

การระบุเงื่อนไขบังคับแบบนอนไปนึ่งดิงในปัญหำหนดการเชิงเส้น  
โดยใช้โครงข่ายประสาทเทียมซึ่งมีการเรียนรู้แบบกำกับดูแล

นางสาววันหยก อติเศรษฐพงศ์

สถาบันวิทยบริการ

จุฬาลงกรณ์มหาวิทยาลัย

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

สาขาวิชาวิทยาการคณนา ภาควิชาคณิตศาสตร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2547

ISBN 974-17-6540-1

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

IDENTIFYING NON-BINDING CONSTRAINTS IN LINEAR PROGRAMMING PROBLEMS  
USING SUPERVISED LEARNING NEURAL NETWORKS

Miss Wanyok Atisattapong



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

A Thesis Submitted in Partial Fulfillment of the Requirements  
for the Degree of Master of Science in Computational Science

Department of Mathematics

Faculty of Science

Chulalongkorn University

Academic Year 2004

ISBN 974-17-6540-1

Thesis Title                    IDENTIFYING NON-BINDING CONSTRAINTS  
   IN LINEAR PROGRAMMING PROBLEMS  
   USING SUPERVISED LEARNING NEURAL NETWORKS

By                                    Miss Wanyok Atisattapong

Field of study                    Computational Science

Thesis Advisor                    Assistant Professor Krung Sinapiromsaran, Ph.D.

---

Accepted by the Faculty of Science, Chulalongkorn University, in Partial  
Fulfillment of the Requirements for the Master's Degree.

..... Dean of the Faculty of Science  
(Professor Piamsak Menasveta, Ph.D.)

Thesis Committee

..... Chairman  
(Associate Professor Wanida Hemakul, Ph.D.)

..... Thesis Advisor  
(Assistant Professor Krung Sinapiromsaran, Ph.D.)

..... Member  
(Professor Chidchanok Lursinsap, Ph.D.)

..... Member  
(Assistant Professor Vimolrat Ngamaramvaranggul, Ph.D.)

วันหยก อติเศรษฐพงศ์ : การระบุเงื่อนไขบังคับแบบนอนไบนารีในปัญหาการกำหนดการเชิงเส้นโดยใช้  
โครงข่ายประสาทเทียมซึ่งมีการเรียนรู้แบบกำกับดูแล. (IDENTIFYING NON-BINDING  
CONSTRAINTS IN LINEAR PROGRAMMING PROBLEMS USING SUPERVISED  
LEARNING NEURAL NETWORKS) อ. ที่ปรึกษา : ผศ. ดร. กรุง สีนอภิรมย์สรานู, 90 หน้า.  
ISBN 974-17-6540-1.

วิทยานิพนธ์นี้เสนอ วิธีการระบุเงื่อนไขบังคับแบบนอนไบนารีในปัญหาการกำหนดการเชิงเส้น โดยนำ  
โครงข่ายประสาทเทียมซึ่งมีการเรียนรู้แบบกำกับดูแล มาประยุกต์ใช้ในขั้นตอนการทำนายเงื่อนไขบังคับ  
ข้อมูลนำเข้าของโครงข่ายประสาทเทียมประกอบด้วย ค่าสัมประสิทธิ์ของฟังก์ชันจุดประสงค์ ค่า  
สัมประสิทธิ์ของเงื่อนไขบังคับ และค่าคงที่ทางขวามือของปัญหาการกำหนดการเชิงเส้น สำหรับค่าเป้าหมาย  
ในการเรียนรู้ของโครงข่ายประสาทเทียมจะกำหนดให้เป็น 1 เมื่อเงื่อนไขบังคับนั้นเป็นแบบไบนารี และให้  
เป็น 0 เมื่อเงื่อนไขบังคับนั้นเป็นแบบนอนไบนารี โดยงานวิจัยนี้จะพิจารณาเฉพาะปัญหาการกำหนดการเชิง  
เส้นที่มีผลเฉลยเพียงคำตอบเดียว และมีขนาด  $m \times n$  มิติ โดย  $n$  มีค่าตั้งแต่ 2 ถึง 4 และ  $m$  มีค่าตั้งแต่  $n$  ถึง  
 $n+2$  และเลือกใช้ขั้นตอนวิธีแบบแบคพรอพาทเกชันในการฝึกสอนโครงข่ายประสาทเทียม ผลของงานวิจัย  
จะนำเสนอในรูปแบบของเปอร์เซ็นต์ความถูกต้องในการทำนายเงื่อนไขบังคับแบบนอนไบนารีของโครงข่าย  
ประสาทเทียม

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

ภาควิชา .....คณิตศาสตร์..... ลายมือชื่อนิสิต.....  
สาขาวิชา .....วิทยาการคณนา..... ลายมือชื่ออาจารย์ที่ปรึกษา.....  
ปีการศึกษา .....2547.....

# # 4572488023 : MAJOR COMPUTATIONAL SCIENCE

KEY WORD: LINEAR PROGRAMMING / NEURAL NETWORKS / NON-BINDING CONSTRAINTS

WANYOK ATISATTAPONG : IDENTIFYING NON-BINDING CONSTRAINTS IN LINEAR PROGRAMMING PROBLEMS USING SUPERVISED LEARNING NEURAL NETWORKS.

THESIS ADVISOR : ASST. PROF. DR. KRUNG SINAPIROMSARAN, 90 pp. ISBN 974-17-6540-1.

This thesis proposed an approach for identifying non-binding constraints in a linear programming problem (LP). A supervised learning neural network (NN) was applied in the prediction method. The inputs of neural network were composed of the coefficients of the objective function, the coefficients of the constraints and the right-hand-side constants of linear programming problem. For each target of neural network, it set to — '1' if the constraint was binding and '0' if the constraint was non-binding. We considered specifically the LP that has a unique optimal solution and fixed the problem size to  $m \times n$  dimensions where  $n$  was varied from 2 to 4 and  $m$  was varied from  $n$  to  $n+2$ . Moreover, the back propagation (BP) algorithm was selected for training neural networks. The result of this research showed the accuracy of neural networks that identified non-binding constraints.

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

Department ..... Mathematics.....Student's signature.....

Field of study .....Computational Science....Advisor's signature.....

Academic year .... 2004.....

## Acknowledgment

I am deeply indebted to my thesis advisor, Assistant Professor Dr. Krung Sinapiromsaran, for his valuable guidance, great encouragement and untiring help. Without his constant support and attention, this thesis would have never been written.

I am also grateful to the thesis committee, Associate Professor Dr. Wanida Hemakul, Professor Dr. Chidchanok Lursinsap, and Assistant Professor Dr. Vimolrat Ngamaramvaranggul, for their constructive criticism and invaluable advise.

I would like to thank all my teachers for their great contributions and my friends for their encouragement and support during my study.

Finally, words are insufficient to express my gratitude towards my parents who always are a source of unconditional love and support for me.



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

# Contents

Abstract in Thai .....	iv
Abstract in English .....	v
Acknowledgment .....	vi
Contents .....	vii
List of Figures .....	ix
List of Tables .....	x
<b>CHAPTER</b>	
<b>1 INTRODUCTION .....</b>	<b>1</b>
1.1 Motivation and Problem Description .....	1
1.2 The Objective of Research .....	3
1.3 The Scope of Study .....	3
<b>2 BACKGROUND KNOWLEDGE.....</b>	<b>4</b>
2.1 Background on Linear Programming Problems .....	4
2.1.1 The Canonical Form .....	5
2.1.2 Feasible Region, Optimal Solution and Extreme Point .....	7
2.1.3 Binding and Non-binding Constraints .....	11
2.1.4 The Normalized Linear Programming Form .....	14
2.2 Background on Neural Networks .....	17
2.2.1 Multilayer Perceptron .....	19
2.2.2 Back propagation Algorithm .....	20

<b>3</b>	<b>IMPLEMENTATION</b> .....	<b>23</b>
3.1	The Algorithm for Generating Patterns . . . . .	23
3.2	The Architecture of Neural Networks . . . . .	31
<b>4</b>	<b>RESULT AND CONCLUSION</b> .....	<b>40</b>
4.1	Result . . . . .	40
4.2	Conclusion . . . . .	63
	<b>REFERENCES</b> .....	<b>66</b>
	<b>APPENDICES</b>	
	Appendix A GNU Linear Programming Kit: GLPK.....	69
	Appendix B Stuttgart Neural Network Simulator: SNNS.....	73
	Appendix C Coding.....	83
	<b>VITAE</b> .....	<b>90</b>

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย



## List of Figures

2.1	Example of convex and nonconvex sets. . . . .	8
2.2	Feasible region ( $F$ ) and extreme points (a,b,c,d,e). . . . .	10
2.3	Binding constraints. . . . .	12
2.4	The original LP and the reduced LP. . . . .	13
2.5	A biological neuron. . . . .	17
2.6	A basic artificial neuron. . . . .	18
2.7	Three-layer feed forward multilayer perceptron. . . . .	19
2.8	Flow chart for training the MLP by BP. . . . .	22
3.1	Flowchart for the program main.c . . . . .	23
3.2	Flow chart for generating an LP. . . . .	24
3.3	Flow chart for the sorting process. . . . .	26
3.4	Flow chart for the encoding step. . . . .	29
3.5	$m + n$ NNs for the LP in $(m \times n)$ dimensions. . . . .	31
3.6	The $k^{th}$ NN for the $k^{th}$ constraint. . . . .	32
3.7	Five NNs for the problem ( $\mathbf{P}$ ). . . . .	33
3.8	The structure of the $1^{st}$ NN for the constraint $0.5y_1 + 0.2y_2 \leq 1$ . . . . .	34
3.9	The structure of the $2^{nd}$ NN for the constraint $-0.1y_1 + 0.3y_2 \leq 1$ . . . . .	35
3.10	The structure of the $3^{rd}$ NN for the constraint $0.1y_1 + 0.1y_2 \leq 1$ . . . . .	36
3.11	The structure of the $4^{th}$ NN for the nonnegative constraint $x_1 \geq 0$ . . . . .	37
3.12	The structure of the $5^{th}$ NN for the nonnegative constraint $x_2 \geq 0$ . . . . .	38
4.1	Both accuracies decreases according to problem size . . . . .	64
A.1	The output from GLPK . . . . .	72
B.1	Bignet for a network creation . . . . .	81

## List of Tables

4.1	NNs for the LP in $m = 2, n = 2$ . . . . .	43
4.2	Selected NNs for the LP in $m = 2, n = 2$ . . . . .	44
4.3	NNs for the LP in $m = 3, n = 2$ . . . . .	45
4.4	Selected NNs for the LP in $m = 3, n = 2$ . . . . .	46
4.5	NNs for the LP in $m = 4, n = 2$ . . . . .	47
4.6	Selected NNs for the LP in $m = 4, n = 2$ . . . . .	48
4.7	NNs for the LP in $m = 3, n = 3$ . . . . .	49
4.8	Selected NNs for the LP in $m = 3, n = 3$ . . . . .	50
4.9	NNs for the LP in $m = 4, n = 3$ . . . . .	51
4.10	Selected NNs for the LP in $m = 4, n = 3$ . . . . .	52
4.11	NNs for the LP in $m = 5, n = 3$ . . . . .	53
4.12	Selected NNs for the LP in $m = 5, n = 3$ . . . . .	54
4.13	NNs for the LP in $m = 4, n = 4$ . . . . .	55
4.14	Selected NNs for the LP in $m = 4, n = 4$ . . . . .	56
4.15	NNs for the LP in $m = 5, n = 4$ . . . . .	58
4.16	Selected NNs for the LP in $m = 5, n = 4$ . . . . .	59
4.17	NNs for the LP in $m = 6, n = 4$ . . . . .	61
4.18	Selected NNs for the LP in $m = 6, n = 4$ . . . . .	62
4.19	The average accuracy . . . . .	63
4.20	The overall accuracy . . . . .	64

# CHAPTER I

## INTRODUCTION

### 1.1 Motivation and Problem Description

Linear programming is one of the active research areas in optimization. It has impact on economics, industry, military and science, such as the inventory problem, the scheduling problem and the transportation problem [1].

Although there are several methods to solve the linear programming problem (LP), such as the simplex method [2] and the interior point method [3], their computational time depends on the problem size which mainly varies according to the number of constraints and the number of variables. In the last 40 years, researchers have proposed various neural network models for solving linear programming problem [4]. Because the models are implemented on RC-circuits, their computational time is insensitive to the problem size [5].

In this research, we consider the canonical form of linear programming problem (LP) [6]:

$$\begin{aligned} &\text{maximize} && c_1x_1 + c_2x_2 + \dots + c_nx_n \\ &\text{subject to} && a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1 \\ &&& a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2 \\ &&& \vdots \\ &&& a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m \\ &&& x_1, x_2, \dots, x_n \geq 0 \end{aligned}$$

where  $a_{ij}$  are the coefficients of the constraints,  $c_j$  are the coefficients of the objective function,  $x_j$  are the decision variables and  $b_i$  are the right-hand-side constants, for  $i = 1, 2, \dots, m$  and  $j = 1, 2, \dots, n$ .

The LP in the canonical form involves  $m$  constraints and  $n$  nonnegative constraints. At the optimal solution, the constraints can be classified into two groups: binding and non-binding constraints [7].

Theoretically, the group of binding constraints is used to obtain the optimal solution, while the other group can be omitted. Therefore, if we can identify only binding constraints and eliminate the others, the time for solving this reduced problem should not be slower than the time for solving the original problem.

A supervised learning neural network (NN) is chosen as the constrained identification tool because of its capability of learning arbitrary nonlinear input patterns. We select a widely popular algorithm known as the back propagation (BP) algorithm in training multilayer perceptron (MLP).

This algorithm is based on adjusting the synaptic weights in accordance with an error between the actual response of the neural network and the desired (target) response. The learning process is maintained on an epoch-by-epoch basis until the synaptic weights of the network stabilize and the average squared error over the entire training set converges to a minimal value [8].

We will generate input patterns for NNs from all coefficients of LP based on assumption that they have an effect on determining the binding constraints of the optimal solution. For output patterns for NNs, we use the following setting:

$$\text{target} = \begin{cases} 1, & \text{if the constraint is binding;} \\ 0, & \text{otherwise.} \end{cases}$$

## 1.2 The Objective of Research

The objective of this research is to predict the non-binding constraints in a linear programming problem using a supervised learning neural network. The result will be presented in terms of the accuracy of NN.

## 1.3 The Scope of Study

In this research, we consider specifically the LP that have a unique optimal solution because at least the number of binding constraints are fixed to be equal to the number of decision variables. To compare the accuracy of trained NNs in different dimensions of LPs, the problem size are fixed to  $m \times n$  dimensions where  $n$  is varied from 2 to 4 and  $m$  is varied from  $n$  to  $n + 2$ .

Any LP can be transformed into the canonical form where the objective coefficient and the right-hand-side constants are not equal to zero, the coefficients of the constraints ( $a_{ij}$ ). In addition, the coefficients of the objective function ( $c_j$ ) and the right-hand-side constants ( $b_i$ ) are randomly generated where  $c_j \neq 0$  and  $b_i \neq 0$  for  $i = 1, \dots, m$  and  $j = 1, \dots, n$ .

For training each NN, the number of maximum epochs is set to 50,000. Generally, we use the heuristic method to determine the number of nodes in the hidden layer. In this research, hidden neurons are increased by the exponential function,  $2^i$ , where  $2^i$  is equal to or larger than the dimension of input patterns. This method help us find the appropriate the number of hidden neurons faster than increasing them by the linear function.

This thesis is organized as follows. Chapter II provides the theoretical background. Chapter III describes the implementation of the proposed method. The result and conclusion are summarized in Chapter IV.

## CHAPTER II

### BACKGROUND KNOWLEDGE

This chapter provides a summary of important theoretical backgrounds that are required in this research. It contains two main sections: linear programming problems (LPs) and neural networks (NNs).

First, we give definitions of related terms in LPs. We also introduce the new form called the *normalized linear programming form* and explain how any LP can be converted into this new form.

Second, we provide an elementary introduction to the concept of NNs. A standard back propagation (BP) learning algorithm and multilayer perceptron (MLP) are used in this research.

#### 2.1 Background on Linear Programming Problems

Mathematical programming problems are concerned with the use or allocation of limited resources like labor, materials and capital in the best possible manner so that costs are minimized or profits are maximized. We will mainly consider a subclass of mathematical programming problems called a linear programming problem (LP) [1]. An LP is an optimization problem in which the objective function and constraints are expressed as linear function.

### 2.1.1 The Canonical Form

There are various forms to represent an LP. In this research, we consider the canonical form of LP with  $m$  constraints and  $n$  nonnegative constraints as:

$$\begin{aligned}
 &\text{maximize} && c_1x_1 + c_2x_2 + \dots + c_nx_n \\
 &\text{subject to} && a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n \leq b_1 \\
 &&& a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n \leq b_2 \\
 &&& \vdots \\
 &&& a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n \leq b_m \\
 &&& x_1, x_2, \dots, x_n \geq 0
 \end{aligned} \tag{2.1}$$

where  $a_{ij}$ , for  $i = 1, 2, \dots, m$  and  $j = 1, 2, \dots, n$ , are the coefficients of the constraints.  $c_1, c_2, \dots, c_n$  are the coefficients of the objective function for nonnegative unknown (decision) variables,  $x_1, x_2, \dots, x_n$ , respectively.  $b_1, b_2, \dots, b_m$  are the right-hand-side constants.

In matrix-vector notation, the above canonical LP can be written in a compact form as:

$$\begin{aligned}
 &\text{maximize} && \mathbf{c}^T \mathbf{x} \\
 &\text{subject to} && \mathbf{A} \mathbf{x} \leq \mathbf{b} \\
 &&& \mathbf{x} \geq \mathbf{0}
 \end{aligned} \tag{2.2}$$

where  $\mathbf{A}$  is an  $(m \times n)$  matrix called the coefficient matrix,  $\mathbf{c}$  is an  $(n \times 1)$  column vector called the cost vector,  $\mathbf{x}$  is an  $(n \times 1)$  column vector called the decision vector and  $\mathbf{b}$  is an  $(m \times 1)$  column vector called the right-hand-side vector [9].

In general, we can convert any LP to the canonical form (2.2). Note that the canonical form requires maximizing the objective function. For the minimized optimization direction, we will multiply the objective function by  $-1$  to reverse its

direction, changing the minimizing problem to the maximizing problem. The optimal solutions of both the maximization problem and the minimization problem are the same, while their optimal values will differ by a negative sign.

$$-\text{maximize } (-\mathbf{c}^T \mathbf{x}) = \text{minimize } (\mathbf{c}^T \mathbf{x}).$$

Any linear inequality constraint of the form  $\geq$  can be converted into the form  $\leq$  by multiplying  $-1$  on both sides of that constraint. For any linear equality constraint, we can express it using two inequality constraints. Consider the conversion of a constraint to the form  $\leq$ .

If the  $k^{\text{th}}$  constraint is of the form

$$a_{k1}x_1 + a_{k2}x_2 + \dots + a_{kn}x_n \geq b_k,$$

then we multiply both sides of this constraint by  $-1$  to get

$$-a_{k1}x_1 - a_{k2}x_2 - \dots - a_{kn}x_n \leq -b_k.$$

If the  $k^{\text{th}}$  constraint is of the form

$$a_{k1}x_1 + a_{k2}x_2 + \dots + a_{kn}x_n = b_k,$$

it can be rewritten using two inequality constraints:

$$a_{k1}x_1 + a_{k2}x_2 + \dots + a_{kn}x_n \leq b_k,$$

and

$$-a_{k1}x_1 - a_{k2}x_2 - \dots - a_{kn}x_n \leq -b_k.$$

Now, we can convert any LP into the canonical form (2.2). In the next part, we will introduce some terminology for finding the solution of the LP.



