. The management wants to minimize the total overtime hours. A direct assignment gives an optimal solution to this problem.

The bus drivers' union has a different objective. Their objective is to maximize the minimum number of hours worked by any driver. Let $\alpha_i = -a_i$, $\beta_j = b_j$ and the integrable functions $f(y) = g(y) = 1$. Thus the weight of the arc (i, j) is defined as

$$w_{i,j} = \alpha_i + \beta_j$$

The objective is to maximize the minimum of the weight. $X = \{(i,i)|i = \{1, 2 \ldots n\}\}$ is a min-max optimal complete matching if $f(y) \geq 0,\ g(y) \geq 0\ \forall$ y.

5.4 Batching Problems

A batching problem can be formulated as follows: Given n jobs, J_i, $i = \{1, 2, \ldots, n\}$ with their corresponding processing times p_i and weights w_i, $i = \{1, 2, \ldots n\}$, the jobs are scheduled in batches. A *batch* is a set of jobs which are processed jointly. The size of the batch is defined as the number of jobs inside that particular batch. The completion time of a job is same as the completion time of the batch in which it resides. All jobs in the same batch will have the same completion time. There is a setup time for each batch which is independent of the batch. Normally the setup time is set to 1.

In the list-batching problem, the order of the jobs will be given and the objective is to batch the jobs such that the value of the objective function is minimized. Albers and Brucker [AB93] show that the problem of finding the optimal batch sequence for a fixed job sequence can be reduced to a shortest path problem. Thus the list-batching problem can be reduced to a shortest path problem and there is a one-to-one correspondence between the shortest path problem and the list-batching problem.

Let the order of the jobs in the list-batching problem to be $J_1, J_2, \dots J_n$. The objective is to find the batch sequence based on the general objective function $F = \sum w_i c_i$ where w_i, c_i represent the weight and completion time of job i. Any solution is of the form as shown in Figure 5.2.

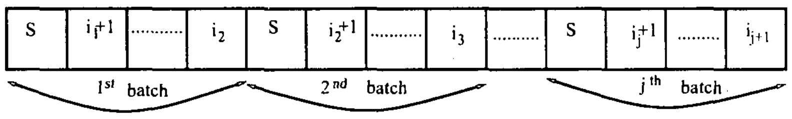

Figure 5.2: Schedule for batching problem

where $i_j + 1$ represents the starting of j^{th} batch.

The processing time of the j^{th} batch equals to

$$P_j = s + \sum_{\gamma=i_j+1}^{i_{j+1}} p_\gamma$$

Thus the value of the schedule is

$$\begin{aligned}
F(S) &= \sum_{i=1}^{n} w_i p_i \\
&= P_1(\sum_{\gamma=i_1+1}^{i_2} w_\gamma) + \dots + (P_1 + \dots + P_k)(\sum_{\gamma=i_k+1}^{i_{k+1}} w_\gamma) \\
&= \sum_{i=1}^{k}(\sum_{\gamma=i_j+1}^{n} w_\gamma) P_j \\
&= \sum_{i=1}^{k}(\sum_{\gamma=i_j+1}^{n} w_i)(s + \sum_{\gamma=i_j+1}^{i_{j+1}} p_\gamma)
\end{aligned}$$

To reduce the list-batching problem to the shortest path problem we construct a weighted acyclic directed graph as shown in Figure 5.3, where the jobs represent the nodes. We add a dummy node 0 to the graph. There is an edge (i, j) if $i < j$.

