NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

A Neural Network Approach to Routing and Flow Allocation Problems in Communications Networks

Faouzi Kamoun

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

November, 1990

ISBN 0-315-64696-9

Canada

ABSTRACT

A Neural Network Approach to Routing and Flow Allocation Problems in Communications Networks

F. Kamoun

Recently, neural networks have been proposed as new computational tools for solving constrained optimization problems. This thesis is concerned with the application of neural networks to flow allocation and routing problems in communications networks. The solutions to these problems involve Linear Programming and shortest-path computations. The existing neural networks have been improved for these applications. The flow allocation and routing problems have been formulated in a convenient way, that make them solvable by neural networks. This will enable these complicated problems to be solved in real time.

In this thesis, the general principles involved in the design of such neural networks to solve routing and flow allocation problems are discussed. The computational power and speed of neural optimization networks are demonstrated through computer simulations. Some of the issues surrounding the applications of neural networks to these routing problems are also addressed. The key features of the neural network approach, namely a potential for high computation power and speed, high degree of robustness and fault tolerance, low power consumption and real time operation are highlighted and suggestions for further research are proposed.

ACKNOWLEDGEMENTS

I would like to express my heartfelt gratitude to my thesis supervisor, Dr.M.K.Mehmet-Ali, for his constant guidance and assistance during the entire preparation of the thesis and for his suggestions and constructive criticism.

I also gratefully acknowledge the graduate scholarship awarded to me by the University Mission of Tunisia, which provided most of the financial support required for the completion of my master's degree.

Last, but not least, I would like to express my deep gratitude to my parents, sister, and brothers for their constant encouragements and support throughout my studies. I would like also to express my appreciation to all my friends at Concordia University for their moral support.

Table of Contents

List of Figures

List of Tables

List of Symbols

C_i = Input capacitance of the i^{th} neuron

c_{xi} = Input capacitance of neuron at location xi

C_{ij} = Cost of the arc from node i to node j

C_a^* = Capacity of arc a

f_a = Aggregate flow on link a

f_a^k = Flow of commodity k on link a

$\vec{f}$ = Vector of path flows

$f(n)$ = Flow carried on path n

g^* = Neural transfer function

L^{sd} = Length of the path connecting node s to node d

P^* = Scalar function of the Linear Programming Problem

(s^k, t^k) = Source-destination pair corresponding to commodity k

$\mathbb{R}$ = The set of real numbers

T_{ij} = Synaptic connection between neurons i and j

T_l = Packet delay at link l queue

T = Total average packet delay

γ = Total packet arrival rate

U_i = Input voltage of the i^{th} neuron

U_{xi} = Input voltage of neuron at location xi

V_i = Output voltage of the i^{th} neuron

V_{xi} = Output voltage of neuron at location xi

v^k = Traffic flow originating at s^k and destined to t^k

$\overrightarrow{V_k}$ = Flow vector corresponding to commodity k.

δt = Incremental time

ΔV_{th} = Voltage threshold value

λ = Neural transfer parameter

λ_l^* = Average packet arrival rate to link l buffer

$\lambda(i)$ = Offered traffic to commodity i

ε = A specified tolerance value

CHAPTER 1

INTRODUCTION

During the past few years, neural networks have become very popular as new candidates for parallel distributed processing systems. From a system's point of view, neural networks are large-scale dynamic systems that can be described by first-order nonlinear difference or differential equations. These systems have been designed in an attempt to capture some of the features of biological neural networks. It is well known today that the computational power of the human brain is derived from the massive interconnections among a huge number of parallel distributed processing elements or neurons. In an attempt to explore this idea of distributed parallel information processing , scientists and engineers have proposed many neural network models, each designed to perform a specific information processing task. All these models, however, have the same general structure:

Conceptually a neural network consists of many nonlinear, typically analog, processing elements which operate independently from each others and in a parallel fashion. Each processing element (PE) gets inputs from many other PEs through a network of variable weight synaptic connections, and produces a single output according to a well defined transfer function.

Most of the neural network models , however, differ in the topology of their synaptic connections (example feedforward against feedback connections), and in the way the weights of their synaptic connections and the neurons' transfer functions are defined.

This thesis is concerned with those neural network models that are intended to be used for solving constrained optimization problems. These models are characterized by the

presence of feedback synaptic connections, and their dynamics follow a gradient descent of a cost function, which combines both the objective function and the constraints. In order to apply neural networks to solve constrained optimization problems, the PEs' transfer functions are to be specified. In addition the synaptic connections and their corresponding weights are to be determined, so that the neural network dynamics evolve towards an equilibrium point, which shall correspond to the desired solution.

This thesis focuses on the application of neural networks to flow allocation and routing problems in communications networks. The problems of flow allocation and traffic routing emerge whenever there are alternative paths for a given source to reach its corresponding destination. All flow allocation and traffic routing algorithms have one common goal, that of finding the best path for each given origin-destination pair, taking into account the fact that the traffic of a given origin-destination pair is likely to interact with that of the remaining pairs. These problems, however, arise in many contexts. For instance the best path between a source-destination pair may be defined as the minimum length path, in which case the routing problem becomes a **shortest path problem.** Alternatively we may wish to route and allocate the traffic of each source-destination pair so as to maximize the total network throughput, in which case the flow allocation and routing problem becomes a **maximum flow problem.** We may also consider the more general flow allocation and routing problem, where the best paths for all source-destination pairs are those which minimize the total cost of the network. If this cost is set to the average network delay, then the flow allocation and routing problem becomes a **minimum delay routing problem.**

The objective of this thesis is to study these three selected routing and flow allocation problems, namely the maximum flow problem, the shortest path problem, and the minimum delay routing problem, and to show how to apply neural network optimization techniques

to solve them. It should be noted that detailed discussion of the many conventional techniques that have already been proposed to solve each of these selected routing problems is beyond the scope of this thesis.

What makes neural optimization networks interesting alternatives for solving traffic routing problems in communications networks is their potential for high computational speed, high degree of robustness and fault tolerance and low power consumption. Neural optimization networks derive most of their high computational speed from the massive parallelism in computation that takes place at each PE. This enables a neural network to process huge amounts of information simultaneously. In addition the search for the optimal solution is performed in real time and this solution is reached almost instantaneously. The high degree of robustness and fault tolerance offered by neural networks is due to the fact that information is distributed among all the PEs in the network, rather than being concentrated at a particular location. When confronted with a sudden failure in some of its neurons, the neural network remains viable and its performance is expected to experience a graceful degradation. This is opposed to what generally happens in a sequential general purpose computer, where the failure of a particular device will completely shutdown the computational process of the whole computer.

This thesis is organized as follows:

In chapter 2 the use of neural networks to solve constrained optimization problems is described. First Hopfield and Tank neural network is reviewed in details then a modified version of Chua and Lin Linear Programming network is suggested. The latter is an extension of Chua and Lin's original network, that makes it possible to handle both equality and inequality constraints in an efficient and cost effective way.

In chapter 3 the computational power of the modified Linear Programming circuit will

be demonstrated, through simulations, by solving the maximum flow problem. It will be shown that the modified Linear Programming circuit is very efficient in solving the problem, with the solution being obtained very fast and with the computational time increasing moderately with problem size.

In chapter 4 the second routing problem, namely that of finding the shortest path between a given origin-destination pair, is solved using Hopfield and Tank neural model. The main steps involved in the design of Hopfield and Tank neural network to solve the shortest path problem will be described. The computational power of the proposed model will be demonstrated through computer simulations. The proposed model combines many features such as flexibility to operate in real time and to adapt to changes in link costs and network topology.

In chapter 5 it will shown that the neural network shortest path algorithm, developed in chapter 4, could effectively be used to solve the optimum minimum delay routing problem in packet-switched communications networks. Since Hopfield and Tank neural network can perform shortest path computations in real time and can adapt to changes in its environment then its application to the proposed minimum delay traffic routing problem (which heavily rely on shortest path computations) becomes very desirable. The applicability of the neural network shortest path problem to the traffic routing problem will be demonstrated through computer simulation and a distributed version of the neural based, minimum delay routing algorithm will be described.

Finally in chapter 6 a summary of the main findings and results of this thesis is given and suggestions for further research are provided.

CHAPTER 2

NEURAL NETWORKS FOR CONSTRAINED OPTIMIZATION

2.1 Introduction

The use of neural networks to solve constrained optimization problems was initiated by Hopfield and Tank [1,2,3]. They proposed a neural network model to solve discrete combinatorial optimization problems. They demonstrated the computational power of their network by solving the Travelling Salesman Problem (TSP), which belongs to the class of nondeterministic polynomial-time (NP) complete problems. Since then many researchers have attempted to apply Hopfield and Tank model to solve other combinatorial optimization problems. Hopfield and Tank have also shown that even the continuous optimization problem of Linear Programming (LP) could be solved by neural networks. Their LP circuit [3] maintained analog (as opposed to binary) values by properly scaling the saturation level of neurons' outputs. This avoids the use of more complicated schemes such as those proposed by Takeda and Goodman [4] for representing analog real values at neurons' outputs [5].

Other attempts to solve continuous optimization problems by neural networks include the work of Chua and Lin [6,7], who proposed a canonical nonlinear programming circuit that simulates both the objective function and the inequality constraints of the more general nonlinear programming problem. Kennedy and Chua [8] analyzed the LP network of Hopfield and Tank and found an error in its design. When this error is corrected, they have shown that the modified Hopfield and Tank LP network becomes a special case of Chua and Lin canonical nonlinear programming circuit. Kennedy and Chua also analyzed the dynamics of the canonical nonlinear programming circuit of Chua and Lin and proposed a

circuit implementation using 'neural' networks. The use of Chua and Lin canonical nonlinear programming circuit to solve LP problems becomes attractive when we consider the relative ease with which the LP circuit (as opposed to the more general nonlinear programming circuit) could be implemented.

The remaining of this chapter is organized as follows:
In the next section we start by reviewing Hopfield and Tank neural network. The error introduced by Hopfield and Tank approximation will be highlighted and a method to correct for this error will be proposed. In section 2.3 an extended version of Chua and Lin Linear Programming network will be described.

2.2 Hopfield and Tank Neural Network

2.2.1 Model Description

The neural computational circuit of Hopfield and Tank is shown in figure 2.1 . This circuit is designed so as to model the basic components of a biological neural network. Each neuron is modeled as a nonlinear device (operational amplifier) with a sigmoid monotonic increasing function relating the output V_i of the i^{th} neuron to its input U_i . The output V_i is allowed to take on any value between 0 and 1. A typical sigmoid function is the logistic function:

$$V_i = g_i(U_i) = \frac{1}{1+e^{-\lambda_i \cdot U_i}} \qquad (2.1)$$

This function approaches a unit step function as λ_i approaches infinity . Figure 2.2 shows the input-output relation of a neuron for three different values of λ_i .

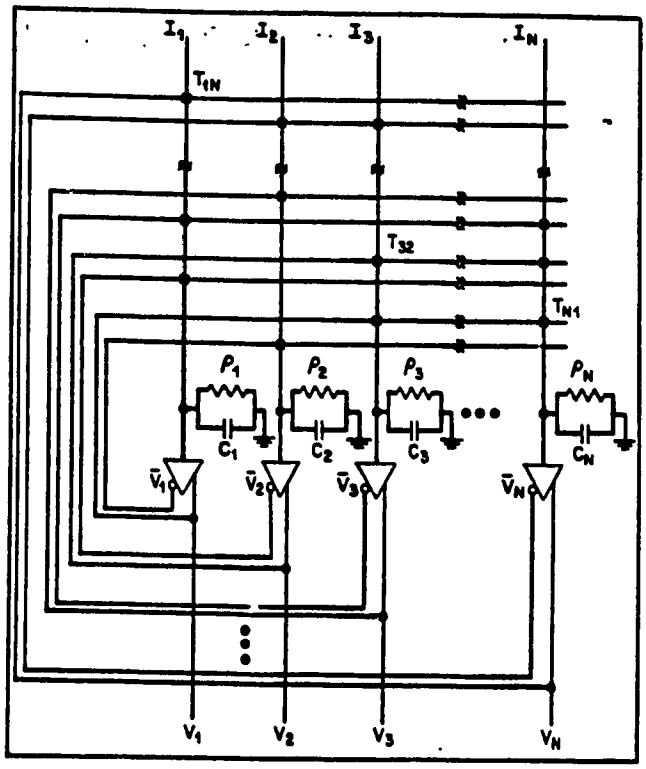


Figure 2.1 Hopfield and Tank Neural Network [10]

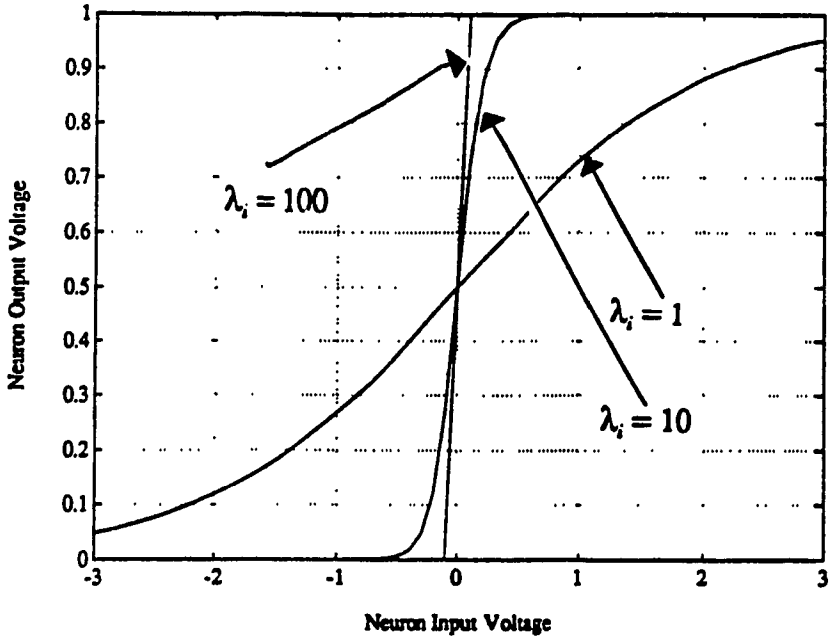


Figure 2.2 The Input-Output Relation of a Neuron

One of the most important characteristics of the sigmoid transfer function is its saturation behavior at both ends. This behavior is behind the computational power of neurons in making decisions [10]. Each neuron has an input resistance ρ_i (to model the trans-membrane resistance of a biological neuron), an input capacitance C_i (to model the capacitance of the cell membrane) and receives resistive connections (to model the synaptic connections) from other neurons. The synaptic connections are allowed to be either excitatory or inhibitory by providing each amplifier with a normal as well as an inverted output. Each synaptic connection is implemented with a resistance $R_{ij} = \frac{1}{|T_{ij}|}$, connecting the input of the i^{th} amplifier to one of the two outputs of the j^{th} amplifier. The selection of the appropriate output depends on the type of synaptic connection to be established. If the synaptic connection is to be excitatory ($T_{ij} > 0$) then the input of the i^{th} amplifier is connected to the normal output of the j^{th} amplifier. In the latter case ($T_{ij} < 0$) it is connected to the inverted output of the j^{th} amplifier. The synaptic connections can be fully described through the matrix $T = [T_{ij}]$, known also as the connection matrix of the network.In addition, as shown in figure 2.1, each neuron receives an external current (known also as a bias) I_i , which could represent actual data provided by the user to the neural network [2,3].

Neglecting the output impedances of the amplifiers, it can be shown from simple circuit theory that the equation of motion of the i^{th} neuron is described by the following nonlinear differential equation:

$$C_i \frac{dU_i}{dt} = \sum_{j=1}^{N} T_{ij} V_j - \frac{U_i}{R_i} + I_i \qquad (2.2.a)$$

$$V_i = g_i(U_i) = \frac{1}{1 + e^{-\lambda_i \cdot U_i}} \qquad (2.2.b)$$

where

$$\frac{1}{R_i}=\frac{1}{\rho_i}+\sum_{j=1}^{N}|T_{ij}| \tag{2.3}$$

Hopfield [11] considers the following energy function :

$$E=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}T_{ij}V_iV_j-\sum_{i=1}^{N}I_iV_i+\sum_{i=1}^{N}\frac{1}{R_i}\cdot\int_0^{V_i}g_i^{*-1}(x)dx \tag{2.4}$$

where g_i^{*-1} is the neuron inverse input-output relation which, for the logistic function in equation (2.1), is given by:

$$U_i=g_i^{*-1}(V_i)=-\frac{1}{\lambda_i}\ln\left(\frac{1-V_i}{V_i}\right) \tag{2.5}$$

For a symmetric connection matrix T , the time derivative of E is :

$$\frac{dE}{dt}=\sum_{i=1}^{N}\frac{\partial E}{\partial V_i}\cdot\frac{dV_i}{dt} \tag{2.6.a}$$

$$=\sum_{i=1}^{N}\left(-\sum_{j=1}^{N}T_{ij}V_j+\frac{U_i}{R_i}-I_i\right)\cdot\frac{dV_i}{dt} \tag{2.6.b}$$

From (2.2) and (2.6) we get :

$$C_i\frac{dU_i}{dt}=-\frac{\partial E}{\partial V_i} \tag{2.7}$$

Substituting (2.7) in (2.6.a), we have:

$$\frac{dE}{dt}=-\sum_{i=1}^{N}C_i\cdot\frac{dU_i}{dt}\cdot\frac{dV_i}{dt} \tag{2.8.a}$$

$$=-\sum_{i=1}^{N}C_i\cdot\frac{dU_i}{dV_i}\cdot\left(\frac{dV_i}{dt}\right)^2 \tag{2.8.b}$$

$$=-\sum_{i=1}^{N}C_i\cdot\frac{d}{dV_i}(g^{*-1}(V_i))\cdot\left(\frac{dV_i}{dt}\right)^2 \tag{2.8.c}$$

Therefore if g_i^{*-1} is a monotonic increasing function then the energy function (2.4) will be monotonically decreasing. In addition at equilibrium :

$$\frac{dE}{dt}=0 \Leftrightarrow \frac{dV_i}{dt}=0 ; \qquad \forall i \in \{1,2,.....,N\} \tag{2.9}$$

Expressions (2.7-2.9) show that the equations of motion of the neurons, for a symmetric connection matrix T, follow a gradient descend of the energy function E and that , starting from some initial conditions, the state of the system (described by the V_i's) evolves towards a minima of E and stabilizes when all neuron outputs remain constant. This final stable state corresponds to a possible solution to the problem. In addition the energy function (2.4) is a Lyapunov function for the neural system [11,12].

Hopfield [11] has shown that if the gains of the amplifiers are sufficiently high ($\lambda_i \rightarrow \infty$) then the last term of (2.4) vanishes and the Lyapunov function reduces to the quadratic expression :

$$E_m=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}T_{ij}V_iV_j-\sum_{i=1}^{N}I_iV_i \tag{2.10}$$

Hopfield [11] has also shown that while the state of the neural system evolves inside the N-dimensional hypercube defined by $V_i \in \{0,1\}$, the minima of the modified energy function (2.10) occur only at the 2^N corners of this space. In terms of the modified Hopfield and Tank energy function, E_m , the dynamics of the i^{th} neuron are described by the following equation:

$$C_i\frac{dU_i}{dt}=-\frac{\partial E_m}{\partial V_i}-\frac{U_i}{R_i} \tag{2.11}$$

The usage of Hopfield and Tank discrete model to solve combinatorial optimization problems involves the following [3] :

- The proper choice of a candidate quadratic energy function. This function incorporates both the objective function and a sequence of penalty functions for constraint violations.
- The derivation of the resistive connections (T_{ij}) and the input biases (I_i) . One simple way to achieve this, is by equating the right side terms of equations (2.2.a) and (2.11) .
- The provision of initial input voltages (U_i 's) to the amplifiers .
- The decoding of the solution from the final stable states of the neurons .

2.2.2 The Hopfield Approximation Error

In this section, the error due to the Hopfield approximation error (2.10) is investigated. It will be shown that for large but finite amplifier gain , λ_i , the last term of (2.4) starts to contribute and can no longer be neglected .

Define the error between the original and the modified energy function by ΔE . Then we have :

$$\Delta E = E - E_m \tag{2.12.a}$$

$$= \sum_{i=1}^{N} \frac{1}{R_i} \int_0^{V_i} g_i^{-1}(x)dx \tag{2.12.b}$$

$$= \sum_{i=1}^{N} \frac{1}{R_i} \int_0^{V_i} -\frac{1}{\lambda_i} \ln\left(\frac{1-x}{x}\right)dx \tag{2.12.c}$$

$$= \sum_{i=1}^{N} \frac{e^*(V_i)}{\lambda_i R_i} \tag{2.12.d}$$

where :

$$e^*(V_i) = -\int_0^{V_i} \ln\left(\frac{1-x}{x}\right)dx = (1-V_i)\ln(1-V_i) + V_i \ln(V_i) \qquad (2.13)$$

The function $e^*(V_i)$ has a maximum at $V_i = 0$ or 1 and a minimum at $V_i = 0.5$, as illustrated in figure 2.3 .

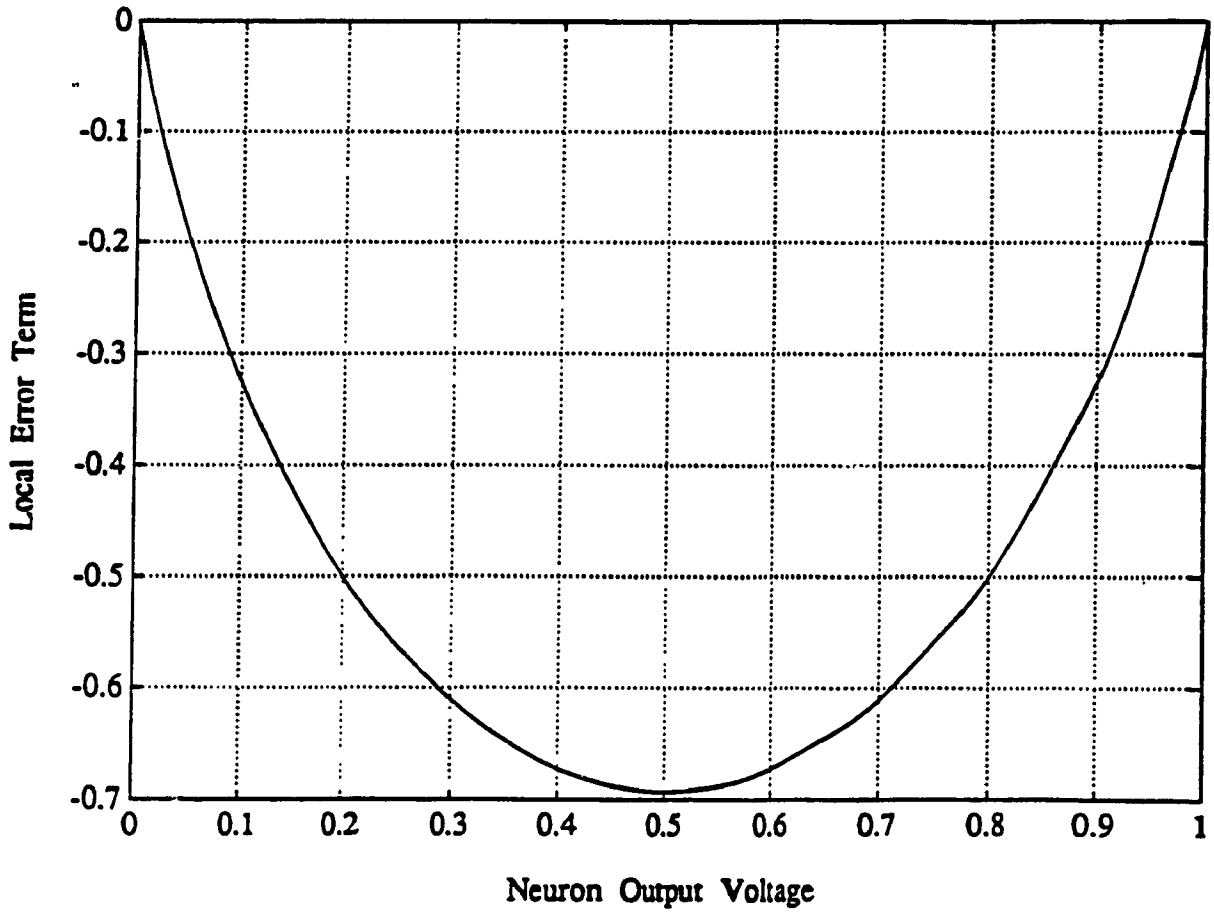


Figure 2.3. Local Error Term $e^*(V_i)$

From the above it becomes clear that while the error ΔE approaches zero as $\lambda_i \rightarrow \infty$, it tends to displace the minima of the energy function towards the center of the hypercube as λ_i becomes finite. This displacement is further accentuated as λ_i is decreased. For very small gain λ_i the effect of ΔE becomes dominant .

One way to compensate for the displacement of the minima from the corners of the hypercube, due to small λ_i is to insert a small positively weighted quadratic term :

$$E^{com} = \frac{S}{2} \sum_{i=1}^{N} V_i(1 - V_i) \tag{2.14}$$

in the energy function. This quadratic term is minimized as V_i approaches 0 or 1 for all i's.

2.3 Extension of Chua and Lin Linear Programming Network

Chua and Lin [6,7,9] proposed a neural network model to solve Linear Programming problems, in which the feasible region restricting the state space of the optimum solution is defined by inequality constraints. In this section an extended version of their model is proposed that will enable us to solve the more general linear programming problem, in which the feasible state space of the optimum solution is defined by inequality as well as equality constraints. Although Chua and Lin Linear Programming network (refer to appendix A for a summary of their original work) can handle such a problem, as an equality constraint can always be written as two inequality constraints, namely:

$$q_j(\vec{V}) = 0 \Leftrightarrow \begin{cases} q_j(\vec{V}) \geq 0 \\ -q_j(\vec{V}) > 0 \end{cases} \tag{2.15}$$

it will be shown that the extended model can solve the same problem in a more efficient and simplified way. The idea behind the proposed model was motivated by earlier work in constrained optimization theory, where exterior point penalty functions or loss functions of quadratic types have been found to be very adequate to handle equality constraints [13,14].

The remaining of this chapter is organized as follows:

In section 2.3.1 we start by describing the extended Linear Programming circuit. Following a similar approach as in [9], we will carry a dynamic analysis of the extended model and show that the proposed model is completely stable. In section 2.3.2 a circuit implementation of the extended model, using solid-state devices, is proposed.

2.3.1 Description of the Extended LP network

Here we consider the more general problem:

Minimize the scalar function:

$$\Phi(\vec{V}) = \vec{A} \cdot \vec{V} \tag{2.16}$$

Subject to the constraints:

$$f_j(\vec{V}) = \vec{B_j} \cdot \vec{V} - E_j \geq 0; \quad j = 1,2,..p \tag{2.17.a}$$

$$q_k(\vec{V}) = \vec{C_k} \cdot \vec{V} - F_k = 0; \quad k = 1,2,...m \tag{2.17.b}$$

or in matrix form:

$$f(\vec{V}) = B\vec{V} - \vec{E} \geq \vec{0} \tag{2.18.a}$$

$$q(\vec{V}) = C\vec{V} - \vec{F} = \vec{0} \tag{2.18.b}$$

where:

$$\vec{A}=\begin{bmatrix}A_1\\A_2\\.\\.\\A_q\end{bmatrix};\ \vec{V}=\begin{bmatrix}V_1\\V_2\\.\\.\\V_q\end{bmatrix};\ \vec{B_j}=\begin{bmatrix}B_{j1}\\B_{j2}\\.\\.\\B_{jq}\end{bmatrix};\ \vec{C_k}=\begin{bmatrix}C_{k1}\\C_{k2}\\.\\.\\C_{kq}\end{bmatrix};\ B=\begin{bmatrix}B_{11}B_{12}..B_{1q}\\B_{21}B_{22}..B_{2q}\\.\ .\ .\ .\ .\\.\ .\ .\ .\ .\\B_{p1}B_{p2}..B_{pq}\end{bmatrix};\ \vec{E}=\begin{bmatrix}E_1\\E_2\\.\\.\\E_p\end{bmatrix} \tag{2.19}$$

$$C=\begin{bmatrix}C_{11}C_{12}..C_{1q}\\C_{21}C_{22}..C_{2q}\\.\ .\ .\ .\ .\\.\ .\ .\ .\ .\\C_{m1}C_{m2}..C_{mq}\end{bmatrix};\vec{F}=\begin{bmatrix}F_1\\F_2\\.\\.\\F_m\end{bmatrix}$$

Note that the (•) symbol in (2.16 - 2.17) refers to the dot product operator, and that the j^{th} row of matrix B corresponds to $\vec{B_j}^T$, while the k^{th} row of matrix C corresponds to vector $\vec{C_k}^T$. The LP formulation (2.16-2.19) is the same as in Chua and Lin original work, except for the equality constraints in (2.17.b and 2.18.b). Our goal is to show that the network shown in figure 2.4 solves this problem with no risk of oscillation.