## 2.1.2 Feasible Region, Optimal Solution and Extreme Point

For any linear programming problem, we are interested in determining the values of the decision variables that satisfy all restrictions and give the optimal value for the objective function. The necessary standard definitions for solving LPs are described as follows [10]:

### Definition 2.1.1 (Feasible Region).

Given an LP in its canonical form (2.2), the *feasible region* is the set of all non-negative solutions that satisfy all the constraints of the LP.

$$F = \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq \mathbf{0}\},$$

where  $F$  is the feasible region.

Any solution in the feasible region of the LP is said to be the *feasible solution*.

Suppose that there are feasible solutions, the goal of the LP is to find the optimal feasible solution, as measured by the value of the objective function.

### Definition 2.1.2 (Optimal Solution).

Consider an LP in its canonical form (2.2), if the feasible region is not empty, an *optimal solution* is a feasible solution that has the largest value of the objective function for the maximization problem. Let  $\mathbf{x}^*$  be an optimal solution to the LP.

$$\mathbf{c}^T \mathbf{x}^* \geq \mathbf{c}^T \mathbf{x}, \forall \mathbf{x} \in F$$

The value of the objective function corresponding to an optimal solution is called the *optimal value*.

The definition of convex set and extreme point that relevant in finding the optimal solution to the LP will be summarized as follows [6]:

**Definition 2.1.3 (Convex Set).**

A set  $S$  in  $\mathbb{R}^n$  is called a *convex set* if given any two points  $\mathbf{x}_1$  and  $\mathbf{x}_2$  in  $S$ , then  $\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2 \in S$  for each  $\lambda \in [0, 1]$ .

Note that  $\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$  for  $\lambda$  in the interval  $[0, 1]$  represents a point on line segment joining  $\mathbf{x}_1$  and  $\mathbf{x}_2$ . Any point of the form  $\lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$  where  $\lambda \in [0, 1]$  is called a *convex combination* of  $\mathbf{x}_1$  and  $\mathbf{x}_2$ . If  $\lambda \in (0, 1)$ , then the convex combination is called *strict*.

Hence convexity of  $S$  can be interpreted geometrically as follows. For each pair of points  $\mathbf{x}_1$  and  $\mathbf{x}_2$  in  $S$ , the line segment joining them must belong to  $S$ .

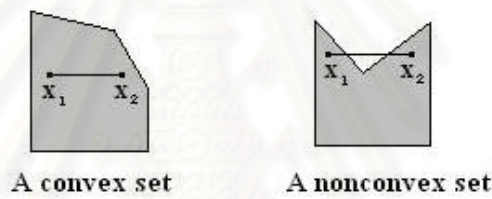


Figure 2.1: Example of convex and nonconvex sets.

In the latter case, we see that not all points on the line segment joining  $\mathbf{x}_1$  and  $\mathbf{x}_2$  belong to  $S$ .

For the feasible region of any LP in its canonical form (2.2), we can show that it is a convex set.

**Definition 2.1.4 (Extreme Point).**

A point  $\mathbf{x}$  in a convex set  $S$  is called an *extreme point* of  $S$ , if  $\mathbf{x}$  cannot be represented as a strict convex combination of two distinct points in  $S$ . In other words, if  $\mathbf{x} = \lambda\mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2$  with  $\lambda \in (0, 1)$  and  $\mathbf{x}_1, \mathbf{x}_2 \in S$ , then  $\mathbf{x} = \mathbf{x}_1 = \mathbf{x}_2$ .

Any LP in its canonical form (2.2) must be in one of the following four cases:

1. LP has the unique optimal solution.

This unique optimal solution must be an extreme point.

2. LP has alternative optimal solutions.

If there are two extreme points  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  being optimal, then any convex combination of  $\mathbf{x}_1^*$  and  $\mathbf{x}_2^*$  is optimal.

3. LP is unbounded.

For a maximization problem, the feasible region is unbounded and the plane  $\mathbf{c}^T \mathbf{x} = z$  can be increased along the unbounded direction of the feasible region. In this case, the objective value is unbounded and no optimal solution exists.

4. LP has an empty feasible region.

In this case, the system of equations and/or inequalities defining the feasible region is inconsistent. This means there is no point satisfying all constraint of the LP. Therefore, no optimal solution exists.

In this research, we considered specifically the LP that has a unique optimal solution. To clarify the concept of optimal solution and extreme point, consider the following example.

**Example 2.1.** Consider the following LP:

$$\begin{array}{ll}
 \text{maximize} & 40x_1 + 36x_2 \\
 \text{subject to} & x_1 \leq 8 \\
 & x_2 \leq 10 \\
 & 5x_1 + 3x_2 \leq 45 \\
 & x_1 \geq 0 \\
 & x_2 \geq 0
 \end{array}$$

The intersection of the five halfspaces give the feasible region as follows:

$$F = \{ \mathbf{x} \in \mathbb{R}^2 \mid x_1 \leq 8, x_2 \leq 10, 5x_1 + 3x_2 \leq 45, x_1 \geq 0, x_2 \geq 0 \}$$

Clearly the set is a convex set and its extreme points are given as:

$$a = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad b = \begin{bmatrix} 8 \\ 0 \end{bmatrix}, \quad c = \begin{bmatrix} 8 \\ \frac{5}{3} \end{bmatrix}, \quad d = \begin{bmatrix} 3 \\ 10 \end{bmatrix}, \quad \text{and} \quad e = \begin{bmatrix} 0 \\ 10 \end{bmatrix}.$$

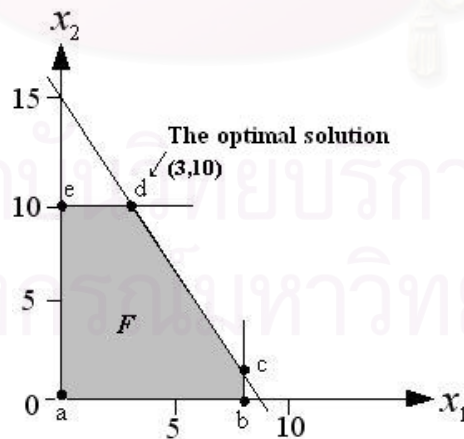


Figure 2.2: Feasible region ( $F$ ) and extreme points ( $a,b,c,d,e$ ).

After solving this LP, we get the unique optimal solution at  $x_1 = 3$  and  $x_2 = 10$  (the extreme point  $d$ ).

In the next part, we will describe the type of constraints that is the main point of this research.

### 2.1.3 Binding and Non-binding Constraints

When the optimal solution to an LP has been found, we will classify each constraint as being a binding constraint or a non-binding constraint using the following definitions [7]:

**Definition 2.1.1 (Binding Constraints).**

For an LP in its canonical form (2.2), a constraint is *binding* at  $\mathbf{x}$  if the left-hand side and the right-hand side of the constraint are *equal* when the feasible solution,  $\mathbf{x}$ , are substituted into the constraint.

**Definition 2.1.2 (Non-binding Constraints).**

For an LP in its canonical form (2.2), a constraint is *non-binding* at  $\mathbf{x}$  if the left-hand side and the right-hand side of the constraint are *unequal* when the feasible solution,  $\mathbf{x}$ , are substituted into the constraint.

**Example 2.2.** Consider the previous LP. The optimal solution of this problem is  $x_1 = 3$  and  $x_2 = 10$ . When we substitute the optimal values of the decision variables into the left-hand side of the constraints, we obtain

$$\begin{array}{ll} \text{maximize} & 40x_1 + 36x_2 \\ \text{subject to} & x_1 \leq 8 \qquad \qquad \qquad 3 \leq 8 \qquad \qquad (1) \end{array}$$

$$x_2 \leq 10 \qquad \qquad \qquad 10 = 10 \qquad \qquad (2)$$

$$5x_1 + 3x_2 \leq 45 \qquad \qquad \qquad 45 = 45 \qquad \qquad (3)$$

$$x_1 \geq 0 \qquad \qquad \qquad 3 \geq 0 \qquad \qquad (4)$$

$$x_2 \geq 0 \qquad \qquad \qquad 10 \geq 0 \qquad \qquad (5)$$

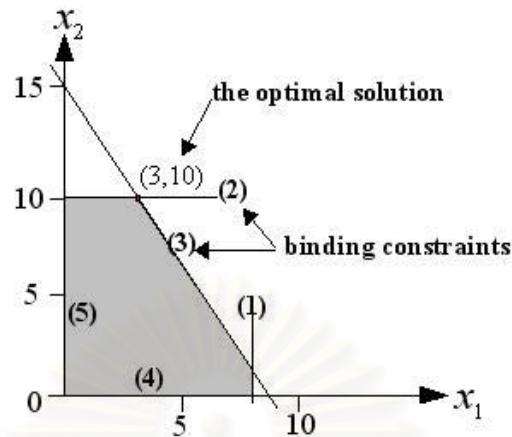


Figure 2.3: Binding constraints.

From the above definition, we can identify that the 2<sup>nd</sup> and 3<sup>rd</sup> constraints are binding at the optimal solution while the others are non-binding at the optimal solution. The point of optimal solution (3, 10) results from the intersection of two binding constraints (Figure 2.3).

From the canonical form of LP (2.2), the concept of extreme points and the definition of binding and non-binding constraints, if there is a unique optimal solution, we can show that the optimal solution must be an extreme point that results from the intersection of at least  $n$  binding constraints.

Therefore, we want to find an approach for identifying only necessary constraints (binding constraints) and eliminating the others to reduce the problem size. The following theorem indicates that the optimal solution of the eliminated non-binding LP is the same as the original LP.

**Theorem 2.1.1.** For an LP in its canonical form (2.2) with a unique optimal solution  $\mathbf{x}^*$ , define

$$J = \{ j \mid \mathbf{A}_j \mathbf{x}^* = \mathbf{b}_j \} \cup \{ j \mid \mathbf{x}_j^* = 0 \}$$

as a set of indices of binding constraints at  $\mathbf{x}^*$  from the set of inequalities from  $\mathbf{A}\mathbf{x} \leq \mathbf{b}$  and  $\mathbf{x} \geq 0$ . The reduced LP that is eliminated non-binding constraints

$$\begin{aligned} & \text{maximize} && \mathbf{c}^T \mathbf{x} \\ & \text{subject to} && \mathbf{A}_i \mathbf{x} \leq \mathbf{b}_i \\ & && \mathbf{x}_i \geq 0 \text{ for } i \in J \end{aligned}$$

has the same optimal solution  $\mathbf{x}^*$  as the original LP.

Observe that if  $\mathbf{x}^*$  is the optimal solution for the original LP, then there cannot be any improving feasible directions at  $\mathbf{x}^*$ . Therefore,  $\mathbf{x}^*$  is also the optimal solution for the reduced LP.

Conversely, suppose that  $F_R$  is the feasible region of the reduced LP. If  $\mathbf{x}^*$  is the optimal solution for the reduced LP, then there cannot be any point in  $F_R$  giving the optimal value better than  $\mathbf{x}^*$ . Obviously,  $F \subseteq F_R$ . Thus  $\mathbf{x}^*$  must be the optimal solution for the original LP.

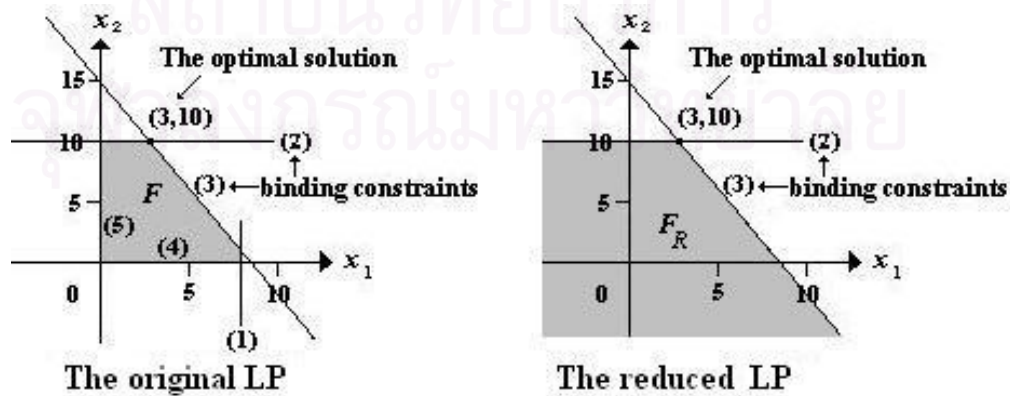


Figure 2.4: The original LP and the reduced LP.

### 2.1.4 The Normalized Linear Programming Form

In this research, we consider LPs in the canonical form (2.2) where  $c_i \neq 0$  for  $i = 1, 2, \dots, n$  and  $b_i \neq 0$  for  $i = 1, 2, \dots, m$ .

To simplify our notation, the decision vector ( $\mathbf{x}$ ) and the coefficient matrix ( $\mathbf{A}$ ) will be divided into a subvector and a submatrix as:

$$\mathbf{x} = [\mathbf{x}^+ \mid \mathbf{x}^-],$$

where  $\mathbf{x}^+$  is a column vector whose the objective coefficient is equal to 1,

$\mathbf{x}^-$  is a column vector whose the objective coefficient is equal to  $-1$ .

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}^+ \\ \mathbf{A}^- \end{bmatrix},$$

where

$\mathbf{A}^+$  is the coefficient matrix whose the right-hand-side constant is equal to 1,

$\mathbf{A}^-$  is the coefficient matrix whose the right-hand-side constant is equal to  $-1$ .

The new form of LPs called the *normalized LP form* can be defined as follows:

$$\begin{aligned} & \text{maximize} && \mathbf{1}^T \mathbf{x}^+ - \mathbf{1}^T \mathbf{x}^- \\ & \text{subject to} && \mathbf{A}^+ [\mathbf{x}^+ \mid \mathbf{x}^-] \leq \mathbf{1} \\ & && \mathbf{A}^- [\mathbf{x}^+ \mid \mathbf{x}^-] \leq -\mathbf{1} \\ & && \mathbf{x}^+, \mathbf{x}^- \geq \mathbf{0} \end{aligned} \tag{2.3}$$

where  $\mathbf{1}$  is the sum vector with all element equal to 1.



The coefficients of the objective function and the right-hand-side constants in the LP has been reduced from the real value in the canonical form (2.2) to 1 or  $-1$  in the normalized LP form (2.3). By fixing that values to 1 or  $-1$ , the LP in the new form have been simplified.

The process for converting any LP into the normalized LP form (2.3) can divided into three steps:

1. Covert any LP into its canonical form (2.2) and check the assumption that  $c_i \neq 0$ , for  $i = 1, 2, \dots, n$  and  $b_i \neq 0$ , for  $i = 1, 2, \dots, m$ .
2. Rename decision variables so that the coefficients of objective function ( $c_i$ , for  $i = 1, \dots, n$ ) of the LP are equal to 1 or  $-1$ .
3. Divide each constraint by the absolute value of its right-hand-side constant ( $|b_i|$ , for  $i = 1, \dots, m$ ).

**Example 2.3.** The LP is given by:

$$\begin{array}{ll}
 \text{maximize} & 5x_1 - 2x_2 \\
 \text{subject to} & 5x_1 - 6x_2 \leq 2 \\
 & -3x_1 + x_2 \leq -1 \\
 & -6x_1 + 12x_2 \leq 3 \\
 & x_1, x_2 \geq 0
 \end{array}$$

We apply the conversion steps to the above problem to get the normalized LP form of (2.3):

1. The LP is already in its canonical form (2.2) and also satisfies the assumption  $c_i \neq 0$  for  $i = 1, 2, \dots, n$  and  $b_i \neq 0$  for  $i = 1, 2, \dots, m$ .
2. Let  $y_1 = 5x_1$  and  $y_2 = 2x_2$  so that the coefficients of the objective function from real values are set to 1 or  $-1$ .

$$\begin{array}{ll}
 \text{maximize} & y_1 - y_2 \\
 \text{subject to} & y_1 - 3y_2 \leq 2 \\
 & -0.6y_1 + 0.5y_2 \leq -1 \\
 & -1.2y_1 + 6y_2 \leq 3 \\
 & y_1, y_2 \geq 0
 \end{array}$$

3. Divide the first constraint by 2 and the third constraint by 3 to get:

$$\begin{array}{ll}
 \text{maximize} & y_1 - y_2 \\
 \text{subject to} & 0.5y_1 - 1.5y_2 \leq 1 \\
 & -0.6y_1 + 0.5y_2 \leq -1 \\
 & -0.4y_1 + 2y_2 \leq 1 \\
 & y_1, y_2 \geq 0
 \end{array}$$

Now, we can convert any LP in the canonical form (2.2) to the normalized LP form (2.3). We can solve any LPs in this form using the software GNU Linear Programming Kit: GLPK\*.

---

\*Appendix A

## 2.2 Background on Neural Networks

A neural network (NN) is widely used machine learning methodology in many diverse fields, including engineering, physics and mathematics. It is a network designed to model the way in which the human brain processes information, such as pattern recognition, data classification and image processing [8].

The human brain consists of a large number (approximately  $10^{11}$ ) of highly connected elements (approximately  $10^4$  connections per element) called neurons [11]. As shown in Figure 2.5, each neuron has four principal components: a cell body, an axon, dendrites and synapses.

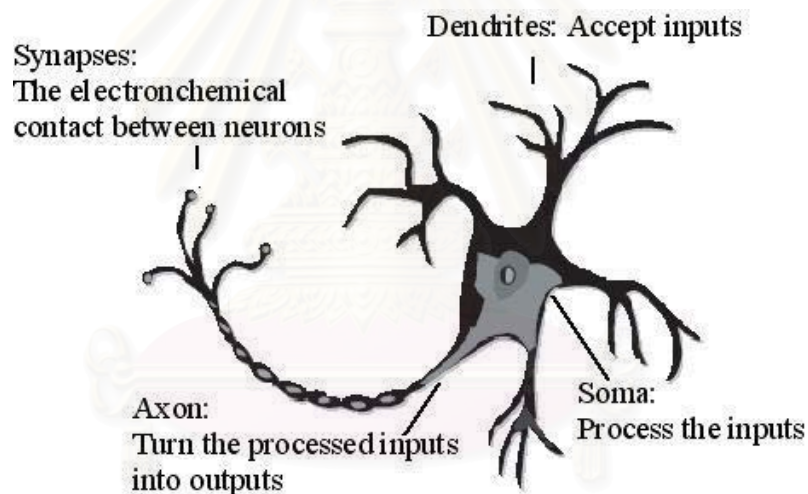


Figure 2.5: A biological neuron.

The dendrites act as a neuron's input receptors for signals coming from other neurons. The cell body effectively evaluates these incoming signal and determines the output. The axon is a single long fiber that carries the signal from the cell body out to other neurons as the neuron's output channel. The gap between an output axon of one neuron and the input dendrites of another is the location of the synapses. The information are transferred across a synapse by electrochemical voltage.