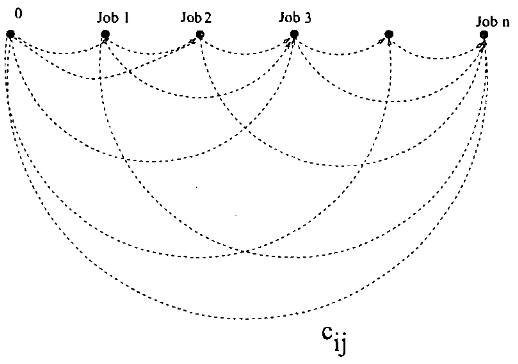

Figure 5.3: Shortest pyramidal path from 0 to n

The edge cost $C_{i,j}$ for $i < j$ is defined as

$$C_{i,j} = (\sum_{\gamma=i+1}^{n} w_\gamma)(s + \sum_{\gamma=i+1}^{j} p_\gamma)$$

The matrix $C_{i,j}$ is a Monge matrix $\forall\ i_1 < i_2$ and $j_1 < j_2$. Let $P_k = \sum_{i=0}^{k} p_i$ and $W_k = \sum_{i=0}^{k} w_i$.

If C is a Monge matrix then

$$\begin{aligned} C_{i_1,j_1} + C_{i_2,j_2} &\leq C_{i_1,j_2} + C_{i_2,j_1} \\ (W_n - W_{i_1})(s + P_{j_1} - P_{i_1}) &+ (W_n - W_{i_2})(s + P_{j_2} - P_{i_2}) \\ &\leq (W_n - W_{i_1})(s + P_{j_2} - P_{i_1}) \\ &+ (W_n - W_{i_2})(s + P_{j_1} - P_{i_2}) \end{aligned} \tag{5.9}$$

We cancel some values in Equation (5.9) and we obtain

$$\begin{aligned} (W_n - W_{i_1})(P_{j_1} - P_{j_2}) &\leq (W_n - W_{i_2})(P_{j_1} - P_{i_2}) \\ (P_{j_1} - P_{j_2})(W_n - W_{i_1} - W_n + W_{i_2}) &\leq 0 \\ (P_{j_1} - P_{j_2})(W_{i_2} - W_{i_1}) &\leq 0 \end{aligned}$$

This is true since $P_{j_1} < P_{j_2}$ and $W_{i_2} > W_{i_1}$.

If there are n jobs then the number of nodes present in the graph G is $n + 1$. They are numbered from 0 to n, where 0 represents the dummy node. The objective of the path problem is to find the shortest path from 0 to n, given all the edge costs.

Any solution of the path problem can be transformed to find a solution for the list-batching problem. The cost of the path $\langle 0, i_1, i_2, \ldots i_k, n\rangle$ gives the $\sum_{i=1}^{n} w_i c_i$ value of the schedule which batches at each job $i_1, i_2, \ldots i_k$. Since there is a one-to-one correspondence between the shortest path problem and the list-batching problem, any batching with cost S corresponds to a path in graph G with the path length S.

For example, consider that we have 5 jobs. Then the number of nodes present in the path problem is 6. Let the shortest path to be $\langle 0, 1, 3, 5\rangle$ as shown in Figure 5.4.

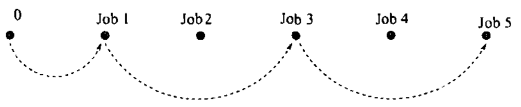

Figure 5.4: shortest path from 0 to 5

Then the solution to the batching problem is shown in Figure 5.5.

S	1	S	2	3	S	4	5

Figure 5.5: Resulting Schedule

$$C_{01} + C_{13} + C_{3,5} = \sum_{i=1}^{5} w_i c_i$$

Albers and Brucker [AB93] give a dynamic programming method that computes the shortest path in $O(n^2)$ time. Let

$$E(l) = \text{cost of the shortest path from 0 to 1}$$

then

$$E(l) = \min_{1 \leq k \leq l} \{E(k) + c_{k,l}\} \text{ with } E(0) = 0$$

This results in a matrix E which is shown in Figure 5.6.