The Linear Programming model, shown in figure 2.4 consists of suitably defined controlled current and voltage sources, linear and nonlinear resistors and linear capacitors. Each rectangular shape symbol in the middle column of figure 2.4 represents a voltage controlled nonlinear resistor, whose characteristic (shown in figure 2.5) is governed by the following equation:

$$g_n(V)=\begin{cases}0 & \text{if } V>0\\ \dfrac{V}{R_n} & \text{if } V\le 0\end{cases} \tag{2.20.a}$$

Now writing the equation for the circuit at the center of figure 2.4, we get:

$$i_n=g_n(f_n(\vec{V})) \tag{2.20.b}$$

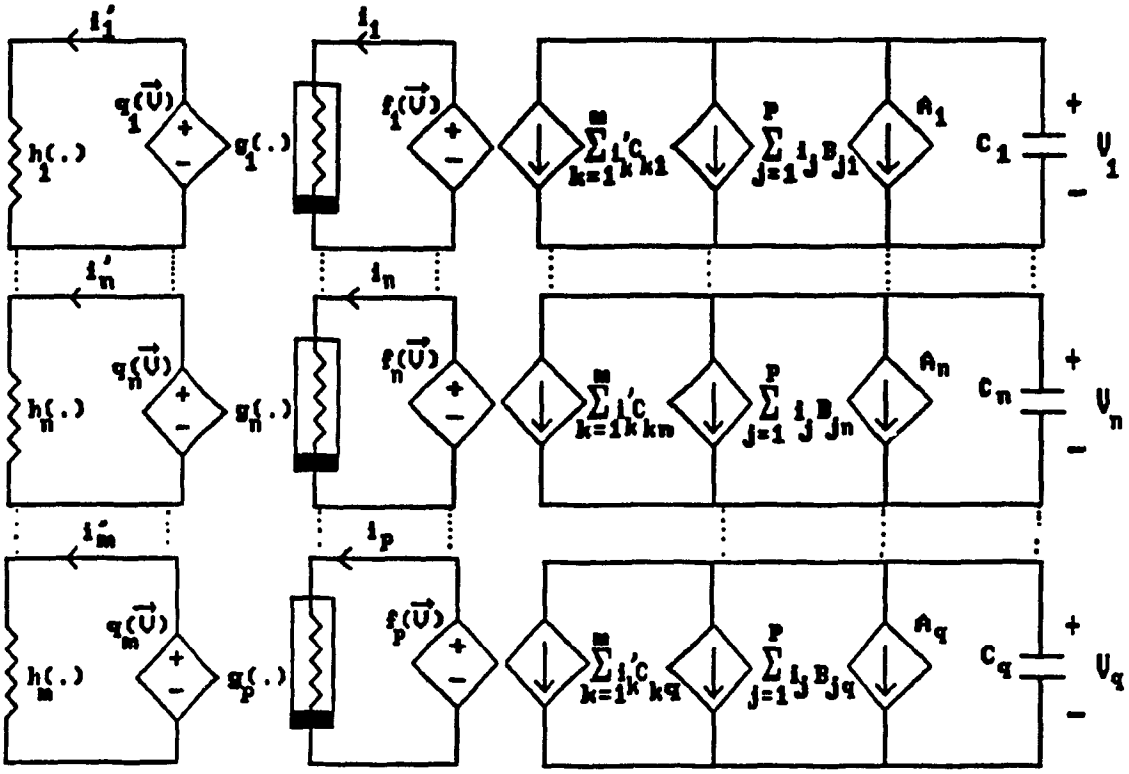


Figure 2.4. The extended Linear Programming circuit

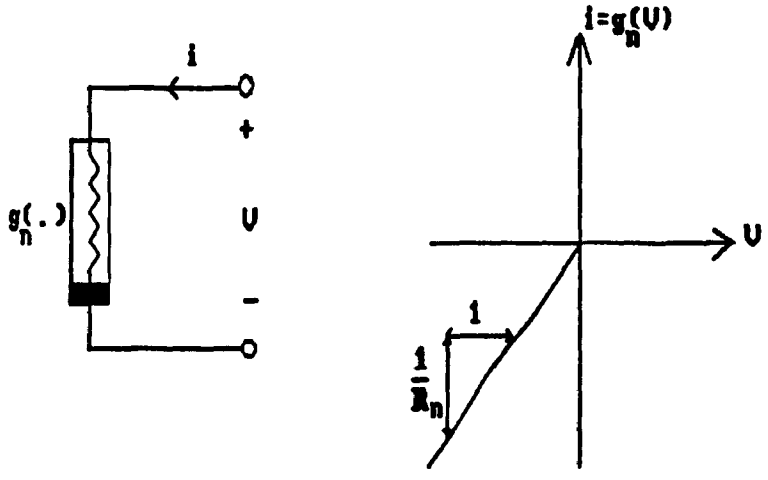


Figure 2.5. Chua and Lin Nonlinear Resistor Characteristic

The equation of the circuit on the left hand side of figure 2.4 is given by:

$$i_n' = h_n(q_n(\vec{V})) = \frac{q_n(\vec{V})}{R_{on}} \tag{2.21}$$

The circuit equation corresponding to the n^{th} row of the network shown in figure 2.4 is given by:

$$C_n \frac{dV_n}{dt} = -A_n - \sum_{j=1}^{p} B_{jn} \cdot i_j - \sum_{k=1}^{m} C_{kn} \cdot i_k' \tag{2.22.a}$$

$$= -A_n - \sum_{j=1}^{p} B_{jn} g_j(f_j(\vec{V})) - \sum_{k=1}^{m} C_{kn} h_k(q_k(\vec{V})) \tag{2.22.b}$$

Next substituting from (2.17.a) and (2.17.b) for $f_j(\vec{V})$ and $q_k(\vec{V})$ respectively, we get:

$$C_n \frac{dV_n}{dt} = -A_n - \sum_{j=1}^{p} B_{jn} g_j(\vec{B_j} \cdot \vec{V} - E_j) - \sum_{k=1}^{m} C_{kn} h_k(\vec{C_k} \cdot \vec{V} - F_k) \tag{2.23}$$

To see how the model shown in figure 2.4 solves the Linear Programming problem formulated in (2.16-2.18) consider the scalar function:

$$P^* : \mathbb{R}^q \longrightarrow \mathbb{R}$$

$$\vec{V} \longrightarrow P^*(\vec{V}) = \vec{A} \cdot \vec{V} + \sum_{j=1}^{p} G_j(\vec{B_j} \cdot \vec{V} - E_j) + \sum_{k=1}^{m} H_k(\vec{C_k} \cdot \vec{V} - F_k) \tag{2.24}$$

where:

$$G_j(V) = \int_0^V g_j(x)dx = \begin{cases} 0 & \text{if } V > 0 \\ \dfrac{V^2}{2R_j} & \text{if } V \le 0 \end{cases} \tag{2.25.a}$$

$$H_k(V) = \int_0^V h_k(x)dx = \frac{V^2}{2R_{ok}} \tag{2.2.b}$$

Note that the scalar function P^* takes care of both the equality and the inequality constraints implicitly through the two sequences of quadratic loss terms, namely:

$$\sum_{k=1}^{m} \frac{(\overrightarrow{C_k} \cdot \overrightarrow{V} - F_k)^2}{2R_{ok}} \text{ and } \sum_{j=1}^{p} \frac{(\overrightarrow{B_j} \cdot \overrightarrow{V} - E_j)^2}{2R_j}.$$

Taking the time derivative of P^* we get :

$$\frac{dP^*}{dt} = \sum_{n=1}^{q} \frac{\partial P^*}{\partial V_n} \cdot \frac{dV_n}{dt} \tag{2.26}$$

From (2.24) we have:

$$\frac{\partial P^*}{\partial V_n} = A_n + \sum_{j=1}^{p} B_{jn} g_j(\overrightarrow{B_j} \cdot \overrightarrow{V} - E_j) + \sum_{k=1}^{m} C_{kn} h_k(\overrightarrow{C_k} \cdot \overrightarrow{V} - F_k) \tag{2.27}$$

Therefore:

$$\frac{dP^*}{dt} = \sum_{n=1}^{q} \left(A_n + \sum_{j=1}^{p} B_{jn} g_j(\overrightarrow{B_j} \cdot \overrightarrow{V} - E_j) + \sum_{k=1}^{m} C_{kn} h_k(\overrightarrow{C_k} \cdot \overrightarrow{V} - F_k) \right) \cdot \frac{dV_n}{dt} \tag{2.28}$$

From (2.23) the expression inside parentheses in (2.28) is nothing but $-C_n \cdot \frac{dV_n}{dt}$. Hence we have:

$$C_n \frac{dV_n}{dt} = -\frac{\partial P^*}{\partial V_n} \tag{2.29.a}$$

$$\frac{dP^*}{dt} = -\sum_{n=1}^{q} C_n \cdot \left(\frac{dV_n}{dt} \right)^2 \leq 0 \tag{2.29.b}$$

and

$$\frac{dP^*}{dt} = 0 \Leftrightarrow \frac{dV_n}{dt} = 0 \qquad ; \forall n \in \{1,2,..q\} \tag{2.30}$$

We therefore conclude that the state of the network described by (2.23) follows a gradient descent of the scalar function P^*, and that starting from some initial condition, this state

evolves towards a minima of P^* and stabilizes towards a stationary point when $\frac{dV_k}{dt} = 0 \ \forall k \in \{1,2,...q\}$.

In order to relate the solution of the Linear Programming problem (2.16-2.19) to the minimum of the scalar function P^* we note that P^* consists of the objective function Φ plus a sequence of quadratic penalty functions for constraints' violations, so that the solution of the new unconstrained problem, formulated in (2.24) approaches that of the original constrained problem (2.16-2.19), provided that the conductance values $\frac{1}{R_{ok}}$ and $\frac{1}{R_j}$ are sufficiently high.

Since the components of the state vector $\vec{V}$ are normally the output voltages of op-amps and since these latter are bounded by their saturation levels, then the state space of the Linear Programming circuit is enclosed within the hypercube in R^q, defined by:

$$H = \{ \vec{V} \in R^q / V_i \in [V_i^{min}, V_i^{max}], i = 1,2,...q \}$$

where V_i^{min} and V_i^{max} are the minimum and maximum values the op-amp outputs can assume respectively. Fortunately one can always scale the state vector $\vec{V}$ so that the stationary point of the Linear Programming network is always kept inside the hypercube H [9].

2.3.2 Network Implementation

The extended model shown in figure 2.4 can be implemented using solid-state devices as illustrated in figure 2.6. The entries of vectors $\vec{A}$, $\vec{E}$ and $\vec{F}$ and those of matrices B and C are implemented as conductance values of some resistors. Each triangular shape symbol enclosing an integral sign is a neuron, which is also known as a variable amplifier, since its output is the value of a variable in the Linear Programming problem. Each elementary neuron is basically a summer followed by an integrator (figure 2.7) whose circuit implementation is shown in figure 2.8 [9].

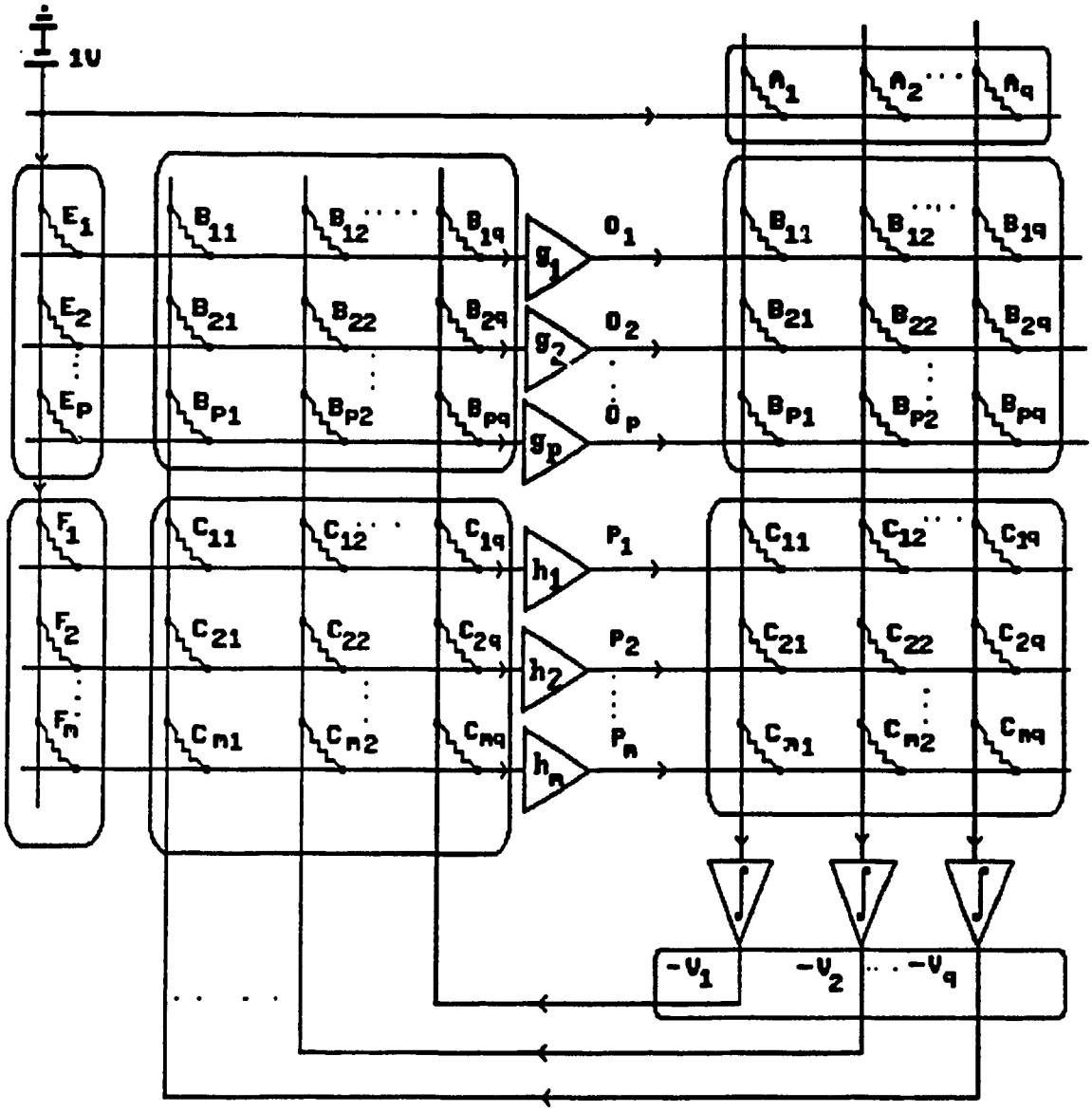


Figure 2.6. A Circuit Implementation of the Extended Model

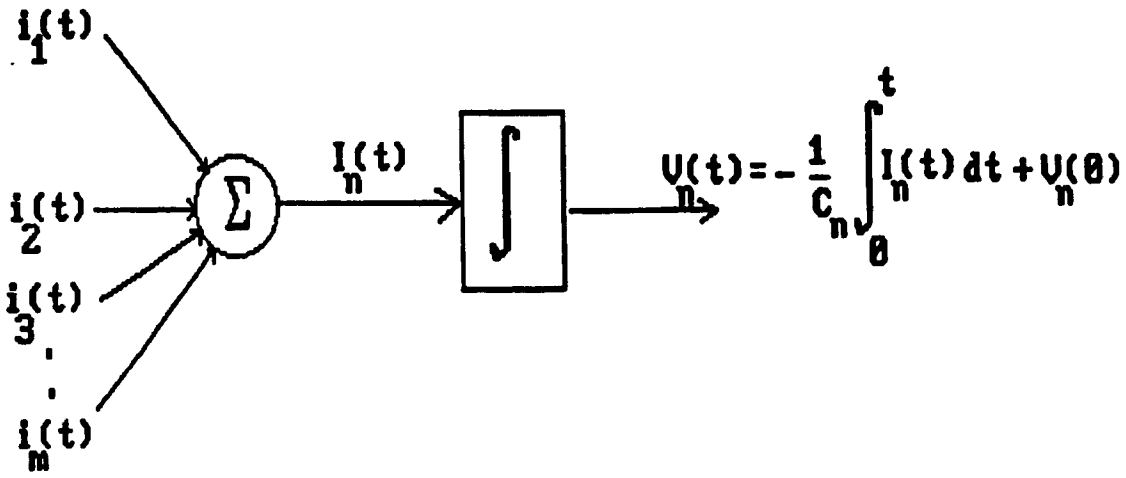


Figure 2.7. Symbol of a Neuron

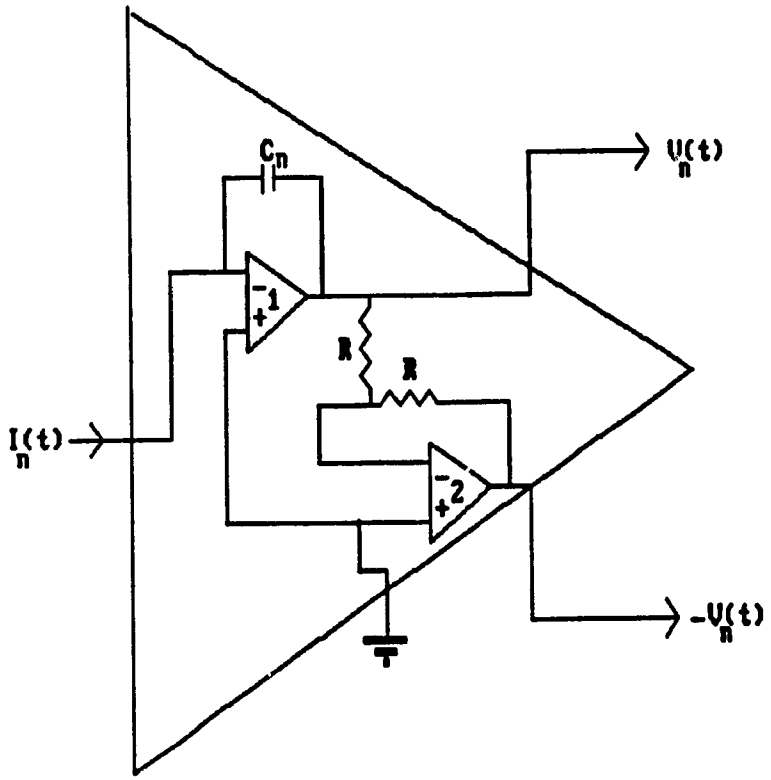


Figure 2.8. Circuit Implementation of a Neuron

Each triangular shape symbol labeled h_k, in figure 2.6 is a Linear Equality Constraint Module whose detailed circuitry is shown in figure 2.9.

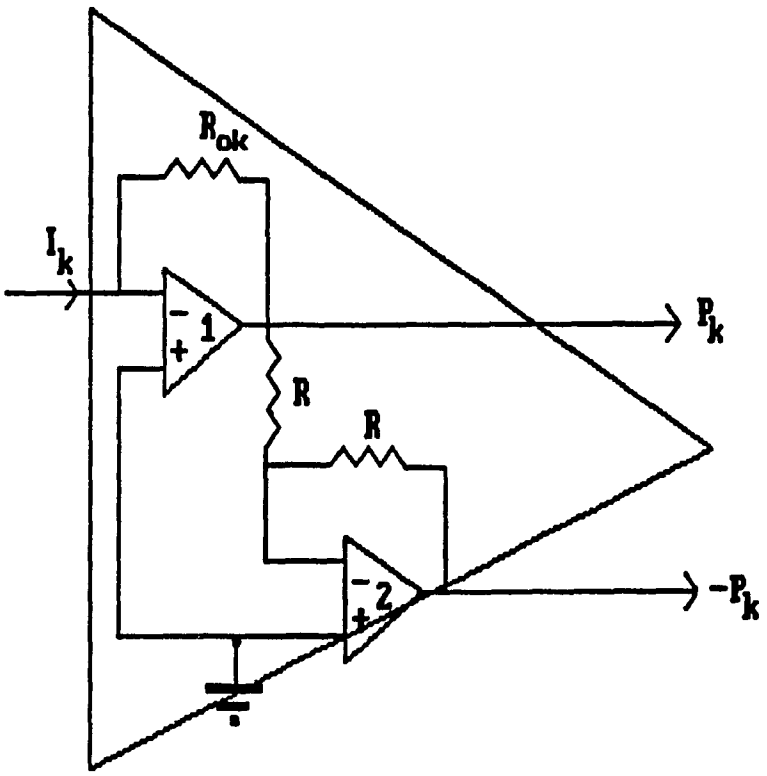


Figure 2.9. Linear Equality Constraint Module

The input-output relation of a Linear Equality Constraint Module is given by :

$$P_k = -R_{ok} I_k \tag{2.31}$$

and an inverted output ($-P_k$) is obtained through op-amp 2 and resistors R.

Each triangular shape symbol labeled g_i in figure 2.6 is a nonlinear inequality constraint module, whose circuit implementation is depicted in figure 2.10 [9].

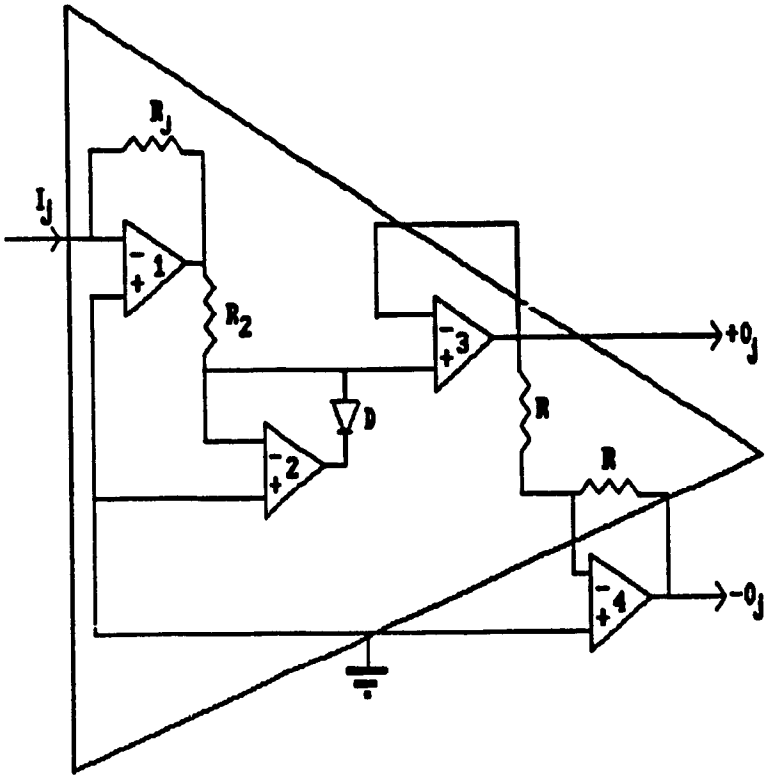


Figure 2.10. Nonlinear Inequality Constraint Module

The input-output relation of a nonlinear inequality constraint module is given by:

$$O_j = g_j(I_j) = \begin{cases} -I_j \cdot R_j & \text{if } I_j \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.32}$$

and op-amp 4 and resistors R are added to obtain an inverted output (- O_j) from the normal output. Referring back to figure 2.5, it can be seen that rather than being the driving point characteristic of a nonlinear resistor, the nonlinearity g_j is now interpreted as the transfer characteristic of a current controlled voltage source. It is also important to note that each neuron and each equality and inequality constraint module maintains a virtual ground at its inputs.

To show how the network of figure 2.6 actually implements the extended linear

Programming model shown in figure 2.4, consider an arbitrary Linear Equality Constraint Module. Its input current (see figures 2.6 and 2.9) is:

$$I_k = -\sum_{j=1}^{q} C_{kj} V_j + F_k \qquad (2.33.a)$$

$$= -(\overrightarrow{C_k} \cdot \vec{V} - F_k) \qquad (2.33.b)$$

Hence from (2.17.b) we have:

$$I_k = -q_k(\vec{V}) \qquad (2.34)$$

Its output voltage (2.31) is :

$$P_k = h_k(q_k(\vec{V})) = R_{ok} q_k(\vec{V}) \qquad (2.35)$$

The linearity h_k may therefore be looked upon as the transfer characteristic of a current controlled voltage source. Next let us consider an arbitrary nonlinear inequality constraint module. Its input current (see figures 2.6 and 2.10) is:

$$I_j = -\sum_{k=1}^{q} B_{jk} V_k + E_j \qquad (2.36.a)$$

$$= -(\overrightarrow{B_j} \cdot \vec{V} - E_j) \qquad (2.36.b)$$

Therefore from (2.17.a) we have:

$$I_j = -f_j(\vec{V}) \qquad (2.37)$$

Its output voltage (2.32) is:

$$O_j = g_j(f_j(\vec{V})) = \begin{cases} R_j \cdot f_j(\vec{V}) & \text{if } f_j(\vec{V}) \le 0 \\ 0 & \text{if } f_j(\vec{V}) > 0 \end{cases} \qquad (2.38)$$

Now consider an arbitrary neuron. Its input current is:

$$I_n = A_n + \sum_{j=1}^{p} B_{jn} O_j + \sum_{k=1}^{m} C_{km} P_k \qquad (2.39.a)$$

$$= A_n + \sum_{j=1}^{p} B_{jn} g_j(f_j(\vec{V})) + \sum_{k=1}^{m} C_{km} h_k(q_k(\vec{V})) \qquad (2.39.b)$$

Its output voltage is described by :

$$C_n \frac{dV_n}{dt} = -A_n - \sum_{j=1}^{p} B_{jn} g_j(\vec{B_j} \bullet \vec{V} - E_j) - \sum_{k=1}^{m} C_{km} h_k(\vec{C_k} \bullet \vec{V} - F_k) \qquad (2.40)$$

The above expression is readily identified as the state equation (2.23) of the extended linear programming model to be implemented.

In the previous analysis it has been assumed that the entries of vectors $\vec{A}$, $\vec{E}$ and $\vec{F}$ and those of matrices B and C are all positive. If any of these entries is negative, then the virtually non-grounded terminal of its corresponding resistor is connected to -1V, $+V_i$, $-O_i$ or $-P_i$, as required. The analysis has also assumed that the outputs of the linear and nonlinear equality constraints modules are updated instantaneously in the feedback path, in response to changes in neuron voltages. Although parasitics in these constraint modules may slow down their output updating rate, the assumption remains reasonable if the capacitances of the elementary neurons are kept large enough so as to allow enough time for the constraint modules to update their outputs [9].

In the special case, where all components of $\vec{F}$ and C are zero, then the proposed model reduces to Chua and Lin Linear Programming network, described in appendix A. Therefore instead of using two nonlinearities to implement (2.15), the extended model offers the alternative to dedicate a simple linearity to handle each equality constraint.

CHAPTER 3

A NEURAL NETWORK APPROACH TO THE MAXIMUM FLOW PROBLEM

3.1 Introduction

In this chapter we will demonstrate the computational power of neural networks by solving a traffic routing problem where the goal is to find the maximum throughput of a communication network. We define a communication network as a set of n nodes interconnected by a given set of m links, each having a finite capacity. We are also given a set of K source-destination pairs, where a message belonging to a given pair defines a commodity. Each commodity originates at a particular source and is to be routed to its corresponding destination through any of the available paths between them. This results in a flow of K commodities in the network. The total throughput of the network, defined as the sum of all flow values of the K commodities, is restricted by the finiteness of link capacities.

By definition, a traffic flow is said to be feasible if it is positive and satisfies both link capacity constraints and stations' flow balance requirements. By link capacity constraint we mean that the aggregate flow due to all commodities cannot exceed the capacity of the link over which it is transmitted. For each commodity, the flow balance requirement at each node depends on the type of that node relative to that commodity. Define the total outflow (inflow) at a particular node as the sum of all flows coming out (in) that station. Then for each commodity, the flow balance requirement is stated as follows:

- If the node is the source of that commodity then its total outflow minus its total inflow must equal commodity flow value.

- If the node is the destination of that commodity then its total inflow minus its total outflow must equal commodity flow value.
- If the station is an intermediate node then its total inflow must equal its total outflow.

Although each commodity flow has its own flow balance requirement at each station, flows of different commodities interact and compete for the residual link capacity, as the flow of a given commodity is likely to share link capacity with flows belonging to other commodities. It should be noted however that flows belonging to two different commodities do not cancel out when flowing in opposite directions [15,16]. The maximum flow (Max-Flow) problem is therefore to find a feasible traffic flow assignment that maximizes the network throughput.

For the one and two-commodity cases efficient labeling techniques based on the Max-Flow-Min-Cut theorem have been proposed [17,18]. In particular Ford and Fulkerson [17] proved that the single commodity flow problem is integer; ie if the arc capacities are integer then the maximum flow is also integer. Hu [19] showed that the two-commodity problem is also integer for even arc capacities. This integrality property can not however be extended to the general multicommodity case [15]. When dealing with three or more commodities, the efficiency of the labeling technique can no longer be exploited except for a special three-commodity problem in which all nodes are either sources or destinations [15].

For general problems involving more than two commodities only Linear Programming techniques have been proposed [20]. However for networks with a large number of arcs, conventional Linear Programming techniques based on the simplex algorithm become inefficient because of their excessive storage and running time requirements. For such problems approximate heuristic methods [21] have been proposed.

3.2 Problem Formulation

Consider a directed graph $G = (\overline{N}, \overline{A})$, where $\overline{N}$ is the set of n nodes and $\overline{A}$ is the set of m links (arcs) forming this graph. To G we assume that K source-destination pairs are priori specified.

Let:

(s^k, t^k) = source-destination pair corresponding to commodity k; $k \in \{1,2,...K\}$.

C_a^* = capacity of link a ; $a \in \overline{A}$.

f_a^k = traffic flow of commodity k on link a ; $k \in \{1,2,..,K\}$; $a \in \overline{A}$.

v^k = traffic flow originating at s^k and destined to t^k ; $k \in \{1,2,..,K\}$

$f_a = \sum_{k=1}^{K} f_a^k$ = aggregate flow on link a ; $a \in \overline{A}$

$\Gamma_i = \{a \in \overline{A}/$ link a is leaving node i $\}$

$\Gamma_i^* = \{a \in \overline{A}/$ link a is entering node i $\}$.

The directed multicommodity Max-Flow problem is then formulated as follows:

Maximize:

$$\sum_{k=1}^{K} v^k \tag{3.1}$$

Subject to:

$$\sum_{a \in \Gamma_i} f_a^k - \sum_{a \in \Gamma_i^*} f_a^k = \begin{cases} v^k & \text{if } i = s^k \\ -v^k & \text{if } i = t^k \\ 0 & \text{if } i \neq s^k, t^k \end{cases} \tag{3.2}$$

$$\forall k \in \{1,2,...K\}$$

$$f_a^k \geq 0 \ ; \forall (a,k) \in \overline{A} \times \{1,2,..K\} \tag{3.3}$$

$$v^k \geq 0 \ ; \forall k \in \{1,2,..,K\} \tag{3.4}$$

$$\sum_{k=1}^{K} f_a^k \le C_a^* ; \forall a \in \overline{A} \qquad (3.5)$$

The objective function (3.1) gives the total throughput of the network. Constraint (3.2) corresponds to the flow balance requirements at each node. Constraints (3.3 - 3.4) ensure that all flow values are positive, while (3.5) ensures that all link capacity constraints are satisfied.