An artificial neuron simulates the four basic functions of the biological neuron: accepts inputs, combines them, performs an operation and outputs a result (Figure 2.6). The inputs to the network are represented by the mathematical symbol  $x_i$ . These are multiplied by their corresponding weights,  $w_i$ . In the simplest case, these products and bias are simply summed, fed through an activation or transfer function to generate an output. The bias acts as a threshold in that it serves to vary the activity of the neuron.

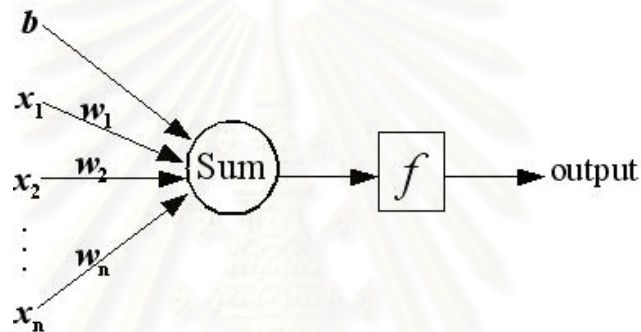


Figure 2.6: A basic artificial neuron.

The output of this model can be denoted by

$$output = f\left(\sum_{i=1}^n x_i w_i + b\right)$$

where  $x_i$  is the input signal,  $w_i$  is the synaptic weight, for  $i = 1, 2, \dots, n$ ,  $b$  is the bias, and  $f$  is the activation function, which is typically a sigmoidal function having a value range from 0 to 1.

$$f(net_i) = \frac{1}{1 + e^{-net_i}}$$

## 2.2.1 Multilayer Perceptron

In this part, we focus on a certain type of NN called the standard feed forward multilayer perceptron (MLP). It is a valuable learning tool when one has little or no knowledge about the form of the relationship between input vectors and their corresponding outputs.

This model consists of a network of neurons organized into several layers. The degree of non-linearity of the MLP network can be changed by varying the number of layers and the number of units in each layer.

Generally, it requires three layers: an input layer, hidden layer and output layer. However, the input layer does not perform any computations. It is only used to distribute the input signal to the hidden layer.

A neuron in any layer of the network is connected to all the neurons in the previous layer by links or weights. The input signal propagates through the network in a forward direction, from left to right and on a layer-by-layer basis [8].

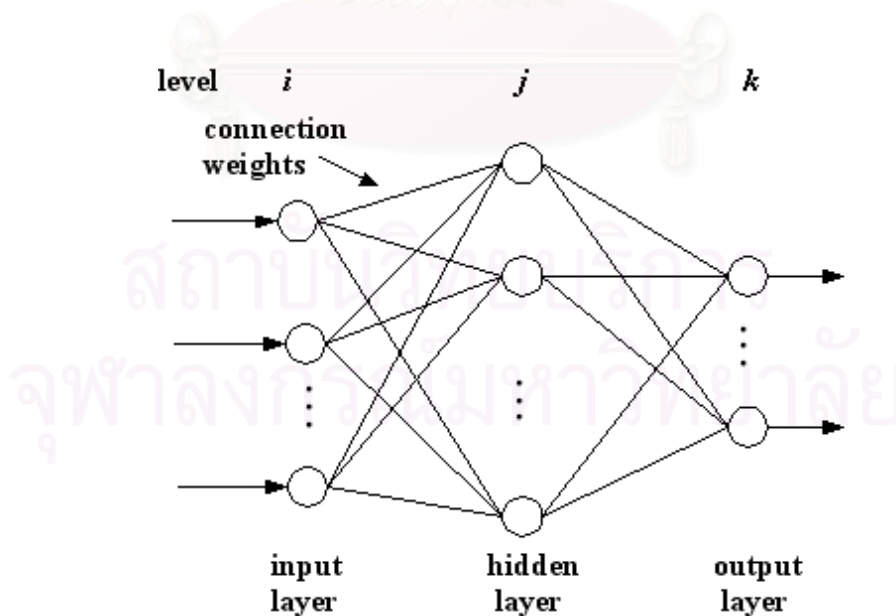


Figure 2.7: Three-layer feed forward multilayer perceptron.

Suppose that the number of neurons in the input layer, hidden layer, and output layer are equal to  $n_1, n_2$ , and  $n_3$  respectively. The output of neuron  $j$  from the hidden layer,  $h_j$ , is computed as:

$$h_j = f_j(net_j)$$

$$net_j = \sum_{i=1}^{n_1} x_i w_{ji} + b_j$$

where  $x_i$  is the input signal,  $w_{ji}$ , for  $i = 1, 2, \dots, n_1$  and  $j = 1, 2, \dots, n_2$ , represents the synaptic weight connecting between the neuron  $i$  in the input layer and the neuron  $j$  in the hidden layer,  $b_j$  is the bias at neuron  $j$  and  $f$  is a sigmoidal function.

The output from the hidden layer is the input to the output layer. The output of neuron  $k$  from the output layer,  $o_k$ , is computed as:

$$o_k = f_k(net_k)$$

$$net_k = \sum_{j=1}^{n_2} h_j w_{kj} + b_k$$

where  $w_{kj}$ , for  $j = 1, 2, \dots, n_2$  and  $k = 1, 2, \dots, n_3$ , represents the synaptic weight connecting between the neuron  $j$  in the hidden layer and the neuron  $k$  in the output layer and  $b_k$  is the bias at neuron  $k$ .

### 2.2.2 Back propagation Algorithm

Back propagation (BP) algorithm is one of the most popular used algorithm for training neural network [12]. It is based on a feed forward multilayer perceptron with supervised learning. Training the MLP with BP is adapting the synaptic weights until the error between the actual output of the NN and the desired response (target) over all training patterns is minimized.

Let  $p$  be the total number of training patterns. The cost function or the total error energy [8] is obtained by:

$$E = \frac{1}{2} \sum_{\mu=1}^p \sum_{k=1}^{n_3} (t_k^\mu - o_k^\mu)^2$$

where  $t_k^\mu$  is the desired response (target) for the  $k^{th}$  dimension of the training pattern  $\mu$ .  $o_k^\mu$  is the output of the  $k^{th}$  neuron in the output layer of the training pattern  $\mu$ .

In this research, we focus on a neural network training technique called the steepest descent method. Mathematically, this technique is accomplished by minimizing a suitable error function with respect to all network weights [12].

Let  $\eta \in (0, 1)$  be a learning rate and  $\delta$  be the error gradient. We can compute the weight adjustment,  $\Delta w$ , to update the synaptic weight,  $w(t)$ , at iteration  $t$  as follows:

- For the weight between hidden and output layers:

$$w_{kj}(t+1) = w_{kj}(t) + \Delta w_{kj},$$

$$\Delta w_{kj} = \eta \delta_k h_j,$$

$$\delta_k = (t_k - o_k) f'(net_k).$$

- For the weight between input and hidden layers:

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji},$$

$$\Delta w_{ji} = \eta \delta_j x_i,$$

$$\delta_j = f'(net_j) \left( \sum_{k=1}^{n_3} \delta_k w_{kj} \right).$$

Training procedures for the MLP by BP are presented in Figure 2.8.

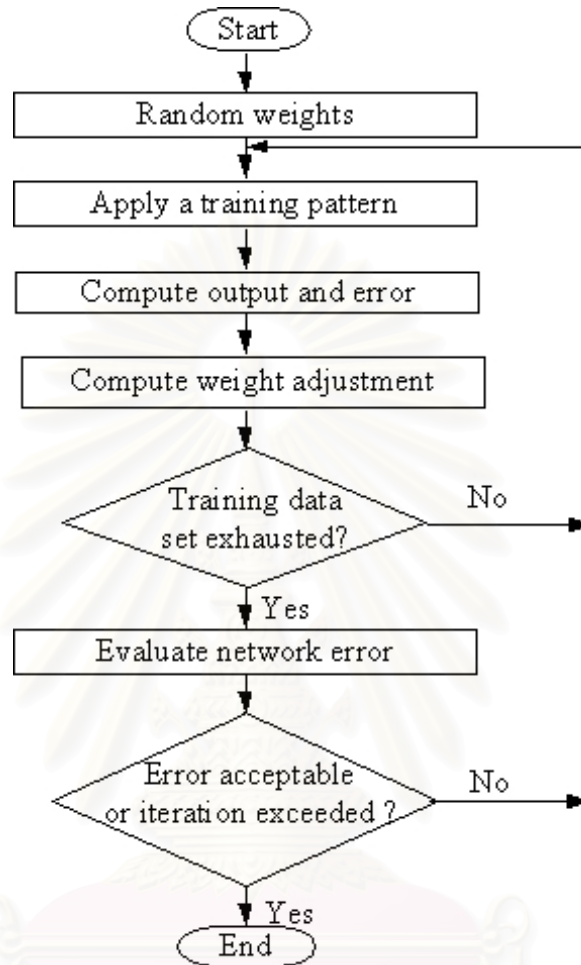


Figure 2.8: Flow chart for training the MLP by BP.

We will use the back propagation algorithm from the software Stuttgart Neural Network Simulator: SNNS<sup>†</sup>.

<sup>†</sup>Appendix B



## CHAPTER III

### IMPLEMENTATION

This chapter describes the implementation of the proposed method. It contains two main sections: the algorithm for generating patterns of NNs from LPs and the architecture of NNs used in this research.

#### 3.1 The Algorithm for Generating Patterns

The input-output training and testing patterns for NNs are generated by the main program (`main.c`<sup>‡</sup>). The flowchart of the program `main.c` is shown in Figure 3.1.

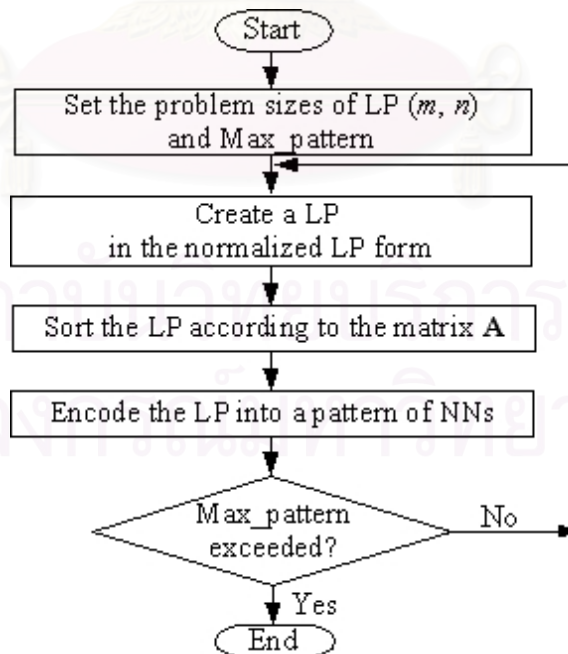


Figure 3.1: Flowchart for the program `main.c`

---

<sup>‡</sup>Appendix C

Let  $n$  be the number of decision variables and  $m$  be the number of constraints.

In the first step, the LPs are randomly created in the normalized LP form (2.3) by the subprogram  $\text{Gen\_LP}()$ <sup>‡</sup>.

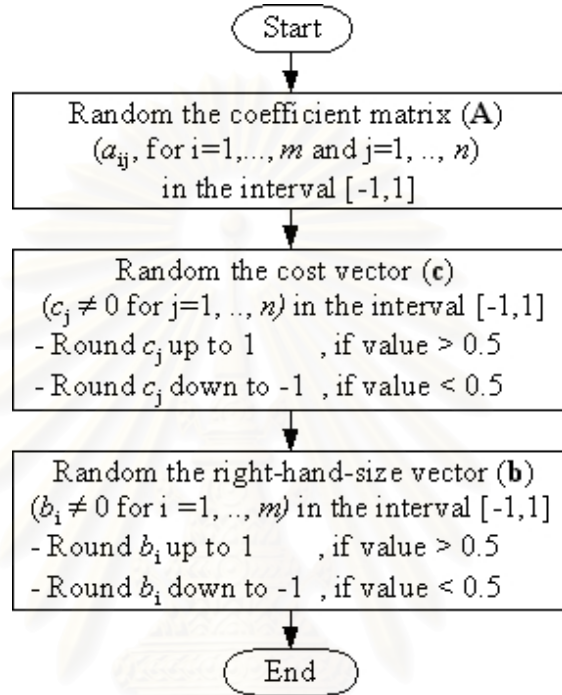


Figure 3.2: Flow chart for generating an LP.

To detect different patterns from the same LP, the generated LP is sorted according to the coefficient matrix ( $\mathbf{A}$ ) by the subprogram  $\text{Sort}()$ <sup>‡</sup>. To clarify the sorting process, we add the notation of the submatrix as follows:

$$\mathbf{A}_i = [a_{rs}], \quad \text{for } r = i, \dots, m \quad \text{and} \quad s = i, \dots, n,$$

and let  $a_{hk}$  be the largest elements in  $\mathbf{A}_i$ .

---

<sup>‡</sup>Appendix C

The sorting process can be separated into two cases:

- Case 1:  $\mathbf{A}$  has  $m \leq n + 1$

At iteration  $i$ ,  $a_{hk}$  of  $\mathbf{A}_i$  will be interchanged to the top left corner of the main diagonal position ( $a_{ii}$ ) of  $\mathbf{A}$ , for  $i = 1, \dots, n$ . When  $\mathbf{A}$  has  $m = n + 1$ , only  $n$  steps are required because  $a_{n+1,n}$  are already smaller than  $a_{nn}$ .

- Case 2:  $\mathbf{A}$  has  $m > n + 1$

For the first  $i = 1, \dots, n$ , it is sorted as in case 1. However, we need to add the step of interchanging the remaining rows of  $\mathbf{A}$  so that the elements in rows  $n + 1, \dots, m$  are sorted in descending order according to the last column  $n$ .

The flowchart of the subprogram  $\text{Sort}()^\ddagger$  is shown in Figure 3.3.




---

<sup>‡</sup>Appendix C

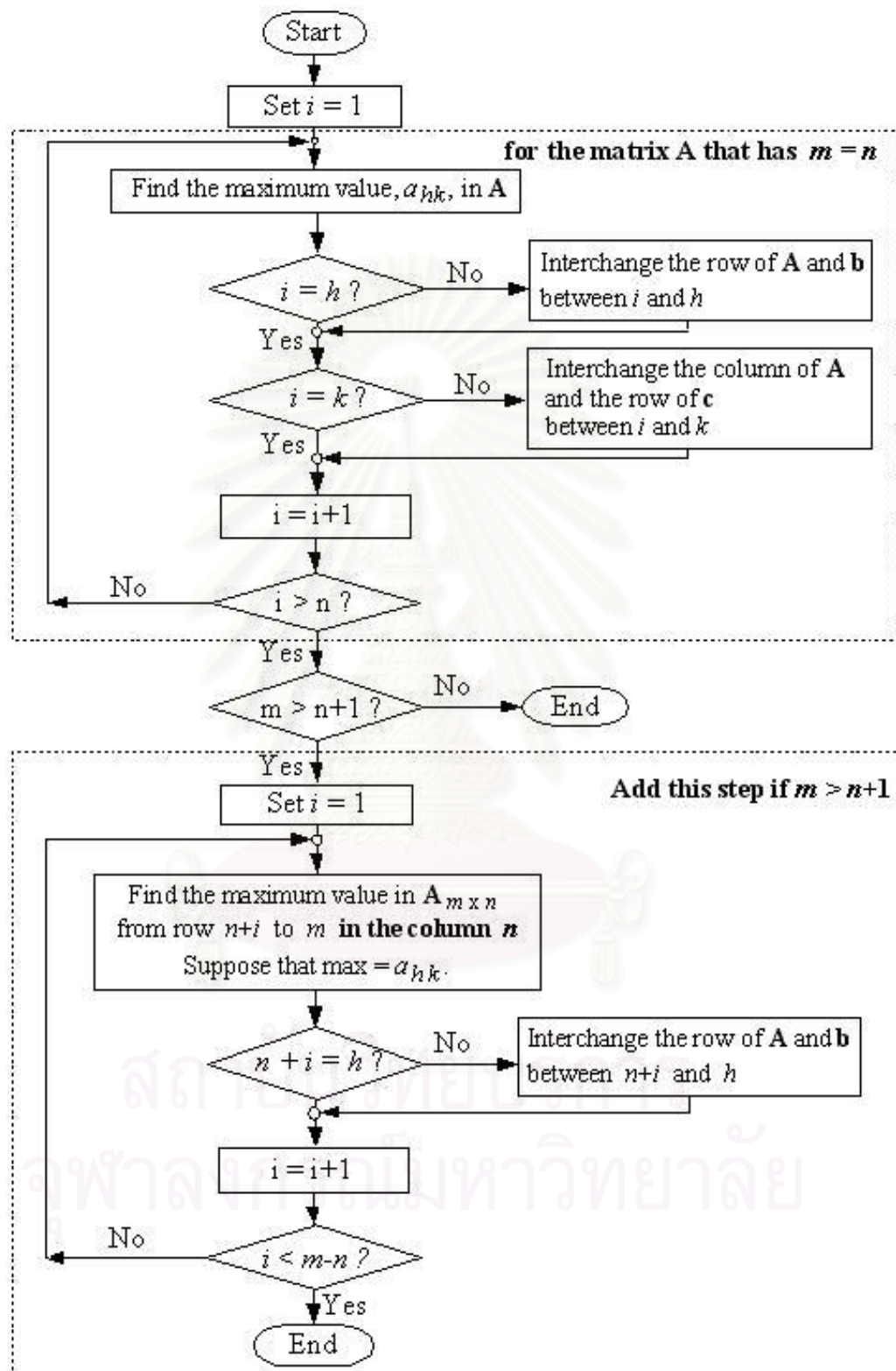


Figure 3.3: Flow chart for the sorting process.

**Example 3.1.** Consider the following LP:

$$\begin{aligned}
 & \text{maximize} && -x_1 + x_2 \\
 & \text{subject to} && 0.3x_1 - 0.1x_2 \leq 1 \\
 & && 0.2x_1 + 0.5x_2 \leq 1 \\
 & && 0.1x_1 + 0.1x_2 \leq 1 && \mathbf{P}^{(1)} \\
 & && x_1, x_2 \geq 0
 \end{aligned}$$

The cost vector, coefficient matrix and right-hand-side vector can be written as

$$\mathbf{c}^{(1)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \quad \mathbf{A}^{(1)} = \begin{bmatrix} 0.3 & -0.1 \\ 0.2 & 0.5 \\ 0.1 & 0.1 \end{bmatrix}, \quad \text{and} \quad \mathbf{b}^{(1)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Additional different five matrices of  $\mathbf{A}^{(1)}$  which interchange rows of the coefficient matrix  $\mathbf{A}^{(1)}$  are

$$\begin{bmatrix} 0.3 & -0.1 \\ 0.1 & 0.1 \\ 0.2 & 0.5 \end{bmatrix}, \quad \begin{bmatrix} 0.2 & 0.5 \\ 0.3 & -0.1 \\ 0.1 & 0.1 \end{bmatrix},$$

$$\begin{bmatrix} 0.2 & 0.5 \\ 0.1 & 0.1 \\ 0.3 & -0.1 \end{bmatrix}, \quad \begin{bmatrix} 0.1 & 0.1 \\ 0.3 & -0.1 \\ 0.2 & 0.5 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} 0.1 & 0.1 \\ 0.2 & 0.5 \\ 0.3 & -0.1 \end{bmatrix}.$$

These matrices still represent the constraints of the same LP.