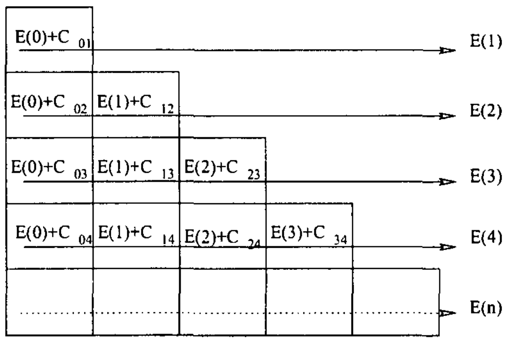

Figure 5.6: Matrix E

The matrix E is defined as follows:

$$E[l,k] = \begin{cases} E[k-1] + c[k-1,l] \; if \; l \geq k \\ \infty \; else \end{cases} \tag{5.10}$$

with $l = 1, 2 \ldots n$ and $k = 1, 2 \ldots n$.

The matrix E is calculated row by row. The protocol for this algorithm is: An element in the column $i+1$ is available once the minimum of i^{th} row is calculated. The matrix E is a Monge matrix since the matrix $C = \{c_{ij}\}$ is a Monge matrix. The row minima of n^{th} row is represented as $E(n)$. Once the minimum of n^{th} row is calculated, the backtracking method is used to find the shortest path. This algorithm takes $O(n^2)$ time to compute all row minima. Another dynamic programming approach to compute all row minima is HIRE/FIRE/RETIRE algorithm.

Hire/Fire/Retire Algorithm This algorithm computes all the row minima in $O(nlog(n))$ time, which is better than the above approach. This algorithm executes row by row. The protocol for this algorithm is: Once the minimum of i^{th} row is known then $(i+1)^{th}$ column is hired. Initially no column is fired as well as retired. The algorithm starts from the first row. After computing the minimum of the first row, the algorithm hires the second column. Then it computes the minimum of second row and hires the third column. Once a column is hired the firing process starts from right to left. A column is fired if none of the elements in that column can be the minimum of any row. Here third and first column will work together to fire the second column. Firing process stops when the newly hired column cannot fire anymore available columns. A column is retired if it cannot contain the minimum value for the upcoming rows. For example if the minimum of 3rd row is in 3rd column then all columns which are before column 3, *i.e.* column 1 and column 2, are retired. Since the matrix is a totally monotone matrix it cannot have the minimum value of rows 4 to n as well as if the value of third column is greater than value of any other column that is currently available then third column is considered as a retired column.

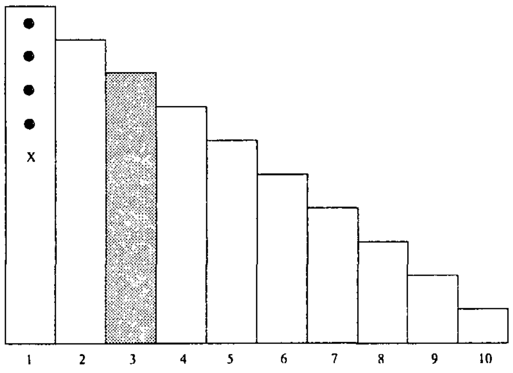

Figure 5.7: Hire/Fire/Retire Algorithm

Consider Figure 5.7, let the minimum value of rows from 1 to 4 are already computed and the column 3 is already fired. Once the minimum of row 4 is known then column 5 becomes available. Column 5 will combine with 2nd available column from left (column 2) to fire the middle column. For firing a column, first do binary search to find the crossover point.

Cross over point is the point where $a_i - b_j > 0$. Since the matrix is a Monge Matrix the difference of adjacent column is non decreasing ie $a_1 - b_1 \leq a_2 - b_2 \leq \ldots a_n - b_n$. If the difference of adjacent column is negative from row 1 to row n then 2nd column can be fired. Similarly if the difference of adjacent column is positive from row 1 to row n then 1st column can be fired. Crossover point can be found using binary search. In Figure 5.8 shaded portion represents the area where minimum cannot be present.