3.3 A Neural Network Approach

Recall that to solve the Max-Flow problem using the extended Linear Programming network, one has to reformulate the problem in the form of (2.16 - 2.19). First the maximization problem is converted to a minimization problem by reversing the sign of the objective function since:

Maximizing $\sum_{k=1}^{K} v^k \Leftrightarrow$ minimizing $-\sum_{k=1}^{K} v^k$.

Then most of the complexity of the neural network approach resides in the programming of the resistive interconnections, which are displayed in the entries of matrices B and C and vectors $\overrightarrow{A}$, $\overrightarrow{E}$ and $\overrightarrow{F}$. In our case the unknowns are the link flows f_a^k and the traffic flows v^k, corresponding to each commodity k.

In addition to constrains ((4.2) to (4.5)), we added K upper-bound constraints on all flow values v^k :

$$v^k \le S_c(s^k, t^k) = \min\{\sum_{a \in \Gamma_{s^k}} C_a^*, \sum_{a \in \Gamma_{t^k}^*} C_a^*\} \qquad (3.6)$$

$$\forall k \in \{1,2,..,K\}$$

These upperbounds on all flow values v^k correspond to the situation where all links leaving s^k or entering t^k are saturated. Additional flows might be lost due to the flow balance

requirements at intermediate nodes and the interaction of traffic flows belonging to different commodities. These upperbound constraints reduce the feasible state space of the network and therefore speed up the search for the optimum solution. This hypotheses was backed by simulation results.

The following notations will be used in the mathematical formulation of the neural network approach to the Max-Flow problem :

I = Identity matrix

O = Zero matrix $(n \times (m+1))$

$W = [w_{ij}]_{(n \times m)}$ = Incidence matrix of the network ; where:

$$w_{ij} = \begin{cases} +1 & \text{if link j} \in \Gamma_i \\ -1 & \text{if link j} \in \Gamma_i^* \\ 0 & \text{otherwise} \end{cases}$$

$\vec{z}$ = Column vector with all zero entries

$\overrightarrow{g_k}$ = Column vector (n × 1) with zero entries except at entries corresponding to s^k and t^k where it takes on the values -1 and +1 respectively

$C^k = [\overrightarrow{g_k} \mid W]_{(n \times (m+1))}$

$I_k^* = [\vec{z}_{(m \times 1)} \mid -I_{(m \times m)}]_{(m \times (m+1))}$

$J_k = [\, j_{xy}]_{(K \times (m+1))}$; where :

$$j_{xy} = \begin{cases} -1 & \text{If } (x,y) = (k,1) \\ 0 & \textit{otherwise} \end{cases}$$

$$\vec{U} = \begin{bmatrix} S_c(s^1, t^1) \\ S_c(s^2, t^2) \\ \cdot \\ \cdot \\ S_c(s^K, t^K) \end{bmatrix}_{(K \times 1)}$$

$$\vec{CAP} = \begin{bmatrix} C_1^* \\ C_2^* \\ \cdot \\ \cdot \\ C_m^* \end{bmatrix}_{(m \times 1)} = \text{Link capacity vector}$$

$\vec{R_k}$ = column vector $((m+1) \times 1)$ with zero entries except at the first entry where it takes on the value -1

$$\vec{V_k} = \begin{bmatrix} v^k \\ f_1^k \\ f_2^k \\ \cdot \\ \cdot \\ f_m^k \end{bmatrix}_{(m+1) \times 1} = \text{flow vector corresponding to commodity k.}$$

With the above notation, vectors $\vec{A}, \vec{V}, \vec{E}, \vec{F}$ and matrices B and C are readily obtained by putting equations (3.1 - 3.6) in the standard form (2.16 - 2.19), giving:

$$\vec{V}=\begin{bmatrix}\vec{V_1}\\---\\\vec{V_2}\\---\\.\\.\\.\\---\\\vec{V_K}\end{bmatrix}_{(K(m+1)\times 1)} ; \vec{A}=\begin{bmatrix}\vec{K_1}\\---\\\vec{R_2}\\---\\.\\.\\.\\---\\\vec{R_K}\end{bmatrix}_{(K(m+1)\times 1)} ; \vec{E}=\begin{bmatrix}\vec{z}_{(K(m+1)\times 1)}\\----\\-\overrightarrow{CAP}\\----\\-\vec{U}\end{bmatrix}_{(2K+m(1+K)\times 1)}$$

$$\vec{F}=\vec{z}_{(Kn+1)} ; B=\begin{bmatrix}I_{K(m+1)\times K(m+1)}\\---------\\I_1^*\,|\,I_2^*\,|\ldots|\,I_K^*\\---------\\J_1\,|\,J_2\,|\ldots|\,J_K\end{bmatrix}_{(2K+m(1+k)\times K(m+1))} ; C=\begin{bmatrix}C^1\,|\,0\,|\,0\,|\ldots|\,0\\----------\\0\,|\,C^2\,|\,0\,|\ldots|\,0\\----------\\0\,|\,0\,|\,C^4\,|\ldots|\,0\\----------\\.....\\.....\\----------\\0\,|\,0\,|\,0\,|\ldots|\,C^K\end{bmatrix}_{(Kn\times K(m+1))}$$

Hence for the Max-Flow problem the extended LP circuit would require :

- K.(m+1) neurons, one for each flow variable.
- 2.K+m.(1+K) nonlinear inequality constraint modules, one for each inequality constraint.
- K.n linear equality constraint modules, one for each equality constraint.

Where m, n and K are the number of links, nodes and commodities, respectively.

3.4 Simulation Results

In this section we report some simulation results of the neural network approach to the Max-Flow problem. We have considered one single commodity and two multicommodity networks as illustrating examples.

3.4.1 A Single Commodity Case

Consider the network shown in figure 3.1, where it is required to find the maximum flow v^1 from node s^1 to node t^1. Each link is labeled by two numbers; with the first number denoting link index and the second number inside parentheses denoting link capacity.

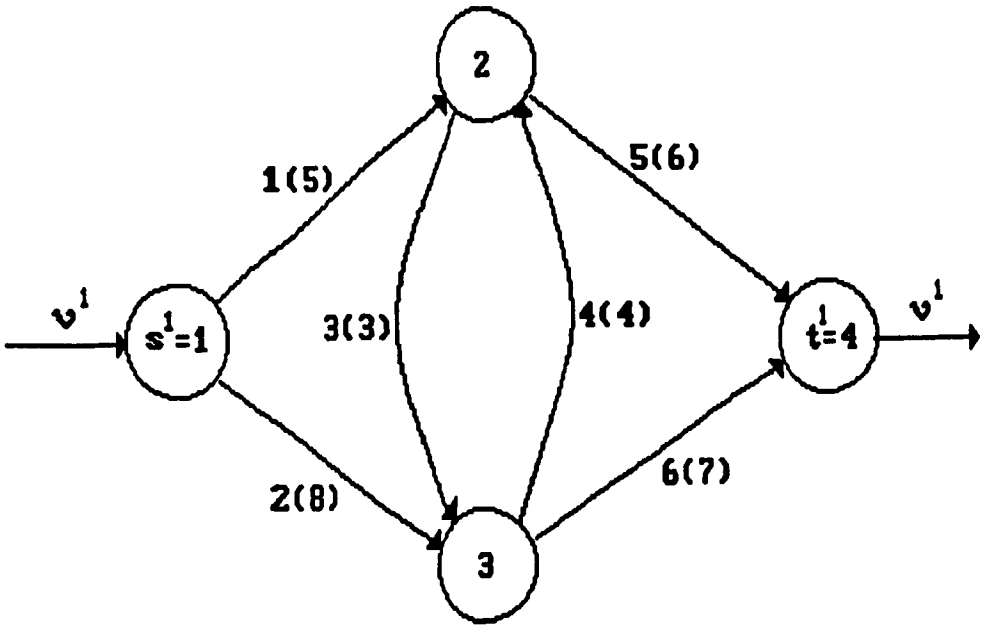


Figure 3.1. A Single Commodity Network

The state vector of the corresponding Linear Programming network is:

$$\vec{V} = \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ V_7 \end{bmatrix} = \begin{bmatrix} v^1 \\ f_1^1 \\ f_2^1 \\ f_3^1 \\ f_4^1 \\ f_5^1 \\ f_6^1 \end{bmatrix}$$

and the resistive interconnections are described by:

$$\vec{A} = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}; \vec{E} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -5 \\ -8 \\ -3 \\ -4 \\ -6 \\ -7 \\ -13 \end{bmatrix}; \vec{F} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}; C = \begin{bmatrix} -1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 1 & 0 \\ 0 & 0 & -1 & -1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & -1 \end{bmatrix}; B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ - & - & - & - & - & - & - \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ - & - & - & - & - & - & - \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

From the above expressions, and referring back to figure 2.6, the extended Linear Programming network that solves this problem is obtained as shown in figure 3.2. As may be seen, there are seven neurons, fourteen inequality constraint modules and 4 equality constraint modules.

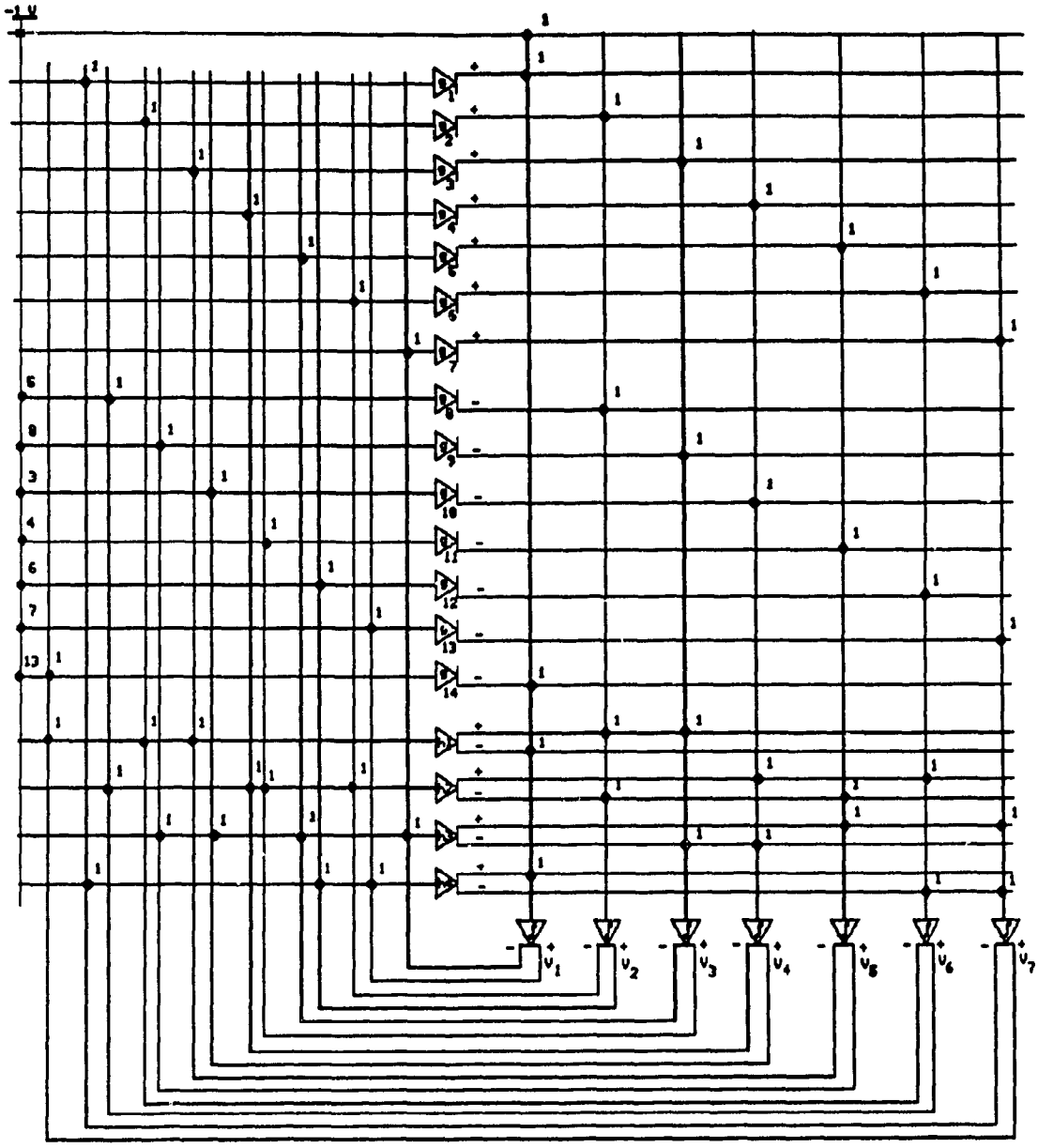


Figure 3.2. The Neural Network for the Single Commodity Example (Black circles at intersections and associated numbers denote resistive connections and their corresponding conductance values)

The equation of motion of a particular neuron (see 2.23) is given by :

$$C_n \frac{dV_n}{dt} = -A_n - \sum_{j=1}^{14} B_{jn} g_j(\vec{B_j} \cdot \vec{V} - E_j) - \sum_{k=1}^{4} C_{kn} h_k(\vec{C_k} \cdot \vec{V} - F_k) \qquad (3.7)$$

$$\forall n \in \{1, 2, \ldots 7\}$$

or

$$\frac{dV_n}{dt} = \frac{-1}{C_n} \cdot \left\{ A_n + \sum_{j=1}^{14} B_{jn} g_j(\vec{B_j} \cdot \vec{V} - E_j) + \sum_{k=1}^{4} C_{kn} h_k(\vec{C_k} \cdot \vec{V} - F_k) \right\} \qquad (3.8)$$

The left side of (3.8) is approximated with its first order difference giving:

$$V_n(t + \delta t) = V_n(t) - \frac{\delta t}{C_n} \cdot \left\{ A_n + \sum_{j=1}^{14} B_{jn} g_j(\vec{B_j} \cdot \vec{V} - E_j) + \sum_{k=1}^{4} C_{kn} h_k(\vec{C_k} \cdot \vec{V} - F_k) \right\} \qquad (3.9)$$

where δt is the incremental time for the updates.

Simulation of the neurons' dynamics therefore involves simultaneous solution of 7 nonlinear differential equations. The simulation, written in Fortran, consists of observing and updating neuron states simultaneously at incremental time steps δt as given in (3.9). Throughout the simulation it was assumed that the system reaches steady state if all elementary neuron voltages did not change by more than a threshold $\Delta V_{th} = 10^{-5}$ volts. To give the system its units, we have chosen:

$$C_n = 10 \text{ nF}$$

$$R_j = R_{ok} = 10K\Omega$$

all independent of their respective subscripts, for simplicity. In addition all remaining resistances in the network (black dots in figure 3.2) have been normalized using a 1 $K\Omega$ resistance, R_{no}. Therefore if a resistor in figure 3.2 reads X, then it was simulated as an $\frac{X}{1000}$ $\mho$ resistor.

A preliminary series of trials revealed that the response time of the extended Linear Programming network is very fast since it was observed that the voltage of a particular neuron oscillates between very small and very large values, without reaching steady state. Therefore it was necessary to reduce the incremental updating time δt to a very small value so as to properly observe the dynamics of the simulated network. For our case a value of 10^{-7} seconds has been chosen for δt. Decreasing this value did not improve our results, but simply increased simulation time. It was also observed that increasing the constraint resistances R_j and R_{ot} yield better results. However if these resistances are allowed to exceed a threshold value then the elementary neuron voltages oscillate without ever converging. Another equally important finding, revealed by simulation, is that the extended Linear Programming model does not require a feasible initial solution in order to converge. Under zero initial conditions, the response of the simulated extended Linear Programming network of figure 3.2 is obtained as shown in figures (3.3-3.9). From (2.24) the corresponding scalar function P^* is also plotted as shown in figure 3.10.

These results confirm that the state of the extended Linear Programming circuit evolves so as to decrease the scalar function P^* and that at steady state $\frac{dP^*}{dt} = 0$. In addition the steady state value of the penalty function, P^*_{ss} , is equal to the normalized minimum objective function :

$$P^*_{ss} = \vec{A} \cdot \vec{V}\,|_{\vec{V}_{ss}} \tag{4.10}$$

where $\vec{V}ss$ denotes the steady state vector.

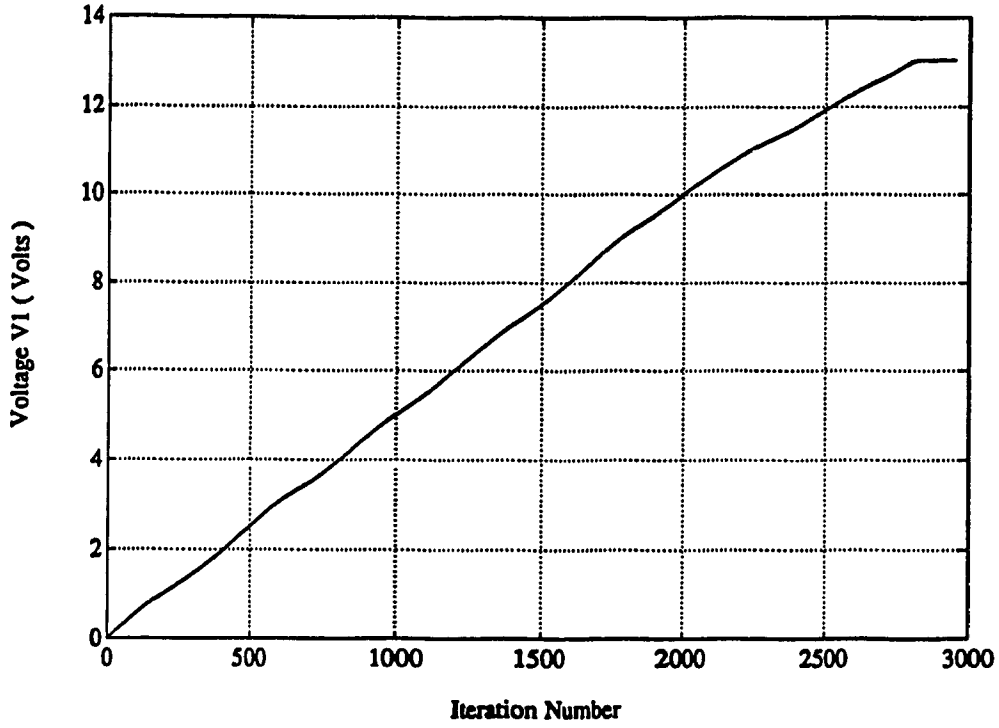


Figure 3.3. Neuron Voltage V_1 (Throughput)

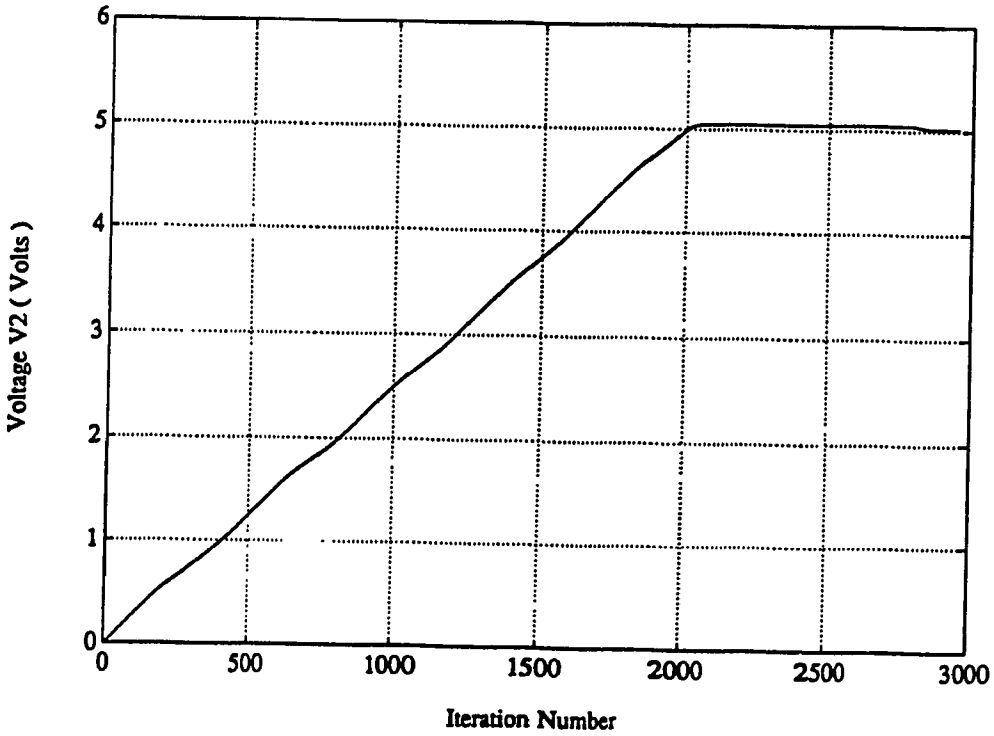


Figure 3.4. Neuron Voltage V_2 (Flow on link 1)

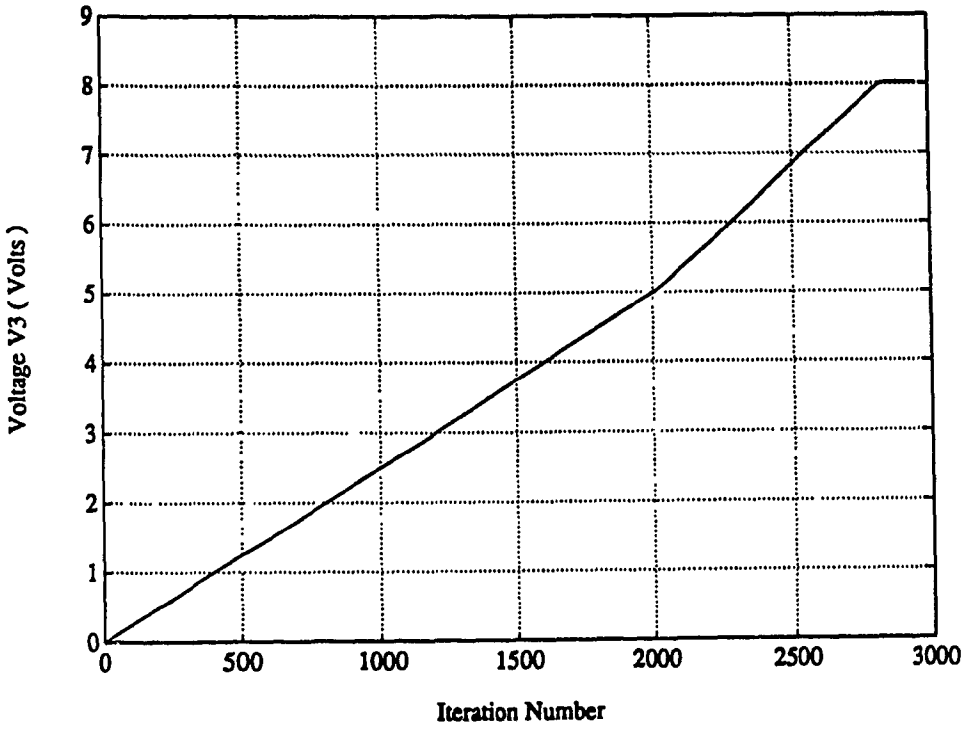


Figure 3.5. Neuron Voltage V_3 (Flow on link 2)

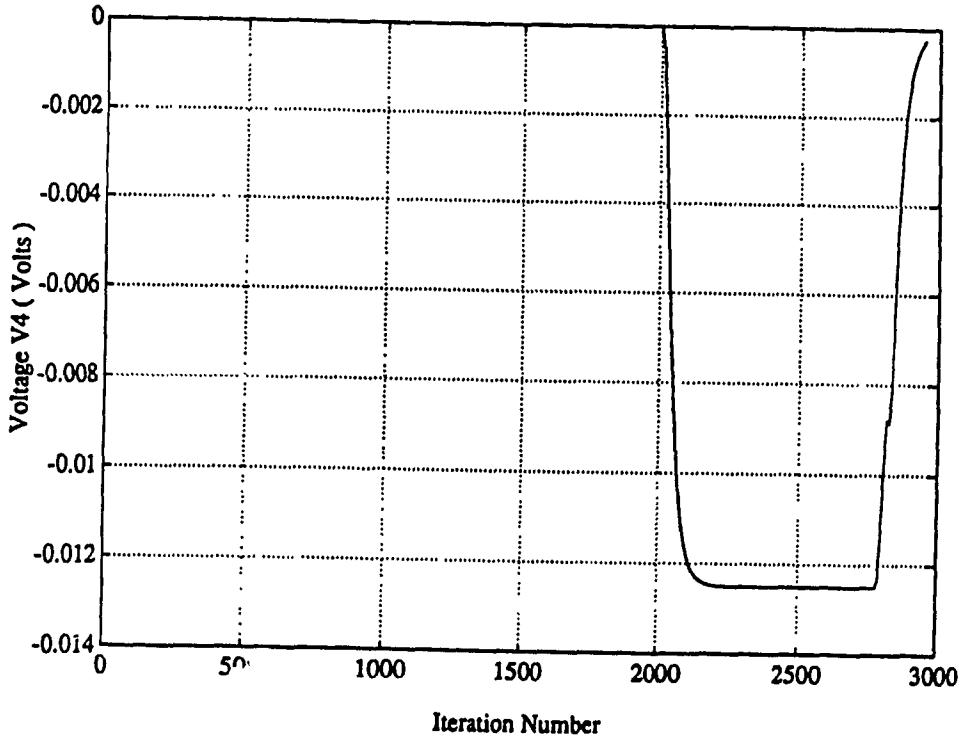


Figure 3.6. Neuron Voltage V_4 (Flow on link 3)

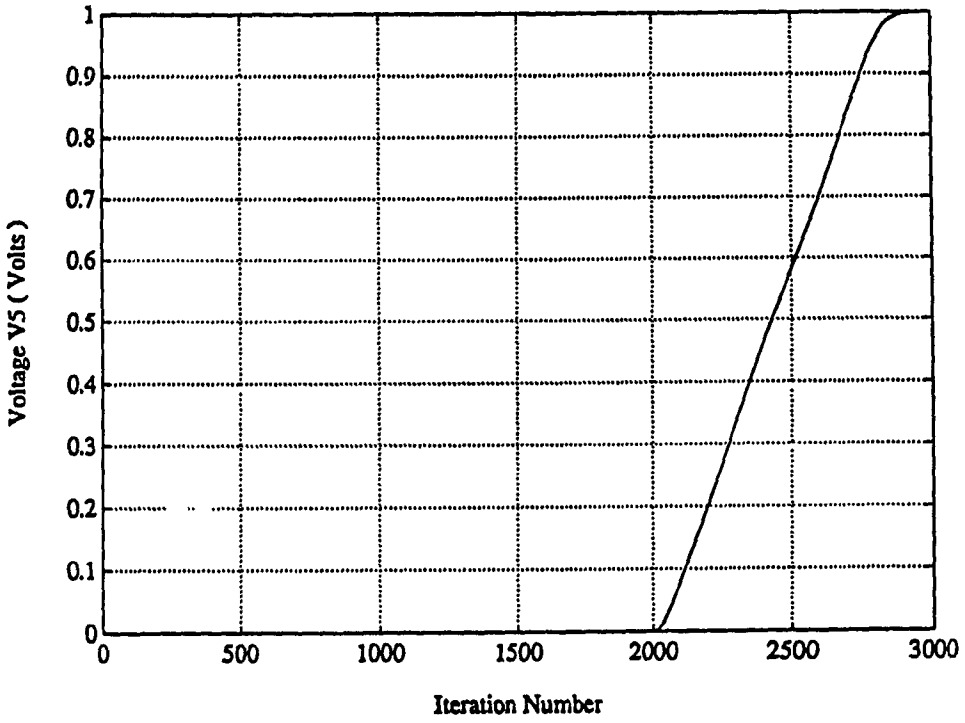


Figure 3.7. Neuron Voltage V_5 (Flow on link 4)

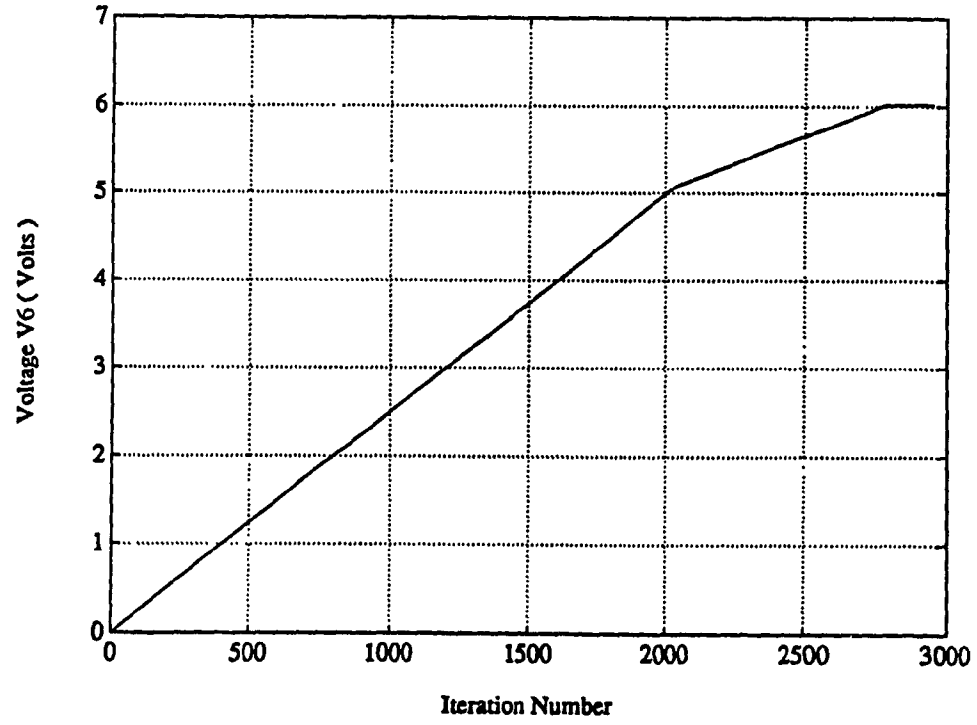


Figure 3.8. Neuron Voltage V_6 (Flow on link 5)