If we consider the column interchange of the problem  $\mathbf{P}^{(1)}$ , its cost vector and coefficient matrix can be written by

$$\mathbf{c}^{(2)} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad \mathbf{A}^{(2)} = \begin{bmatrix} -0.1 & 0.3 \\ 0.5 & 0.2 \\ 0.1 & 0.1 \end{bmatrix}, \quad \text{and} \quad \mathbf{b}^{(2)} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

Similarly, there are the other five matrices which are the results of interchanging rows of the coefficient matrix  $\mathbf{A}^{(2)}$

$$\begin{bmatrix} -0.1 & 0.3 \\ 0.1 & 0.1 \\ 0.5 & 0.2 \end{bmatrix}, \quad \begin{bmatrix} 0.5 & 0.2 \\ -0.1 & 0.3 \\ 0.1 & 0.1 \end{bmatrix},$$

$$\begin{bmatrix} 0.5 & 0.2 \\ 0.1 & 0.1 \\ -0.1 & 0.3 \end{bmatrix}, \quad \begin{bmatrix} 0.1 & 0.1 \\ -0.1 & 0.3 \\ 0.5 & 0.2 \end{bmatrix}, \quad \text{and} \quad \begin{bmatrix} 0.1 & 0.1 \\ 0.5 & 0.2 \\ -0.1 & 0.3 \end{bmatrix}.$$

We can see that one LP can be represented by different patterns of various cost vectors, right-hand-side vectors and coefficient matrices. However, we need to identify the unique LP before we encode it into an input pattern of NNs.

Sorted by subprogram `Sort()`, the above problem ( $\mathbf{P}^{(1)}$ ) can be represented by:

$$\begin{aligned} &\text{maximize} && y_1 - y_2 \\ &\text{subject to} && 0.5y_1 + 0.2y_2 \leq 1 \\ &&& -0.1y_1 + 0.3y_2 \leq 1 \\ &&& 0.1y_1 + 0.1y_2 \leq 1 \\ &&& y_1, y_2 \geq 0 \end{aligned} \quad (\mathbf{P})$$

where  $y_1 = x_2$  and  $y_2 = x_1$ ,

$$\mathbf{c} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \quad \mathbf{A} = \begin{bmatrix} 0.5 & 0.2 \\ -0.1 & 0.3 \\ 0.1 & 0.1 \end{bmatrix}, \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}.$$

In the last step, we encode the LP into a input-output pattern of NNs as follows:

$$\mathbf{xn} = [c_1 \ \dots \ c_n \ a_{11} \ \dots \ a_{1n} \ b_1 \ \dots \ a_{m1} \ \dots \ a_{mn} \ b_m]^T$$

$$\mathbf{t} = [t_1 \ \dots \ t_n \ t_{n+1} \ \dots \ t_{m+n}]^T$$

where

$\mathbf{xn}$  is the  $((m+n+mn) \times 1)$  vector called the input vector and its  $i^{th}$  dimension represents the input signal of neuron  $i$  in the input layer of NNs.

$\mathbf{t}$  is  $((m+n) \times 1)$  vector called the target vector and its  $i^{th}$  dimension represents the output signal that is

$$t_i = \begin{cases} 1, & \text{if the } i^{th} \text{ constraint is binding;} \\ 0, & \text{otherwise.} \end{cases}$$

We solve the LP by the subprogram `Solve_LP()`<sup>‡</sup> to identify the binding and non-binding constraints using the subprogram `Binding()`<sup>‡</sup>. This construct the vector  $\mathbf{t}$ . In this research, the number of binding constraints is fixed to the number of decision variables, so we ignore the LPs that do not have a unique solution.

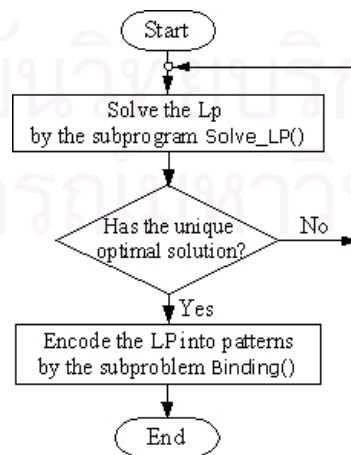


Figure 3.4: Flow chart for the encoding step.

<sup>‡</sup>Appendix C

For a testing set generations, we also solve LP so that we can compare the target with the output from trained NNs and compute the accuracy of trained NNs.

**Example 3.2.** From the previous problem **P**, the  $(11 \times 1)$  input vector can be written as follows:

$$\mathbf{xn} = [1 \quad -1 \quad 0.5 \quad 0.2 \quad 1 \quad -0.1 \quad 0.3 \quad 1 \quad 0.1 \quad 0.1 \quad 1]^T$$

After we solve the LP, we get the only one optimal solution which is  $y_1 = 2$  and  $y_2 = 0$ . When we substitute the optimal values of the decision variables into the left-hand side of the constraints, we obtain

$$\begin{aligned} &\text{maximize} && y_1 - y_2 \\ &\text{subject to} && 0.5y_1 + 0.2y_2 \leq 1 && 1 = 1 && (1) \\ &&& -0.1y_1 + 0.3y_2 \leq 1 && -0.2 \leq 1 && (2) \\ &&& 0.1y_1 + 0.1y_2 \leq 1 && 0.2 \leq 1 && (3) \\ &&& y_1 \geq 0 && 2 \geq 0 && (4) \\ &&& y_2 \geq 0 && 0 = 0 && (5) \end{aligned}$$

The 1<sup>st</sup> and 5<sup>th</sup> constraints are binding and the others are non-binding. Therefore, the targets for NNs can be written in the vector form as:

$$\mathbf{t} = [1 \quad 0 \quad 0 \quad 0 \quad 1]^T$$

The training and testing sets for NNs are generated this way. In the next part, we will describe the design of NNs.



### 3.2 The Architecture of Neural Networks

In this research, the back-propagation (BP) learning algorithm is used to train the three-layer multilayer perceptron (MLP).

The input patterns for training NNs ( $\mathbf{xn}$ ) are composed of the coefficient of the objective function, the coefficient of constraints and the right-hand-side constants of LP. Therefore, the number of input neurons is specified by the dimension of the input vector ( $\mathbf{xn}$ ).

To identify that each constraint of the LP in  $(m \times n)$  dimensions is binding or non-binding, we use  $m+n$  NNs with one output to adjust the weight in accordance with an error from only one constraint instead of all  $m+n$  constraints. By reducing the target size for training NNs, the network error can converge to an acceptable threshold faster.

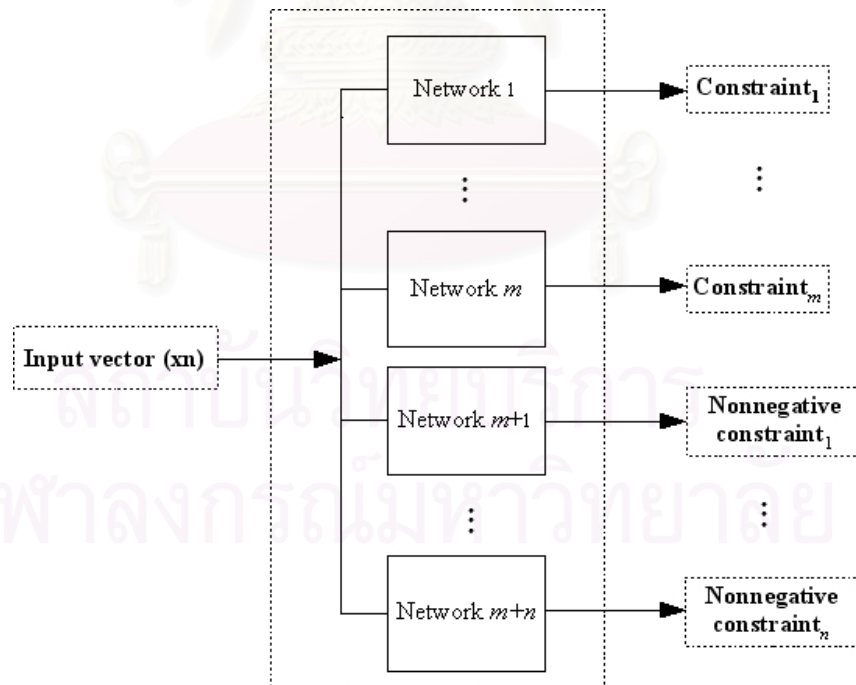


Figure 3.5:  $m + n$  NNs for the LP in  $(m \times n)$  dimensions.

The number of hidden nodes is a crucial parameter of a feed forward multilayer perceptron. An NN with too many hidden neurons may over fit the data, causing poor classification on unseen data, while too few hidden units under fit the model, and is not sufficiently accurate.

In this research, we determine the appropriate number of hidden neurons by increasing them based on the exponential function,  $2^j$  where  $2^j$  is equal to or larger than the number of input nodes.

Suppose that the input, hidden, output neurons are equal to  $n_1, n_2$  and  $n_3$  respectively. Therefore, one NN has

$$n_1 = |\mathbf{x}\mathbf{n}| = m + n + mn,$$

$$n_2 = 2^j, \text{ where } j \geq \log_2(m + n + mn) \text{ and } n_3 = 1.$$

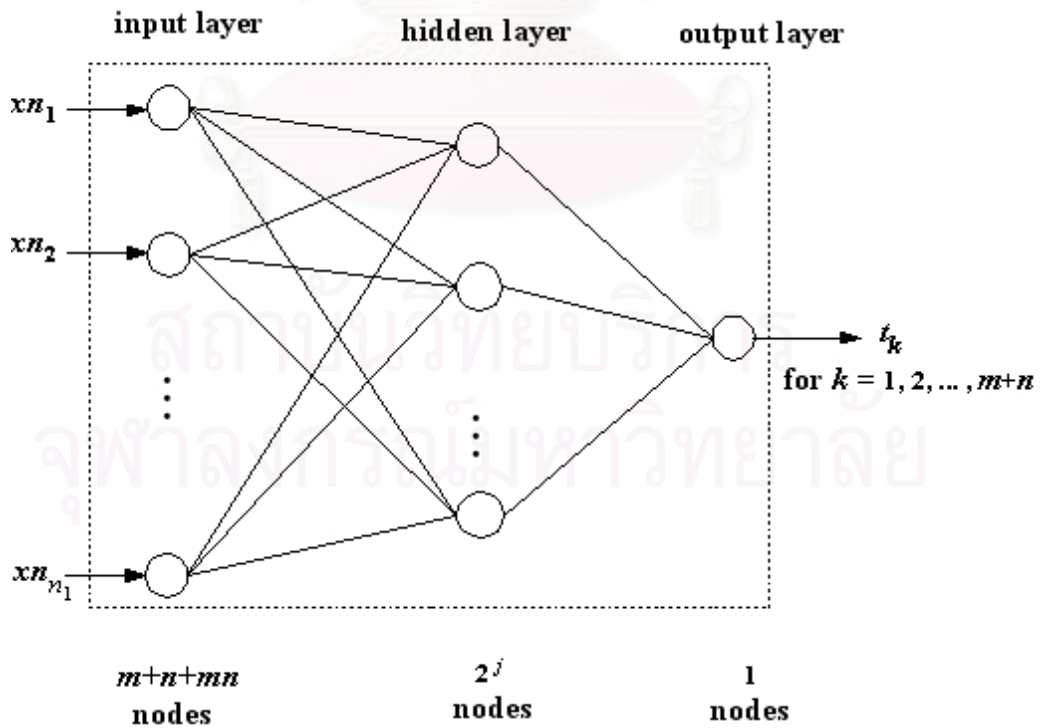


Figure 3.6: The  $k^{th}$  NN for the  $k^{th}$  constraint.

**Example 3.3.** For our previous example (**P**), the input and target vectors can be written as:

$$\mathbf{xn} = [1 \quad -1 \quad 0.5 \quad 0.2 \quad 1 \quad -0.1 \quad 0.3 \quad 1 \quad 0.1 \quad 0.1 \quad 1]^T$$

and  $\mathbf{t} = [1 \quad 0 \quad 0 \quad 0 \quad 1]^T$

From the above LP, the problem involves five constraints. We also need five corresponding NNs where the  $k^{\text{th}}$  NN identifies that the  $k^{\text{th}}$  constraint is binding or non-binding, for  $k = 1, \dots, 5$ .

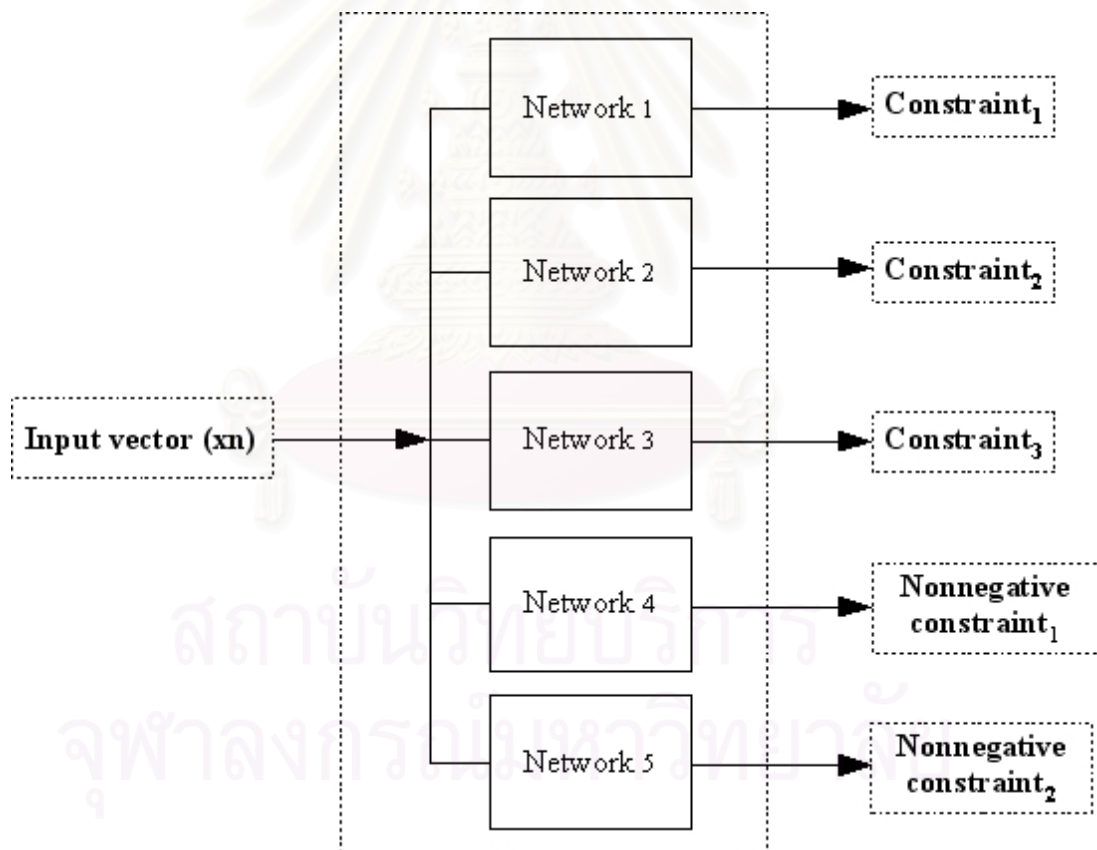


Figure 3.7: Five NNs for the problem (**P**).

For the hidden neurons,  $2^j$ ,  $j$  starts from 4 because  $2^4 = 16$  is larger than the dimension of input patterns ( $|\mathbf{xn}| = 11$ ).

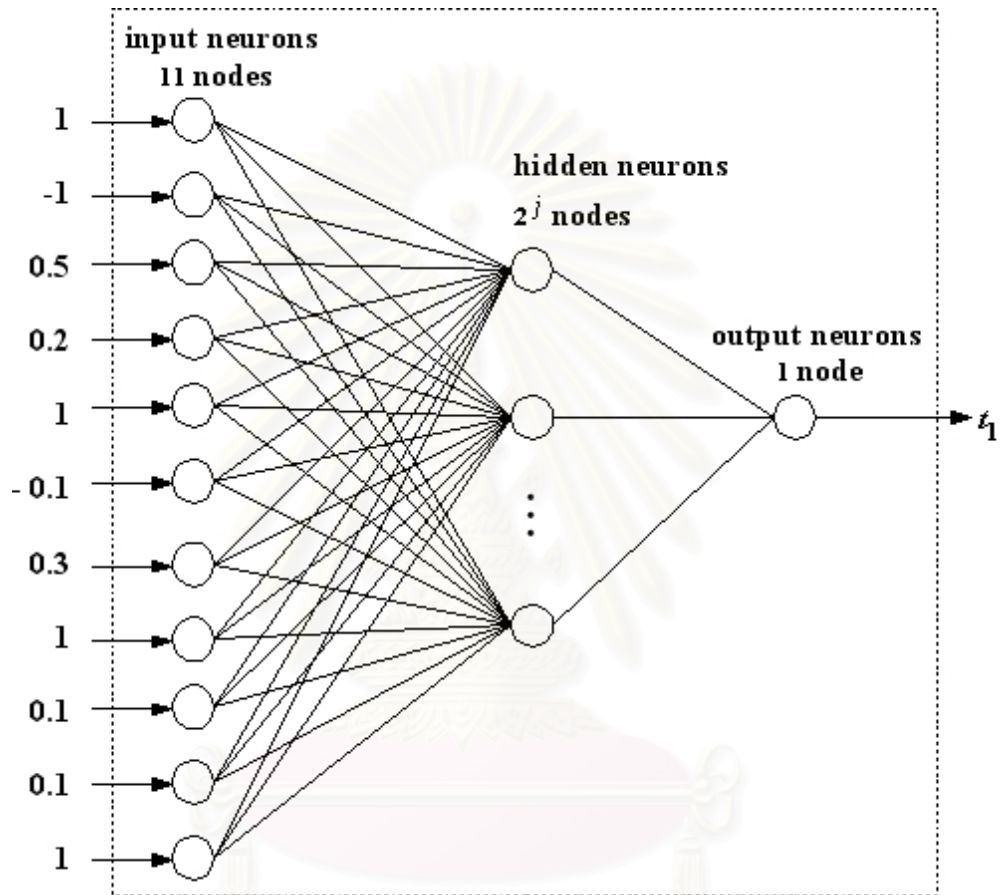


Figure 3.8: The structure of the 1<sup>st</sup> NN for the constraint  $0.5y_1 + 0.2y_2 \leq 1$ .

The target of the 1<sup>st</sup> NNs is equal to 1 which means the first constraint ( $0.5y_1 + 0.2y_2 \leq 1$ ) is binding.