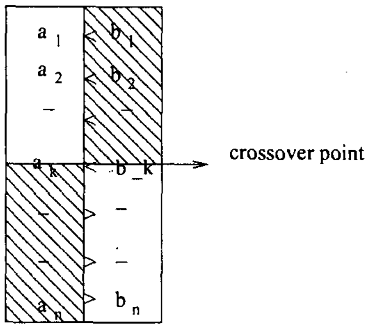

Figure 5.8: Crossover Point

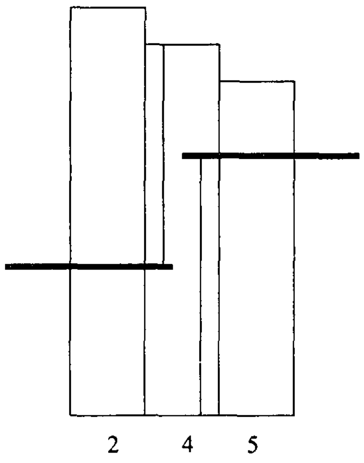

Figure 5.9: Crossover point of Two columns

For firing column 4 first do binary search on column 2 and 4. Find the left crossover point. Then do binary search on column 4 and 5 and find the right crossover point. If the right crossover point is above the left crossover point then the column 4 can be fired. This indicates that minimum of any of the rows cannot be in the 4^{th} column. Figure 5.9 shows the case where right crossover point is above left crossover point. If the right crossover point is below the left crossover point then that particular column cannot be fired.

When no more columns can be fired then the firing process stops. For finding the minimum value in the next row, the value X, as shown in Figure 5.7, is compared with value of next available column from right. If the available column's value is less than the value X then the column containing X will be retired. Then the new column value is compared with the next available column value. This process continues until the new columns value is less than the next available column value. Then the value which is present in that new column value becomes the minimum value in that particular row. Once the minimum value of a row is found then all columns which come before the column index where the minimum value resides will be retired. This process continues until all row minima are calculated.

5.5 d-Dimensional Transportation Problem

The transportation problem can be stated as follows: Given m supply vectors, n demand vectors and an $m \times n$ cost matrix, the objective is to satisfy all the demands, so that the total cost of meeting the demand, given the supply constraints, is minimized(as shown in Figure 5.10).

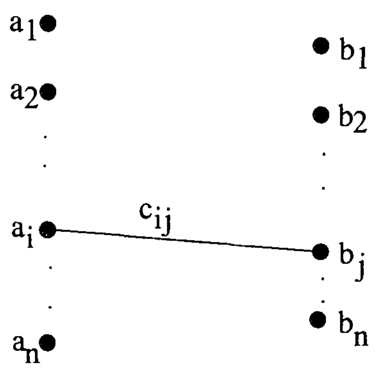

Figure 5.10: Transportation Problem

The fastest algorithm for the general transportation problem given by Orlin [Orl88] runs in $O(mn^2 \lg n + n^2 \lg^2 n)$ time. In 1963, Hoffman gave the necessary conditions under which the family of greedy algorithms solve the two-dimensional transportation problem in $O(mn)$ time. He proved that, if the problems cost array satisfies the Monge property then the northwest-corner rule greedy algorithm solves the two-dimensional transportation problem in $O(m + n)$ time.

The paper [BBPP95] gives the condition under which the northwest-corner rule greedy algorithm solves the d-dimensional transportation problem. The paper shows that if the problems cost array satisfies the d-dimensional Monge property then the northwest corner rule greedy algorithm solves an instance of the d-dimensional transportation problem.

In a mathematical way, the transportation problem can be stated as follows: Given an m-entry supply vector $A = \{a[i]\}$, an n-entry demand vector B=$b[j]$, and an $m \times n$ cost matrix $C = \{c[i,j]\}$ such that all entries of A and B are positive and $\sum_i a[i] = \sum_j b[j]$, the objective is to find the variable array $X = \{x[i][j]\}$ in order to

$$\text{minimize} \sum_{i=1}^{m} \sum_{j=1}^{n} c[i,j]x[i,j]$$

s.t

$$\begin{aligned} \sum_{j=1}^{n} x[I,j] &= a[I] \text{ for } 1 \leq I \leq m \\ \sum_{i=1}^{m} x[i,J] &= b[J] \text{ for } 1 \leq J \leq n \\ x[i,j] &\geq 0,\ i = \{1,2,\ldots m\}, j = \{1,2,\ldots n\} \end{aligned}$$

If the cost matrix is a Monge matrix then the northwest-corner rule greedy algorithm solves the two-dimensional transportation problem in $O(m+n)$ time.