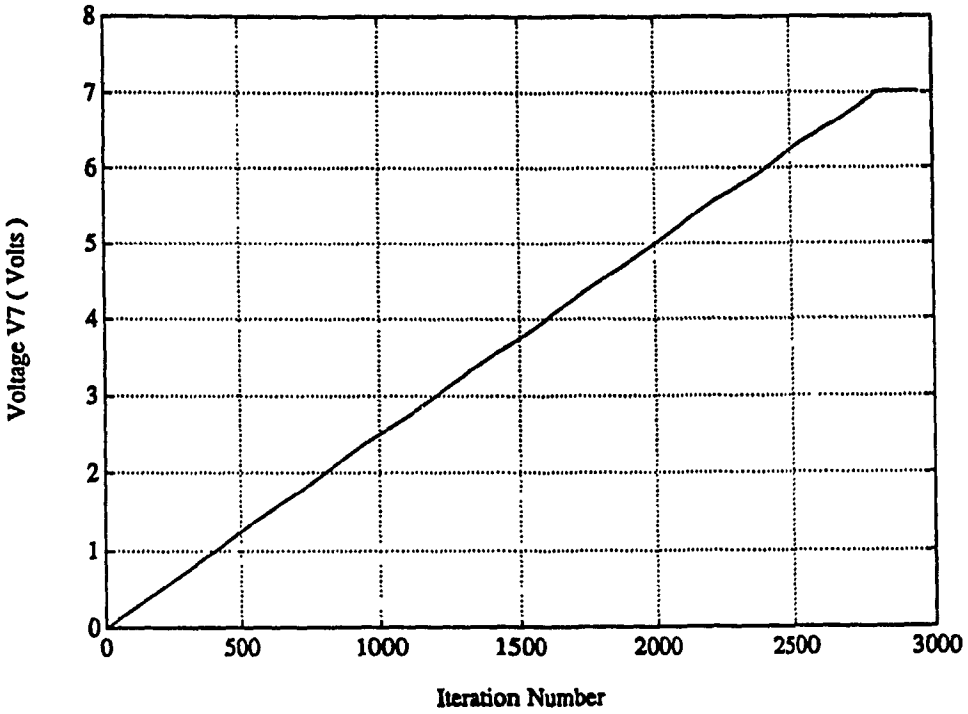


Figure 3.9. Neuron Voltage V_7 (Flow on link 6)

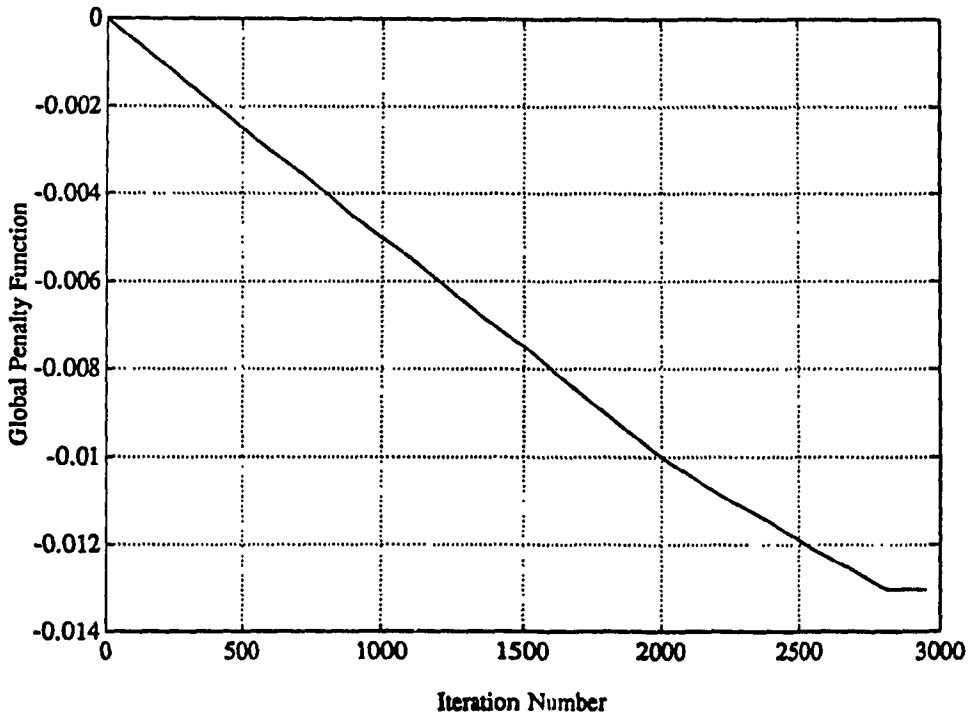


Figure 3.10. Scalar Function of the Single Commodity Network

Simulation results for three different initial conditions are illustrated in table 3.1, where the initial state vectors are given by:

$$\vec{V}_{in}^{(1)}=\begin{bmatrix}0\\0\\0\\0\\0\\0\\0\end{bmatrix};\ \vec{V}_{in}^{(2)}=\begin{bmatrix}6\\4\\5\\2\\4\\5\\5\end{bmatrix};\ \vec{V}_{in}^{(3)}=\begin{bmatrix}14\\14\\14\\8\\9\\2\\9\end{bmatrix} \tag{3.11}$$

Table 3.1. Simulation Results for the Single Commodity Case

(Numbers in brackets denote exact values)

Initial state vector	Throughput v^1	f_1^1	f_2^1	f_3^1	f_4^1	f_5^1	f_6^1	Number of iterations
$\vec{V}_{in}^{(1)}$	13.06 (13)	5.82 (5)	8.81 (8)	-1.36×10^{-4} (0)	8.999 (1)	6.819 (6)	7.82 (7)	2978
$\vec{V}_{in}^{(2)}$	13.059 (13)	5.82 (5)	8.819 (8)	2.00 (2)	3.00 (3)	6.82 (6)	7.819 (7)	1262
$\vec{V}_{in}^{(3)}$	13.06 (13)	5.82 (5)	8.819 (8)	2.991 (3)	3.991 (4)	6.819 (6)	7.82 (7)	142

These results show that the maximum throughput of this single commodity network is 13 units and that there are more than one flow allocation strategy to achieve this purpose. In addition the closer the initial state vector is to the optimum solution, the faster is the response time of the extended Linear Programming network.

3.4.2 A Multicommodity Case

3.4.2.1. A Five Commodity Network

Consider the network shown in figure 3.11 where it is required to find the maximum throughput for the five commodities specified in table 3.2.

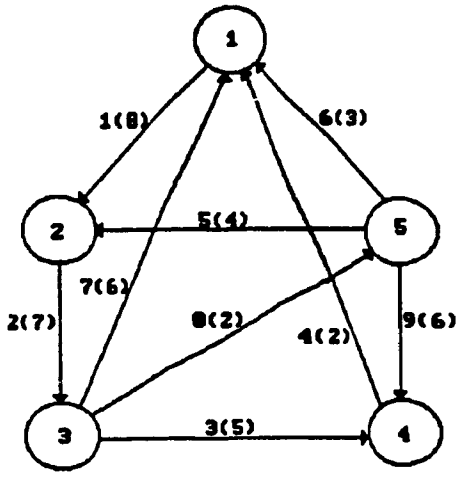


Figure 3.11. A Five Commodity Network

(Numbers inside (outside) brackets denote link capacities (link indexes))

Table 3.2. Commodities of the Network Shown in Figure 3.11

Commodity k	Source s^k	Destination t^k
1	1	5
2	1	4
3	5	4
4	2	5
5	3	1

In this case the equation of motion of a particular elementary neuron is:

$$C_n \frac{dV_n}{dt} = -A_n - \sum_{j=1}^{64} B_{jn} g_j(\overrightarrow{B_j} \cdot \vec{V} - E_j) - \sum_{k=1}^{25} C_{kn} h_k(\overrightarrow{C_k} \cdot \vec{V} - F_k) \qquad (3.11)$$

$$\forall n \in \{1,2,..,50\}$$

This problem requires 50 neurons, 64 nonlinear inequality constraint modules and 25 linear equality constraint modules. The dynamics of the corresponding extended Linear Programming network are simulated as in section 3.4.1 with $\delta t = 10^{-7}$ seconds and $C_n = 100nF$. After a series of trials the following constraint resistance values are found to yield good results:

$$R_{ok} = 550K\Omega$$

$$R_j = 650K\Omega$$

all independent of their respective subscripts. In addition all remaining resistances are normalized using a $1K\Omega$ resistance. It was assumed that steady state is reached if all elementary neuron voltages did not change by more than $\Delta V_{ss} = 10^{-4}$ volts. Decreasing this value did not improve the results, but just increased simulation time.

Under zero initial conditions, the extended Linear Programming network reached steady state after 7433 iterations, giving a maximum throughput of 19 units. Table 3.3 summarizes the corresponding flow allocation.

Table 3.3. Simulation Results for the Five Commodity Network Under Zero Initial Conditions

{ f_a^k is the flow on link a due to commodity k }

(Numbers inside brackets denote expected values)

Commodity k	f_1^k	f_2^k	f_3^k	f_4^k	f_5^k	f_6^k	f_7^k	f_8^k	f_9^k	v^k
1	0.883 (0.88)	0.881 (0.88)	3.683×10^{-4} (0)	-1.28×10^{-3} (0)	-1.56×10^{-3} (0)	-1.68×10^{-3} (0)	-1.54×10^{-3} (0)	0.882 (0.88)	-1.88×10^{-3} (0)	0.888 (0.88)
2	1.315 (1.30)	1.314 (1.30)	1.316 (1.30)	-1.56×10^{-3} (0)	-4.18×10^{-3} (0)	-3.98×10^{-4} (0)	-1.51×10^{-3} (0)	-1.24×10^{-3} (0)	-3.86×10^{-4} (0)	1.319 (1.30)
3	4.899×10^{-3} (0)	1.703 (1.7)	1.705 (1.7)	-1.54×10^{-3} (0)	1.699 (1.7)	7.54×10^{-3} (0)	-1.56×10^{-3} (0)	-1.55×10^{-3} (0)	6.806 (6.8)	7.715 (7.7)
4	-1.222×10^{-3} (0)	1.122 (1.12)	2.998×10^{-4} (0)	-4.48×10^{-4} (0)	-1.56×10^{-4} (0)	-4.68×10^{-4} (0)	-5.13×10^{-4} (0)	1.122 (1.12)	-1.59×10^{-3} (0)	1.126 (1.12)
5	-4.747×10^{-4} (0)	-1.87×10^{-3} (0)	1.979 (2.0)	1.978 (2.0)	-4.87×10^{-4} (0)	2.48×10^{-3} (0)	6.806 (6.8)	-2.43×10^{-5} (0)	-1.55×10^{-3} (0)	7.988 (8.0)
									Maximum throughput = $\sum_{k=1}^{5} v^k \rightarrow$	19.836 (19.8)

As shown in figure 3.12, again the dynamics of the extended Linear Programming circuit evolve so as to gradually reduce the scalar function P^*; until the maximum allowable

throughput is reached (figure 3.13). Once again it was found that this maximum flow of 19 units could be achieved through many other routing strategies. Table 3.4 gives one alternate flow allocation, which corresponds to each neuron having a 1 volt initial voltage. This flow allocation was obtained after 4925 iterations.

Table 3.4. An Alternate Flow Allocation for the Five Commodity Network

(f_a^k is the flow on link a due to commodity k)

(Numbers inside brackets denote expected values)

Commodity k	f_1^k	f_2^k	f_3^k	f_4^k	f_5^k	f_6^k	f_7^k	f_8^k	f_9^k	v^k
1	1.824 (1.83)	1.821 (1.83)	2.726×10^{-4} (0)	-1.26×10^{-3} (0)	-1.53×10^{-3} (0)	-1.53×10^{-3} (0)	-1.54×10^{-3} (0)	1.822 (1.83)	-1.81×10^{-3} (0)	1.829 (1.83)
2	2.334 (2.33)	2.333 (2.33)	2.335 (2.33)	-1.54×10^{-3} (0)	-3.83×10^{-4} (0)	-3.87×10^{-4} (0)	-1.53×10^{-3} (0)	-1.48×10^{-3} (0)	-3.89×10^{-4} (0)	2.338 (2.33)
3	0.332 (0.33)	1.572 (1.57)	1.575 (1.57)	-1.53×10^{-3} (0)	1.220 (1.24)	0.336 (0.33)	-1.53×10^{-3} (0)	-1.54×10^{-3} (0)	6.006 (6.0)	7.584 (7.57)
4	-1.878×10^{-3} (0)	0.165 (0.16)	2.384×10^{-4} (0)	-4.59×10^{-4} (0)	-1.53×10^{-3} (0)	-4.57×10^{-4} (0)	-4.53×10^{-4} (0)	0.164 (0.16)	-1.53×10^{-3} (0)	0.168 (0.16)
5	-4.826×10^{-4} (0)	-1.85×10^{-3} (0)	1.891 (1.81)	1.089 (1.1)	-4.82×10^{-4} (0)	1.93×10^{-2} (0)	6.006 (6.0)	1.689×10^{-2} (0)	-1.53×10^{-3} (0)	7.116 (7.11)
									Maximum throughput $= \sum_{k=1}^{5} v^k \rightarrow$	19.035 (19.0)

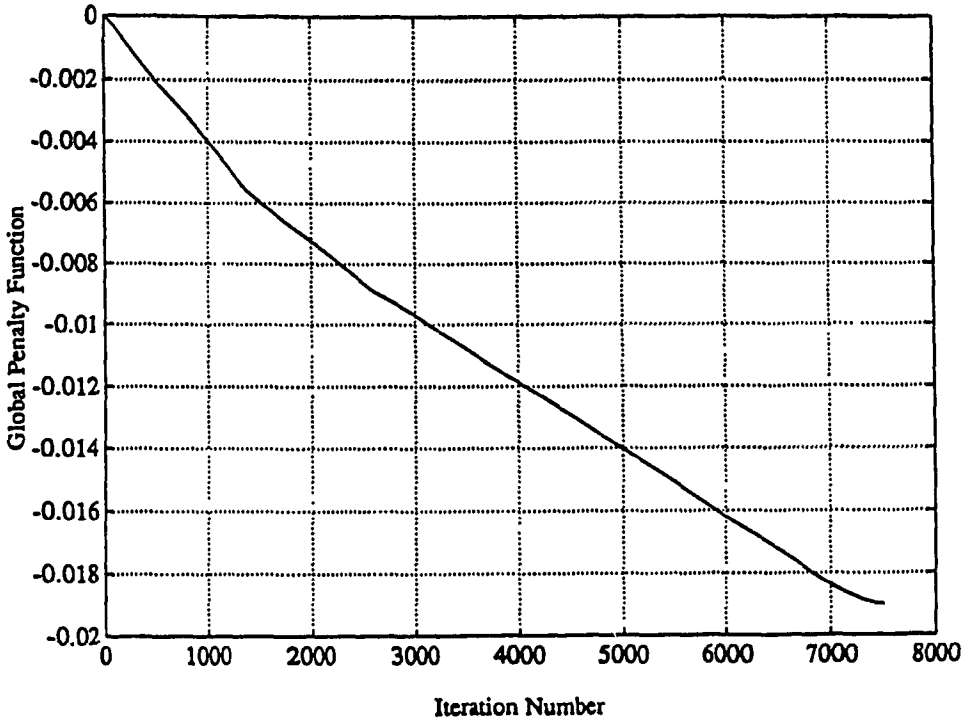


Figure 3.12. Scalar Function of the Five Commodity Network

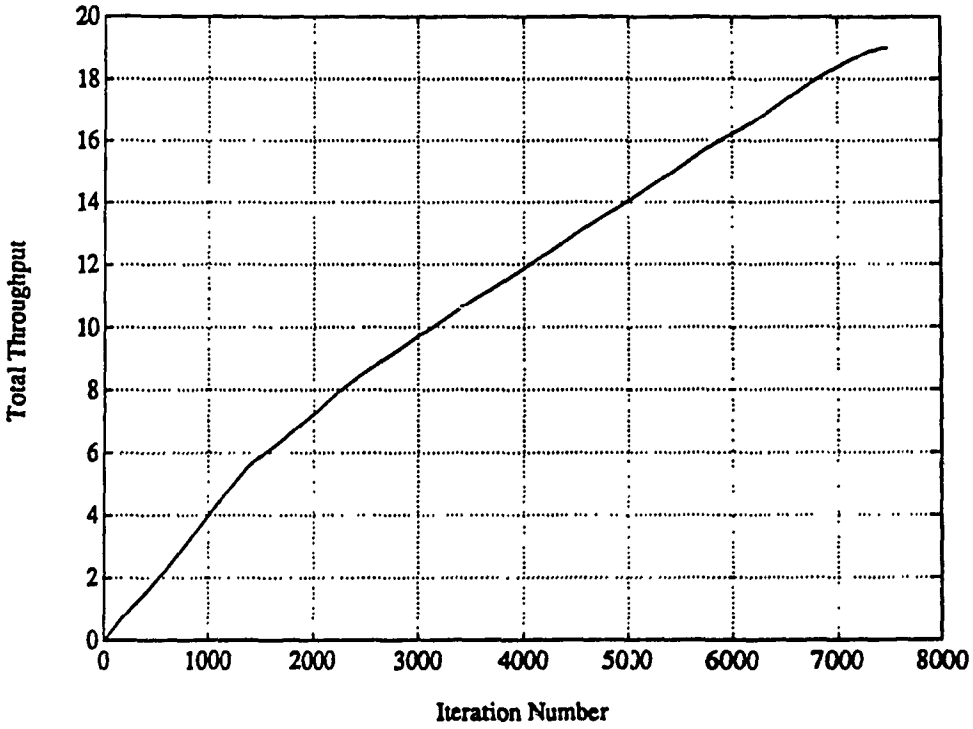


Figure 3.13. Total Throughput of the Five Commodity Network

3.4.2.2. A Ten Commodity Network

In this section we consider a larger network consisting of 14 links and 7 nodes as shown in figure 3.14. Correspondingly we define 10 commodities to be accommodated, where each commodity is defined by its source-destination pair as specified in table 3.5.

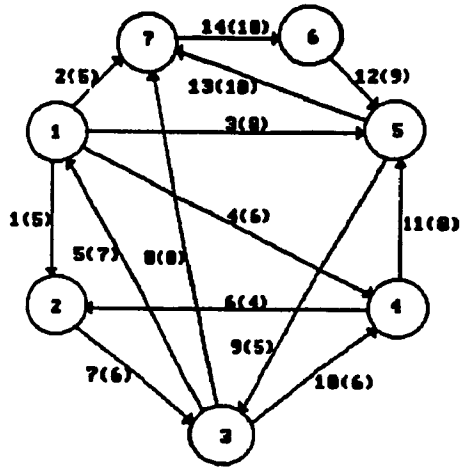


Figure 3.14. A Ten Commodity Network

(Numbers inside (outside) brackets denote link capacities (link indexes))

Table 3.5. The Ten Commodities for the Network of Figure 3.14

Commodity k	Source o^k	Destination d^k
1	1	2
2	1	4
3	1	2
4	2	4
5	2	5
6	3	2
7	3	4
8	7	3
9	4	3
10	4	6

The corresponding extended Linear Programming network that solves this Max-Flow problem requires 150 neurons, one for each variable, 174 nonlinear inequality constraint modules and 70 linear equality constraint modules. The equation of motion of a particular neuron is:

$$C_n \frac{dV_n}{dt} = -A_n - \sum_{j=1}^{174} B_{jn} g_j(\overrightarrow{B_j} \cdot \overrightarrow{V} - E_j) - \sum_{k=1}^{70} C_{kn} h_k(\overrightarrow{C_k} \cdot \overrightarrow{V} - F_k) \qquad (3.12)$$

$$\forall n \in \{1,2,..150\}$$

The dynamics of the corresponding extended Linear Programming network, with a zero initial state vector, are simulated as before with the following parameters :

$$C_n = 100nF \; ; \; \delta t = 10^{-7} \text{ sec} \; ; \; R_{nk} = 550K\Omega$$

$$R_j = 650K\Omega \; ; \; R_{no} = 1K\Omega \; ; \; \Delta V_{nk} = 10^{-4}V$$

The dynamics of the simulated system settle to equilibrium after 8064 iterations, giving a maximum network throughput of 40 units. The corresponding flow allocation is shown in table 3.6. The global penalty function and the total throughput are also obtained as shown in figures 3.15 and 3.16.

Table 3.6. Simulation Results of the Ten Commodity Network

(Zero Initial Conditions)

(Numbers inside brackets denote expected values)

commodity k	r_1^k	r_2^k	r_3^k	r_4^k	r_5^k	r_6^k	r_7^k	r_8^k	r_9^k	r_{10}^k	r_{11}^k	r_{12}^k	r_{13}^k	r_{14}^k	v^k
1	-4.818×10^{-4} (0)	-1.268×10^{-4} (0)	1.548 (1.54)	0.818 (0.81)	-1.6×10^{-3} (0)	-6.845×10^{-4} (0)	-7.477×10^{-4} (0)	-1.484×10^{-3} (0)	2.363 (2.35)	-1.484×10^{-3} (0)	0.817 (0.81)	-6.445×10^{-4} (0)	1.589×10^{-3} (0)	-8.69×10^{-4} (0)	2.348 (2.35)
2	-8.637×10^{-4} (0)	3.437 (3.435)	0.265 (0.265)	8.327×10^{-4} (0)	-6.747×10^{-4} (0)	-8.279×10^{-4} (0)	-1.483×10^{-2} (0)	-6.817×10^{-4} (0)	-9.434×10^{-4} (0)	-6.879×10^{-4} (0)	4.874×10^{-4} (0)	-1.543×10^{-3} (0)	0.265 (0.265)	3.71 (3.7)	3.704 (3.7)
3	3.584 (3.5)	-8.943×10^{-6} (0)	1.032×10^{-3} (0)	1.534 (1.53)	-7.782×10^{-4} (0)	1.533 (1.53)	-2.888×10^{-3} (0)	-8.571×10^{-4} (0)	-7.844×10^{-4} (0)	-7.854×10^{-4} (0)	-7.672×10^{-4} (0)	-4.498×10^{-4} (0)	5.647×10^{-4} (0)	-8.846×10^{-4} (0)	5.841 (5.83)
4	-3.634×10^{-3} (0)	-4.222×10^{-4} (0)	-4.322×10^{-4} (0)	1.278 (1.28)	1.288 (1.28)	-3.819×10^{-3} (0)	3.862 (3.85)	-3.778×10^{-4} (0)	-1.281×10^{-2} (0)	2.572 (2.57)	-2.286×10^{-4} (0)	-4.882×10^{-4} (0)	-5.597×10^{-5} (0)	-9.899×10^{-4} (0)	3.869 (3.85)
5	-3.857×10^{-4} (0)	-3.236×10^{-4} (0)	1.981 (1.98)	-2.826×10^{-4} (0)	1.877 (1.88)	-2.997×10^{-3} (0)	2.1521 (2.15)	-2.493×10^{-4} (0)	-1.783×10^{-3} (0)	1.872 (1.87)	1.876 (1.87)	-3.158×10^{-4} (0)	-2.788×10^{-4} (0)	-9.318×10^{-4} (0)	2.159 (2.15)
6	1.588 (1.5)	-1.92×10^{-4} (0)	-1.647×10^{-4} (0)	0.687 (0.68)	2.196 (2.19)	2.478 (2.47)	-3.889×10^{-3} (0)	-1.878×10^{-4} (0)	-1.498×10^{-3} (0)	1.791 (1.79)	-2.865×10^{-4} (0)	-6.234×10^{-4} (0)	1.287×10^{-4} (0)	-8.833×10^{-4} (0)	3.991 (3.97)
7	-1.886×10^{-2} (0)	1.544 (1.54)	0.343 (0.34)	4.844×10^{-5} (0)	1.987 (1.9)	-1.848×10^{-3} (0)	-1.967×10^{-3} (0)	1.558 (1.56)	-1.446×10^{-3} (0)	-1.183×10^{-4} (0)	0.937×10^{-4} (0)	-1.535×10^{-3} (0)	0.344 (0.34)	3.446 (3.46)	3.449 (3.46)
8	-8.885×10^{-4} (0)	-6.449×10^{-4} (0)	-8.812×10^{-5} (0)	2.885×10^{-4} (0)	-9.914×10^{-4} (0)	-8.559×10^{-4} (0)	-1.184×10^{-2} (0)	-1.545×10^{-3} (0)	-5.218×10^{-4} (0)	-1.819×10^{-3} (0)	-1.894×10^{-4} (0)	-4.472×10^{-4} (0)	-4.614×10^{-4} (0)	-6.281×10^{-4} (0)	2.812×10^{-3} (0)
9	-8.292×10^{-4} (0)	-2.617×10^{-4} (0)	-2.768×10^{-4} (0)	-2.172×10^{-4} (0)	-1.214×10^{-4} (0)	-5.987×10^{-4} (0)	-8.884×10^{-4} (0)	-1.616×10^{-3} (0)	2.647 (2.65)	1.551×10^{-3} (0)	2.658 (2.65)	-6.484×10^{-4} (0)	1.422×10^{-3} (0)	-8.531×10^{-4} (0)	2.652 (2.65)
10	-9.161×10^{-4} (0)	-4.528×10^{-5} (0)	-5.197×10^{-5} (0)	-5.636×10^{-4} (0)	-6.394×10^{-4} (0)	-8.364×10^{-4} (0)	-1.463×10^{-3} (0)	-7.862×10^{-4} (0)	-9.179×10^{-4} (0)	-7.854×10^{-4} (0)	2.841 (2.84)	-1.537×10^{-3} (0)	2.848 (2.84)	2.839 (2.84)	2.842 (2.84)
														Maximum throughput $= \sum_{k=1}^{10} v^k \longrightarrow$	30.897 (30)

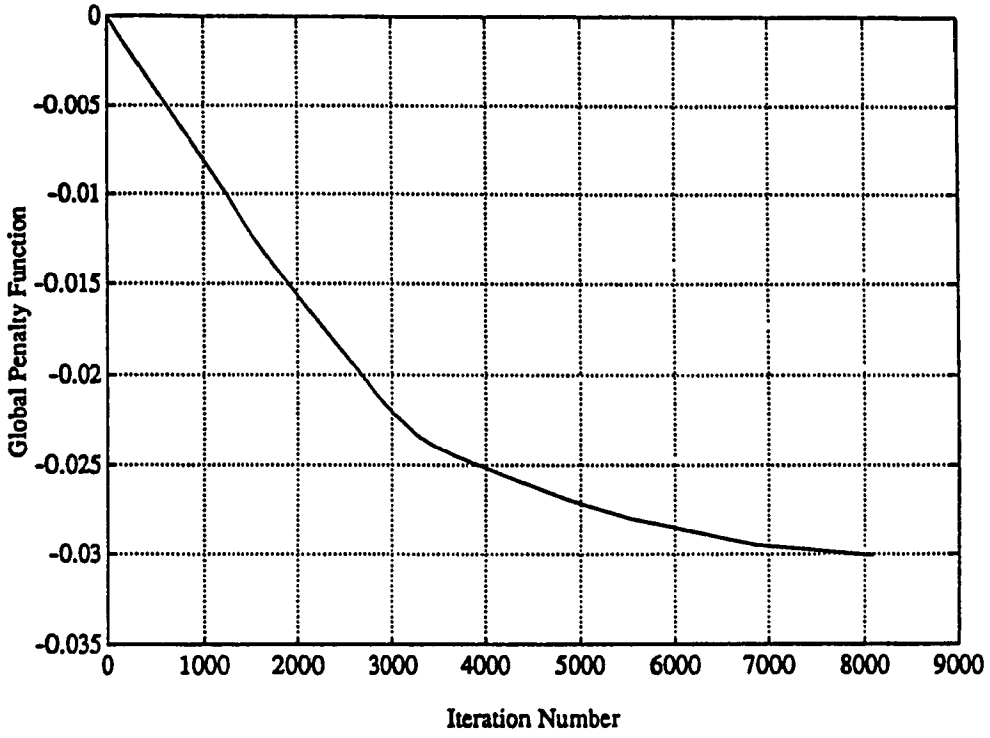


Figure 3.15. Scalar Function of the Ten Commodity Network

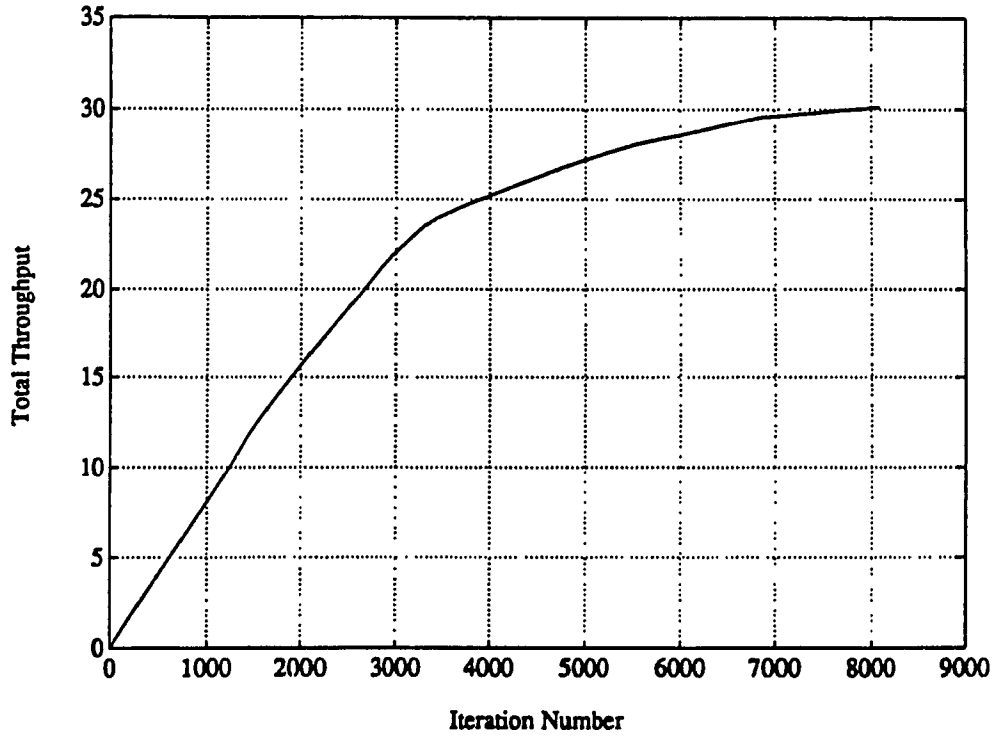


Figure 3.16. Total Throughput of the Ten Commodity Network

An alternate flow allocation, obtained with a different initial state vector and which gives the same throughput is illustrated in table 3.7.