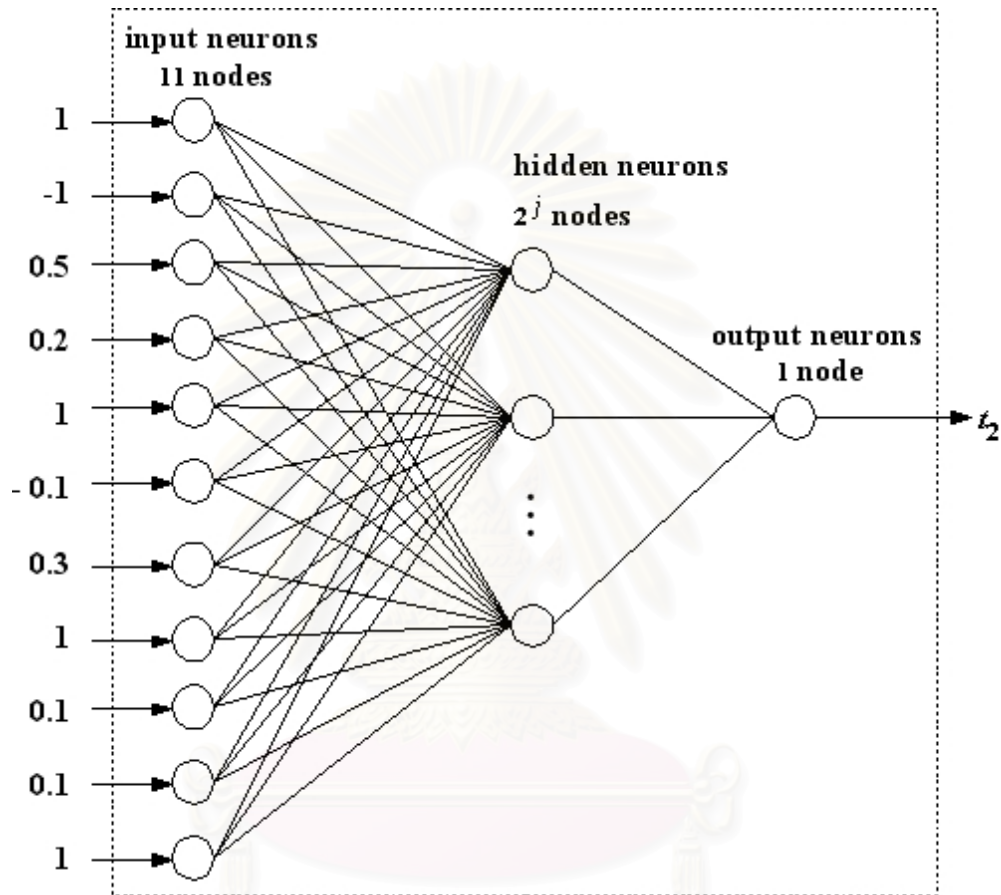


Figure 3.9: The structure of the 2<sup>nd</sup> NN for the constraint  $-0.1y_1 + 0.3y_2 \leq 1$ .

The target of the 2<sup>nd</sup> NNs is equal to 0 which means the second constraint ( $-0.1y_1 + 0.3y_2 \leq 1$ ) is non-binding.

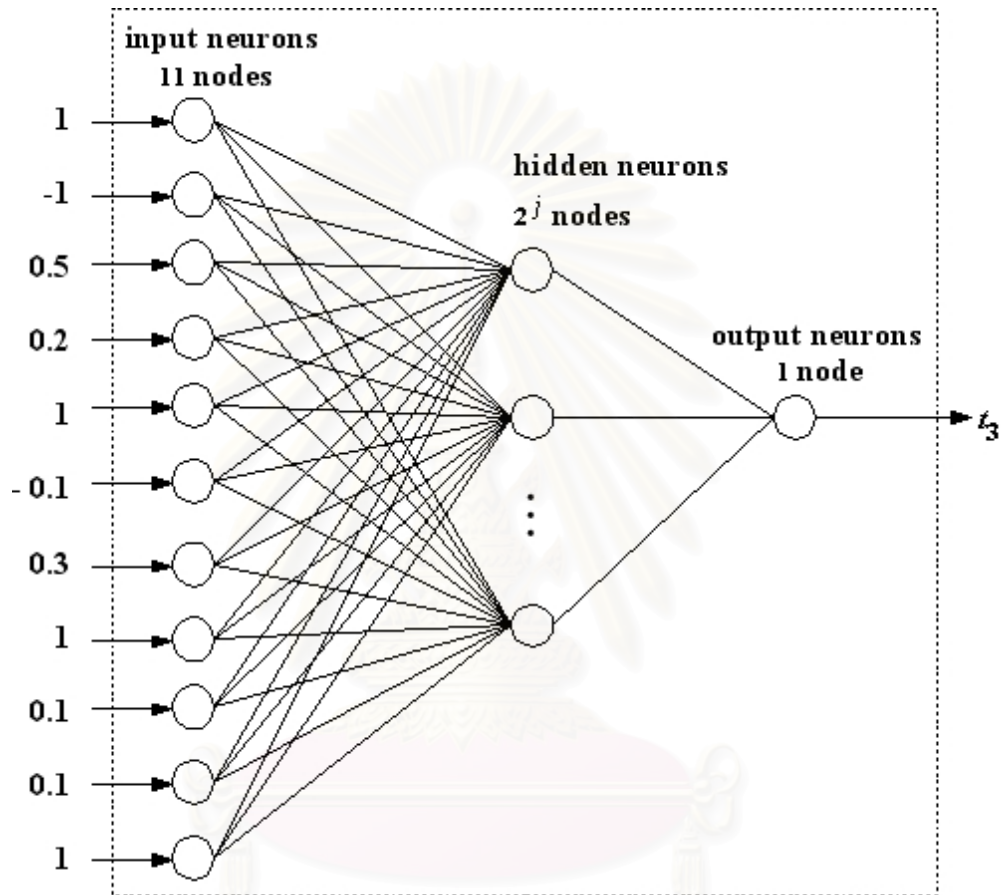


Figure 3.10: The structure of the 3<sup>rd</sup> NN for the constraint  $0.1y_1 + 0.1y_2 \leq 1$ .

The target of the 3<sup>rd</sup> NNs is equal to 0 which means the third constraint ( $0.1y_1 + 0.1y_2 \leq 1$ ) is non-binding.

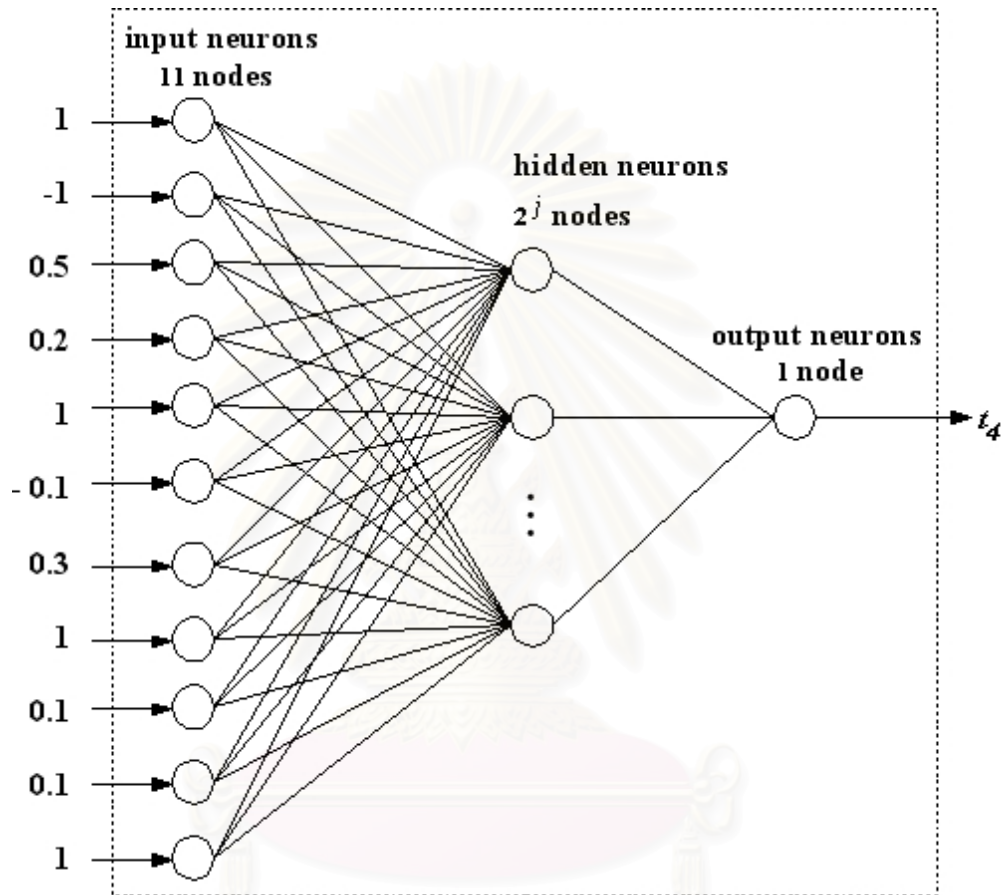


Figure 3.11: The structure of the 4<sup>th</sup> NN for the nonnegative constraint  $x_1 \geq 0$ .

The target of the 4<sup>th</sup> NNs is equal to 0 which means the nonnegative constraint ( $x_1 \geq 0$ ) is non-binding.

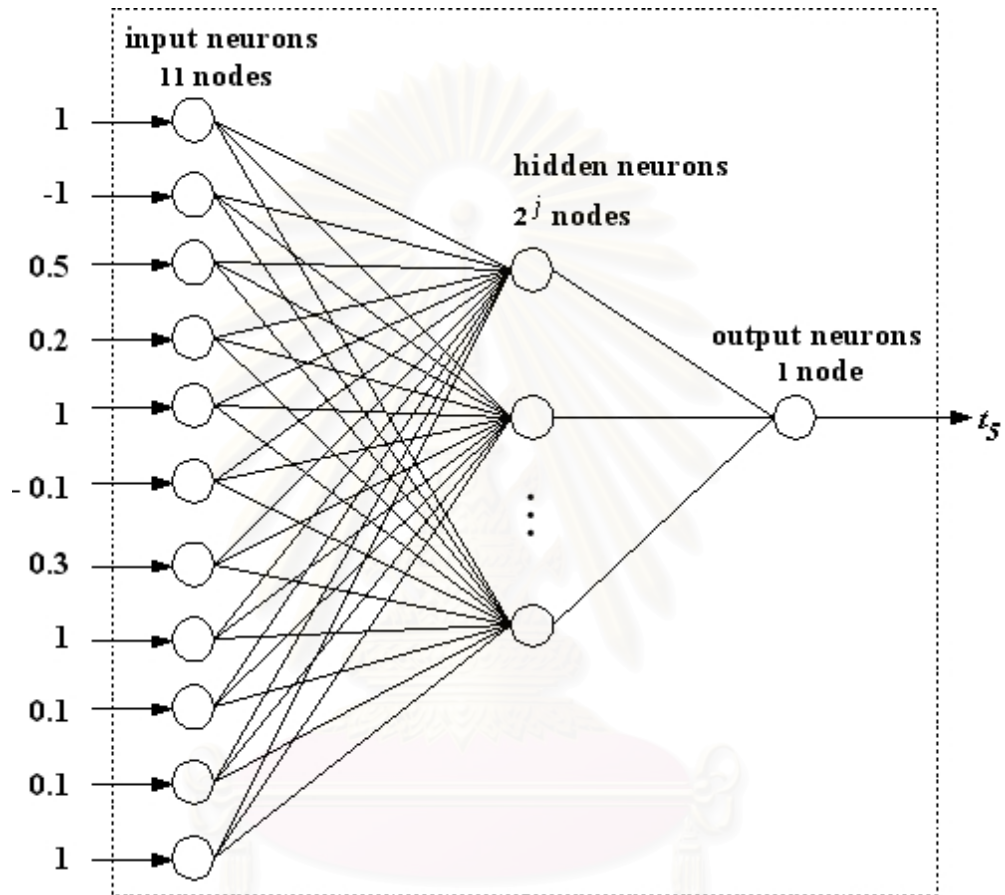


Figure 3.12: The structure of the 5<sup>th</sup> NN for the nonnegative constraint  $x_2 \geq 0$ .

The target of the 5<sup>th</sup> NNs is equal to 1 which means the nonnegative constraint ( $x_2 \geq 0$ ) is binding.



As for the activation function, we use the sigmoidal function because the target of our patterns is only 0 or 1. After the training process, its performance and classification capabilities are evaluated. Before comparing with its target, the output from the trained NNs is rounded by the threshold value 0.5 as:

$$\text{output}(\text{net}) = \begin{cases} 1, & \text{if net} \geq 0.5. \\ 0, & \text{otherwise.} \end{cases}$$

The predicted results are shown in terms of the accuracy of trained NNs in the next chapter .



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## CHAPTER IV

### RESULT AND CONCLUSION

#### 4.1 Result

In the thesis, we perform our experiments based on Linux operating system with the CPU Intel Pentium IV 2 GHz and RAM 256 MB using the SNNS software.

The problem size of LP is fixed to  $m \times n$  dimensions where  $n$  is the number of decision variables varied from 2 to 4 and  $m$  is the number of constraints varied from  $n$  to  $n + 2$ . Therefore, we have nine different LP sizes, namely  $2 \times 2$ ,  $3 \times 2$ ,  $4 \times 2$ ,  $3 \times 3$ ,  $4 \times 3$ ,  $5 \times 3$ ,  $4 \times 4$ ,  $5 \times 4$ , and  $6 \times 4$ .

For training NNs, the training sets contain 10,000 input vectors. The maximum epochs and the acceptable error (sum square error: SSE) are set to 50,000 and  $10^{-6}$ , respectively.

The learning process of the back propagation algorithm is terminated when either the iteration exceeded (learning at the maximum iteration) or the network error was below acceptable threshold (learning successfully).

If NN can learn successfully, the newly 1,000 generated patterns will be simulated to estimate the accuracy of the unseen data of the NN.

In our experiment, each binding and non-binding constraint at the optimal solution is treated differently. Since our aim of the research is to identify non-binding constraints, the accumulated error is defined as the incorrectly classifying the binding constraint as the non-binding output.

To clarify this concept, consider the following example.

**Example 4.1.** Consider the previous problem **P**,

$$\begin{aligned}
 & \text{maximize} && y_1 - y_2 \\
 & \text{subject to} && 0.5y_1 + 0.2y_2 \leq 1 && (1) \\
 & && -0.1y_1 + 0.3y_2 \leq 1 && (2) \\
 & && 0.1y_1 + 0.1y_2 \leq 1 && (3) \\
 & && y_1 \geq 0 && (4) \\
 & && y_2 \geq 0 && (5)
 \end{aligned}$$

$$\mathbf{xn} = [1 \quad -1 \quad 0.5 \quad 0.2 \quad 1 \quad -0.1 \quad 0.3 \quad 1 \quad 0.1 \quad 0.1 \quad 1]^T$$

$$\mathbf{t} = [1 \quad 0 \quad 0 \quad 0 \quad 1]^T$$

This problem involves five constraints, we thus have five corresponding NNs. At the optimal solution, the 1<sup>st</sup> and 5<sup>th</sup> constraints are binding and the others are non-binding.

The output after  $\mathbf{xn}$  is fed to the NNs of the 1<sup>st</sup> and 5<sup>th</sup> NNs must not be equal to 0, otherwise the optimal solution of the original LP may not reach the actual optimal solution. The NN is said to be incorrectly classified.

In contrast, the output of the other NNs can be equal to 0 or 1 because the other constraints are non-binding. Whether we eliminate them or not, the optimal solution of this updated problem will be the same as the original problem.

The accuracy of the  $j^{\text{th}}$  NN can be denoted by

$$\text{Accuracy}(\text{NN}_j) = \frac{\text{The number of correctly predicted patterns}}{\text{Total number of testing patterns}}.$$

If our experiment, we train NNs with the different number of hidden nodes. Since the accuracy of trained NNs on testing data is more important than the training time, the NN with the appropriate hidden nodes that has the best accuracy will be selected. If there are two or more NNs having the same accuracy, the NN with the shortest running time is selected.

Suppose that the selected NN for the  $k^{\text{th}}$  constraint is denoted by  $\text{NN}_k$ , for  $k = 1, \dots, m + n$ .

The average accuracy of the  $m + n$  NNs is evaluated as:

$$\text{Average accuracy} = \frac{\sum_{k=1}^{m+n} \text{Accuracy}(\text{NN}_k)}{m + n}$$

Note that if all  $m + n$  NNs predict the correct non-binding constraints then the optimal solution of the eliminated non-binding LP is the same as the original problem.

However, if one of the NNs misclassifies the binding constraint as the non-binding one then the optimal solution of the eliminated non-binding LP does not guarantee to be the same as the original LP.

Therefore, we define the overall accuracy of all  $m + n$  NNs as

$$\text{Overall accuracy} = \frac{\text{The number of LP that all } m + n \text{ selected NNs predict correctly}}{\text{Total number of testing patterns}}$$

Our first learned NNs composes 2 + 2 NNs for the LP in the  $2 \times 2$  dimensions.

The input and target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad a_{11} \quad a_{12} \quad b_1 \quad a_{21} \quad a_{22} \quad b_2]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4]^T$$

The problem involves 4 constraints, we thus have 4 corresponding NNs. Each of them has 8 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^3$  to  $2^6$ .

The experimental results is presented in Table 4.1.

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	8	50,000	-	4.48369	-
	16	15,217	1,154.40	-	99.80
	<b>*32</b>	<b>4,198</b>	<b>591.89</b>	-	<b>99.90</b>
	64	2,315	632.95	-	99.70
2	<b>*8</b>	<b>19,890</b>	<b>960.68</b>	-	<b>99.90</b>
	16	3,807	291.88	-	99.60
	32	2,527	363.75	-	99.60
	64	1,732	467.80	-	99.50
$x_1 \geq 0$	8	50,000	-	9.80923	-
	16	10,004	759.08	-	99.80
	32	3,523	491.70	-	99.70
	<b>*64</b>	<b>2,279</b>	<b>606.59</b>	-	<b>100.00</b>
$x_2 \geq 0$	<b>8</b>	<b>2,369</b>	<b>*119.07</b>	-	<b>99.90</b>
	16	1,769	144.46	-	99.90
	32	1,409	207.07	-	99.60
	64	1,135	306.05	-	99.90

\* marks the selected  $NN_k$  for  $k = 1, \dots, 4$ .

Table 4.1: NNs for the LP in  $m = 2, n = 2$

Table 4.1 shows that the accuracy of the NNs with 8 hidden neurons for the 1<sup>st</sup> and  $x_1 \geq 0$  constraints are not evaluated (-) because the NNs learn unsuccessfully. They reach the maximum iteration (50,000 epochs) due to having too few hidden neurons to capture the complexity of training patterns.

The NN with the appropriate hidden nodes for the  $k^{\text{th}}$  constraint,  $k = 1, \dots, 4$ , is selected from Table 4.1 as shown in Table 4.2

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	32	591.89	99.90
2	NN <sub>2</sub>	8	960.68	99.90
$x_1 \geq 0$	NN <sub>3</sub>	64	606.59	100.00
$x_2 \geq 0$	NN <sub>4</sub>	8	119.07	99.90
<b>Average accuracy</b>				<b>99.93</b>

Table 4.2: Selected NNs for the LP in  $m = 2, n = 2$

After checking the number of LP that all four selected NNs predict correctly, we get the overall accuracy is equal to 99.7%. This means that there are 0.03% of LP in testing patterns misclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

The second learned NN composes of 3 + 2 NNs for the LP in the  $3 \times 2$  dimensions. The input and target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad a_{11} \quad a_{12} \quad b_1 \quad a_{21} \quad a_{22} \quad b_2 \quad a_{31} \quad a_{32} \quad b_3]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5]^T$$

The problem involves 5 constraints, we thus have 5 corresponding NNs. Each of them has 11 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^4$  to  $2^7$ .

The experimental results is presented in Table 4.3.