The d-dimensional transportation problem can be stated as follows: Given d-supply demand vectors $A_1, A_2 \ldots A_d$ where the k^{th} vector $A_k = \{a_k[i]\}$ has n_k entries and an $n_1 \times n_2 \times \ldots \times n_d$ cost array $C = \{c[i_1, i_2, \ldots i_d]\}$ such that the entries of $A_1, A_2 \ldots A_d$ are positive and $\sum_i a_1[i] = \sum_i a_2[i] = \ldots = \sum_i a_d[i]$, the objective is to find the variable array $X = \{x[i_1, i_2 \ldots i_d]\}$ in order to

$$\text{minimize} \sum_{i_1, i_2 \ldots i_d} c[i_1, i_2 \ldots i_d] x[i_1, i_2 \ldots i_d]$$

s.t

$$\begin{aligned} \sum_{i_1, i_2 \ldots i_d s.t i_k = I} x[i_1, i_2, \ldots i_d] &= a_k[I] \text{ for } 1 \leq k \leq d \text{ and } 1 \leq I \leq n_k \\ x[i_1, i_2 \ldots i_d] &\geq 0 \ \forall\ i_1, i_2 \ldots i_d \end{aligned}$$

A d-dimensional matrix is called as a *Monge matrix* if it satisfies d-dimensional Monge property. The paper [BBPP95] gives some of the properties of d-dimensional Monge matrices.

Definition 5.1 (**d-dimensional Distribution Array**) *For any $d \geq 2$, an $n_1 \times n_2 \times \ldots n_d$ d-dimensional array $C = c[i_1, i_2 \ldots i_d]$ is called a* distribution array *if*

$$c[i_1, i_2 \ldots i_d] \;=\; \sum_{l_1, l_2 \ldots l_d} p[l_1, l_2 \ldots l_d]$$

where $p[l_1, l_2 \ldots l_d] \leq 0$, $\forall\ l_1, l_2 \ldots l_d$.

Definition 5.2 (**d-dimensional Monotone Matrix**) *For any $d \geq 2$, let $C = c[i_1, i_2 \ldots i_d]$ denote a d-dimensional array and let $f_2(I_1), f_3(I_1) \ldots f_d(I_1)$ denote the 2^{nd} through d^{th} coordinates of the minimum entry in the $(d-1)$-dimensional plane consisting of those entries whose first coordinates is I_1 so that*

$$c[I_1, f_2(I_1), f_3(I_1) \ldots f_d(I_1)]$$

If C is a Monotone matrix then $\forall\ I_1 < J_1$, $f_k(I_1) \leq f_k(J_1)$ for all k, $2 \leq k \leq d$.

The paper [BBPP95] presents a northwest-corner rule greedy algorithm for the d-dimensional transportation problem. The algorithm works as follows:

1. First we set $x[1, 1, \ldots, 1] = \min\{a_1[1], a_2[1], \ldots, a_d[1]\}$.
2. Then we reduce each of $a_1[1], a_2[1], \ldots, a_n[1]$ by $\min\{a_1[1], a_2[1], \ldots, a_d[1]\}$. At least one of $a_1[1], a_2[1], \ldots, a_d[1]$ is reduced to 0, *i.e.* one of the problem's dimensions $n_1, n_2, \ldots, n_d$ has been reduced by 1.
3. The smaller transportation problem obtained this way is solved recursively.

The running time of the algorithm is $O(d(n_1 + n_2 + \ldots + n_d))$. Each variable assignment takes $O(d)$ time and at each iteration, at least one of the problem's dimension has been reduced by 1.