Table 3.7. An Alternate Flow Allocation for the Ten Commodity Network

Commodity k	ρ_1^k	ρ_2^k	ρ_3^k	ρ_4^k	ρ_5^k	ρ_6^k	ρ_7^k	ρ_8^k	ρ_9^k	ρ_{10}^k	ρ_{11}^k	ρ_{12}^k	ρ_{13}^k	ρ_{14}^k	v^k
1	-6.648×10^{-4} (0)	1.454×10^{-3} (0)	1.889 (1.88)	0.702 (0.70)	-1.484×10^{-2} (0)	-6.737×10^{-4} (0)	-8.274×10^{-4} (0)	-1.848×10^{-2} (0)	2.648 (2.64)	-1.482×10^{-3} (0)	0.701 (0.70)	-6.979×10^{-4} (0)	-1.233×10^{-4} (0)	-8.351×10^{-4} (0)	2.648 (2.64)
2	-8.983×10^{-4} (0)	2.582 (2.58)	0.977 (0.98)	0.204 (0.2)	-6.718×10^{-4} (0)	-8.678×10^{-4} (0)	-1.472×10^{-2} (0)	-6.611×10^{-4} (0)	-2.261×10^{-4} (0)	-8.888×10^{-4} (0)	0.204 (0.2)	-1.888×10^{-2} (0)	1.189 (1.18)	3.761 (3.76)	3.764 (3.76)
3	3.168 (3.15)	-2.397×10^{-7} (0)	-1.889×10^{-2} (0)	1.889 (1.81)	-8.112×10^{-4} (0)	1.818 (1.81)	-2.288×10^{-2} (0)	-8.899×10^{-4} (0)	-6.488×10^{-4} (0)	-8.187×10^{-4} (0)	9.841×10^{-1} (0)	-6.424×10^{-4} (0)	6.289×10^{-4} (0)	-8.988×10^{-4} (0)	4.982 (4.96)
4	-3.868×10^{-2} (0)	-4.838×10^{-4} (0)	-3.58×10^{-4} (0)	1.838 (1.82)	1.826 (1.82)	-3.851×10^{-2} (0)	3.147 (3.14)	-4.879×10^{-4} (0)	-1.128×10^{-2} (0)	2.128 (2.12)	-3.315×10^{-4} (0)	-4.715×10^{-6} (0)	-6.927×10^{-6} (0)	-8.975×10^{-4} (0)	3.155 (3.14)
5	-3.867×10^{-2} (0)	-2.899×10^{-4} (0)	1.476 (1.47)	1.843×10^{-1} (0.1)	1.576 (1.57)	-3.894×10^{-2} (0)	2.866 (2.84)	-2.779×10^{-4} (0)	-1.448×10^{-2} (0)	1.289 (1.29)	1.396 (1.39)	-3.161×10^{-4} (0)	-2.788×10^{-4} (0)	-8.325×10^{-4} (0)	2.874 (2.86)
6	1.862 (1.85)	-1.736×10^{-4} (0)	-1.193×10^{-4} (0)	0.192 (0.19)	2.844 (2.84)	2.194 (2.19)	-3.854×10^{-2} (0)	-1.651×10^{-4} (0)	-1.282×10^{-2} (0)	2.882 (2.8)	-1.427×10^{-4} (0)	-6.125×10^{-4} (0)	-4.717×10^{-6} (0)	-8.895×10^{-4} (0)	4.858 (4.84)
7	-1.898×10^{-2} (0)	1.168 (1.16)	0.183 (0.18)	1.381×10^{-2} (0.01)	1.356 (1.35)	-1.879×10^{-2} (0)	-1.981×10^{-2} (0)	2.155 (2.15)	-1.418×10^{-3} (0)	0.336 (0.33)	0.361 (0.36)	-1.531×10^{-2} (0)	8.534 (8.54)	3.86 (3.88)	3.853 (3.85)
8	-8.729×10^{-4} (0)	-8.821×10^{-4} (0)	-8.785×10^{-4} (0)	2.784×10^{-4} (0)	-8.884×10^{-4} (0)	-8.884×10^{-4} (0)	-1.883×10^{-3} (0)	-1.854×10^{-2} (0)	-8.878×10^{-4} (0)	-8.764×10^{-4} (0)	-6.384×10^{-4} (0)	-4.187×10^{-4} (0)	-4.968×10^{-4} (0)	-6.129×10^{-4} (0)	2.883×10^{-2} (0)
9	-8.367×10^{-4} (0)	-2.159×10^{-4} (0)	-2.368×10^{-4} (0)	-2.699×10^{-4} (0)	-1.278×10^{-4} (0)	-6.887×10^{-4} (0)	-8.991×10^{-4} (0)	-1.479×10^{-3} (0)	2.345 (2.34)	1.511×10^{-3} (0)	2.348 (2.34)	-6.998×10^{-4} (0)	1.376×10^{-2} (0)	-8.763×10^{-4} (0)	2.358 (2.34)
10	-8.431×10^{-4} (0)	-1.644×10^{-4} (0)	-3.729×10^{-4} (0)	-3.658×10^{-4} (0)	-6.288×10^{-4} (0)	-8.619×10^{-4} (0)	-1.621×10^{-4} (0)	-6.963×10^{-4} (0)	-7.705×10^{-4} (0)	-6.962×10^{-4} (0)	2.398 (2.39)	-1.648×10^{-2} (0)	2.396 (2.39)	2.396 (2.39)	2.398 (2.39)
														Maximum throughput = $\sum_{k=1}^{10} v^k \rightarrow$	38.894 (38)

Next is a number of remarks to follow regarding these simulations:

- First the computer simulation results obtained in sections 3.4.1 and 3.4.2 are based on the assumption that all the components of the extended Linear Programming circuit are ideal, which is not the case in most practical situations. However, when the LP network is implemented in hardware, the solution, which may not be hundred percent exact, will be obtained very fast. In many real-time control problems, requiring Linear Programming computations, the controller's decisions are to made very fast. In such cases it would be better to get a very good solution in a very short time, using a neural network hardware implementation approach, rather than exhausting considerable amount of time to get very accurate solutions using conventional iterative optimization methods.

- Second the simulation results for the single commodity and the two multicommodity cases, as obtained in sections 3.4.1 and 3.4.2, reveal that the computation time of the extended Linear Programming network does not increase very rapidly with problem size. In fact the solution to the single commodity network (7 variables, 14 inequality constraints and 4 equality constraints) was obtained in about 2970 iterations while the five-commodity (50 variables, 64 inequality constraints and 25 equality constraints) and the ten-commodity (150 variables, 174 inequality constraints and 71 equality constraints) problems were solved after 7440 and 8070 iterations respectively. This prominent characteristic of the neural network approach can be explained by the fact that as the problem size gets larger the number of required neurons increases and so does the amount of parallelism used during the distributed processing of neural computation [4].

- Third convergence of the extended Linear Programming network to the optimum solution is guaranteed thanks to the convex property of the solution space of all Linear Programming problems. Hence the local minimum reached by the gradient descent optimization process is also a global minimum regardless of the initial state.

- Fourth although a hardware implementation of the extended Linear Programming network can solve the maximum flow problem very fast and in real time, there are two challenges that will be hopefully overcome, especially with the recent advances in VLSI technology. The first challenge is the software problem of how to program the proper resistive inter-connections among all 'neurons'. It is clearly understood that the computational time invested in programming the inter-neurons' resistive connections should not shade the very fast response brought by the hardware implementation. The second challenge is the hardware problem of implementing large resistive connection matrices and large number of integrated op-amps on a single VLSI chip. During the past few years there have been many attempts to solve these problems using analog VLSI techniques. Although the results obtained so far are very encouraging, there is a lot of work to be done before the microfibrication of thousands of resistive connections and integrated op-amps on a single silicon wafer becomes plausible. Nevertheless since the extended Linear Programming circuit has a well defined structure (figure 2.6) then it might be possible to design a special purpose Linear Programming network, where the general topology of the neural network is stored in a special program, with the user specifying the parameters required to solve his particular problem (example: number of elementary neurons, linear equality constraint modules and nonlinear inequality constraint modules, conductance values of the resistive connection matrices, etc..) [6].

- Finally for small size Linear Programming problems, special purpose softwares such as SPICE could be used to simulate in a more realistic way the dynamics of the extended Linear Programming network.

CHAPTER 4

A NEURAL NETWORK SOLUTION TO THE SHORTEST PATH PROBLEM

4.1.Introduction

In this chapter, another important network routing problem is considered, where the goal is to find the shortest path from some source node (s) to a destination node (d) through a connecting network. Once again we define a network as a directed graph $G = (\overline{N}, \overline{A})$, with n nodes and m arcs. Corresponding to each arc (i,j) there is a nonnegative number C_{ij}, called length, distance or transit time from node i to node j. Defining a directed path P^{sd} as an ordered sequence of nodes connecting s to d:

$$P^{sd} = (s, i, j, k, \ldots, r, d) \tag{4.1}$$

Then the length of this path will be $L^{sd} = C_{si} + C_{ij} + C_{jk} + \ldots + C_{rd}$. The problem is therefore to find the path which has the minimum length, L^{sd}.

The applicability of the shortest path (SP) problem to telecommunication and transportation networks is diverse. For instance, if C_{ij} is the cost of sending data on arc (i,j) then the shortest path from s to d is the optimum route for data transmission. Alternatively if C_{ij} is set to -ln $[p_{ij}]$, where p_{ij} is the probability that arc (i,j) is usable, then the shortest path from s to d is the most reliable path for data transmission, provided that the usability of a given arc is independent from that of the remaining arcs. In addition there is a large class of network optimization problems whose solution requires solving SP problems as subproblems [22].

The SP problem can be solved using many well known graph theory algorithms. The

two most commonly used are Dijkstra's and Ford-Fulkerson's algorithms. Dijkstra's (or forward search) algorithm finds the shortest path from a given source node to all other nodes. Ford-Fulkerson (or backward search) algorithm finds the shortest path to a given destination node from all other nodes. Discussion of these graph theoretic algorithms however is beyond the scope of this thesis.

The use of neural networks to find the shortest path between a given source-destination pair was initiated by Rauch and Winarske (R&W) [23]. They proposed a neural network architecture arranged in a two dimensional array of size nxM; where M is the number of nodes forming the path. The output V_{xi} of the neuron at location (x,i) is defined as follows:

$$V_{xi} = \begin{cases} 1 & \text{if node x is the } i^{th} \text{ node to be visited in the path} \\ 0 & \text{otherwise} \end{cases} \qquad (4.2)$$

One obvious limitation of the above representation is that it requires a prior knowledge of the number of nodes forming the shortest path. This number however is unknown. R&W fixed M to the minimum number of nodes forming a path. It is obvious then that R&W algorithm finds the shortest path only among those M node paths. Therefore R&W algorithm is not adequate to solve the SP problem since it gives only a suboptimal solution; after all the shortest path may consist of more than M nodes.

To correct for this error, Zhang and Thomopoulos (Z&T) [24] extended M to the total number of nodes in the network, which is also the maximum number of nodes the SP may consists of. They assigned zero-cost self-loops connecting each node to itself and very large costs to non-existing links. In this way a self-loop can be included in the SP without increasing the path cost. To illustrate Z&T approach, consider the six node network shown in figure 4.1 where each link is labeled by its corresponding cost. The cost matrix $C^{ad} = [C_{ij}]$ associated with this network is given in table 4.1, where L denotes some large positive

number.

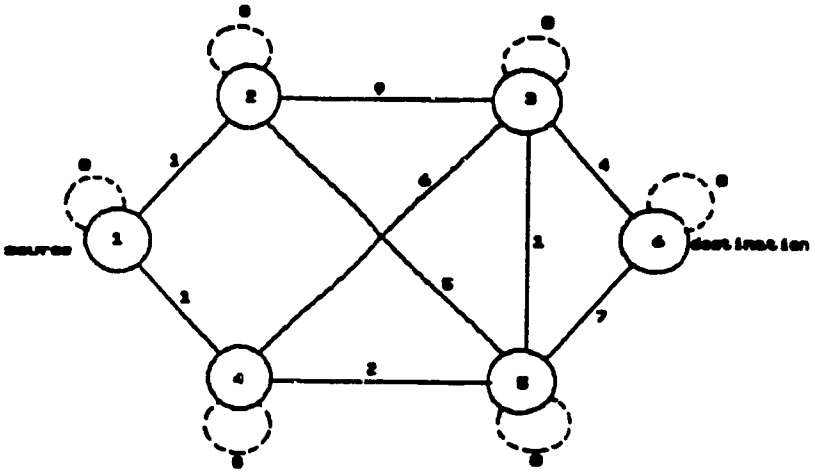


Figure 4.1. A Six Node Network With Self-loops

(Links are labeled by their corresponding costs)

Table 4.1. Cost Matrix of the Network of Figure 4.1

(L is a large positive number)

	1	2	3	4	5	6
1	0	1	L	1	L	L
2	1	0	9	L	5	L
3	L	2	0	6	1	4
4	1	L	6	0	2	L
5	L	5	1	2	0	7
6	L	L	4	L	7	0

The shortest path between nodes 1 and 6 is $P^{16} = (1,4,5,3,6)$ and it has many neural representations as shown in figure 4.2. Each neural representation in figure 4.2 corresponds to a particular self loop, which is introduced to compensate to the fact that node 2 is not in the SP solution.

	1	2	3	4	5	6
1	1	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	1	1	0
4	0	1	0	0	0	0
5	0	0	1	0	0	0
6	0	0	0	0	0	1

a) 1—4—5—3—3—6

	1	2	3	4	5	6
1	1	1	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	1	0
4	0	0	1	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

b) 1—1—4—5—3—6

	1	2	3	4	5	6
1	1	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	1	0
4	0	1	1	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

c) 1—4—4—5—3—6

Figure 4.2. Some Neural Representations of the Shortest Path P^{16}

($V_{xi} = 1$ means that node x is the i^{th} node to be visited)

Using Hopfield and Tank discrete neural model, Zhang and Thomopoulos proposed the following energy function to be minimized:

$$E_m = \frac{A}{2}\sum_{k=1}^{n-1}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}C_{ij}V_{jk+1} + \frac{B}{2}\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}V_{jk} + \frac{C}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ij} - n\right)^2 \qquad (4.3)$$

In the above the A term represents the total cost of the path, while the B and C terms are constraints introduced to force the neural network state to converge to a valid path. The B term is minimized if each column contains at most a single 1, which corresponds to at most one node visited at a time. The C term ensures that there will be exactly n 1's in the final solution. When combined together, the B and C term ensure that each column will have exactly a single 1. The connection weights and the biases, corresponding to the energy function (4.3) are given below:

$$T_{xi,yj} = -AC_{xy}(\delta_{j,i+1} + \delta_{j,i-1}) - B\delta_{ij}(1-\delta_{xy}) - C \qquad (4.4)$$

$$I_{xi} = Cn \qquad (4.5)$$

where δ is the Kronecker delta defined by:

$$\delta_{ab} = \begin{cases} 1 & \text{if a=b} \\ 0 & \text{otherwise} \end{cases} \qquad (4.6)$$

For a given source destination pair, Z&T fix the state of all neurons located at the first and last column, while allowing the remaining neurons to evolve so as to minimize the energy function (4.3). For example in figure 4.2, the state of the neurons belonging to column 1 is fixed, corresponding to a single one at row 1 since it is known that the source node (1) is the first to be visited. Similarly the state of the neurons located in the 6th column is fixed, with a single one at row 6 since destination node 6 is the last one to be visited.

Although the neural network formulation proposed by Z&T is more adequate than that of R&W, it still has limitations. Since each neuron belonging to the first and last column

has a fixed output voltage, the neural network is designed to find the shortest path between only a given source destination pair. To find the SP between another pair, the neural network configuration has to be changed. More importantly, since the A term in the energy function (4.3) is quadratic, the connection strengths among neurons, as given in (4.4), depend on link costs. In practice the link costs in a communication network are usually time varying since they normally depend on the flow the links are carrying. Subsequently if Z&T neural network is to be used for traffic routing as suggested in [24] then the resistances of the synaptic connections are to be changed continuously in order to adapt to changes in link costs. This makes the circuit implementation of Z&T neural network not suitable for finding shortest paths under time varying link costs. In addition, in terms of neural representation, the presence of self-loops in the final solution is not desirable. Referring back to figure 4.2 it can be seen that a path which does not pass through all the nodes of the network has many neural representations, each defined by a particular set of self loops.

In what follows a new approach to the shortest path problem, based on Hopfield and Tank neural network is proposed. This solution is more appropriate for the optimal routing application to be studied in the following chapter.

4.2.Problem Formulation

To formulate the SP problem in terms of Hopfield and Tank neural network, a suitable representation scheme is to be found so that the shortest path can be decoded from the final state of the neural network. The proposed model is organized in an (nxn) matrix, with all diagonal elements removed, since not needed. Therefore the computational network requires n(n-1) neurons, which is less than the number of neurons required by Z&T model. Each

neuron is described by double indices (x,i), where the row subscript x and the column subscript i denote node numbers, and a neuron at location (x,i) is characterized by its output V_{xi}, defined as:

$$V_{xi} = \begin{cases} 1 & \text{if the arc from node x to node i is in the shortest path} \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

We also define ρ_{xi} as:

$$\rho_{xi} = \begin{cases} 1 & \text{if the arc from node x to node i does not exist} \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

In addition the cost of an arc from node x to node i will be denoted by C_{xi}, a finite real positive number. For non-existing arcs this cost will be assumed to be zero, but non-existing arcs will be eliminated from the solution by being invalid.

4.3 The SP Energy Function

In order to solve the SP problem, using Hopfield and Tank neural model, we first have to define an energy function whose minimization process drives the neural network into its lowest energy state. This final stable state shall correspond to the shortest path solution. The energy function must favor states that correspond to valid paths between the specified origin-destination pair. Among these valid paths it must also favor the one with the shortest length.

An energy function that satisfies such requirements is given by:

$$E_m=\frac{\mu_1}{2}\sum_{x=1}^{n}\sum_{\substack{i=1\\ i\neq x\\ (x,i)\neq(d,s)}}^{n} C_{xi}\cdot V_{xi}+\frac{\mu_2}{2}\sum_{x=1}^{n}\sum_{\substack{i=1\\ i\neq x\\ (x,i)\neq(d,s)}}^{n} \rho_{xi}\cdot V_{xi}+\frac{\mu_3}{2}\sum_{x=1}^{n}\left\{\sum_{\substack{i=1\\ i\neq x}}^{n} V_{xi}-\sum_{\substack{i=1\\ i\neq x}}^{n} V_{ix}\right\}^2$$

$$+\frac{\mu_4}{2}\sum_{i=1}^{n}\sum_{\substack{x=1\\ x\neq i}}^{n} V_{xi}\cdot(1-V_{xi})+\frac{\mu_5}{2}(1-V_{ds}) \qquad (4.9)$$

In (4.9) the μ_1 term minimizes the total cost of a path by taking into account the cost of existing links. The μ_2 term prevents nonexistent links from being included in the chosen path. The μ_3 term is zero if for every node in the solution, the number of ingoing arcs equals the number of outgoing arcs. This makes sure that if a node has been entered it will also be exited by a path. The μ_4 term is a compensating term that pushes the state of the neural network to converge to one of the 2^{n^2-n} corners of the hypercube, defined by $V_{xi} \in \{0,1\}$. The μ_5 term is zero when the neuron at location (d,s) has a unity output . Although the link from d to s is not part of the solution, it is introduced to enforce the construction of a path, which must originate at s and terminate at d. This makes sure that the final solution contains the arc from d to s and therefore both source and destination nodes will be in the solution. Thus the final solution will always be a loop, with nodes s and d included. This loop consists of two parts; a directed path from s to d and an arc from d to s.

If there are no zero length loops in the network, then the μ_3 term will ensure that there will be at most a single 1 at each row and at each column. This will also guarantee that to each path P^{sd} there corresponds one and only one neural output. As a result the decoding of the selected path from the final stable state of the neural network will become easier, as there will be a one to one relation between the set of feasible neural outputs and the set of paths. To illustrate this consider the example depicted in figure 4.3. For the network in figure 4.3.a, let us assume that the shortest path is $P^{sd} = (1,2,5,6)$. The corresponding arcs which

are to be included in the final solution are shown in figure 4.3.b. These arcs form a loop, thereafter referred as a primary loop. Correspondingly, the neural output will be represented as shown in figure 4.3.c, where each node included in the shortest path has a single 1 in its corresponding row\column. Note that since nodes 3 and 4 are not part of the shortest path, they have all zero entries in their corresponding rows / columns.

Now let us suppose that node 2 has two outgoing arcs in the final solution, corresponding to two ones in row 2. Then besides the primary loop, forming the shortest path, secondary loops will be forced into the solution due to the μ_3 and μ_5 terms. Examples of secondary loops are shown in figures 4.3.d and 4.3.e. As long as the length of the secondary loop is not null, the energy function, through the μ_1 cost term, will prevent this secondary loop from being part of the final solution as its inclusion results in an extra cost to be added to the primary loop cost. Therefore path $P^{sd} = (1,2,5,6)$ will have one and only one corresponding neural representation, the one shown in figure 4.3.c.

4.4. The Connection Matrix and the Biases

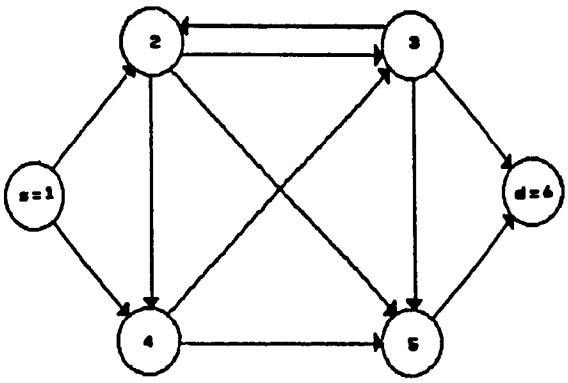
The connection matrix and the biases of the neural network can be found by comparing (2.2.a) with (2.11), which are now written as :

$$c_{xi} \cdot \frac{dU_{xi}}{dt} = -\frac{U_{xi}}{R_{xi}} + \sum_{y=1}^{n} \sum_{\substack{j=1 \\ j \neq y}}^{n} T_{xi,yj} \cdot V_{yj} + I_{xi} \tag{4.10}$$

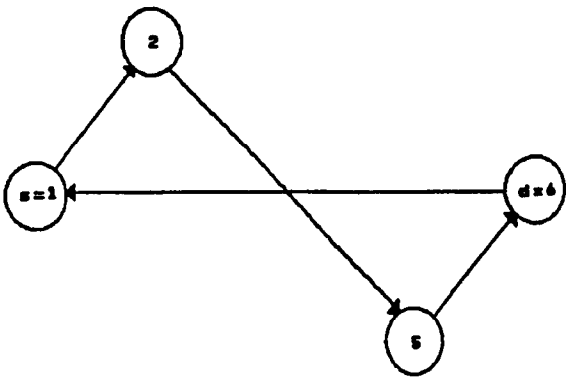
$$c_{xi} \cdot \frac{dU_{xi}}{dt} = -\frac{U_{xi}}{R_{xi}} - \frac{\partial E_m}{\partial V_{xi}} \tag{4.11}$$

$$\forall (x,i) \in \overline{N} \times \overline{N} / x \neq i$$

By substituting (4.9) in (4.11) , the equation of motion of the neural network is readily obtained:



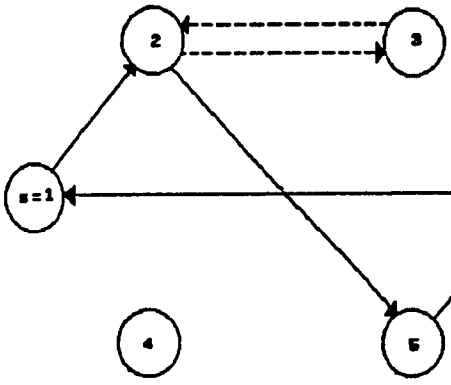
4.3.a. The Original Network



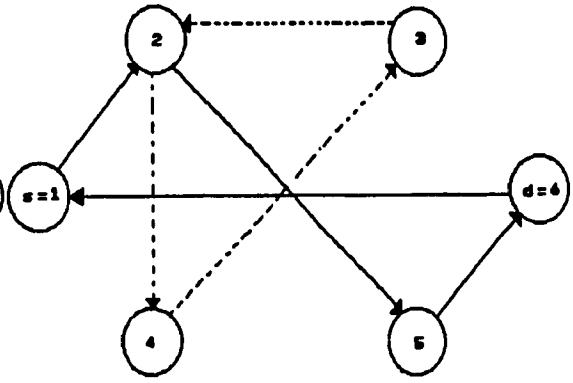
4.3.b. The Primary Loop Forming The SP Solution

	1	2	3	4	5	6
1		1	0	0	0	0
2	0		0	0	1	0
3	0	0		0	0	0
4	0	0	0		0	0
5	0	0	0	0		1
6	1	0	0	0	0	

4.3.c. The Neural Output Representation



4.3.d. A secondary Loop (2,3,2)



4.3.e. Another Secondary Loop (2,4,3,2)

: A primry loop arc

: A secondary loop arc

Figure 4.3. An Illustrating example

$$c_{xi}\cdot\frac{dU_{xi}}{dt}=-\frac{U_{xi}}{R_{xi}}-\frac{\mu_1}{2}C_{xi}(1-\delta_{xd}\cdot\delta_{is})-\frac{\mu_2}{2}\rho_{xi}(1-\delta_{xd}\cdot\delta_{is}) \tag{4.12}$$

$$-\mu_3\sum_{\substack{y=1\\y\neq x}}^{n}(V_{xy}-V_{yi})+\mu_3\sum_{\substack{y=1\\y\neq i}}^{n}(V_{iy}-V_{yi})-\frac{\mu_4}{2}(1-2V_{xi})$$

$$+\frac{\mu_5}{2}\delta_{xd}\cdot\delta_{is}$$

$$\forall(x,i)\in\overline{N}\times\overline{N}/x\neq i$$

where δ is as defined in (4.6).

By comparing the corresponding coefficients in (4.10) and (4.12), the connection strengths and the biases are derived as:

$$T_{xi,yj}=\mu_4\delta_{xy}\delta_{ij}-\mu_3\delta_{xy}-\mu_3\delta_{ij}+\mu_3\delta_{jx}+\mu_3\delta_{iy} \tag{4.13}$$

$$I_{xi}=-\frac{\mu_1}{2}C_{xi}(1-\delta_{xd}\cdot\delta_{is})-\frac{\mu_2}{2}\rho_{xi}(1-\delta_{xd}\cdot\delta_{is})-\frac{\mu_4}{2}+\frac{\mu_5}{2}\delta_{xd}\delta_{is}$$

$$=\begin{cases}\dfrac{\mu_5}{2}-\dfrac{\mu_4}{2} & \text{if } (x,i)=(d,s)\\[2ex] -\dfrac{\mu_1}{2}C_{xi}-\dfrac{\mu_2}{2}\rho_{xi}-\dfrac{\mu_4}{2} & \text{otherwise}\end{cases} \tag{4.14}$$

$$\forall(x\neq i),\forall(y\neq j)$$

The first term in (4.13) represents excitatory self-feedbacks, the second and third terms represent local inhibitory connections among neurons in the same row and in the same column respectively. The last two terms represent excitatory cross connections among neurons.

Unlike the previously proposed neural networks, solving the SP (23&24) and TSP (2) problems, the proposed model maps the data (here defined by link costs and node connectivity information) into the biases rather than into the neural interconnections. This is

due to the fact that the data terms are associated with linear rather than quadratic expressions in the energy function (4.9). One advantage of the proposed representation scheme is a flexibility reflected by the fact that the link costs C_{xi}'s and the network topology information, embedded in the ρ_{xi}'s terms, can be changed through the biases. This will make the neural network very attractive to operate in real time and to adapt to changes in network topology and link costs. Another advantage is that the inter-connection strengths do not depend on a particular source or destination. Hence the neural network can find the shortest path between any given two nodes by properly choosing the input biases as given in equation (4.14).