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	16	50,000	-	68.51823	-
	32	50,000	-	16.12836	-
	<b>*64</b>	<b>2,780</b>	<b>948.61</b>	-	<b>98.20</b>
	128	1,345	881.92	-	97.80
2	16	50,000	-	59.77855	-
	32	50,000	-	9.67427	-
	<b>*64</b>	<b>5,603</b>	<b>1,887.96</b>	-	<b>98.60</b>
	128	1,763	1,147.47	-	98.10
3	16	50,000	-	44.51794	-
	<b>*32</b>	<b>13,447</b>	<b>2,401.58</b>	-	<b>99.50</b>
	64	4,064	1,374.76	-	99.20
	128	1,432	941.20	-	99.40
$x_1 \geq 0$	16	50,000	-	8.08606	-
	32	15,960	2,734.58	-	98.90
	64	2,526	833.11	-	98.90
	<b>*128</b>	<b>1,005</b>	<b>589.71</b>	-	<b>99.10</b>
$x_2 \geq 0$	<b>*16</b>	<b>2,760</b>	<b>273.82</b>	-	<b>99.80</b>
	32	1,805	311.08	-	99.40
	64	1,398	470.16	-	99.60
	128	1,139	743.05	-	99.50

\* marks the selected NN<sub>k</sub> for  $k = 1, \dots, 5$ .

Table 4.3: NNs for the LP in  $m = 3, n = 2$

Table 4.3 shows that the NNs with 16 hidden neurons for all constraint except the  $x_2 \geq 0$  constraint and 32 hidden neurons for the 1<sup>st</sup> and 2<sup>nd</sup> constraints reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{\text{th}}$  constraint,  $k = 1, \dots, 5$ , is selected from Table 4.3 as shown in Table 4.4

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	64	948.61	98.20
2	NN <sub>2</sub>	64	1,887.96	98.60
3	NN <sub>3</sub>	32	2,401.58	99.50
$x_1 \geq 0$	NN <sub>4</sub>	128	589.71	99.10
$x_2 \geq 0$	NN <sub>5</sub>	16	273.82	99.80
<b>Average accuracy</b>				<b>99.04</b>

Table 4.4: Selected NNs for the LP in  $m = 3, n = 2$

After checking the number of LP that all five selected NNs predict correctly, we get the overall accuracy is equal to 95.20%. This means that there are 4.80% of LP in testing patterns missclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

Next, we train the 4 + 2 NNs for the LP in the  $4 \times 2$  dimensions. The input and target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad a_{11} \quad a_{12} \quad b_1 \quad a_{21} \quad a_{22} \quad b_2 \quad a_{31} \quad a_{32} \quad b_3 \quad a_{41} \quad a_{42} \quad b_4]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6]^T$$

The problem involves 6 constraints, we thus have 6 corresponding NNs. Each of them has 14 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^4$  to  $2^7$ .

The experimental results is presented in Table 4.5.



Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	16	50,000	-	220.01929	-
	32	50,000	-	90.70090	-
	64	22,158	8081.71	-	96.80
	<b>*128</b>	<b>2,162</b>	<b>1,564.69</b>	-	<b>96.90</b>
2	16	50,000	-	144.23007	-
	32	50,000	-	42.77959	-
	64	10,053	3,711.40	-	97.50
	<b>*128</b>	<b>2,335</b>	<b>1,719.71</b>	-	<b>97.80</b>
3	16	50,000	-	101.22366	-
	32	50,000	-	28.36635	-
	<b>64</b>	<b>2,831</b>	<b>*1,041.43</b>	-	<b>97.90</b>
	128	7,520	5,482.10	-	97.90
4	16	50,000	-	51.59223	-
	32	21,346	4,030.50	-	98.20
	64	5,272	1,945.18	-	98.30
	<b>*128</b>	<b>2,183</b>	<b>1,574.30</b>	-	<b>98.70</b>
$x_1 \geq 0$	16	50,000	-	96.61501	-
	32	50,000	-	15.81535	-
	<b>*64</b>	<b>2,781</b>	<b>1,033.00</b>	-	<b>98.20</b>
	128	1,281	932.60	-	97.50
$x_2 \geq 0$	<b>16</b>	<b>1,677</b>	<b>*173.11</b>	-	<b>99.50</b>
	32	963	182.12	-	99.40
	64	904	384.64	-	99.50
	128	485	357.78	-	99.50

\* marks the selected  $NN_k$  for  $k = 1, \dots, 6$ .

Table 4.5: NNs for the LP in  $m = 4, n = 2$

Table 4.5 shows that the NNs with 16 hidden neurons for all constraint except the  $x_2 \geq 0$  constraint and 32 hidden neurons for all constraint except the 4<sup>th</sup> and  $x_2 \geq 0$  constraints reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{\text{th}}$  constraint,  $k = 1, \dots, 6$ , is selected from Table 4.5 as shown in Table 4.6

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	128	1,564.69	96.90
2	NN <sub>2</sub>	128	1,719.71	97.80
3	NN <sub>3</sub>	64	1,041.43	97.90
4	NN <sub>4</sub>	128	1,574.30	98.70
$x_1 \geq 0$	NN <sub>5</sub>	64	1,033.00	98.20
$x_2 \geq 0$	NN <sub>6</sub>	16	173.11	99.50
<b>Average accuracy</b>				<b>98.17</b>

Table 4.6: Selected NNs for the LP in  $m = 4, n = 2$

After checking the number of LP that all six selected NNs predict correctly, we get the overall accuracy is equal to 89.40%. This means that there are 10.60% of LP in testing patterns missclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

The next learned NNs composes of 3+3 NNs for the LP in the  $3 \times 3$  dimensions. The input and target vectors can be represented by

$$\mathbf{x}\mathbf{n} = [c_1 \quad c_2 \quad c_3 \quad a_{11} \quad a_{12} \quad a_{13} \quad b_1 \quad a_{21} \quad a_{22} \quad a_{23} \quad b_2 \quad a_{31} \quad a_{32} \quad a_{33} \quad b_3]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6]^T$$

The problem involves 6 constraints, we thus have 6 corresponding NNs. Each of them has 15 inputs ( $|\mathbf{x}\mathbf{n}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^4$  to  $2^8$ .

The experimental results is presented in Table 4.7.

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	16	50,000	-	323.92490	-
	32	50,000	-	201.72466	-
	64	50,000	-	33.36899	-
	<b>*128</b>	<b>3,688</b>	<b>2,659.00</b>	-	<b>95.80</b>
	256	3,023	4,204.74	-	95.30
2	16	50,000	-	302.52875	-
	32	50,000	-	201.60039	-
	64	50,000	-	26.59995	-
	128	2,992	2,127.22	-	94.80
	<b>*256</b>	<b>2,097</b>	<b>2,881.96</b>	-	<b>95.00</b>
3	16	50,000	-	302.52875	-
	32	50,000	-	180.22064	-
	64	50,000	-	26.13642	-
	128	2,751	1,941.99	-	95.20
	<b>*256</b>	<b>1,545</b>	<b>2,168.17</b>	-	<b>95.70</b>
$x_1 \geq 0$	16	50,000	-	346.91119	-
	32	50,000	-	224.78435	-
	64	50,000	-	27.86832	-
	<b>*128</b>	<b>2,397</b>	<b>1,699.12</b>	-	<b>95.10</b>
	256	688	965.54	-	94.30
$x_2 \geq 0$	16	50,000	-	312.15701	-
	32	50,000	-	192.91785	-
	64	50,000	-	9.30723	-
	<b>*128</b>	<b>2,493</b>	<b>1,771.57</b>	-	<b>95.90</b>
	256	2,799	4,025.36	-	94.60
$x_3 \geq 0$	16	50,000	-	203.40755	-
	32	50,000	-	76.81271	-
	64	10,898	3,959.65	-	96.40
	128	3,181	2,245.13	-	95.80
	<b>256</b>	<b>2,104</b>	<b>*2,993.36</b>	-	<b>96.40</b>

\* marks the selected  $NN_k$  for  $k = 1, \dots, 6$ .

Table 4.7: NNs for the LP in  $m = 3, n = 3$

Table 4.7 shows that the NNs with 16, 32, and 64 hidden neurons for all constraint except the NNs with 64 hidden neurons for the  $x_3 \geq 0$  constraint reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{th}$  constraint,  $k = 1, \dots, 6$ , is selected from Table 4.7 as shown in Table 4.8

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	128	2,659.00	95.80
2	NN <sub>2</sub>	256	2,881.96	95.00
3	NN <sub>3</sub>	256	2,168.17	95.70
$x_1 \geq 0$	NN <sub>4</sub>	128	1,699.12	95.10
$x_2 \geq 0$	NN <sub>5</sub>	128	1,771.57	95.90
$x_3 \geq 0$	NN <sub>6</sub>	256	2,993.36	96.40
<b>Average accuracy</b>				<b>95.65</b>

Table 4.8: Selected NNs for the LP in  $m = 3, n = 3$

After checking the number of LP that all six selected NNs predict correctly, we get the overall accuracy is equal to 74.90%. This means that there are 25.10% of LP in testing patterns misclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

The next learned NNs composes of 4+3 NNs for the LP in the  $4 \times 3$  dimensions. The input and target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad c_3 \quad a_{11} \quad a_{12} \quad a_{13} \quad b_1 \quad a_{21} \quad a_{22} \quad a_{23} \quad b_2$$

$$a_{31} \quad a_{32} \quad a_{33} \quad b_3 \quad a_{41} \quad a_{42} \quad a_{43} \quad b_4]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7]^T$$

The problem involves 7 constraints, we thus have 7 corresponding NNs. Each of them has 19 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^5$  to  $2^8$ .

The experimental results is presented in Table 4.9.

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	32	50,000	-	313.71573	-
	64	50,000	-	92.86191	-
	128	2,426	1,950.49	-	91.50
	<b>*256</b>	<b>2,223</b>	<b>3,602.47</b>	-	<b>92.50</b>
2	32	50,000	-	294.47223	-
	64	50,000	-	64.82623	-
	<b>128</b>	<b>1,846</b>	<b>*1,499.14</b>	-	<b>91.20</b>
	256	1,325	2,135.12	-	91.20
3	32	50,000	-	250.20413	-
	64	50,000	-	50.28577	-
	<b>*128</b>	<b>1,817</b>	<b>1,477.70</b>	-	<b>94.40</b>
	256	1,207	1,933.6	-	94.30
4	32	50,000	-	159.51068	-
	64	50,000	-	1.04256	-
	128	1,232	991.65	-	94.50
	<b>*256</b>	<b>1,202</b>	<b>1,938.42</b>	-	<b>95.30</b>
$x_1 \geq 0$	32	50,000	-	308.46732	-
	64	50,000	-	61.96968	-
	<b>*128</b>	<b>2,314</b>	<b>1,858.59</b>	-	<b>92.20</b>
	256	2,080	3,332.15	-	91.40
$x_2 \geq 0$	32	50,000	-	267.73502	-
	64	50,000	-	54.29050	-
	<b>*128</b>	<b>1,581</b>	<b>1,280.21</b>	-	<b>95.90</b>
	256	1,372	2,241.31	-	91.80
$x_3 \geq 0$	32	50,000	-	80.38234	-
	64	4,176	1,704.03	-	95.70
	128	1,062	861.76	-	95.20
	<b>*256</b>	<b>1,083</b>	<b>1,714.70</b>	-	<b>96.10</b>

\* marks the selected  $NN_k$  for  $k = 1, \dots, 7$ .

Table 4.9: NNs for the LP in  $m = 4, n = 3$

Table 4.9 shows that the NNs with 32, and 64 hidden neurons for all constraint except the NNs with 64 hidden neurons for the  $x_3 \geq 0$  constraint reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{th}$  constraint,  $k = 1, \dots, 7$ , is selected from Table 4.9 as shown in Table 4.10

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	256	3,602.47	92.50
2	NN <sub>2</sub>	128	1,499.14	91.20
3	NN <sub>3</sub>	128	1,477.70	94.40
4	NN <sub>4</sub>	256	1,938.42	95.30
$x_1 \geq 0$	NN <sub>5</sub>	128	1,858.59	92.20
$x_2 \geq 0$	NN <sub>6</sub>	128	1,280.21	95.90
$x_3 \geq 0$	NN <sub>7</sub>	256	1,714.70	96.10
<b>Average accuracy</b>				<b>93.94</b>

Table 4.10: Selected NNs for the LP in  $m = 4, n = 3$

After checking the number of LP that all seven selected NNs predict correctly, we get the overall accuracy is equal to 59.30%. This means that there are 40.70% of LP in testing patterns missclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

The next learned NNs composes of 5+3 NNs for the LP in the  $5 \times 3$  dimensions. The input and the target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad c_3 \quad a_{11} \quad a_{12} \quad a_{13} \quad b_1 \quad a_{21} \quad a_{22} \quad a_{23} \quad b_2 \quad a_{31} \quad a_{32} \quad a_{33} \quad b_3$$

$$a_{41} \quad a_{42} \quad a_{43} \quad b_4 \quad a_{51} \quad a_{52} \quad a_{53} \quad b_5]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8]^T$$

The problem involves 8 constraints, we thus have 8 corresponding NNs. Each of them has 23 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^5$  to  $2^8$ .

The experimental results is presented in Table 4.11.

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	32	50,000	-	395.51593	-
	64	50,000	-	120.07862	-
	128	1,652	1,504.46	-	88.40
	<b>*256</b>	<b>1,040</b>	<b>1,841.59</b>	-	<b>90.00</b>
2	32	50,000	-	334.06140	-
	64	50,000	-	101.96539	-
	128	2,685	2,440.01	-	91.20
	<b>*256</b>	<b>1,248</b>	<b>2,210.47</b>	-	<b>91.80</b>
3	32	50,000	-	315.96420	-
	64	50,000	-	75.28999	-
	128	1,483	1,346.76	-	91.10
	<b>*256</b>	<b>835</b>	<b>1,517.04</b>	-	<b>92.10</b>
4	32	50,000	-	240.11996	-
	64	50,000	-	4.20835	-
	128	1,374	1,236.81	-	94.00
	<b>*256</b>	<b>756</b>	<b>1,371.65</b>	-	<b>95.40</b>
5	32	50,000	-	92.58267	-
	<b>*64</b>	<b>4,067</b>	<b>1,867.23</b>	-	<b>96.70</b>
	128	893	806.19	-	95.40
	256	485	879.64	-	95.80
$x_1 \geq 0$	32	50,000	-	335.90030	-
	64	50,000	-	81.96362	-
	<b>*128</b>	<b>1,930</b>	<b>1,737.46</b>	-	<b>91.60</b>
	256	1,167	2,093.19	-	91.10
$x_2 \geq 0$	32	50,000	-	285.33115	-
	64	50,000	-	53.00173	-
	128	1,243	1,127.71	-	93.40
	<b>*256</b>	<b>1,004</b>	<b>1,812.19</b>	-	<b>93.60</b>
$x_3 \geq 0$	32	50,000	-	42.98532	-
	<b>*64</b>	<b>1,136</b>	<b>522.56</b>	-	<b>96.10</b>
	128	516	467.45	-	95.60
	256	337	607.67	-	95.90

\* marks the selected  $NN_k$  for  $k = 1, \dots, 8$ .

Table 4.11: NNs for the LP in  $m = 5, n = 3$

Table 4.11 shows that the NNs with 32 ,and 64 hidden neurons for all constraint except the NNs with 64 hidden neurons for the 5<sup>th</sup> and  $x_3 \geq 0$  constraints reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{th}$  constraint,  $k = 1, \dots, 8$ , is selected from Table 4.11 as shown in Table 4.12

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	256	1,841.59	90.00
2	NN <sub>2</sub>	256	2,210.47	91.80
3	NN <sub>3</sub>	256	1,517.04	92.10
4	NN <sub>4</sub>	256	1,371.65	95.40
5	NN <sub>5</sub>	64	1,867.23	96.70
$x_1 \geq 0$	NN <sub>6</sub>	128	1,737.46	91.60
$x_2 \geq 0$	NN <sub>7</sub>	256	1,812.19	93.60
$x_3 \geq 0$	NN <sub>8</sub>	64	522.56	96.10
<b>Average accuracy</b>				<b>93.41</b>

Table 4.12: Selected NNs for the LP in  $m = 5, n = 3$

After checking the number of LP that all eight selected NNs predict correctly, we get the overall accuracy is equal to 54.90%. This means that there are 45.10% of LP in testing patterns missclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

The next learned NNs composes of 4+4 NNs for the LP in the  $4 \times 4$  dimensions. The input and target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad c_3 \quad c_4 \quad a_{11} \quad a_{12} \quad a_{13} \quad a_{14} \quad b_1 \quad a_{21} \quad a_{22} \quad a_{23} \quad a_{24} \quad b_2$$

$$a_{31} \quad a_{32} \quad a_{33} \quad a_{34} \quad b_3 \quad a_{41} \quad a_{42} \quad a_{43} \quad a_{44} \quad b_4]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8]^T$$

The problem involves 8 constraints, we thus have 8 corresponding NNs. Each of them has 24 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^5$  to  $2^8$ .



The experimental results is presented in Table 4.13.

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	32	50,000	-	351.40891	-
	64	50,000	-	47.71244	-
	128	1,488	1,422.90	-	88.90
	<b>*256</b>	<b>877</b>	<b>1,669.31</b>	-	<b>91.20</b>
2	32	50,000	-	334.82507	-
	64	50,000	-	99.72289	-
	<b>*128</b>	<b>1,501</b>	<b>1,429.86</b>	-	<b>90.00</b>
	256	877	1,669.31	-	91.20
3	32	50,000	-	353.66815	-
	64	50,000	-	76.27777	-
	128	1,578	1,505.59	-	91.40
	<b>*256</b>	<b>729</b>	<b>1,391.13</b>	-	<b>92.50</b>
4	32	50,000	-	324.51324	-
	64	50,000	-	55.45377	-
	128	793	761.57	-	90.60
	<b>*256</b>	<b>830</b>	<b>1,585.39</b>	-	<b>92.60</b>
$x_1 \geq 0$	32	50,000	-	334.82507	-
	64	50,000	-	69.87735	-
	<b>*128</b>	<b>1,367</b>	<b>1,342.07</b>	-	<b>92.50</b>
	256	840	1,600.71	-	92.10
$x_2 \geq 0$	32	50,000	-	360.21011	-
	64	50,000	-	69.50634	-
	<b>*128</b>	<b>1,154</b>	<b>1,220.00</b>	-	<b>90.60</b>
	256	804	1,570.97	-	88.80
$x_3 \geq 0$	32	50,000	-	328.40054	-
	64	50,000	-	51.65095	-
	<b>*128</b>	<b>1,011</b>	<b>968.51</b>	-	<b>91.20</b>
	256	829	1,579.03	-	90.50
$x_4 \geq 0$	32	50,000	-	244.22375	-
	64	50,000	-	7.52560	-
	<b>*128</b>	<b>999</b>	<b>952.96</b>	-	<b>91.30</b>
	256	944	1,818.33	-	90.20

\* marks the selected  $NN_k$  for  $k = 1, \dots, 8$ .