5.6 A Scheduling Problem - $P||\sum C_i$

The scheduling problem $P||\sum C_i$ can be stated as follows: Given n jobs with the corresponding processing times and m identical parallel machines, the objective is to schedule the jobs such that sum of completion time of jobs or the mean flow time is minimized. There are no precedence constraints between the jobs.

The scheduling problem $P||\sum C_i$ can be transformed to an assignment problem by doing the following steps:

1. First we sort the jobs based on the processing time such that $p_1 \geq p_2 \geq ...p_n$.

2. Then we partition the set of jobs into into m disjoint sets $I_1, I_1, \ldots, I_m$. The jobs in I_j are scheduled on machine m_j starting at time 0 without any idle time. We arrange the jobs on the left-hand side and machines on the right-hand side.

The following corollary derived from the theorem given by Brucker [Bru04] helps to solve the scheduling problem efficiently:

Corollary 5.6.1 *Let a_i and b_j be arrays of real numbers $a_1 \geq a_2 \geq \ldots a_n$ and $b_1 \leq b_2 \leq \ldots b_n$ where $n \leq m$. Then the assignment problem given by the array*

$$C = c_{i,j} \text{ with } c_{i,j} = a_i * b_j$$

has an optimal solution given by

$$x_{i,j} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases} \tag{5.11}$$

Since the array $c_{i,j} = a_i * b_j$, where $a_1 \geq a_2 \geq \ldots a_n$ and $b_1 \leq b_2 \leq \ldots b_n$, is a Monge matrix, the direct assignment gives the minimum value for the mean flow time.

We consider the scheduling problem $P||\sum C_i$ where the number of jobs given is 7 and the number of machines given is 3. To transform the scheduling problem into an assignment problem, we first sort the jobs based on their processing time such that $p_1 \geq p_2 \geq ....p_7$. Then we arrange the jobs as in Figure 5.11.

Let the schedule be as shown in Figure 5.12

The completion time of Job 7 is $P_4 + P_3 + P_7$.

The completion time of Job 3 is $P_4 + P_3$.

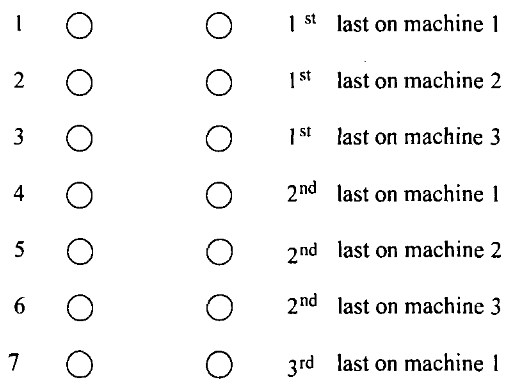

Figure 5.11: Scheduling Problem

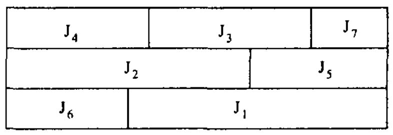

Figure 5.12: Schedule of the jobs

The completion time of Job 4 is P_4.

In the value of the objective function, the processing time of the 4^{th} job is added 3 times. Thus the cost of assigning a job to a machine depends on whether the job has been processed and on the processing time of that job.

Thus the cost of assigning job i to j^{th} last machine is

$$\begin{aligned} c_{ij} &= p_i * j \\ c_{ij} &= a_i * b_j \end{aligned}$$

The matrix $C = c_{i,j} = a_i * b_j$ where $a_i = p_i$ for $i = 1, 2, \ldots n$ and $b_j = \lceil j/m \rceil$ for $j = 1, 2, \ldots n$ is a Monge matrix as per above corollary. Thus the direct assignment will give the minimum value for $\sum_{i=1} nC_i$.

After sorting the jobs, the direct assignment gives the solution. Thus the scheduling problem $P||\sum C_i$ can be solved in $O(n \log n)$ time.