4.5. The SP Simulation Results

Recall from (4.10) that the dynamics of the neural network are described by:

$$c_{xi} \cdot \frac{dU_{xi}}{dt} = \sum_{y=1}^{n} \sum_{\substack{j=1 \\ j \neq y}}^{n} T_{xi,yj} \cdot V_{yj} - \frac{U_{xi}}{R_{xi}} + I_{xi} \qquad (4.15.a)$$

$$V_{xi} = g_{xi}^{*}(U_{xi}) = \frac{1}{1 - e^{-\lambda_{xi} \cdot U_{xi}}} \qquad (4.15.b)$$

For simplicity it will be assumed that $c_{xi} = C$, $R_{xi} = R$, $\lambda_{xi} = \lambda$, and $g_{xi}^{*} = g^{*}$, all independent of the subscript (x,i). Then dividing by C and redefining $\frac{T_{xi,yj}}{C}$ and $\frac{I_{xi}}{C}$ as $T_{xi,yj}$ and I_{xi} respectively, we get:

$$\frac{dU_{xi}}{\mathrm{dt}} = \sum_{y=1}^{n} \sum_{\substack{j=1 \\ j \neq y}}^{n} T_{xi,yj} V_{yj} - \frac{U_{xi}}{\tau} + I_{xi} \qquad (4.16.a)$$

$$\tau = RC \qquad (4.16.b)$$

$$V_{xi} = g^{*}(U_{xi}) = \frac{1}{1 + e^{-\lambda \cdot U_{xi}}} \qquad (4.16.c)$$

With the above simplifications, the time it takes for the neural network to reach steady state is in arbitrary units. Therefore the time constant τ of each neuron is arbitrary set to 1.0. In addition the equation of motion describing the dynamics of Hopfield and Tank model now becomes:

$$\frac{dU_{xi}}{dt} = h(U_{xi}) = -\frac{U_{xi}}{\tau} - \frac{\mu_1}{2} C_{xi}(1-\delta_{xi}\delta_{ix}) - \frac{\mu_2}{2} \rho_{xi}(1-\delta_{xi}\delta_{ix}) - \mu_3 \sum_{\substack{y=1 \\ y \neq x}}^{n} (V_{xy} - V_{yx})$$

$$+ \mu_3 \sum_{\substack{y=1 \\ y \neq i}}^{n} (V_{iy} - V_{yi}) - \frac{\mu_4}{2}(1-2V_{xi}) + \frac{\mu_5}{2}\delta_{xi}\delta_{ix} \qquad (4.17.a)$$

$$V_{xi} = \frac{1}{1+e^{-\lambda \cdot U_{xi}}} \qquad (4.17.b)$$

Given the initial neurons' input voltages U_{xi}'s at time t=0, the time evolution of the state of the neural network is simulated by numerically solving (4.17). This corresponds to solving a system of n(n-1) nonlinear differential equations, where the variables are the neurons' output voltages V_{xi}'s. To achieve this the fourth order Range-Kutta method [25] has been used. Simulation has shown that this method always gives better results than the classical Euler method. Accordingly the updating of neuron voltages is simulated as follows:

$$K_1 = \delta t \cdot h(U_{xi})$$

$$K_2 = \delta t \cdot h\left(U_{xi} + \frac{K_1}{2}\right)$$

$$K_3 = \delta t \cdot h\left(U_{xi} + \frac{k_2}{2}\right) \qquad (4.18)$$

$$K_4 = \delta t \cdot h(U_{xi} + K_3)$$

$$U_{xi} = U_{xi} + \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$$

where δt is the incremental size of the updates.

Simulation has shown that a good value for δt is 10^{-5}. Reducing this value increases the simulation time without improving the results. Another important parameter in the simulation is the neuron initial input voltages U_{xi}'s. Since the neural network should have no a prior favor for a particular path, all the U_{xi}'s are set to zero. This corresponds to the initial state of the neural network being concentrated at the center of the hypercube, with all neuron output voltages V_{xi}'s set to 0.5V. The simulation is stopped when the system reaches a stable final state. This is supposed to occur when all neuron voltages do not change by more than a threshold value $\Delta V_{th} = 10^{-5}V$ from one update to the next.

As an example, the five node network shown in figure 4.4.a is considered. There are six feasible paths between the source node 1 and the destination node 5,whose corresponding neural output representations are shown in figure 4.4.b.

After a series of preliminary trials the heuristic coefficients, μ_i's are chosen as follows:

$$\mu_1 = 950$$

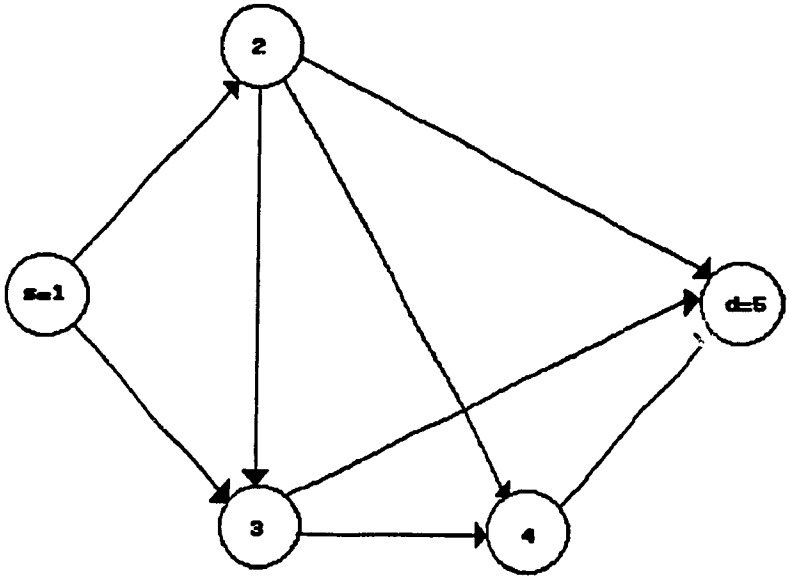
$$\mu_2 = 2500$$

$$\mu_3 = 1500$$

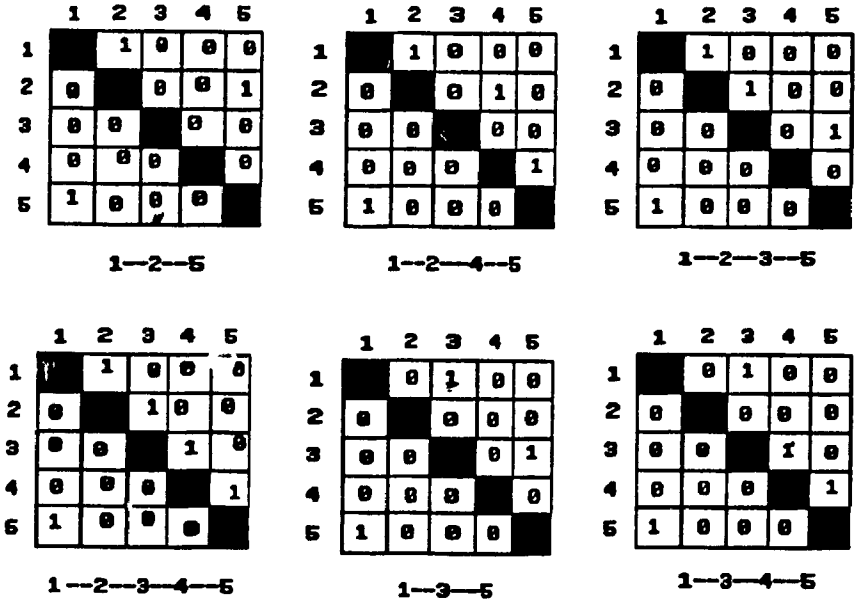
$$\mu_4 = 475$$

$$\mu_5 = 2500$$

Simulation has shown that there is a compromise between chosing a small or a large value for the neural transfer parameter λ.



4.4.a. A Network Example Used in the Simulation



4.4.b. Output Representation of the Feasible Paths

Figure 4.4. A simulation Example

While a large λ gives rise to a fast neural response for which the solution is not always a global minimum, a small λ yields a slower response which always guarantees an optimum solution. In our case we have chosen $\lambda = 1$. For the network of figure 4.4.a, the simulated neural algorithm is run 100 times using different randomly generated link costs, between 0 and 1. In all runs the simulated algorithm has converged to states corresponding to valid solutions within 3000 to 8000 iterations. In addition the global optimum was obtained in all runs. Typical examples are illustrated in figures 4.5 and 4.6. A second set of 100 runs is then performed on the eight node network shown in figure 4.7, using the following parameters:

$$\mu_1 = 650 \; ; \; \mu_2 = 2500 \; ; \; \mu_3 = 2000 \; ; \; \mu_4 = 85$$

$$\mu_5 = 2500 \; ; \; \lambda = 1 \; ; \Delta V_{ik} = 10^{-5} V ; \; \delta t = 10^{-5}$$

For this network there are 21 paths between the source node 1 and the destination node 8. All the runs converged to states representing valid and optimum solutions within 5000 to 16000 iterations. Typical results are also shown in figures 4.8 and 4.9.

Simulation has also shown that the number of iterations required in the SP neural computation increases as the second best solution gets closer to the optimum solution. This could be explained by the fact that it takes more time for the neural network to decide to which path to converge, taking into account that it should favor states corresponding to shortest paths.

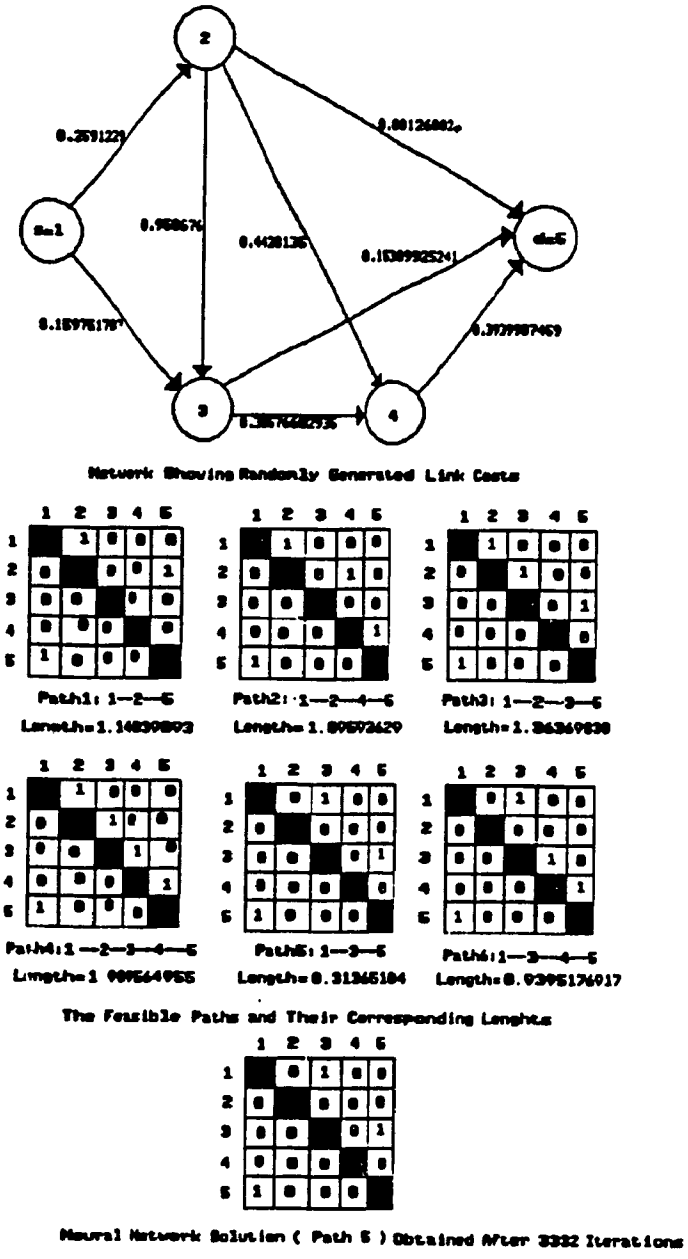


Figure 4.5. Typical results for the Five Node network

($V_{xi} = 1$ means that the arc from node x to node i is part of the path)

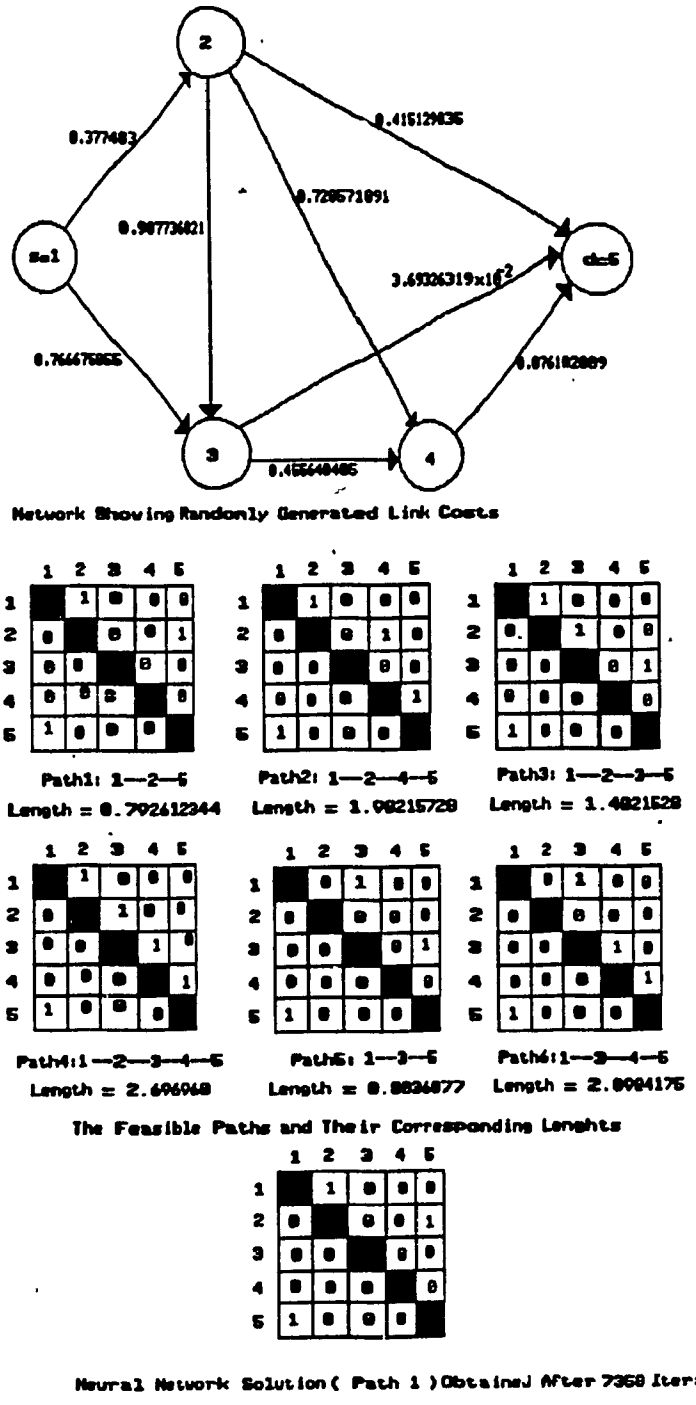


Figure 4.6. Another Simulation Example for the Five Node Network

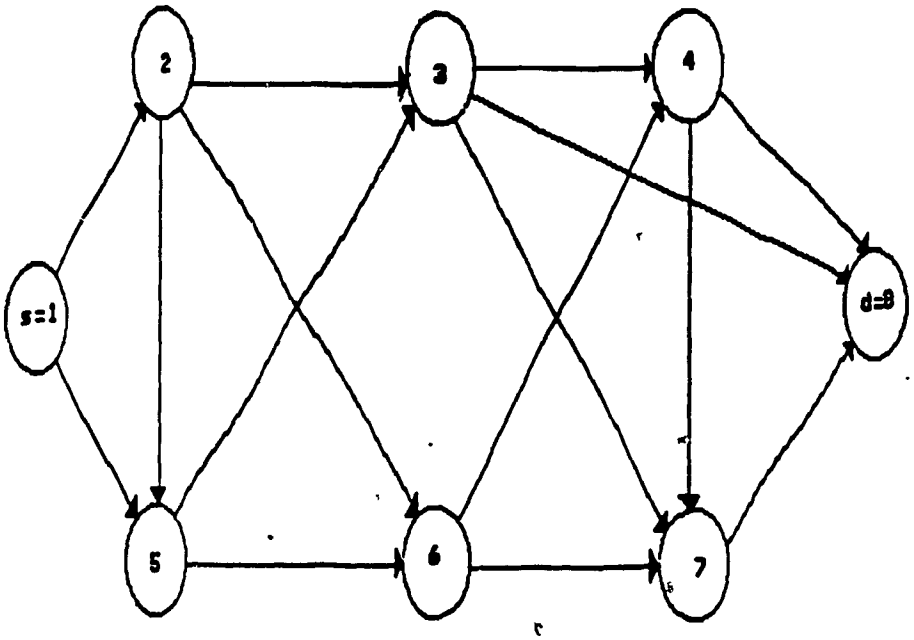


Figure 4.7. An Eight Node Network Used in the SP Simulation

Path	Node Sequence	Length
1	1-2-3-4-8	1.194396389
2	1-2-3-8	1.279696567
3	1-2-3-4-7-8	1.837431758
4	1-2-3-7-8	0.795678783
5	1-2-6-4-8	1.648637241
6	1-2-6-4-7-8	2.283672698
7	1-2-6-7-8	1.559066386
8	1-2-6-3-4-8	1.686396994
9	1-2-6-3-4-7-8	2.328432433
10	1-2-5-3-8	1.778697241
11	1-2-5-3-7-8	1.286579377
12	1-2-6-6-7-8	1.639048264
13	1-2-5-6-4-8	1.7286111118
14	1-2-5-6-4-7-8	2.263646655
15	1-6-6-7-8	2.191798677
16	1-6-6-4-8	2.273271434
17	1-6-3-4-8	2.238667387
18	1-6-3-8	2.323257565
19	1-6-3-7-8	1.839239781
20	1-6-3-4-7-8	2.881892756
21	1-6-6-4-7-8	2.916386883

b. Feasible Paths and Corresponding Lengths

	1	2	3	4	5	6	7	8
1		0.38888883	0	0	0.91384868	0	0	0
2	0		0.29825712	0	5.9687x10^{-2}	0.734762118	0	0
3	0	0		0.5944822	0	0	5.76186x10^{-2}	0.6886386
4	0	0	0		0	0	0.6847612	9.8736x10^{-2}
5	0	0	0.7216782	0		0.7651845	0	0
6	0	0	0	0.5863819	0		0.3888E389	0
7	0	0	0	0	0	0		0.13781812
8	0	0	0	0	0	0	0	

a. Randomly Generated Cost Matrix

(Number at location (x, i) denotes the length of the arc from node x to node i)

	1	2	3	4	5	6	7	8
1		1	0	0	0	0	0	0
2	0		1	0	0	0	0	0
3	0	0		0	0	0	1	0
4	0	0	0		0	0	0	0
5	0	0	0	0		0	0	0
6	0	0	0	0	0		0	0
7	0	0	0	0	0	0		1
8	1	0	0	0	0	0	0	

c. Neural Network Final State Corresponding to Path 4

(Solution was obtained After 8886 iterations)

Figure 4.8. Typical Results for the Eight Node Network

Path	Node Sequence	Length
1	1-2-3-4-8	1.984676437
2	1-2-3-8	1.588733877
3	1-2-3-4-7-8	2.636474288
4	1-2-3-7-8	2.722866819
5	1-2-6-4-8	2.173857943
6	1-2-6-4-7-8	2.884855714
7	1-2-6-7-8	2.312681823
8	1-2-6-3-4-8	2.488319618
9	1-2-6-3-4-7-8	3.339317381
10	1-2-6-3-8	2.284376258
11	1-2-6-3-7-8	3.425789992
12	1-2-6-6-7-8	3.165691554
13	1-2-6-6-4-8	3.826847674
14	1-2-6-6-4-7-8	3.767865445
15	1-6-6-7-8	2.338582615
16	1-6-6-4-8	2.190878736
17	1-5-3-4-8	1.781139671
18	1-5-3-8	1.377187311
19	1-5-3-7-8	2.598521853
20	1-5-3-4-7-8	2.512128442
21	1-5-6-4-7-8	2.929876586

b. Feasible Paths and Corresponding Lengths

	1	2	3	4	5	6	7	8
1		0.88469495	0	0	0.53164432	0	0	0
2	0		0.25595633	0	0.47832377	0.41536146	0	0
3	0	0		0.4476268	0	0	0.92478526	0.35636824
4	0	0	0		0	0	0.39878658	0.31278553
5	0	0	0.48927474	0		0.798837489	0	0
6	0	0	0	0.85661146	0		0.36592415	0
7	0	0	0	0	0	0		0.8529671
8	0	0	0	0	0	0	0	

a. Randomly Generated Cost Matrix

(Number at location (x, i) denotes the length of the arc from node x to node i)

	1	2	3	4	5	6	7	8
1		0	0	0	1	0	0	0
2	0		0	0	0	0	0	0
3	0	0		0	0	0	0	1
4	0	0	0		0	0	0	0
5	0	0	1	0		0	0	0
6	0	0	0	0	0		0	0
7	0	0	0	0	0	0		0
8	1	0	0	0	0	0	0	

c. Neural Network Final State Corresponding to Path 18
(Solution was obtained After 12151 iterations)

Figure 4.9. Another Simulation Example for the Eight Node Network

Referring back to figures 4.5 and 4.6, it can be seen that the number of iterations required to compute the SP solution has more than doubled in the second example due to the fact that the cost of path 5 has become very close to that of the optimum path 1, hence it takes more time before the neural network decides to which pattern it shall converge. Also it should be noted that the ability of the SP neural algorithm to separate between an optimum and a very good suboptimal solution is heavily enforced by the smoothness of the neural transfer function g^* (which corresponds to a low neural transfer parameter λ). This is explained by the fact that a large λ (such as 10 or 100) may not allow enough time for a neuron to optimize its performance, as the neural response time becomes very fast and hence less accurate.

These results show that the performance of Hopfield and Tank neural network gets better when solving the SP problem as opposed to the TSP. In our case the neural network does not require any random initialization and always converges to valid solutions which are also global optima. This is also supported by earlier simulation results reported by Protzel [26], where it was found that neural networks with linear cost functions generally perform much better than those with quadratic cost functions.

Finally convergence problems due to Hopfield approximation error could have been avoided by omitting the $\frac{U_{xi}}{R_{xi}}$ term in (4.15), in which case the dynamics of the new model, described by:

$$E_m = -\frac{1}{2}\sum_{x=1}^{n}\sum_{\substack{i=1\\ i\neq x}}^{n}\sum_{y=1}^{n}\sum_{\substack{j=1\\ j\neq y}}^{n} T_{xi,yj}V_{xi}V_{yj} - \sum_{x=1}^{n}\sum_{\substack{i=1\\ i\neq x}}^{n} I_{xi}V_{xi} \tag{4.19}$$

$$c_{xi}\cdot\frac{dU_{xi}}{dt} = \sum_{y=1}^{n}\sum_{\substack{j=1\\ j\neq y}}^{n} T_{xi,yj}V_{yj} + I_{xi} \tag{4.20.a}$$

$$= -\frac{\partial E_m}{\partial V_{xi}} \tag{4.20.b}$$

$$V_{xi} = g_{xi}^{\bullet}(U_{xi}) = \frac{1}{1-e^{-\lambda_{xi}\cdot U_{xi}}} \tag{4.20.c}$$

are guaranteed to converge to one of the 2^{n^2-n} corners of the n^2-n dimensional hypercube, defined by $V_{xi} \in \{0,1\} / x \neq i$. Although there are no doubts that computer simulation of (4.20) will reveal better convergence properties, as the neurons' dynamics will follow a gradient descent of the modified energy function E_m, the usage of (4.20) was avoided simply because there is no trivial way to find an analog hardware implementation that simulates it [5]. Since Hopfield and Tank neural network derives most of its computational power from its hardware implementation, through parallel distributed processing, then it becomes redundant to proclaim all the features of neural networks by just numerically solving (4.20), regardless of its feasibility to analog hardware implementation.

CHAPTER 5

NEURAL NETWORKS FOR OPTIMUM ROUTING IN PACKET-SWITCHED COMMUNICATIONS NETWORKS

5.1.Introduction

The choice of an optimum routing policy to forward packets from sources to destinations in a packet switched computer network is an important factor that has to be dealt with care in order to optimize some performance measures such as mean packet delay and network throughput. For many years the area of network routing has been the subject of intensive research because of its direct impact on the performance of computer networks.

Routing algorithms can generally be classified as centralized or decentralized, static, quasi-static or dynamic, deterministic or stochastic [27]. This chapter, however, focuses on the optimum quasi-static bifurcated routing problem, where the goal is to minimize the network-wide average time delay. Bifurcated or multiple path routing arises in situations where the traffic of a given commodity is allowed to be distributed over several paths. This is opposed to virtual circuit (VC) routing, where a single path is assigned to all traffic belonging to a given commodity. Here the term commodity refers to the traffic offered to a given source node (S) and destined to a given destination node (D). Although VC routing has many advantages such as simplicity reflected by the fact that all packets reach their destination in their proper sequence, it is not optimum if the goal of the network routing strategy is to minimize the long-term average network delay [27,28]. A quasi-static routing strategy assumes that the statistics of the traffic entering the network exhibit slow variations relative to their mean values, so that near optimum solutions are often obtained [29].

In the next section it will be shown that under appropriate assumptions the optimum bifurcated minimum delay routing problem can be formulated as a nonlinear multicommodity flow problem, whose solution is well known using nonlinear mathematical programming techniques. However direct application of these mathematical programming techniques to the routing problem in computer networks is not very efficient because in computer networks the optimum routing computations are to be executed in real time and usually in a distributed fashion. This makes neural networks very good candidates for implementing most of the computations involved in the routing problem, since they can operate in real time and can guarantee a high degree of robustness and fault tolerance.

The remaining of this chapter is organized as follows:
In section 5.2 the general minimum delay routing problem is formulated. Then in section 5.3 the optimum routing solution is characterized. The use of Hopfield and Tank neural network to achieve this optimum solution is highlighted. In section 5.4 the implementation of the routing algorithm in a distributed fashion is described. In section 5.5 simulation results for a single commodity and a multicommodity network are reported.

5.2. Problem Formulation of the Optimum Routing Algorithm

Consider a packet-switched (known also as a store-and-forward) communication network with N nodes and L directed links (channels) connecting them. Here a message is segmented into packets and a packet originating from node S and destined to node D is stored at any intermediate node K and then forwarded to the next node R in the route from S to D whenever channel (K,R) becomes available [30]. In a packet-switched communication network, a packet experiences delay at various stages during its journey from its source node to its corresponding destination node. First there is a processing delay at each node due to

packet header processing, routing computations and error checking. There is also a delay due to error control re-transmissions and, most importantly, there is a significant delay due to buffering of packets at each node. This chapter focuses only on link buffer packet delay. Here it will be assumed that the packet arrival process at each link buffer (queue) is random, derived from a Poisson probability distribution. Each queue is assumed to have enough capacity to accommodate all incoming packets, so that packets are always allowed to join the queue and hence are never blocked. In addition packets are processed on a First Come First Served (FCFS) basis, with an exponential distributed service time. This permits to model each queue as an M/M/1 queue.

Let:

l = link index, $l \in \{1,2,...,L\}$

C_l = capacity of link l (Data units/sec)

$\frac{1}{\mu}$ = average packet length (Data units/packet) , assumed equal for all links.

λ_l^* = average packet arrival rate to link l buffer (packets/sec).

where data units could be bits, Kbits etc...

The average waiting time or packet delay at link l queue is the sum of queuing and transmission (service) time and, for an M/M/1 queue is given by:

$$T_l = \frac{1}{\mu C_l - \lambda_l^*} \quad \text{(sec/packet)} \tag{5.1}$$

Under Kleinrock [30] independence assumption, namely independence of service time at successive nodes, a communication net can be modeled as a network of independent M/M/1

queues, in which case the expression of the average network time delay (here the delay is averaged over time and over all commodities) is given by the well known formula:

$$T = \sum_{l=1}^{L} \frac{\lambda_l^*}{\gamma} T_l \tag{5.2}$$

where:

T = total average end-to-end packet delay in sec/packet.

γ = Total packet arrival rate offered to the network (throughput) in packets/sec.

Equation (5.2) can now be written as:

$$T = \frac{1}{\gamma} \sum_{l=1}^{L} \frac{\lambda_l^*}{\mu C_l - \lambda_l^*} \tag{5.3.a}$$

$$= \frac{1}{\gamma} \sum_{l=1}^{L} \frac{\frac{\lambda_l^*}{\mu}}{C_l - \frac{\lambda_l^*}{\mu}} \tag{5.3.b}$$

$$= \frac{1}{\gamma} \sum_{l=1}^{L} \frac{f_l}{C_l - f_l} \tag{5.3.c}$$

$$= \frac{1}{\gamma} \sum_{l=1}^{L} D_l(f_l) \tag{5.3.d}$$

where :

$f_l = \frac{\lambda_l^*}{\mu}$ = average flow on link l in Data units/sec

and

$$D_l(f_l) = \frac{f_l}{C_l - f_l} \tag{5.4}$$

Since the total external load, γ, is constant then the object of the routing algorithm is to minimize :

$$T^* = \sum_{i=1}^{L} D_i(f_i) = \sum_{i=1}^{L} \frac{f_i}{C_i - f_i} \tag{5.5}$$

which, from Little's formula, has the interpretation of the average number of packets in the network.

The minimization of (5.5) is to be carried over the set of feasible multicommodity flows, in which flow balance requirements, non-negativity of flows and capacity constraint requirements are to be met. It is also convenient to formulate the routing optimization problem in terms of path rather than link flows as above, in which case the routing problem is formulated as follows:

Given a directed network of N nodes and L directed links. Let NC = {1,2,..K} be the set of K commodities (S-D pairs) in the network. Following a similar approach as in [22], the corresponding paths will be numbered sequentially, so that the sets of directed paths corresponding to the K commodities are given by:

$$\begin{aligned} P_1 &= \{1, 2, \ldots, n_1\} \\ P_2 &= \{n_1 + 1, n_1 + 2, \ldots, n_2\} \\ &\vdots \\ P_K &= \{n_{K-1} + 1, \ldots, n_K\} \end{aligned} \tag{5.6}$$

where n_1, n_2, ...n_K are some integers satisfying $n_1 < n_2 < \ldots n_K$. In (5.6), P_i denotes the set of directed paths that could be used in routing commodity i, and the integer n_K is the total number of paths corresponding to all K commodities.

Let:

$f(n)$ = traffic flow (Data units/sec) carried on path n.

$$\vec{f} = \begin{bmatrix} f(1) \\ f(2) \\ . \\ . \\ f(n_K) \end{bmatrix} = \text{vector of path flows.}$$

$\lambda(i)$ = offered external traffic (Data units/sec) to commodity i ; $i \in NC$

Then:

$$\gamma = \mu \sum_{i \in NC} \lambda(i) \tag{5.7}$$

The average link flow, f_l, can be expressed in terms of path flows as follows:

$$f_l = \sum_{n=1}^{n_K} f(n)\zeta(n,l) \tag{5.8}$$

where:

$$\zeta(n,l) = \begin{cases} 1 & \text{if path n contains link l} \\ 0 & \text{otherwise} \end{cases} \tag{5.9}$$

With the preceding definitions, the minimum average delay routing problem can be concisely formulated as follows:

Given:

- Topology and link capacities of the packet-switched network.
- A set of K commodities and their corresponding traffic requirements $\lambda(i)$'s.