Table 4.13: NNs for the LP in  $m = 4, n = 4$

Table 4.13 shows that the NNs with 32 and 64 hidden neurons for all constraint reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{th}$  constraint,  $k = 1, \dots, 8$ , is selected from Table 4.13 as shown in Table 4.14

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	256	1,669.31	91.20
2	NN <sub>2</sub>	128	1,429.86	90.00
3	NN <sub>3</sub>	256	1,391.13	92.50
4	NN <sub>4</sub>	256	1,585.39	92.60
$x_1 \geq 0$	NN <sub>5</sub>	128	1,342.07	92.50
$x_2 \geq 0$	NN <sub>6</sub>	128	1,220.00	90.60
$x_3 \geq 0$	NN <sub>7</sub>	128	968.51	91.20
$x_4 \geq 0$	NN <sub>8</sub>	128	952.96	91.30
<b>Average accuracy</b>				<b>91.48</b>

Table 4.14: Selected NNs for the LP in  $m = 4, n = 4$

After checking the number of LP that all eight selected NNs predict correctly, we get the overall accuracy is equal to 44.20%. This means that there are 55.80% of LP in testing patterns missclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

The next learned NNs composes of 5+4 NNs for the LP in the  $5 \times 4$  dimensions.

The input and target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad c_3 \quad c_4 \quad a_{11} \quad a_{12} \quad a_{13} \quad a_{14} \quad b_1 \quad a_{21} \quad a_{22} \quad a_{23} \quad a_{24} \quad b_2$$

$$a_{31} \quad a_{32} \quad a_{33} \quad a_{34} \quad b_3 \quad a_{41} \quad a_{42} \quad a_{43} \quad a_{44} \quad b_4$$

$$a_{51} \quad a_{52} \quad a_{53} \quad a_{54} \quad b_5]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \quad t_9]^T$$

The problem involves 9 constraints, we thus have 9 corresponding NNs. Each of them has 29 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^5$  to  $2^8$ .

The experimental results is presented in Table 4.15.

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	32	50,000	-	402.96899	-
	64	50,000	-	71.98686	-
	128	983	1,063.28	-	87.50
	<b>*256</b>	<b>665</b>	<b>1,430.42</b>	-	<b>89.00</b>
2	32	50,000	-	434.57349	-
	64	50,000	-	114.58553	-
	128	1,895	2,054.32	-	89.60
	<b>*256</b>	<b>815</b>	<b>1,761.21</b>	-	<b>91.70</b>
3	32	50,000	-	399.22968	-
	64	50,000	-	92.48386	-
	128	1,053	1,145.90	-	89.00
	<b>*256</b>	<b>558</b>	<b>1,207.57</b>	-	<b>89.30</b>
4	32	50,000	-	339.72290	-
	64	50,000	-	40.49747	-
	128	760	832.19	-	83.7
	<b>*256</b>	<b>565</b>	<b>1,224.02</b>	-	<b>88.90</b>
5	32	50,000	-	189.69020	-
	64	9,453	5,213.91	-	92.10
	<b>*128</b>	<b>498</b>	<b>536.48</b>	-	<b>92.40</b>
	256	553	1,194.48	-	91.40
$x_1 \geq 0$	32	50,000	-	380.92621	-
	64	50,000	-	97.01405	-
	128	1,085	1,178.07	-	88.90
	<b>*256</b>	<b>582</b>	<b>1,251.37</b>	-	<b>89.70</b>
$x_2 \geq 0$	32	50,000	-	405.59177	-
	64	50,000	-	66.69518	-
	<b>*128</b>	<b>855</b>	<b>928.62</b>	-	<b>90.80</b>
	256	705	1,527.03	-	88.60
$x_3 \geq 0$	32	50,000	-	367.77017	-
	64	50,000	-	77.19629	-
	128	850	922.95	-	90.10
	<b>*256</b>	<b>655</b>	<b>1,408.90</b>	-	<b>90.80</b>
$x_4 \geq 0$	32	50,000	-	207.37527	-
	64	50,000	-	1.04151	-
	128	571	620.28	-	90.60
	<b>*256</b>	<b>513</b>	<b>1,112.52</b>	-	<b>91.80</b>

\* marks the selected  $NN_k$  for  $k = 1, \dots, 9$ .

Table 4.15: NNs for the LP in  $m = 5, n = 4$

Table 4.15 shows that the NNs with 32 and 64 hidden neurons for all constraint except the NNs with 64 hidden neurons for the 5<sup>th</sup> constraint reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{th}$  constraint,  $k = 1, \dots, 9$ , is selected from Table 4.15 as shown in Table 4.16

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	256	1,430.42	89.00
2	NN <sub>2</sub>	256	1,761.21	91.70
3	NN <sub>3</sub>	256	1,207.57	89.30
4	NN <sub>4</sub>	256	1,224.02	88.90
5	NN <sub>5</sub>	128	536.48	92.40
$x_1 \geq 0$	NN <sub>6</sub>	256	1,251.37	89.70
$x_2 \geq 0$	NN <sub>7</sub>	128	928.62	90.80
$x_3 \geq 0$	NN <sub>8</sub>	256	1,408.90	90.80
$x_4 \geq 0$	NN <sub>9</sub>	256	1,112.52	91.80
<b>Average accuracy</b>				<b>90.49</b>

Table 4.16: Selected NNs for the LP in  $m = 5, n = 4$

After checking the number of LP that all nine selected NNs predict correctly, we get the overall accuracy is equal to 36.00%. This means that there are 64.00% of LP in testing patterns misclassified the binding constraint, which causes inaccurate optimal solution of the original LP.

The last learned NNs composes of 6+4 NNs for the LP in the  $6 \times 4$  dimensions.

The input and target vectors can be represented by

$$\mathbf{xn} = [c_1 \quad c_2 \quad c_3 \quad c_4 \quad a_{11} \quad a_{12} \quad a_{13} \quad a_{14} \quad b_1 \quad a_{21} \quad a_{22} \quad a_{23} \quad a_{24} \quad b_2$$

$$a_{31} \quad a_{32} \quad a_{33} \quad a_{34} \quad b_3 \quad a_{41} \quad a_{42} \quad a_{43} \quad a_{44} \quad b_4$$

$$a_{51} \quad a_{52} \quad a_{53} \quad a_{54} \quad b_5 \quad a_{61} \quad a_{62} \quad a_{63} \quad a_{64} \quad b_6]^T$$

$$\mathbf{t} = [t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6 \quad t_7 \quad t_8 \quad t_9 \quad t_{10}]^T$$

The problem involves 10 constraints, we thus have 10 corresponding NNs. Each of them has 34 inputs ( $|\mathbf{xn}|$ ) and one output. As for the number of hidden neuron, it is varied from  $2^6$  to  $2^8$ .

The experimental results is presented in Table 4.17.

Constraint#	#Hidden	Epochs	Times(sec)	Error(sse)	Accuracy(%)
1	64	50,000	-	78.91619	-
	<b>*128</b>	<b>696</b>	<b>820.74</b>	-	<b>87.30</b>
	256	429	1,125.77	-	87.10
2	64	50,000	-	96.64111	-
	<b>*128</b>	<b>967</b>	<b>1,139.89</b>	-	<b>89.80</b>
	256	417	975.09	-	88.70
3	64	50,000	-	53.38405	-
	128	681	799.97	-	86.60
	<b>*256</b>	<b>427</b>	<b>1,015.72</b>	-	<b>87.30</b>
4	64	50,000	-	13.29587	-
	128	601	715.08	-	88.6
	<b>*256</b>	<b>472</b>	<b>1,097.12</b>	-	<b>90.9</b>
5	64	50,000	-	0.53562	-
	128	483	567.42	-	89.60
	<b>*256</b>	<b>390</b>	<b>919.06</b>	-	91.10
6	64	985	591.98	-	92.30
	<b>128</b>	<b>352</b>	<b>*413.09</b>	-	<b>92.70</b>
	256	326	759.98	-	92.70
$x_1 \geq 0$	64	50,000	-	52.79285	-
	128	1,013	1,186.10	-	88.00
	<b>*256</b>	<b>533</b>	<b>1,258.20</b>	-	<b>88.40</b>
$x_2 \geq 0$	64	50,000	-	54.08652	-
	128	879	1,030.46	-	87.20
	<b>*256</b>	<b>328</b>	<b>790.87</b>	-	<b>88.70</b>
$x_3 \geq 0$	64	50,000	-	1.51025	-
	128	574	673.88	-	88.70
	<b>*256</b>	<b>391</b>	<b>975.63</b>	-	<b>89.50</b>
$x_4 \geq 0$	64	1,863	1,119.96	-	92.80
	128	376	444.15	-	93.90
	<b>*256</b>	<b>334</b>	<b>791.43</b>	-	<b>94.40</b>

\* marks the selected  $NN_k$  for  $k = 1, \dots, 10$ .

Table 4.17: NNs for the LP in  $m = 6, n = 4$

Table 4.17 shows that the NNs with 64 hidden neurons for all constraint except the 6<sup>th</sup> and  $x_4 \geq 0$  constraints reach the maximum iteration (learning unsuccessfully). Therefore, their accuracy is not evaluated (-).

The NN with the appropriate hidden nodes for the  $k^{\text{th}}$  constraint,  $k = 1, \dots, 10$ , is selected from Table 4.17 as shown in Table 4.18

Constraint#	Network	#Hidden	Times(sec)	Accuracy(%)
1	NN <sub>1</sub>	128	820.74	87.30
2	NN <sub>2</sub>	128	1,139.89	89.80
3	NN <sub>3</sub>	256	1,015.72	87.30
4	NN <sub>4</sub>	256	1,097.12	90.90
5	NN <sub>5</sub>	256	919.06	91.10
6	NN <sub>6</sub>	128	413.09	92.70
$x_1 \geq 0$	NN <sub>7</sub>	256	1,258.20	88.40
$x_2 \geq 0$	NN <sub>8</sub>	256	790.87	88.70
$x_3 \geq 0$	NN <sub>9</sub>	256	975.63	89.50
$x_4 \geq 0$	NN <sub>10</sub>	256	791.43	94.40
<b>Average accuracy</b>				<b>90.01</b>

Table 4.18: Selected NNs for the LP in  $m = 6, n = 4$

After checking the number of LP that all ten selected NNs predict correctly, we get the overall accuracy is equal to 27.90%. This means that there are 72.10% of LP in testing patterns misclassified the binding constraint, which causes inaccurate optimal solution of the original LP.



## 4.2 Conclusion

The aim of this research is to use a supervised learning neural network for identifying non-binding constraints in linear programming problems. The back propagation algorithm is selected to train three layer MLPs.

We considered specifically nine different LP sizes:  $2 \times 2$ ,  $3 \times 2$ ,  $4 \times 2$ ,  $3 \times 3$ ,  $4 \times 3$ ,  $5 \times 3$ ,  $4 \times 4$ ,  $5 \times 4$ , and  $6 \times 4$ .

For the LP in the  $m \times n$  dimensions, the problem consists of  $m + n$  constraints, we thus use  $m + n$  NNs with one input to identify each constraint where the  $k^{th}$  NNs identifies the  $k^{th}$  constraints.

After the training process, the NNs will be measured their classification capabilities as the average accuracy and the overall accuracy.

The average accuracy for nine different LP is summarized in Table 4.19.

LP size ( $m \times n$ )	#Input ( $ xn $ )	Average accuracy(%)
$2 \times 2$	8	99.93
$3 \times 2$	11	99.04
$4 \times 2$	14	98.17
$3 \times 3$	15	95.65
$4 \times 3$	19	93.94
$5 \times 3$	23	93.41
$4 \times 4$	24	91.48
$5 \times 4$	29	90.49
$6 \times 4$	34	90.01

Table 4.19: The average accuracy

Table 4.19 shows the decreasing of the average accuracy varied by the number of input nodes. The 90% or above means using the NNs to learn individual non-binding constraint shows acceptable accuracy.

The overall accuracy for nine different LP is summarized in Table 4.20.

LP size ( $m \times n$ )	#NNs	Overall accuracy(%)
$2 \times 2$	4	99.70
$3 \times 2$	5	95.20
$4 \times 2$	6	89.40
$3 \times 3$	6	74.90
$4 \times 3$	7	59.30
$5 \times 3$	8	54.40
$4 \times 4$	8	44.20
$5 \times 4$	9	36.00
$6 \times 4$	10	27.90

Table 4.20: The overall accuracy

Table 4.20 shows the decreasing of the overall accuracy varied by the number of NNs. The small accuracy of NNs shows that the large number of NNs does not help to apply the non-binding elimination of the original LP.

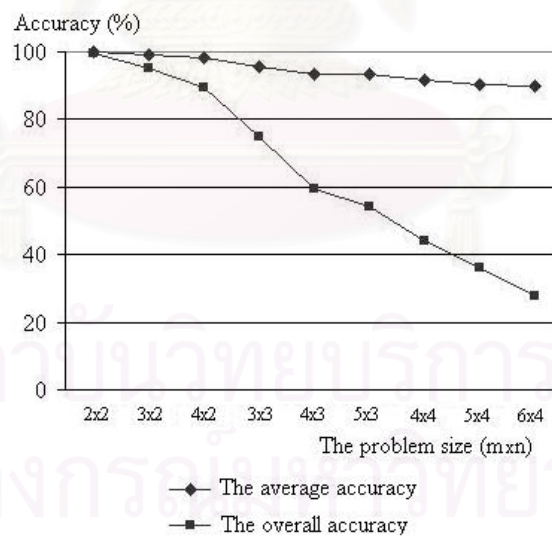


Figure 4.1: Both accuracies decreases according to problem size

The results indicate the impact of this method to both accuracies. The individual accuracy of NNs is slowly decrease while the overall accuracy of NNs is sharply decrease as the problem size becomes larger as shown in Figure 4.1.

An unacceptable overall accuracy of the larger LP shows an inappropriate naive construction of input of the LP cannot be trained for all constraints, only individual one is possible.

This shows that the use of the NN to learn the non-binding constraints is possible. However, the appropriate structure of inputs of the LP is required in order to reach the acceptable overall accuracy.

Note that for our input vector, no interrelationship between coefficients from different rows has been identified. In addition, the distinctions among the objective coefficient and the right-hand-side constant are given according to the index of the input.

In this research, NNs are used as classification tool for identifying non-binding constraints. The back propagation algorithm may be unsuitable for training NNs. Therefore, the future work of this research may concentrate on selecting other algorithms in the training process, such as Radial Basis Function (RBF) or Support Vector Machine (SVM).

However, this method is suitable to use in the small problem size. Moreover, the time for predicting non-binding constraints does not depend on the number of LPs. If we have a lot of LPs with two decision variables and less than four constraints, this method can help us predict non-binding constraints correctly more than 89%.

## References

- [1] Hiller, F. S., and Lieberman, G. J. Introduction to mathematical programming. New York: McGraw-Hill, 1991.
- [2] Dantzig, G. B. Linear programming and extensions. New Jersey: Prentice-Hall, 1963.
- [3] Karmarkar, N. A new polynomial time algorithm for linear programming. *Combinatorica* 4, pp. 373 - 395, 1984.
- [4] Zak, S. H., Upatizing, V., and Hui, S. Solving linear programming problems with neural networks: a comparative study. *IEEE Trans. Neural Networks*, vol. 6, pp. 94 - 104, Jan. 1995.
- [5] Yuan, J. L., and Chiang, H. D. A neural net linear programming solver. *IEEE Trans. Circ. Syst.*, vol. 2, pp. 1117 - 1120, June. 1991.
- [6] Bazaraa, M. S., Jarvis, J. J., and Sherali, H. D. Linear programming and network flow. New York: John Wiley and Sons, 1990.
- [7] Winston, W. L. Operations research: Applications and algorithms. Boston: PWS-Kent, 1991.
- [8] Haykin, S. S. Neural networks: A comprehensive foundation. New Jersey: Prentice-Hall, 1999.
- [9] Philips, D. T., Ravindran, A., and Solberg, J. Operations research: Principles and practice. New York: John Wiley and Sons, 1976.
- [10] Gass, S. I. Linear programming. 5<sup>th</sup> ed. New York: McGraw-Hill, 1994.
- [11] Hagan, M. T., Demuth, H. B., and Beale, M. Neural network design. Boston: PWS Pub. Co., 1995.
- [12] Chester, M. Neural networks: A tutorial. New Jersey: PTR Prentice Hall, 1993.

- [13] Makhorin, A. GNU linear programming kit. Moscow: Moscow Aviation Institute, Russia, 2001.
- [14] Zell, A., and others. Stuttgart neural network simulator. German: The Institute for Parallel and Distributed High Performance Systems, The University of Stuttgart, 1998.



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย



## APPENDICES

สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย

## Appendix A

### GNU Linear Programming Kit: GLPK

GLPK (GNU LinearLinear Programming Kit) is a set of routines written in the ANSI-C programming language and organized in the form of a callable library [13]. It is intended for solving mathematical programming problems. The GLPK package can be downloaded from <http://www.gnu.org/gnu/glpk>.