5.7 Geometric Application-Finding the farthest neighbor

The paper [AKM+87] gives a geometric application of Monge matrices. The paper shows that problem of finding the farthest neighbors of all vertices in a convex polygon can be done in linear time by applying Monge properties. Let the convex polygon be P and the distances are measured in clockwise order, see Figure 5.13. We define a matrix A as follows

$$\text{if } i < j \leq i + n - 1 \text{ then } A(i,j) = d(p_i, p_j).$$

$$\text{if } j \leq i \text{ then } A(i,j) = j - i, \text{ and if } j \geq i + n \text{then } A(i,j) = -1.$$

For example a convex polygon with 5 vertices as shown in Figure 5.13. In the poygon vertices are numbered in clockwise order.

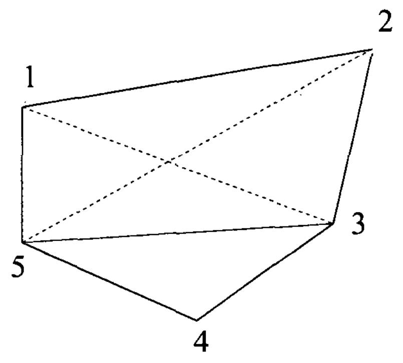

Figure 5.13: Convex Polygon

Then coresponding matrix A is shown in Figure 5.14.

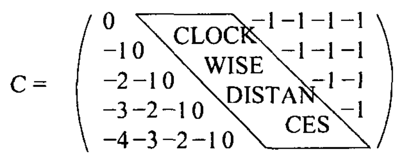

Figure 5.14: Distance Matrix

If a 2×2 submatrix of $A[i, j : k, l]$, has only positive values then $i < j < k < l$ and the vertices p_i, p_j, p_k, and p_l are in clokwise order. From triangular inequality

$$d(p_i, p_k) + d(p_j, p_l) \geq d(p_i, p_l) + d(p_j, p_k)$$

Since the matrix A is an inverse Monge Matrix the maximum problem on A gives the solution to farthest neighbor problem and the problem of finding the maximum value in each row of A can be done in linear time.

CHAPTER 6

CONCLUSION

In this thesis we survey the broad research done on a large number of various Monge properties and find out how these properties play an important role in different applications. Some of the Monge properties which we considered in the thesis are monotonicity property, algebraic Monge property, higher-dimensional Monge property and bottleneck Monge property. In this thesis we show that some of the optimization problems, like the Traveling Salesman Problem which is an NP-hard problem, can be solved in polynomial time when is restricted to a Monge structure. We also show how the Monge arrays can be used to speed up the dynamic programming algorithms. We also discuss several algorithms for computing the row minima in Monge arrays. One of them is SMAWK algorithm, which is an off-line algorithm that computes all row minima of a Monge matrix in linear time. For computing the row minima in linear time, the SMAWK algorithm uses the monotonicity property of the Monge matrices.

We show how the northwest-corner rule greedy algorithm solves the d-dimensional transportation problem when the problem's cost array satisfies the higher dimensional Monge property. Another important application of Monge properties is for the assignment problem. We show that the assignment problem has a fixed structure when the underlying cost matrix satisfies the Monge property. We also discuss how the Monge properties can be applied in some scheduling problems to improve their running time. In this thesis we have considered $1|list-s-batch|\sum C_i$ batching problem and we show that it can be solved by a dynamic programming approach in $O(n^2)$ time. Next we show that hire-fire-retire algorithm solves the above batching problem in $O(n \log n)$ time. Another scheduling problem that we consider in the thesis is $P||\sum C_i$. This problem is similar to an assignment problem and it can be solved in $O(n \log n)$ time. Even though we concentrated mainly on application of

Monge matrix in combinatorial optimization problems, we also mentioned an application of Monge matrix in geometric problems.

Further research on the topic of Monge matrix can be directed in finding other applications of Monge matrix and it variations. One possible area could be the use of Monge matrix in the field of Bioinformatics since Monge properties occurs naturally. Also, investigation on Monge properties to speed up the dynamic programming algorithm can be a good research topic.

There are several papers published on applications which lead to Monge structures. In this thesis we present an overview of different properties and we draw a connection between them.