Object:

Minimize average packet delay or equivalently :

$$\textit{Minimize}: \qquad F(\vec{f}) = \sum_{l=1}^{L} D_l(f_l) = \sum_{l=1}^{L} \frac{\sum_{n=1}^{n_K} f(n)\zeta(n,l)}{C_l - \sum_{n=1}^{n_K} f(n)\zeta(n,l)} \tag{5.10}$$

Variables:

Path flow vector $\vec{f}$.

Constraints:

$$\lambda(i) = \sum_{n \in P_i} f(n) \quad \forall i \in \mathrm{NC} \tag{5.11}$$

$$f(n) \geq 0 \quad \forall n \in P_1 \cup P_2 \cup \ldots P_K \tag{5.12}$$

Note that the link capacity constraints, $f_l < C_l$, are not included here since, starting from a feasible solution, the objective function will implicitly take care of these constraints, as $F(\vec{f}) \rightarrow \infty$ when $f_l \rightarrow C_l$ [22].

5.3. Characterization of Optimum routing

The solution to the optimum routing problem, formulated in (5.10 - 5.12) can be found using many iterative algorithms such as the Frank-Wolfe or flow deviation (FD) method [31], the extremal flow (EF) method [32] and the gradient projection (GP) method [33]. All these algorithms rely on the fact that the objective function (5.10) is a convex function of path flows and that the multicommodity constraints (5.11 and 5.12) form a convex set in terms of path flows. Therefore a stationary point, corresponding to a local minimum is also the global minimum. A comparison of the iterative algorithms cited above (see for example [27,34]) reveals that the GP method is suitable only for networks having

small number of commodities, which is not the case in most computer networks. The EF method requires huge memory space for networks with large number of nodes. The FD method requires less memory space and less computations, but converges slowly.

In this chapter the FD method is chosen for solving the minimum delay routing problem. A neural network algorithm is incorporated into the FD method in order to enable its real time implementation.

The application of the FD method to optimal routing in packet-switched computer networks was suggested in [31]. The FD algorithm approximates the objective function (5.10) by its tangent hyperplane, defined by the partial derivatives $\frac{\partial F}{\partial f(i)}$. Then starting from some initial feasible path flow vector $\vec{f}^{(0)}$ the algorithm changes $\vec{f}^{(0)}$ along the feasible steepest direction until the global minimum is reached. The general minimum routing algorithm, based on the FD method is illustrated in figure 5.1.

The main steps of this algorithm are the search for an initial set of feasible path flows (step 1), the shortest path computation (step 2), the minimization process (step 4) and the stopping rule (step 5) [35]. Step 1 consists of finding a feasible initial path flow vector $\vec{f}^{(0)}$, which should preferably be close to the optimum path flow vector, if the number of iterations is to be reduced. Here it will be assumed that a feasible starting flow is available, although there are methods to find it (see for example [35]). Step 2 starts by computing the first derivatives D_i' at the current path flow vector $\vec{f}^{(k)}$, where:

$$\vec{f}^{(k)} = \begin{bmatrix} f^k(1) \\ f^k(2) \\ . \\ . \\ f^k(n_K) \end{bmatrix}$$

k = iteration number

$f^k(n)$ = flow on path n at the k^{th} iteration

$$D_l'(f_l) = \frac{C_l}{(C_l - f_l)^2} \qquad \forall l \in \{1,2,..L\} \tag{5.13}$$

Under this metric, each node computes the shortest path (known also as the minimum first derivative length (MFDL) path) from itself to each of its destinations. The choice of (5.13) as a measure of link cost is derived from the Kuhn-Tucker optimality conditions, applied to the objective function (5.10).

Step 3 finds the path flow vector $\vec{f}^{(k)} = \begin{bmatrix} \vec{f}^k(1) \\ \vec{f}^k(2) \\ . \\ . \\ \vec{f}^k(n_K) \end{bmatrix}$, obtained by routing all input traffic, $\lambda(i)$, for each commodity along its corresponding shortest path.

Step 4 forms the new flow $\vec{f}^{(k+1)}$, expressed as a convex combination of $\vec{f}^{(k)}$ and $\vec{\bar{f}}^{(k)}$:

$$\vec{f}^{(k+1)} = \vec{f}^{(k)} + \alpha_k\left(\vec{\bar{f}}^{(k)} - \vec{f}^{(k)}\right); \qquad \alpha_k \in [01] \tag{5.14}$$

where α_k is chosen so as to minimize

$$G(\alpha_k) = F(\vec{f}^{(k+1)}) = F\left(\vec{f}^{(k)} + \alpha_k\left(\vec{\bar{f}}^{(k)} - \vec{f}^{(k)}\right)\right) \tag{5.15}$$

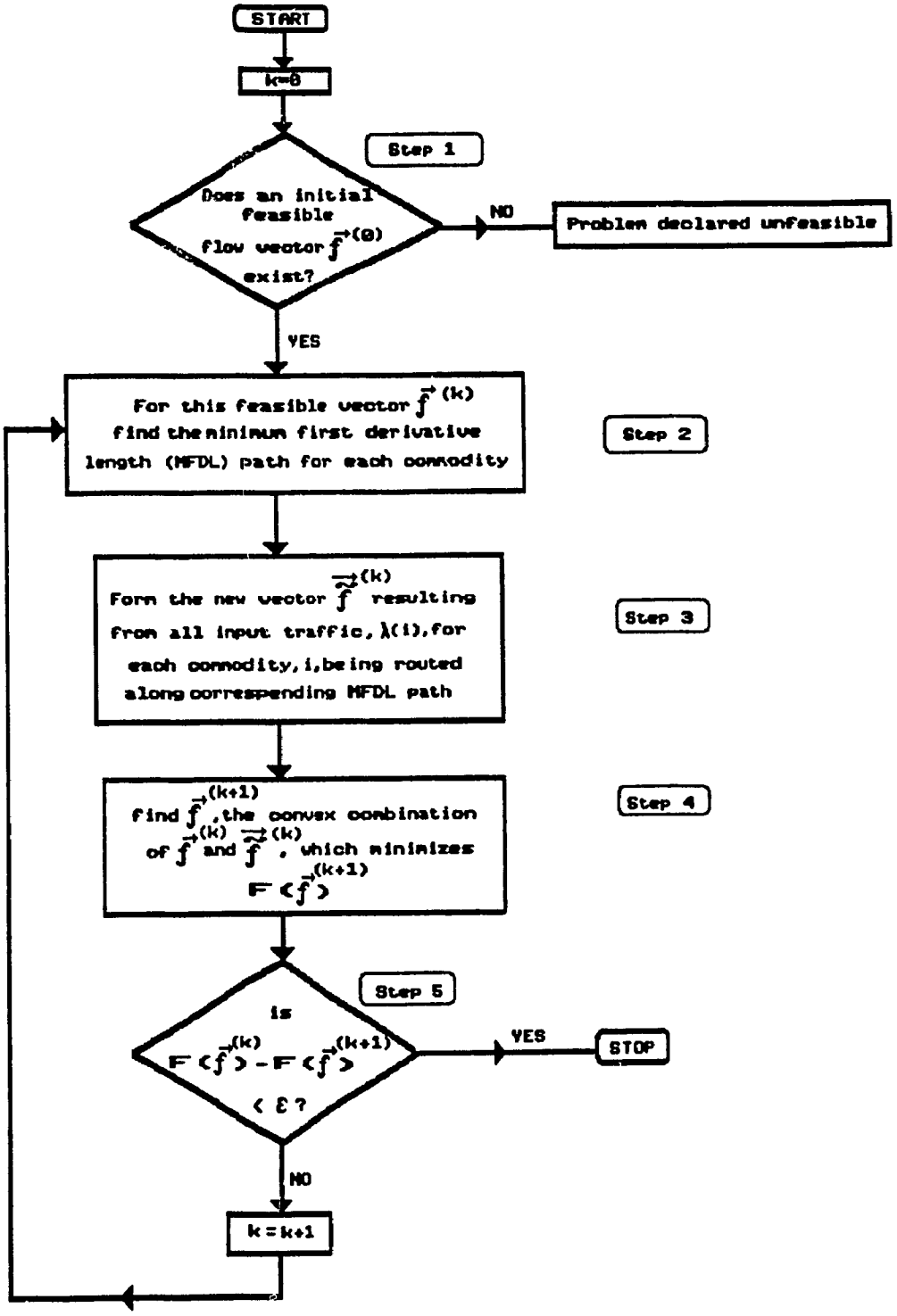


Figure 5.1. Flowchart of the Minimum Delay Routing Algorithm

One way to find the optimal α_k is to approximate $G(\alpha_k)$ by its second order Taylor series around $\alpha_k = 0$, giving [22]:

$$G(\alpha_k) \doteq \sum_{l=1}^{L} \left\{ D_l(f_l^k) + \alpha_k D_l'(f_l^k) \cdot (\bar{f}_l^k - f_l^k) + \left(\frac{\alpha_k^2}{2} \right) D_l''(f_l^k) \cdot (\bar{f}_l^k - f_l^k)^2 \right\} \tag{5.16}$$

where f_l^k and $\bar{f}_l^k$ are the total link flows corresponding to $\vec{f}^{(k)}$ and $\vec{\bar{f}}^{(k)}$ respectively; ie:

$$f_l^k = \sum_{n=1}^{n_x} f^k(n) \zeta_n(n,l) \tag{5.17.a}$$

$$\bar{f}_l^k = \sum_{n=1}^{n_x} \bar{f}^k(n) \zeta_n(n,l) \tag{5.17.b}$$

In (5.16) the first derivative D_l' is as given in (5.13), while the second derivative D'' is given by:

$$D_l''(f_l) = \frac{2 \cdot c_l}{(c_l - f_l)^3} \tag{5.18}$$

Minimization of (5.16) with respect to α_k over the interval [0,1] is obtained by setting $\frac{\partial G(\alpha_k)}{\partial \alpha_k}$ to zero, giving:

$$\alpha_k = \min \left\{ 1, -\frac{\sum_{l=1}^{L} (\bar{f}_l^k - f_l^k) D'(f_l^k)}{\sum_{l=1}^{L} (\bar{f}_l^k - f_l^k) D''(f_l^k)} \right\} \tag{5.19}$$

Step 5 decides whether further iterations are necessary by comparing the improvement brought up by the last iteration to some predetermined level of tolerance ε. If no significant improvement is made then the routing algorithm is stopped.

From the above it can be seen that the shortest path computations play a key role in the optimal routing algorithm, especially that they have to be carried out at each iteration. Therefore the choice of an efficient shortest path algorithm is a crucial factor in the per-

formance of the routing algorithm, especially for large networks where thousands of iterations (hence thousands of shortest path computations) might be required by the FD algorithm. In a packet switched network, routing and flow allocation decisions are to be made very fast, otherwise the network performance may be subject to severe degradation, at the customer dissatisfaction. As a result it is suggested that the use of the neural network SP algorithm, described in chapter 4, is highly recommended as it will reduce the execution time per iteration required by the FD method. The incorporation of the neural network SP algorithm into the optimum routing method requires that at every iteration, each node compute the MFDL path from itself to each of its corresponding destinations. Note that the link from node x to node i is labeled l, then (from (5.13)) its corresponding cost is:

$$C_{xi} = \frac{C_l}{(C_l - f_l)^2} \tag{5.20}$$

Here the link costs are time varying since they depend on link flows which change from one iteration to another. Along with the fixed connectivity information terms ρ_{xi}'s, the link costs (5.20) constitute the inputs to the neural network.

5.4. Implementation of the Routing Algorithm in a Distributed Manner

The minimum delay routing algorithm using the neural network SP algorithm can be implemented in a distributed fashion, where the computational task required at each iteration by the FD algorithm is shared among all the nodes of the network [29].

Each node measures the average input traffic $\lambda(i)$ for all commodities for which it is the source. At the beginning of the algorithm each node broadcasts to all other nodes the average link flows, f_l^*, of all its outgoing links. This could be easily done through a flooding algorithm or through a spanning tree originating from the broadcasting node. Each node can

then calculate the first and second derivatives $D'(f_l^k)$ and $D''(f_l^k)$ for all links l. Subsequently each node computes the MFDL path to each of its destinations, using the metric $D'(f_l^k)$ as link costs. This requires repeated or preferably parallel application of the neural network SP algorithm for each destination node. Each node then broadcasts to each of its destinations the current value $\lambda(i)$ of the offered traffic along the corresponding shortest path. Then every node computes the flow value $\bar{f}_l^k$ for each outgoing link and transmits the difference $(\bar{f}_l^k - f_l^k)$ to all other nodes. At this stage each node computes α_k (5.19) and updates the link flows f_l^{k+1} according to:

$$f_l^{k+1} = f_l^k + \alpha_k \cdot (\bar{f}_l^k - f_l^k) \tag{5.21}$$

The process is then repeated until the optimum path flow vector is reached.

The above implementation calls for certain degree of synchronization when broadcasting, flow updating and computing shortest paths. This synchronization is however required in order to guarantee convergence to the FD algorithm. It should also be noted that the minimum delay algorithm can also be implemented even for time-varying external flows $\lambda(i)$'s. In this case, at the k^{th} iteration, each node keeps track of the fractions of flows

$$\pi^k(n) = \frac{f^k(n)}{\lambda(i)} \qquad ; \forall n \in P_i \tag{5.22}$$

for all destinations and subsequently routes each commodity flow according to these fractions [22].

5.5. Simulation Results

The minimum delay algorithm described in section 5.3 is simulated in Fortran to solve a single commodity and a five-commodity network routing problem. The shortest path algorithm was also simulated using the proposed neural network approach as explained in section 4.5. The routing problem consists of allocating some desired commodity flow levels $\lambda(i)$'s among their corresponding paths so as to minimize the network average delay.

5.5.1. A Single Commodity Case

Consider the network shown in figure 5.2.a, where a desired flow of 20 data units per second is offered to node 1 and is to be routed to destination node 5. There are six possible paths between nodes 1 and 5, whose neural output representations are shown in figure 5.2.b.

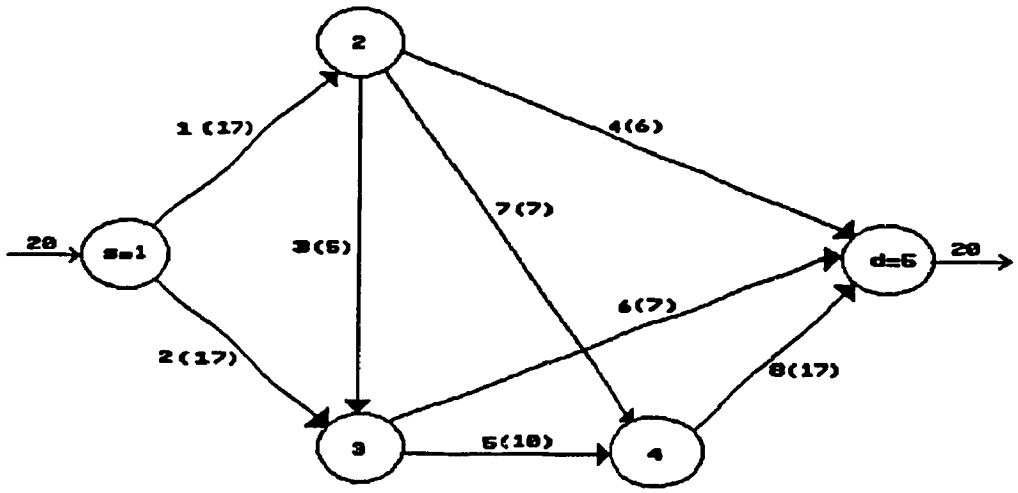
The path flow vector, $\vec{f} = \begin{bmatrix} f(1) \\ f(2) \\ f(3) \\ f(4) \\ f(5) \\ f(6) \end{bmatrix}$, is intially set to $\vec{f}^{(0)} = \begin{bmatrix} 4 \\ 5 \\ 4 \\ 0 \\ 2 \\ 5 \end{bmatrix}$.

This corresponds to the initial flow allocation depicted in figure 5.3.a, and which gives an initial cost of 20.8786.

In this case the neural network shortest path algorithm requires 20 neurons, whose dynamics are simulated according to (4.16 & 4.17) and using the following parameters:

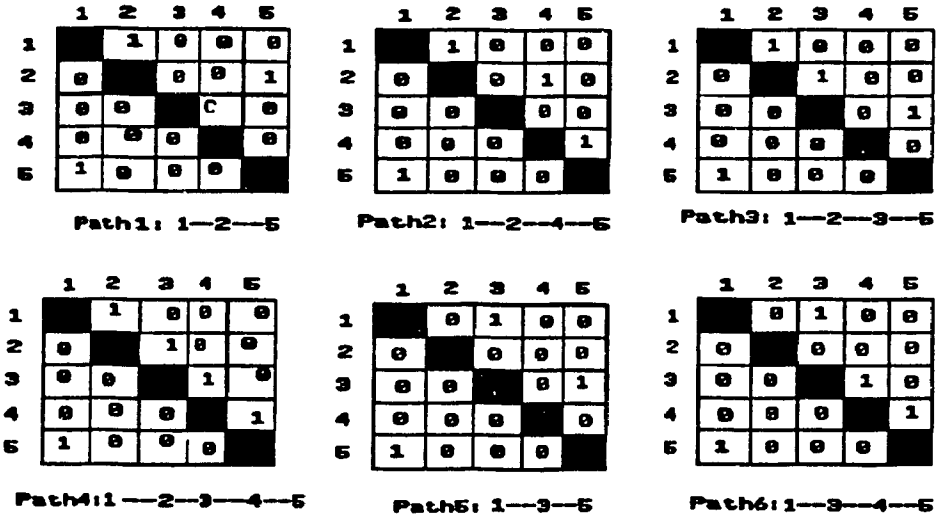
$$\mu_1 = 950 \; ; \; \mu_2 = 2500 \; ; \; \mu_3 = 1500 \; ; \; \mu_4 = 175$$

$$\mu_5 = 2500 \; ; \; \lambda = 1 \; ; \; \delta t = 10^{-5} \; ; \; \Delta V_{xi} = 10^{-5} \, V$$



5.2.a. A Network Example Used in the Simulation

(Numbers inside (outside) brackets denote link capacities (link indexes))



5.2.b. Output Representation of the Feasible Paths

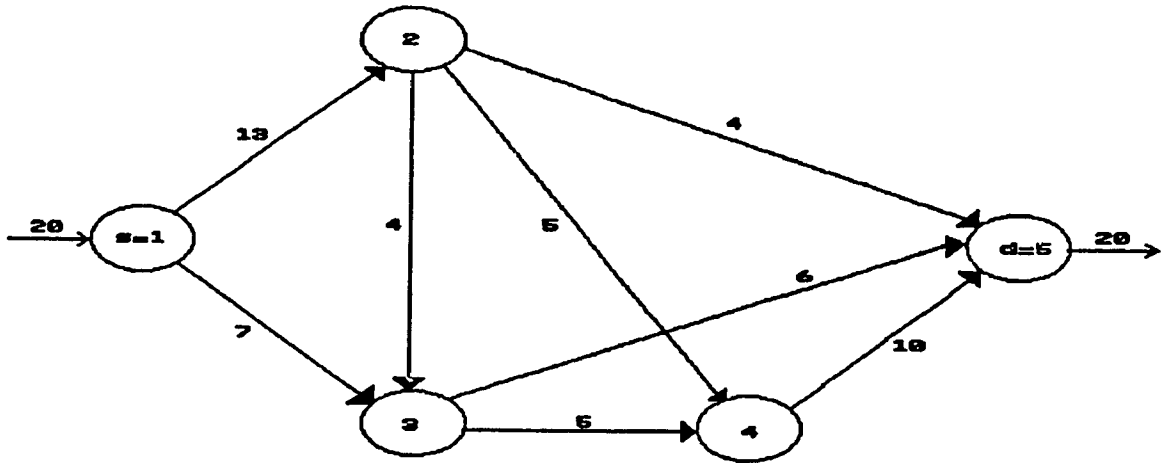
Figure 5.2. The Single Commodity Example

For the simulated FD algorithm, a tolerance value of 10^{-2} was assigned to ε (step 5). With these sets of parameters, the simulated minimum delay algorithm converged to the desired solution after 29 iterations.

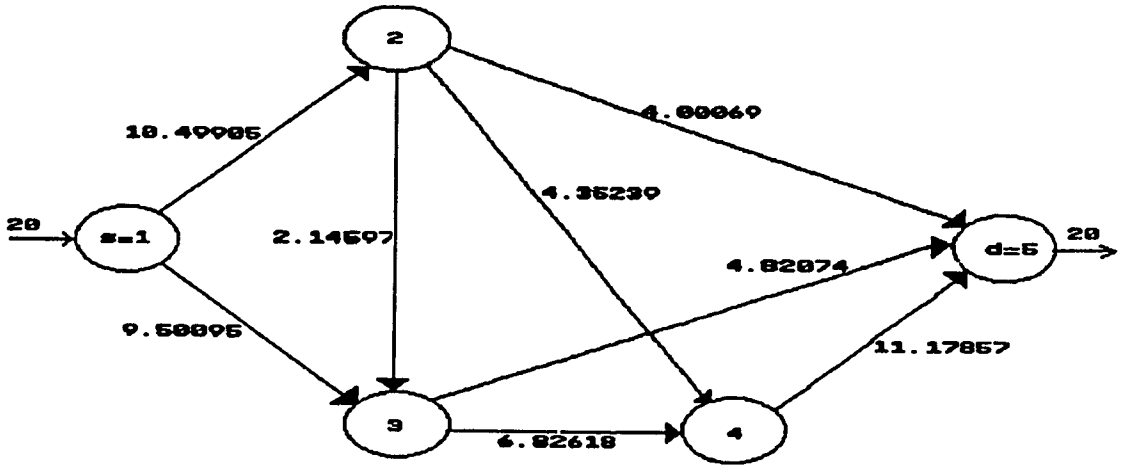
The final path flow vector is $\vec{f}^{(29)} = \begin{bmatrix} 4.00069 \\ 4.35239 \\ 2.14597 \\ 0 \\ 2.67477 \\ 6.82618 \end{bmatrix}$.

Correspondingly the optimum flow allocation is obtained as shown in figure 5.3.b. In figure 5.4, the objective function, given by equation (5.10) is also plotted against number of iterations.

During each of the 29 FD iterations, the neural network SP algorithm has successfully converged to valid paths, which are also global optima. Figures 5.5 and 5.6 illustrate two neural network shortest path computations, corresponding to two different iterations.



5.3.a. Initial Flow Allocation
Cost = 20.8786



5.3.b. Optimal Flow Allocation
Cost = 13.5619

Figure 5.3. Initial and Final Flow Allocation for the Single Commodity Example

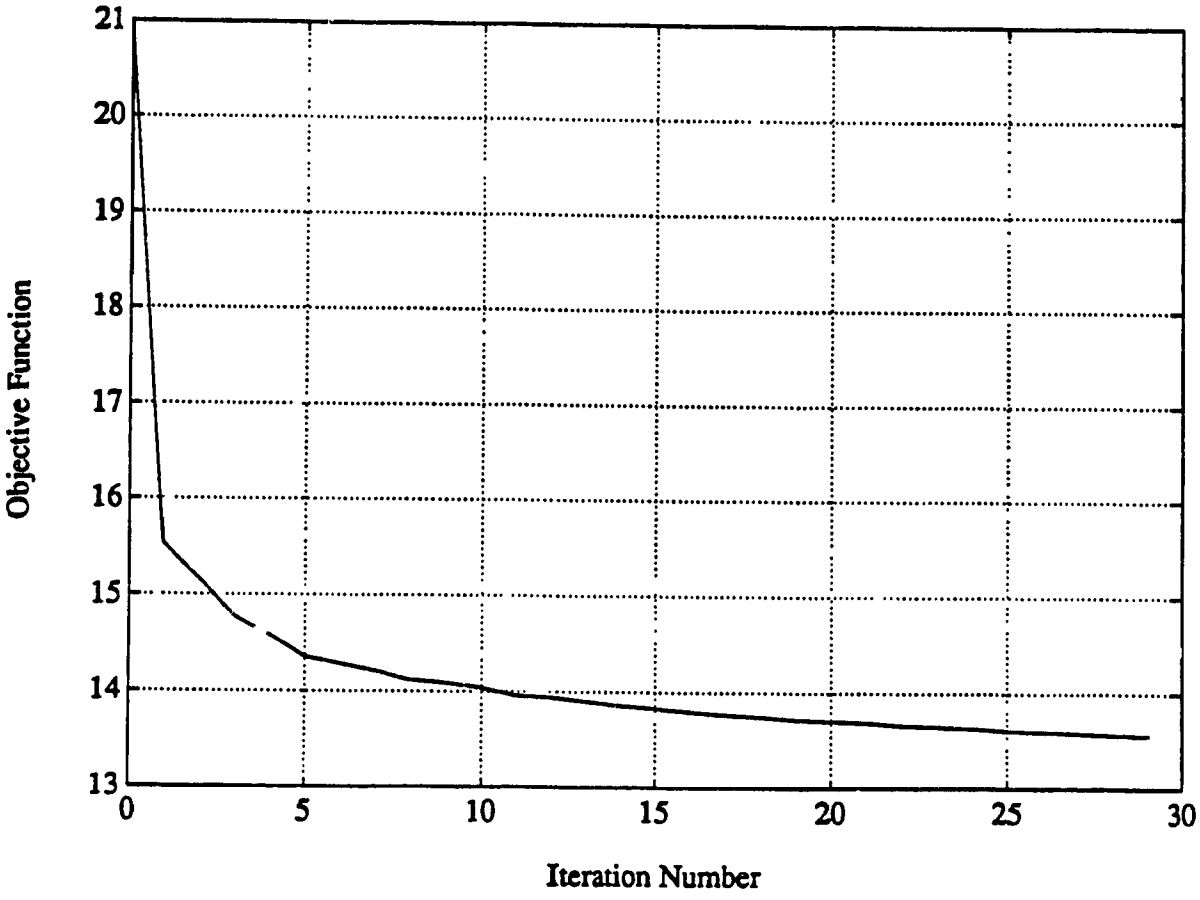


Figure 5.4. Simulation Results for the Single Commodity Network

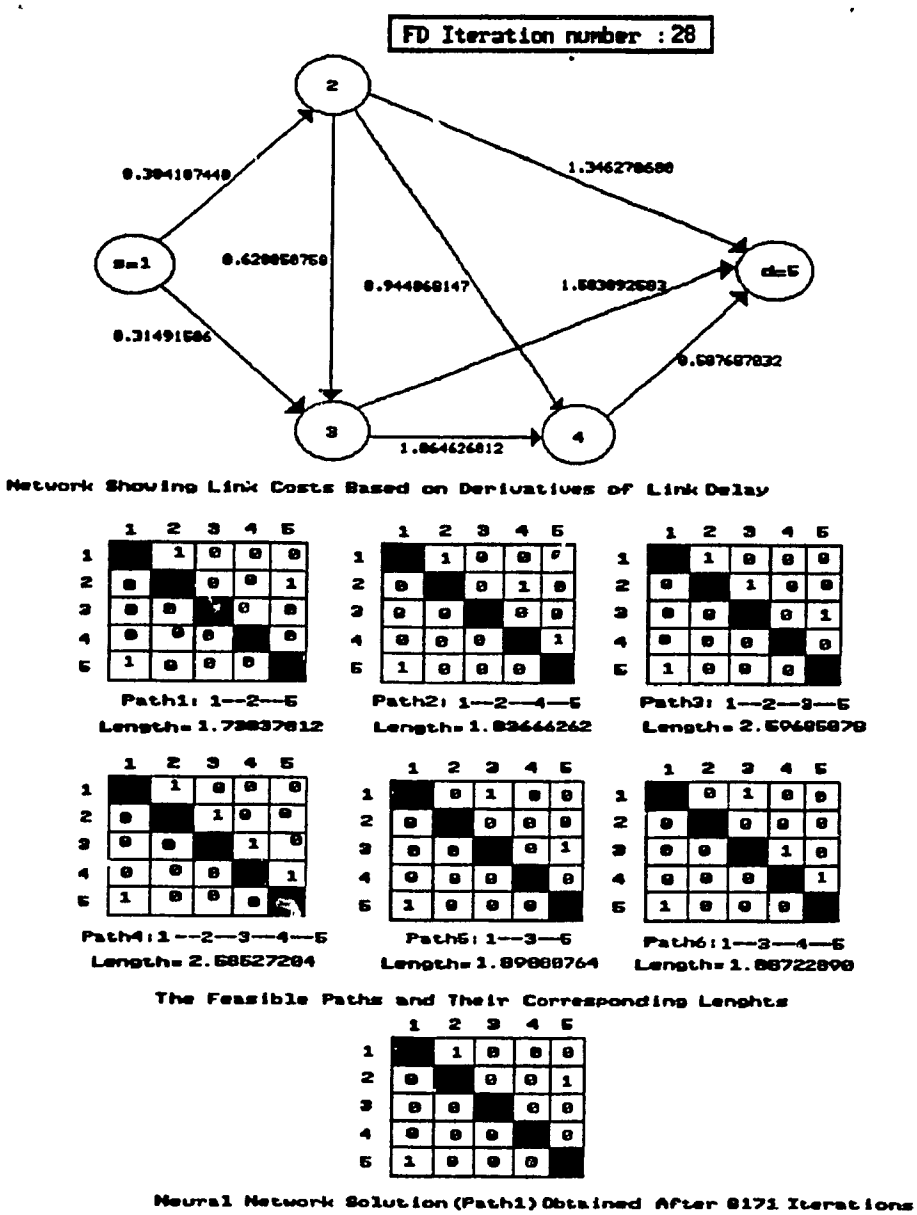


Figure 5.5. Example 1 of Shortest Path Computation

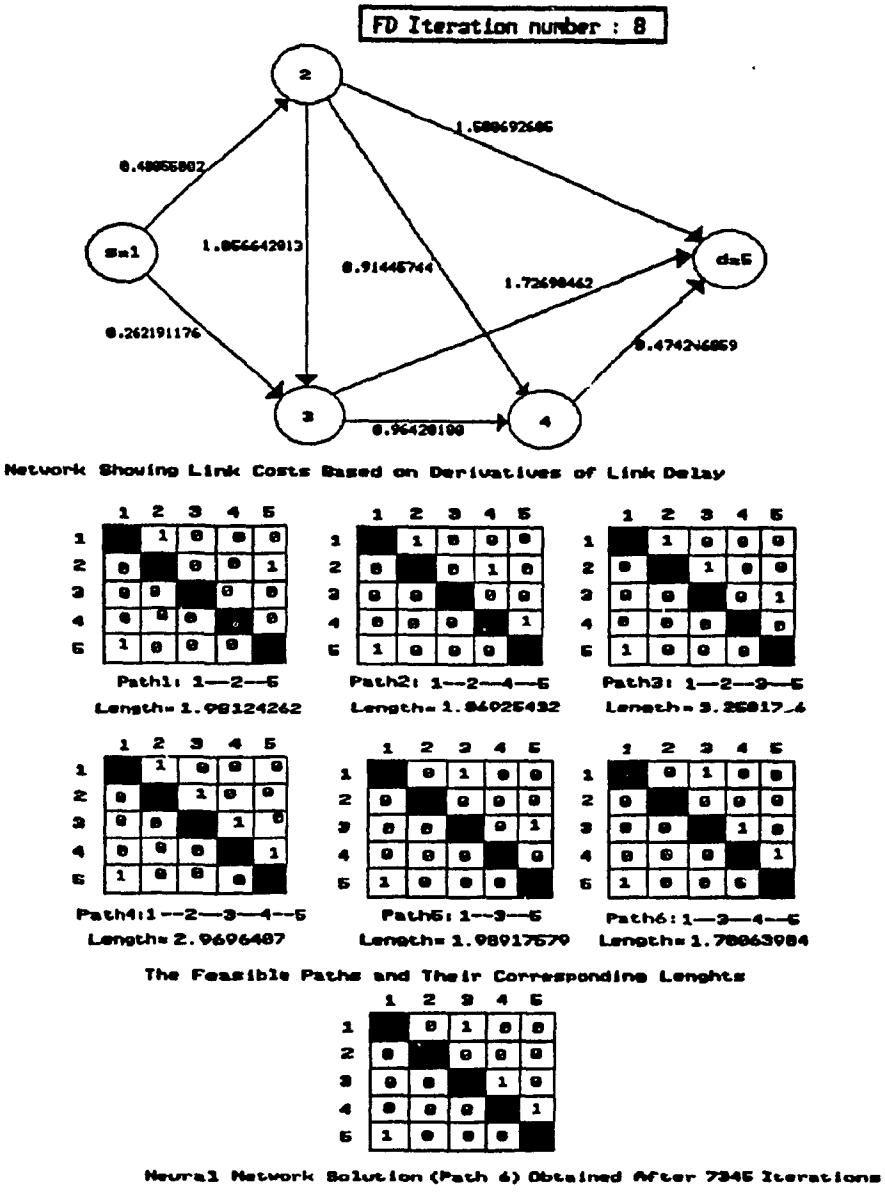


Figure 5.6. Example 2 of Shortest Path Computation

5.5.2. A Five Commodity Case

Here we consider the 5 commodity network shown in figure 5.7, where the goal is to route the traffic inputs $\lambda(i)$'s for all the commodities specified in table 5.1 so as to minimize the average traffic delay.