In order to understand how GLPK can solve the LP, consider the following example:

**Example A.1.** Suppose that we have an LP in the normalized LP form as:

$$\begin{aligned} & \text{maximize} && x_1 - x_2 \\ & \text{subject to} && 0.5x_1 + -1.5x_2 \leq 1 \\ & && -0.6x_1 + 0.5x_2 \leq -1 \\ & && -0.4x_1 + 2x_2 \leq 1 \\ & && x_1, x_2 \geq 0 \end{aligned}$$

We can solve this LP using GLPK API routines by the following C program.

```
/**An example code C for solving the LP that has m=3,n=2***/  
/**Declare the relevant libraries***/  
#include <stdio.h>  
#include <stdlib.h>  
#include "glpk.h"  
int main(void)  
{  
    /**Declare the relevant variables***/  
    LPX *lp;  
    int rn[1+6], cn[1+6];  
    double a[1+6], Z, x1, x2;  
  
    /**Create a LP object that initially is empty***/  
    s1: lp=lp_create_prob();  
  
    /**Add rows and set upper bounds to the problem object***/  
    s2: lp_add_rows(lp,3);  
    s3: lp_set_rows_bnds(lp, 1, LPX_UP, 0.0, 1.0);  
    s4: lp_set_rows_bnds(lp, 2, LPX_UP, 0.0, -1.0);  
    s5: lp_set_rows_bnds(lp, 3, LPX_UP, 0.0, 1.0);  
  
    /**Add columns and set lower bounds to the problem object***/  
    s6: lp_add_cols(lp,2);  
    s7: lp_set_cols_bnds(lp, 1, LPX_LO, 0.0, 0.0);  
    s8: lp_set_cols_bnds(lp, 2, LPX_LO, 0.0, 0.0);
```



```
/**Assign the constraint coefficients to three arrays***/  
s9:  rn[1]=1, cn[1]=1, a[1]= 0.5;  
s10: rn[2]=1, cn[2]=2, a[2]=-1.5;  
s11: rn[3]=2, cn[3]=1, a[3]=-0.6;  
s12: rn[4]=2, cn[4]=2, a[4]= 0.5;  
s13: rn[5]=3, cn[5]=1, a[5]=-0.4;  
s14: rn[6]=3, cn[6]=2, a[6]= 2.0;  
  
/**Load information into the problem object***/  
s15: lpx_load_mat3(lp, 6, rn, cn, a);  
  
/**Set the optimization direction***/  
s16: lpx_set_obj_dir(lp, LPX_MAX);  
  
/**Set the coefficients of the objective function***/  
s17: lpx_set_cols_coef(lp, 1, 1.0);  
s18: lpx_set_cols_coef(lp, 2, -1.0);  
  
/**Solve the Lp***/  
s19: lpx_simplex(lp);  
  
/**Get the optimal value and optimal solution***/  
s20: Z=lpx_get_obj_val(lp);  
s21: lpx_get_col_info(lp, 1, NULL, &x1, NULL);  
s22: lpx_get_col_info(lp, 2, NULL, &x2, NULL);
```

```

/**Print the output***/
s23: printf("Z = %lf; x1 = %lf, x2 = %lf\n", Z, x1, x2);

/**Delete the problem object***/
s24: lpx_delete_prob(lp);

return 0;
}

```

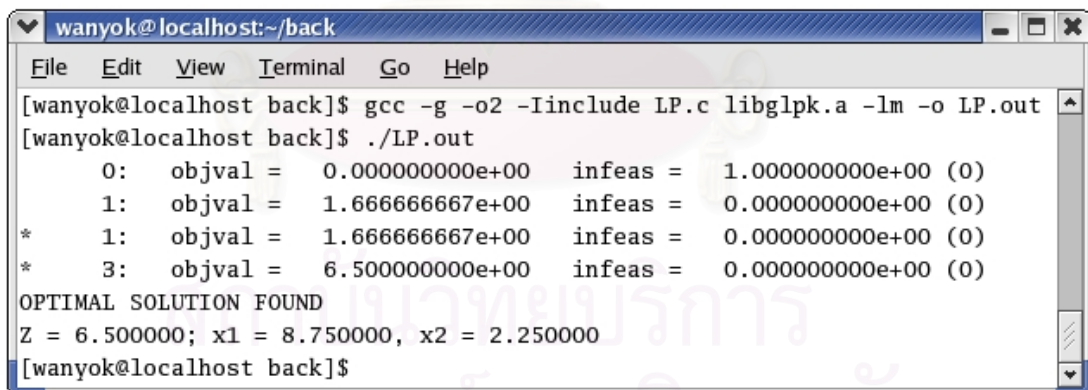
After we get the C program (LP.c), the next step is to link the relevant libraries and compile it by the following command:

```
>>gcc -g -o2 -Iinclude LP.c ../libglpk.a -lm -o LP.out
```

The LP.out can be executed using the following command:

```
>> ./LP.out
```

The result is shown in the following figure:



```

wanyok@localhost:~/back
File Edit View Terminal Go Help
[wanyok@localhost back]$ gcc -g -o2 -Iinclude LP.c libglpk.a -lm -o LP.out
[wanyok@localhost back]$ ./LP.out
  0:  objval =  0.00000000e+00   infeas =  1.00000000e+00 (0)
  1:  objval =  1.66666667e+00   infeas =  0.00000000e+00 (0)
 *  1:  objval =  1.66666667e+00   infeas =  0.00000000e+00 (0)
 *  3:  objval =  6.50000000e+00   infeas =  0.00000000e+00 (0)
OPTIMAL SOLUTION FOUND
Z = 6.500000; x1 = 8.750000, x2 = 2.250000
[wanyok@localhost back]$

```

Figure A.1: The output from GLPK

The optimal value ( $z$ ) is 6.5 and the optimal solution of this problem is  $x_1 = 8.75$  and  $x_2 = 2.25$ .

## Appendix B

### Stuttgart Neural Network Simulator: SNNS

SNNS (Stuttgart Neural Network Simulator) is a software simulator for research on and application of NNs [14]. The SNNS package can be obtained from <http://www-ra.informatik.uni-tuebingen.de/SNNS/>

To train an NN, we have to prepare data files using the SNNS format. SNNS supports five file types as follows:

#### 1. Pattern files (.pat)

The pattern file can be divided into two components: a header and a data. The header defines how many patterns the file contains and the dimension of the input and target vectors. An example of the pattern file (`xor.pat`) for the XOR problem is given as:

```
# Header
SNNS pattern definition file V1.4-3D
generated at Thu Sep 16 10:00:20 2004

No. of patterns : 4

No. of input units : 2

No. of output units : 1

# Data

# Pattern 1:
```

```

1 1
0
# Pattern 2:
0 1
1
# Pattern 3:
1 0
1
# Pattern 4:
0 0
0

```

## 2. Network file (.net)

The network file consists of the information about the weight, bias and the link between neurons of NN. An example of the network file (`xor.net`) is given as follows:

```

SNNS network definition file V1.4-3D
generated at Thu Sep 16 10:10:19 2004

```

```

network name : xor
source files :
no. of units : 8
no. of connections : 15
no. of unit types : 0
no. of site types : 0
learning function : Std_Backpropagation

```

update function : Topological\_Order

unit default section :

act	bias	st	subnet	layer	act func	out func
0.00000	0.00000	h	0	1	Act_Logistic	Out_Identity

unit definition section :

no.	typeName	unitName	act	bias	st	...
1			1.00000	-1.00000	i	
....						
7			0.03148	-1.96954	h	
8			0.09881	-2.13637	o	
----	-----	-----	-----	-----	----	...

connection definition section :

target	site	source:weight
3		1:-3.24972, 2:-2.69466
....		
8		3:-3.09026, 4:-1.59093, ... , 7: 5.77850
-----	-----	-----

### 3. Configuration files (.cfg)

The configuration file defines the location of relevant files and the value of all parameters, such as, the maximum iteration, maximum error and learning rate. An example of the configuration file (`xor.cfg`) is given as follows:

```
Type: SNNSBATCH_2
# If a key is given twice, the second appearance is taken.
# Keys that are not required for a special run maybe omitted.
# If a key is omitted but required, a default value is assumed.
# The lines may be separated with comments.
#
# Please note the mandatory file type specification at
# the beginning and the colon following the key.
#
NetworkFile: xor.net
InitFunction: Randomize_Weights
NoOfInitParam: 2
InitParam: -1.0 1.0
#
LearnPatternFile: xor.pat
#
NoOfLearnParam: 2
LearnParam: 0.8 0.1
MaxLearnCycles: 10000
MaxErrorToStop: 0.001
Shuffle: YES
```

TrainedNetworkFile: xor.net

ResultFile: xor.res

ResultMinMaxPattern: 1 4

ResultIncludeInput: YES

ResultIncludeOutput: YES

#### 4. Result file (.res)

The result file is the output of the trained NN written after the SNNS finished processing. An example of the result file (`xor.res`) is given as follow:

SNNS result file V1.4-3D

generated at Thu Sep 16 10:10:19 2004

No. of patterns : 4

No. of input units : 2

No. of output units : 1

startpattern : 1

endpattern : 4

input patterns included

teaching output included

#1.1

0 0

0

0.09794

```

#2.1
0 1
1
0.90076
#3.1
1 0
1
0.90016
#4.1
1 1
0
0.09881

```

## 5. Log file (.log)

The log file created automatically reports the statistical data about the process of training an NN, such as, the error in each iteration, CPU time used in learning and the number of learned cycles. An example of the log file (`snsbat.log`) is given as follow:

```

SNNS 3D-Kernel V4.20 Batchlearning Program
Configuration file: 'xor.cfg'
Log file           : 'snsbat.log'

```

```

##### SNNS batch execution run. Loop 1 #####
Networkfile 'xor.net' loaded.
Patternfile 'xor.pat' loaded.
No. of patterns: 4

```



No. of cycles: 100000

Max. network error to stop: 0.001000

Patterns are shuffled

Network name : xor

No. of units : 8

No. of sites : 0

No. of links : 15

Learning Function : Std\_Backpropagation

Update Function : Topological\_Order

Learning Parameter #1 : 0.800000

Learning Parameter #2 : 0.100000

Init Function: Randomize\_Weights

Init Parameter #1 : -1.000000

Init Parameter #2 : 1.000000

Result File : 'xor.res'

Result File Start Pattern: 1

Result File End Pattern : 4

Result File Input Pattern included

Result File Output Pattern included

\*\*\*\*\*

SNNS 3D-Kernel V4.20 Batchlearning started

at Thu Sep 16 10:11:27 2004

Network initialized with

Randomize\_Weights -1.00 1.00

Result file saved.

Network saved to xor.net.

SNNS 3D-Kernel V4.20 Batchlearning terminated

at Thu Sep 16 10:11:29 2004

Node: unknown

\*\*\*\*\*

---- STATISTICS ----

No. of learned cycles: 611

No. of units updated : 19552

No. of sites updated : 0

No. of links updated : 36660

CPU Time used: 0.06 seconds

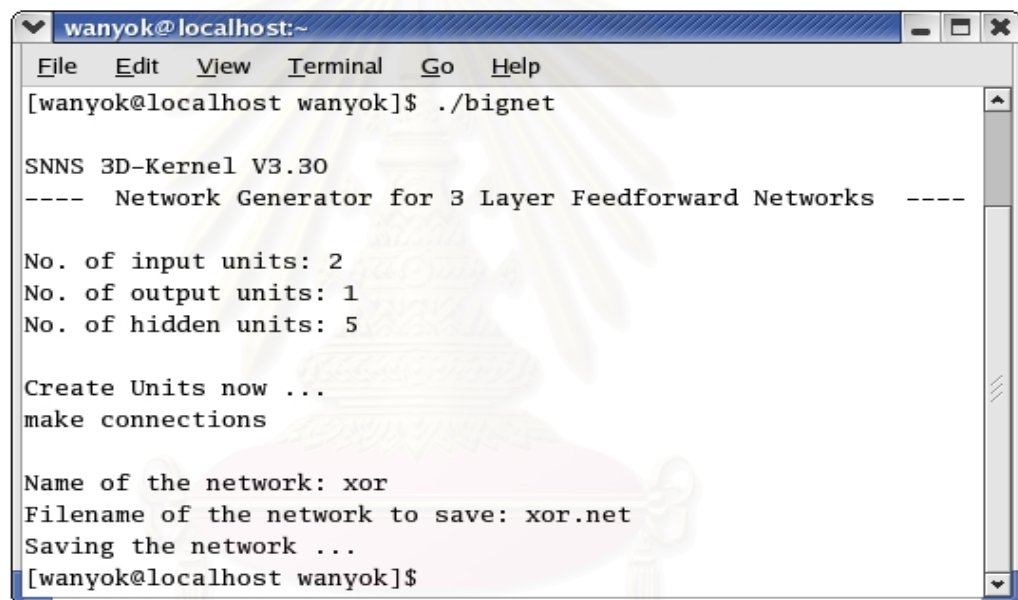
User time: 2 seconds

No. of connection updates per second (CUPS): 6.110000e+05

The process for using SNNS can be divided into three steps:

### 1. Network Creation

The network file for `xor.net` can be created by the SNNS tool called Bignet using the following command: `>> ./bignet`. The number of nodes in input, output and hidden layers must be entered, respectively. We also need to enter the name of the network and the file name that we want to save the network.



```
wanyok@localhost:~
File Edit View Terminal Go Help
[wanyok@localhost wanyok]$ ./bignet

SNNS 3D-Kernel V3.30
---- Network Generator for 3 Layer Feedforward Networks ----

No. of input units: 2
No. of output units: 1
No. of hidden units: 5

Create Units now ...
make connections

Name of the network: xor
Filename of the network to save: xor.net
Saving the network ...
[wanyok@localhost wanyok]$
```

Figure B.1: Bignet for a network creation

Bignet will create a file in our directory with the name `xor.net`. This network has 2 input neurons, 5 hidden neurons and one output unit. However, we must also have a pattern file before we proceed with the simulation.

### 2. Pattern Creation

A pattern file is a text file with the `.pat` extension. It contains a header and a data. The header is composed of a number of training patterns and

the number of input and output units. Then, we append training patterns after the header information. An example of the pattern file has shown in `xor.pat`.

### 3. Simulation

After we have a network file (`xor.net`) and a pattern file (`xor.pat`), the next step is to create the configuration file that defines the location of relevant files and the value of all parameters. An example of the configuration file has shown in `xor.cfg`.

The configuration file can be executed using the following command:

```
>> ./snnsbat xor.cfg
```

This execution loads a network and pattern files, initializes the weights, trains it for 10000 cycles (or stops, if the error is less than 0.001), and finally generates the result file (`xor.res`).

In order to measure the accuracy of our testing patterns, the configuration file will be change as follows:

```
NetworkFile: xor.net
```

```
LearnPatternFile: xor_test.pat
```

```
ResultFile: xor_test.res
```

```
ResultMinMaxPattern: 1 4
```

```
ResultIncludeInput: YES
```

```
ResultIncludeOutput: YES
```

We can see our results and network file in `xor_test.res` and `xor.net`. For any statistical data, we can see it in the log file (`snnsbat.log`) that is automatically created when we train an NN.

## Appendix C

### Coding

The program `main.c` is written in the ANSI-C programming language following the algorithm for generating patterns for NNs from LPs in chapter III. Its coding can be shown as follows:

```
/**Declare the relevant libraries***/
#include <stdio.h>
#include <stdlib.h>
#include <glpk.h>

/**Declare the relevant variables***/
int m,n;
int Max_pat,LP;
int h,k;

/**Declare subprograms***/
void Set_problemsize();
void Gen_LP();
void Sort(double c[n+1],double aa[m+1][n+1],double b[m+1]);
void Find_max(double aa[m+1][n+1],int sr,int sc);
void Switch_row(double b[m+1],double aa[m+1][n+1],int r1,int r2);
void Switch_col(double c[n+1],double aa[m+1][n+1],int c1,int c2);
void Solve_LP(double c[n+1],double aa[m+1][n+1],double b[m+1]);
```

```

void Binding(double c[n+1],double aa[m+1][n+1],double b[m+1],
            double x[n+1]);

void Print_output(double c[n+1],double aa[m+1][n+1],double b[m+1],
                int t[m+n+1]);

```

```

/**Program main.c for generating input patterns**/

int main(void)
{
    Set_problemsize(); //Set problem sizes and maximum patterns.
    for(LP=1;LP<=Max_pat;LP++)Gen_LP();
    return 0;
}

```

```

void Set_problemsize()
{
    /**Set problem sizes and maximum patterns**/
    printf("Enter the number of constraints (m): ");
    scanf("%d",&m);
    printf("Enter the number of variables (n): ");
    scanf("%d",&n);

    /*Set the number of training + testing patterns*/
    printf("Enter the maximum patterns (Max_pattern): ");
    scanf("%d",&Max_pat);
}

```

```

void Gen_LP()
/**Random Matrix A, vector c and vector b***/
/**in the normalized LP from***/
/**c!=0 and b!=0***/
{
double aa[m+1][n+1];
double c[n+1],b[m+1];
double temp;
int i,j;

for(i=1;i<=m;i++){
    for(j=1;j<=n;j++){
        aa[i][j]=2.0*(rand()/(double)RAND_MAX)-1.0;
        temp=rand()/(double)RAND_MAX;
        if(temp<=0.5)c[j]=1;
        else c[j]=-1;
    }
    temp=rand()/(double)RAND_MAX;
    if(temp<=0.5)b[i]=1;
    else b[i]=-1;
}

Sort(c,aa,b); //Sort the LP according to the matrix A
Solve_LP(c,aa,b); //Solve the LP to get the optimal solution
}

```

```

void Sort(double c[n+1],double aa[m+1][n+1],double b[m+1])
/**Sort the LP according to the matrix A***/
{
    int sr,sc,i;
    for(i=1;i<=n;i++){
        Find_max(aa,i,i);
        if(h!=i)Switch_row(b,aa,i,h);
        if(k!=i)Switch_col(c,aa,i,k);
    }
    if(m>n+1){
        for(i=1;i<m-n;i++){
            Find_max(aa,n+i,n);
            if(h!=n+i)Switch_row(b,aa,n+i,h);
        }
    }
}

void Find_max(double aa[m+1][n+1],int sr,int sc)
{
    int i,j; double max=aa[sr][sc];
    h=sr,k=sc;
    for(i=sr;i<=m;i++){
        for(j=sc;j<=n;j++){
            if(max<aa[i][j]){max=aa[i][j];h=i;k=j;}
        }
    }
}

```



```
void Switch_row(double b[m+1],double aa[m+1][n+1],int r1,int r2)
{
    double temp;
    int i;

    for(i=1;i<=n;i++){temp=aa[r1][i];
        aa[r1][i]=aa[r2][i];
        aa[r2][i]=temp;}
    if(b[r1]!=b[r2]){temp=b[r1];
        b[r1]=b[r2];
        b[r2]=temp;}
}

void Switch_col(double c[n+1],double aa[m+1][n+1],int c1,int c2)
{
    double temp;
    int i;

    for(i=1;i<=m;i++){temp=aa[i][c1];
        aa[i][c1]=aa[i][c2];
        aa[i][c2]=temp;}
    if(c[c1]!=c[c2]){temp=c[c1];
        c[c1]=c[c2];
        c[c2]=temp;}
}
```

```

void Solve_LP(double c[n+1],double aa[m+1][n+1],double b[m+1])
{
    LPX *lp;
    int i,j,cu=1,rz=0,rn[m*n+1],cn[m*n+1];
    double a[m*n+1],x[n+1];

    s1: lp=lp_create_prob();
    s2: lp_add_rows(lp,m);
    s3: for(i=1;i<=m;i++)lp_set_row_bnds(lp,i,LPX_UP,0.0,b[i]);
    s4: lp_add_cols(lp,n);
    s5: for(i=1;i<=n;i++)lp_set_col_bnds(lp,i,LPX_LO,0.0,0.0);
    s6: for(i=1;i<=m;i++){
        for(j=1;j<=n;j++,cu++){
            if(fabs(aa[i][j])>=1E-6){
                rn[cu-rz]=i, cn[cu-rz]=j, a[cu-rz]=aa[i][j];}
            else rz++;} }
    s7: lp_load_mat3(lp,m*n-rz,rn,cn,a);
    s8: lp_set_obj_dir(lp,LPX_MAX);
    s9: for(i=1;i<=n;i++)lp_set_col_coef(lp,i,c[i]);
    s10: lp_simplex(lp);
    s11: if(lp_get_status(lp)==LPX_OPT){
        for(i=1;i<=n;i++)lp_get_col_info(lp,i,NULL,&x[i],NULL);
        Binding(c,aa,b,x);}
        else LP--;
    s12: lp_delete_prob(lp);
}

```

```

void Binding(double c[n+1],double aa[m+1][n+1],double b[m+1],
            double x[n+1])
{
    int i,j,t[m+n+1];
    double sum;
    for(i=1;i<=n;i++){if(fabs(x[i])<=1E-6)t[i]=1;else t[i]=0;}
    for(i=1;i<=m;i++){sum=0.0;
        for(j=1;j<=n;j++)sum+=(aa[i][j]*x[j]);
        if(fabs(sum-b[i])<=1E-6)t[n+i]=1;
        else t[n+i]=0;}
    Print_output(c,aa,b,t);
}

void Print_output(double c[n+1],double aa[m+1][n+1],double b[m+1],
                int t[m+n+1])
{
    int i,j;
    FILE *fpat;
    fpat=fopen("Lp.pat","a+");
    for(i=1;i<=n;i++)fprintf(fpat,"%01f\t",c[i]);
    for(i=1;i<=m;i++){for(j=1;j<=n;j++)fprintf(fpat,"%1f\t",aa[i][j]);
        fprintf(fpat,"%01f\t",b[i]);}
    for(i=1;i<=m+n;i++)fprintf(fpat,"%d\t",t[i]);
    fprintf(fpat,"\n");
    fclose(fpat);
}

```

## Vitae

Wanyok Atisattapong was born in March 5, 1982, in Bangkok. She obtained her Bachelor's degree in Applied Mathematics from the Faculty of Science and Technology, Thammasat University in 2002.



สถาบันวิทยบริการ  
จุฬาลงกรณ์มหาวิทยาลัย