BIBLIOGRAPHY

[AB93] S. Albers and P. Brucker. The complexity of one-machine batching problem. *Discrete Applied Mathematics*, 47:87–107, 1993.

[AKL$^+$89] M. J. Atallah, S. R. Kosarraju, L. L .Larmore, G. Miller, and S. Teng. Constructing trees in parallel. *1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 421–431, 1989.

[AKM$^+$87] A. Aggarwal, M.M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric Applications of a Matrix-Searching Algorithm. *Algorithmica*, 2:195–208, 1987.

[AP88a] A. Aggarwal and J.K. Park. Parallel searching in multidimensional monotone arrays. Research report RC 14826, IBM T. J. Watson Research Center, Yorktowns Heights, NY,August 1989. Portion of this paper appear in. *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 497–512, 1988.

[AP88b] A. Aggarwal and J.K. Park. Sequential searching in multidimensional monotone arrays. Research report RC 15128, IBM T. J. Watson Research Center, Yorktowns Heights, NY,November 1989. Portion of this paper appear in. *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 497–512, 1988.

[BBH93] W. W. Bein, P. Brucker, and A. J. Hoffman. Series parallel composition of greedy linear programming problems. *Mathematical Programming*, pages 1–14, 1993.

[BBPP95] W.W. Bein, P. Brucker, J.K. Park, and P.K. Pathak. A Monge property for the d-Dimensional Transportation problem. *Discrete Applied Mathematics*, 58:97–109, 1995.

[BP96] W.W. Bein and P.K. Pathak. A Characterization of the Monge property. *Demonstratio Mathematica*, XXIX:451–457, 1996.

[Bru04] P. Brucker. *Scheduling Algorithms.* Springer Verlang, 2004.

[D.E90] D.Eppstein. Sequence comparison with mixed covex and concave cost. *Journal of Algorithms*, 11:85–101, 1990.

[EZGI90] D. Eppstein, R. Giancarlo Z. Galil, and G. F. Italiano. Sparse dynamic programming. *1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 513–522, 1990.

[FP89] M. Fahimi and P. K. Pathak. Applications of a size reduction technique for transportation problem in sampling. *First International Symposium on Optimization and Statistics*, 1989.

[GLS85] P.C. Gilmore, E.L. Lawler, and D.B. Shmoys. Well solved special cases of Traveling Salesman Problem. *Wiley*, 4:87–143, 1985.

[Hof63] A. J. Hoffman. On simple linear programming problems. *V Klee ed Proceedings of Symposia in pure Mathematics*, VII:317–327, 1963.

[Kuh04] H.W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics*, 52:7–21, 2004.

[LS91] L.L. Larmore and B. Schieber. On-line dynamic programming with applications to the prediction of RNA secondary structure. *J. Algorithms*, 12:490–515, 1991.

[Orl88] J. B. Orlin. A faster strongly polynomial minimum cost flow algorithm. *Proceedings of the 20th Annual ACM Symposium on the Theory of Computing*, pages 377–387, 1988.

[Par91] J. K Park. Special case of the n-vertex traveling salesman problem that can be solved in O(n) time. *Information Processing Letters*, 40:247–254, 1991.

[Yao80] F. F. Yao. Efficent dyanamic programming using quadrangle inequalities. *12th Annual ACM Symposium on Theory of Computing*, pages 429–435, 1980.

VITA

Graduate College
University of Nevada, Las Vegas

Swetha Sethumadhavan

Home Address:
6109 S 30th Dr
Phoenix, AZ 85041

Degree:
Bachelor of Information Technology
Cochin University of Science and Technology, India

Thesis Title: A Survey of Monge Properties

Thesis Examination Committee:
Chairperson, Dr. Wolfgang Bein, Ph.D.
Committee Member, Dr. Ajoy K Datta, Ph.D.
Committee Member, Dr. Yoohwan Kim, Ph.D.
Graduate Faculty Representative, Dr. Venkatesan Muthukumar, Ph.D.