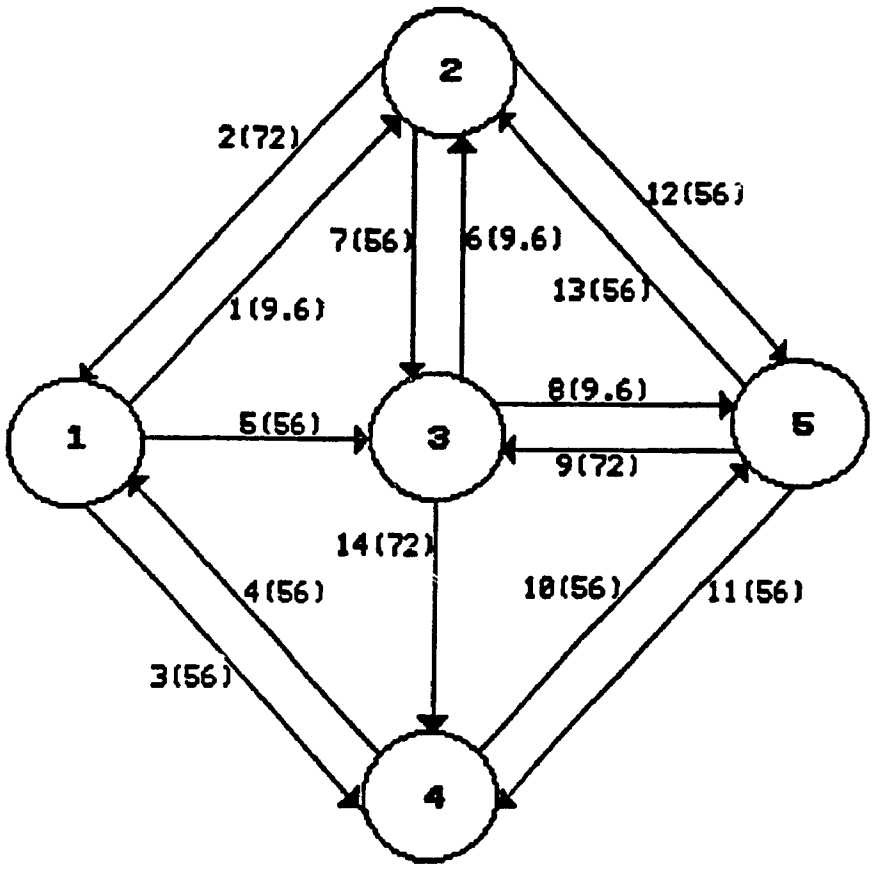


Figure 5.7. The Five Commodity Network

(Numbers inside (outside) brackets denote link capacities (link indexes))

Table 5.1. Traffic Requirements of the Multicommodity Network

Commodity i	Source	Destination	Traffic input λ(i)
1	1	5	50
2	1	3	45
3	2	4	35
4	4	3	15
5	5	1	30

In this example the set of commodities is NC = {1,2,3,4,5} and the corresponding sets of directed paths are:

$$P_1 = \{1,2,3,4,5,6,7\}$$

$$P_2 = \{8,9,10,11,12\}$$

$$P_3 = \{12,13,14,15,16,17,18,19\}$$

$$P_4 = \{20,21,22,23,24,25\}$$

$$P_5 = \{26,27,28,29,30\}$$

where each path is defined by its node sequence, as specified in table 5.2.

Table 5.2. Characterization of the Paths of the Multicommodity network

Commodity	Path	Node sequence
1	1	1-2-5
	2	1-2-3-5
	3	1-2-3-4-5
	4	1-3-2-5
	5	1-3-5
	6	1-3-4-5
	7	1-4-5
2	8	1-3
	9	1-2-3
	10	1-2-5-3
	11	1-4-5-3
	12	1-4-5-2-3
3	13	2-3-4
	14	2-1-4
	15	2-5-4
	16	2-5-3-4
	17	2-5-3-4
	18	2-1-3-4
	19	2-1-3-5-4
4	20	4-1-3
	21	4-1-2-3
	22	4-1-2-5-3
	23	4-5-3
	24	4-5-2-3
	25	4-5-2-1-3
5	26	5-2-1
	27	5-2-3-4-1
	28	5-3-2-1
	29	5-3-4-1
	30	5-4-1

The path flow vector is initially set to the feasible flow $\vec{f}^{(0)} = \begin{bmatrix} f^0(1) \\ f^0(2) \\ . \\ . \\ f^0(30) \end{bmatrix}$ where:

$$f^0(7) = 50 \; ; \; f^0(8) = 45 \; ; \; f^0(13) = 35$$
$$f^0(20) = 10 \; ; \; f^0(21) = 5 \; ; \; f^0(26) = 30$$

with all remaining path flow components being set to zero. This corresponds to the initial flow allocation shown in figure 5.8.a, and which gives an initial cost of 78.4336.

Once again the dynamics of the neural network (consisting of 20 neurons)are simulated using (4.16 & 4.17) and with the following parameters:

$$\mu_1 = 550 \; ; \; \mu_2 = 2500 \; ; \; \mu_3 = 2000 \; ; \; \mu_4 = 85$$
$$\mu_5 = 2500 \; ; \; \lambda = 1 \; ; \; \delta t = 10^{-5} \; ; \; \Delta V_{ik} = 10^{-5} V$$

In all runs the neural network SP computations have converged to optimum solutions. In addition, with a tolerance value (ε) of 10^{-2}, the simulated minimum delay routing algorithm has reached the optimum solution after 38 iterations. The components of the optimum path flow vector, $\vec{f}^{(38)}$, have converged to the following values:

$$f^{38}(1) = 1.88967 ; f^{38}(4) = 0.270102 ; f^{38}(5) = 1.29328$$

$$f^{38}(7) = 46.5469 ; f^{38}(8) = 42.1885 ; f^{38}(10) = 1.70069$$

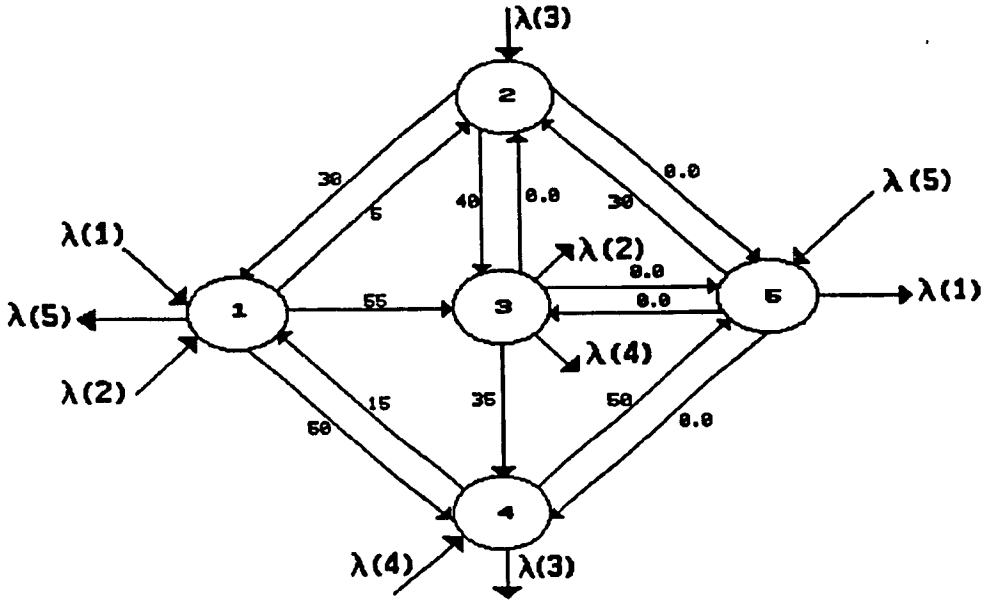
$$f^{38}(11) = 1.11084 ; f^{38}(13) = 27.1009 ; f^{38}(15) = 7.89910$$

$$f^{38}(20) = 7.74312 ; f^{38}(21) = 3.87156 ; f^{38}(22) = 0.29370$$

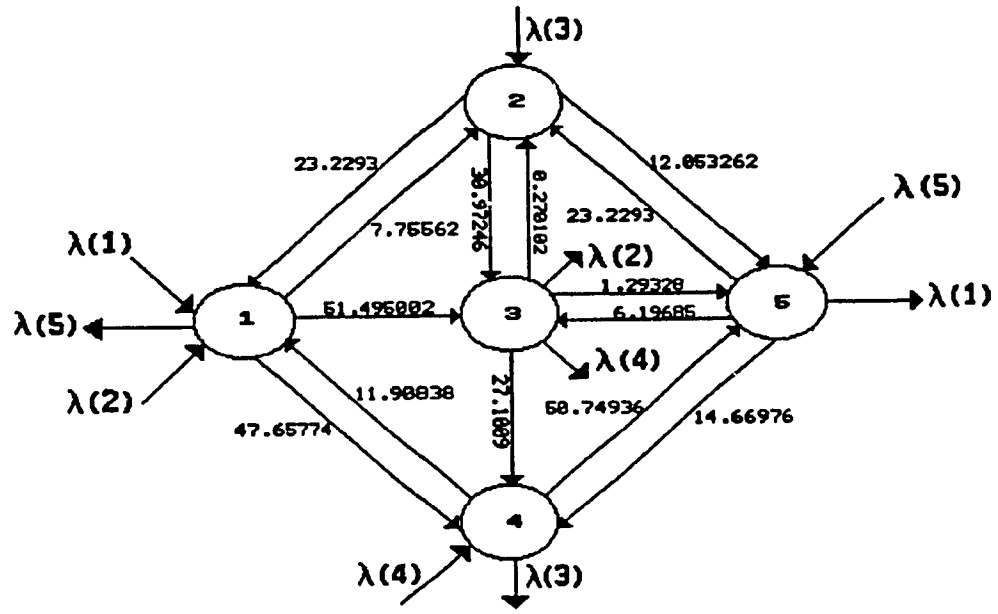
$$f^{38}(23) = 3.09162 ; f^{38}(26) = 23.2293 ; f^{38}(30) = 6.77066$$

with all remaining components set to zero. This corresponds to the optimum flow allocation shown in figure 5.8.b and whose cost is 35.4485. In figure 5.9 the objective function (equation 5.10) is also plotted as function of number of iterations.

Finally it should be noted that while the applicability of the SP neural network algorithm to traffic routing was demonstrated in conjunction with the FD method, other routing algorithms can as well benefit from the SP neural implementation. In fact most of the current operating packet-switched networks use some form of shortest path computation, where a cost measure (fixed or variable) is assigned to each link and a least-cost path between each source-destination pair is sought. Here again the cost of a path is defined as the sum of the costs of each link. Most of today's routing algorithms, however, differ in the way the link costs are defined and computed and they also differ in the way the routing computations are performed.



5.8.a. Initial Flow Allocation
Cost = 78.4336



5.8.b. Optimal Flow Allocation
Cost = 35.4485

Figure 5.8. Initial and Final Flow Allocation for the Five Commodity Example

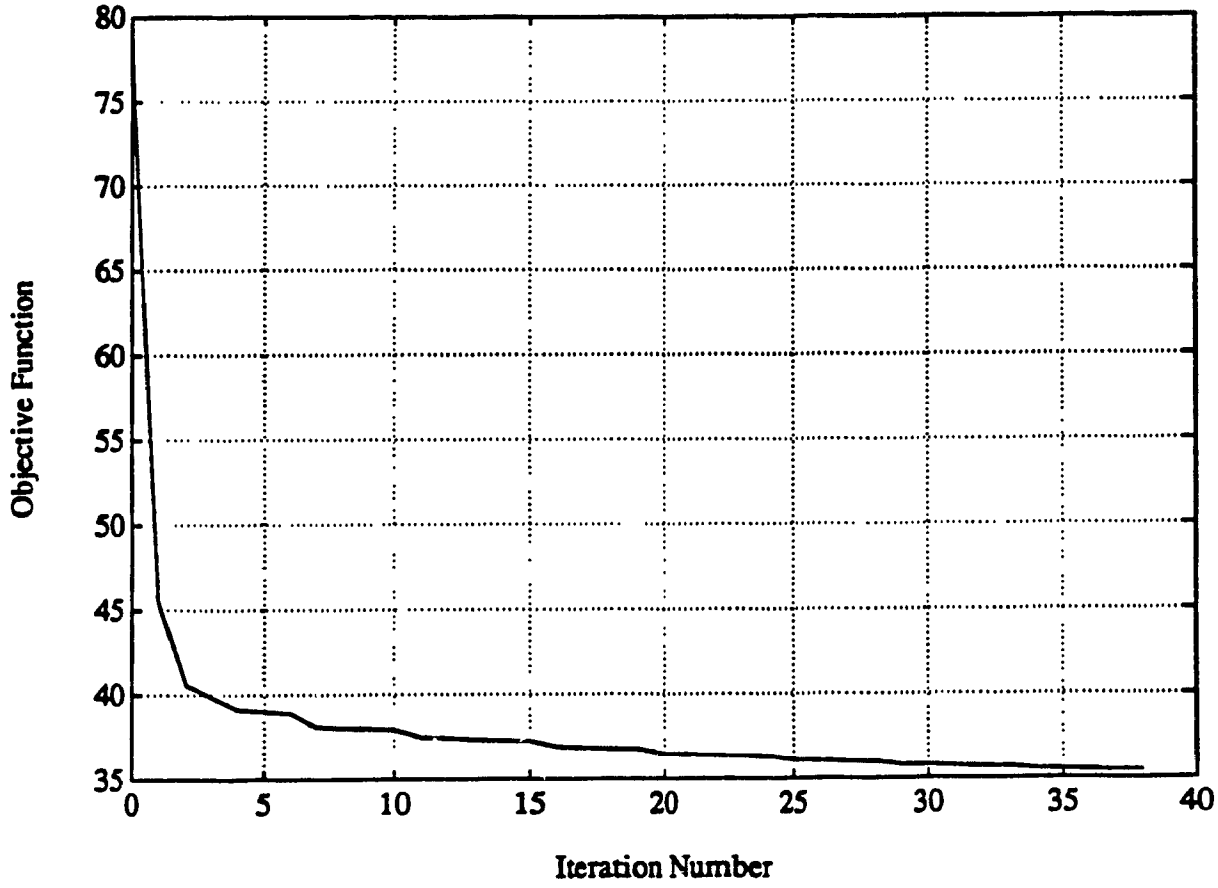


Figure 5.9. Simulation Results of the Five Commodity Network

CHAPTER 6

CONCLUSION

The computational power of neural optimization networks in solving routing and flow allocation problems in communication networks was quantitatively and qualitatively demonstrated.

The potential use of neural networks to solve routing and flow allocation optimization problems is motivated by the inherent features of these highly interconnected networks of analog processors. These features, most of which are brought by parallel distributed information processing, include a fascinating computational power and speed, a robustness and fault tolerance with respect to the failure of individual processors, a low power consumption and finally an aptness to real time operation, combined with an adaptivity to fluctuations in traffic characteristics.

In chapter 2, Hopfield and Tank neural network was described in details and an extended version of Chua and Lin Linear Programming network was proposed. These two neural optimization networks are of feedback type and are characterized by their dynamic behavior which follows a gradient descent of a global penalty function that combines both the objective function and a series of local penalty functions for constraints' violations. Starting from a given initial state, the dynamics of the neural network, described by the output voltages of its neurons, converge to a stationary state which corresponds to the desired solution.

The application of the Extended Linear Programming circuit to the maximum throughput, routing problem was successfully demonstrated in chapter 3. Simulation of the

circuits' dynamics, using a digital computer, has revealed that from a computational point of view, the Extended Linear Programming network is very efficient in solving the problem. It was also shown that the computation time of the Extended Linear Programming network does not increase very rapidly with the problem size. Further, since the feasible state space of linear programming problems is always convex, then the Extended Linear Programming circuit is guaranteed to converge to the optimum solution provided that the constraint resistances associated with the constraint processors are sufficiently high. It should be noted that the simulation results obtained in chapter 3 are just approximations to what would have been obtained by hardware implementation. These results were obtained based on many assumptions such as ideality of all circuit components and no inherent delay in the response of the constraint processors. In addition the numerical solution technique used to solve the differential equations, describing the dynamics of the Extended Linear Programming networks is approximate. More accurate solution techniques could have been considered, but it was found that they often result in a prohibitive simulation time, especially for large size problems. To this effect one has to keep in mind that the neural dynamics were simulated on a sequential digital computer.

In chapter 4, a new solution to the shortest path problem was proposed, using Hopfield type neural network. The general principles involved in the design of Hopfield and Tank neural network to solve the SP problem were discussed. The proposed model combines many features, such as flexibility to operate efficiently in real time and to adapt to changes in network topology and link costs. By properly chosing the heuristic coefficients μ_i's it was found that, for the SP problem, Hopfield and Tank neural model always converges to valid solutions which are also global minima.

In chapter 5 The proposed neural network SP algorithm was applied to the optimum

minimum delay routing problem in packet-switched computer communications networks. It was found that in a quasi-static environment, Hopfield and Tank neural network SP algorithm could be used in conjunction with the FD method to route traffic so as to minimize the average network delay. The applicability of the proposed neural network SP algorithm to the routing problem was successfully demonstrated through computer simulation. When implemented in real time, it is expected that Hopfield and Tank neural network SP algorithm will speed-up the execution time of the FD algorithm. The implementation of the neural network based, routing algorithm in a distributed fashion was also considered.

Finally there are some obstacles that have to be overcome if the practical use of neural optimization networks in solving routing and flow allocation problems is to be concretized:

- The first obstacle is the hardware challenge of microfibricating very large resistive connection matrices and a huge number of integrated op-amps on a single VLSI chip. Although advances in analog VLSI technology have been escalating during the past few years, there is a lot to be done before VLSI chips with thousands of resistive connections and op-amps become available. In addition three-dimensional optical technology may be considered to solve the communication bottleneck problem encountered when implementing neural networks on VLSI chips. However advances in this very promising area are still at their primitive stages. Another hardware problem is the inability, with today's available technology, to design very accurate resistors on a silicon wafer. The fact that the applicability of neural optimization networks to large scale optimization problems is tightly coupled to future technological developments is not a very new situation for engineers; after all in most previous successful engineering achievements the ideas have preceded the technology.
- The second problem that has to be tackled is the software challenge of programming a huge number of resistive interconnections and biases on a VLSI chip.

- Another equally important problem is how to design Hopfield and Tank neural optimization network with a unique equilibrium point, corresponding to the global minimum. This would prevent the network from being trapped in a local minimum. Further research is required to find a systematic way to estimate the values of the μ_i's coefficients, so that the global minimum would always be obtained.

Finally neural networks which are based on learning algorithms (example: the back-propagation network [36]) can be investigated for possible use in dynamic routing control. These neural networks are well suited to dynamic routing environments because they operate in real time and they rapidly adapt to changes in traffic characteristics by learning through actual examples. A neural network that uses a learning algorithm, such as back-propagation, can learn a nonlinear function between the observed input data and the optimal routing decision by training itself with sufficient number of sample data. When confronted with new input data that it has never been trained with, the neural network can successfully generate its own rule and take the appropriate routing action. Future work that concentrates on the application of these learning algorithms to dynamic routing problems is recommended.

References

[1] J.J.Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc.Natl.Acad.Sci.USA, Vol.79, 1982, pp.2554-2558.

[2] J.J.Hopfield and D.W Tank, " 'Neural' Computation of Decisions in Optimization Problems", Biological Cybernetics, Vol.52, 1986, pp141-152.

[3] D.W.Tank and J.J.Hopfield, "Simple 'Neural' Optimization Networks: an A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit ", IEEE Trans.Circuits.Syst. CAS-33, No.5, 1986, pp.533-541.

[4] M.Takeda and J.W.Goodman, " Neural Networks for Computation: Number Representations and Programming Complexity ", Applied Optics, Vol.25, No.18, 1986, pp.3033-3046.

[5] B.R.Copeland, "Global Minima Within the Hopfield Hypercube ", International Joint Conference on Neural Networks, January 1990.pp.I377-I380.

[6] L.O.Chua and G.N.Lin, " Nonlinear Programming Without Computation ", IEEE Trans. Circuits. Syst, 1984, Vol.Cas-31, pp.182-188.

[7] L.O.Chua and G.N.Lin, " Errata to 'Nonlinear Programming Without Computation ' ", IEEE Trans. Circuits. Syst. 1985. Vol.Cas-32, pp. 736.

[8] M.P.Kennedy and L.O.Chua, "Unifying the Tank and Hopfield Linear Programming Circuit and the Canonical Nonlinear Programming Circuit of Chua and Lin", IEEE Trans.Circuits. Syst, 1987, Vol.CAS-34, pp.210-214.

[9] M.P.Kennedy and L.O.Chua, "Neural Networks For Nonlinear Programming", IEEE Trans Circuits. Syst, 1988, Vol-CAS-35, pp.554-562.

[10] J.J.Hopfield, "Artificial Neural Networks", IEEE Circuits and Devices Magazine, Vol.4, No.5, Sept 1988, pp.3-10.

[11] J.J.Hopfield, "Neurons With Graded Response Have Collective Computational Properties Like Those of Two-State Neurons", Proc.Natl.Acad.Sci. USA, 1984, Vol81, pp.3088-3092.

[12] Y.Yao, "Dynamic Tunneling Algorithm for Global Optimization", IEEE Trans On Systems, Man, and Cybernetics, 1989, Vol.19, No.5, pp.1222-1230.

[13] P.R.Adby and M.A.H.Dempster, "Introduction To Optimization Methods", 1982, pp.119-136, Chapman and Hall Ltd.

[14] A.V.Fiacco and G.P.McComik, "NonLinear Programming: Sequential Unconstrained Minimization Techniques", 1968, John Wiley & Sons Inc .

[15] A.A.Assad, "Multicommodity Network Flows - A Survey ", Networks, 1978, John Wiley & Sons, Inc, Vol.8, pp.37-91.

[16] T.B.Boffey, "Graph Theory in Operation Research", The Macmillan Press Ltd, 1982, pp.227-228.

[17] L.R.Ford and D.R.Fulkerson, "Flows In Networks", Princeton University, 1962.

[18] T.C.Hu, "Multicommodity Network Flows", Operations Res, Vol.11, No.3, May-June 1963, pp.344-360.

[19] T.C.Hu, "Integer Programming and Network Flow", Addison-Wesley, Reading, Masachusetts, 1969.

[20] L.R.Ford and D.R.Fulkerson, "A Suggested Computation For Maximal Multicommodity Network Flows", Management Sci, 1958, Vol.5, pp.97-101.

[21] F.J.Gratzer and K.Steiglitz, "A Heuristic Approach to Large Multicommodity Flow Problems", Proceedings of The Symposium on Computer Communications Networks and

Teletraffic, New York , April 1972, pp.311-324.

[22] D.P.Berteskas and R.G.Gallager, "Data Networks", Prentice-Hall INC, 1987.

[23] H.E.Rauch and T.Winarske, "Neural Networks for Routing Communication Traffic", IEEE Control System Magazine, April 1988, pp.26-30.

[24] L.Zhang and S.C.A.Thomopoulos, "Neural Network Implementation of the Shortest Path Algorithm for Traffic Routing in Communication Networks", International Joint Conference On Neural Networks, June 1989, p II591.

[25] A.W.Alkhafaji and J.R.Tooley, "Numerical Methods in Engineering Practice ", H.R.W.Inc, 1986.

[26] P.W.Protzel, "Comparative Performance Measure for Neural Networks Solving Optimization Problems", International Joint Conference On Neural Networks, 1990, pp.II.523-526

[27] M.Schwartz, "Computer-Communications Network Design and Analysis ", Prentice-Hall Inc, Englewood Cliffs, New Jersey, 1977.

[28] M.Schwartz and T.Stern, "Routing Techniques Used in Computer Communication Networks ", IEEE Transaction on Comm, Vol.COM-28, No.4, April 1980, pp.539-552.

[29] J.F.Hayes, "Modeling and Analysis of Computer Communication Networks", Plenum Press, 1984.

[30] L.Kleinrock, "Communication Networks: Stochastic Message Flow and Delay", McGraw-Hill Book Co, New York, 1964.

[31] L.Fratta, M.Gerla, and L.Kleinrock, "The Flow Deviation Method: An Approach to Store-and-Forward Communication Network Design", Networks, Vol.3, 1973, pp.97-133.

[32] D.G.Cantor and M.Gerla, "Optimal Routing in a Packet Switch Computer Network", IEEE Trans on Comm, Vol.C-23, No.10, Oct 1975, pp.1062-1069.

[33] M.Schwartz and C.K.Cheung, "The Gradient Projection Algorithm for Multiple Routing in Message Switched Systems", IEEE Trans on Comm, Vol.Com-24, No.4, April 1976, pp.449-456.

[34] M.Schwartz, "Telecommunication Networks: Protocols, Modeling and Analysis", Addison-Wesley, 1987.

[35] P.J.Courtois and P.Semal, "A Flow Assignment Algorithm Based on the Flow Deviation Method", ICCC 1980, pp.77-83.

[36] D.E.Rumelhart, J.L.Mc Clelland and the PDP research group, "Parallel Distributed Processing", Vol.1.MIT Press, 1987.

APPENDIX A

CHUA AND LIN CANONICAL LINEAR PROGRAMMING CIRCUIT

A.1 Network Description

Chua and Lin [6,7,9] consider the following LP problem:

Minimize the scalar function:

$$\Phi(\vec{V}) = \vec{A} \cdot \vec{V} \qquad (a.1)$$

Subject to the constraints:

$$f_j(\vec{V}) = \vec{B_j} \cdot \vec{V} - E_j \geq 0\,; \qquad j = 1,2,\ldots p \qquad (a.2)$$

Or equivalently:

$$f(\vec{V}) = B\vec{V} - \vec{E} \geq \vec{0} \qquad (a.3)$$

where:

$$\vec{A} = \begin{bmatrix} A_1 \\ A_2 \\ . \\ . \\ . \\ A_q \end{bmatrix}; \vec{V} = \begin{bmatrix} V_1 \\ V_2 \\ . \\ . \\ . \\ V_q \end{bmatrix}; \vec{B_j} = \begin{bmatrix} B_{j1} \\ B_{j2} \\ . \\ . \\ . \\ B_{jq} \end{bmatrix}; \vec{E} = \begin{bmatrix} E_1 \\ E_2 \\ . \\ . \\ . \\ E_p \end{bmatrix}; B = \begin{bmatrix} B_{11}B_{12} \ldots B_{1q} \\ B_{21}B_{22} \ldots B_{2q} \\ \ldots\ldots \\ \ldots\ldots \\ B_{p1}B_{p2} \ldots B_{pq} \end{bmatrix} \qquad (a.4)$$

They have shown [9] that the network shown in figure a.1 solves this problem with no risk of oscillation.

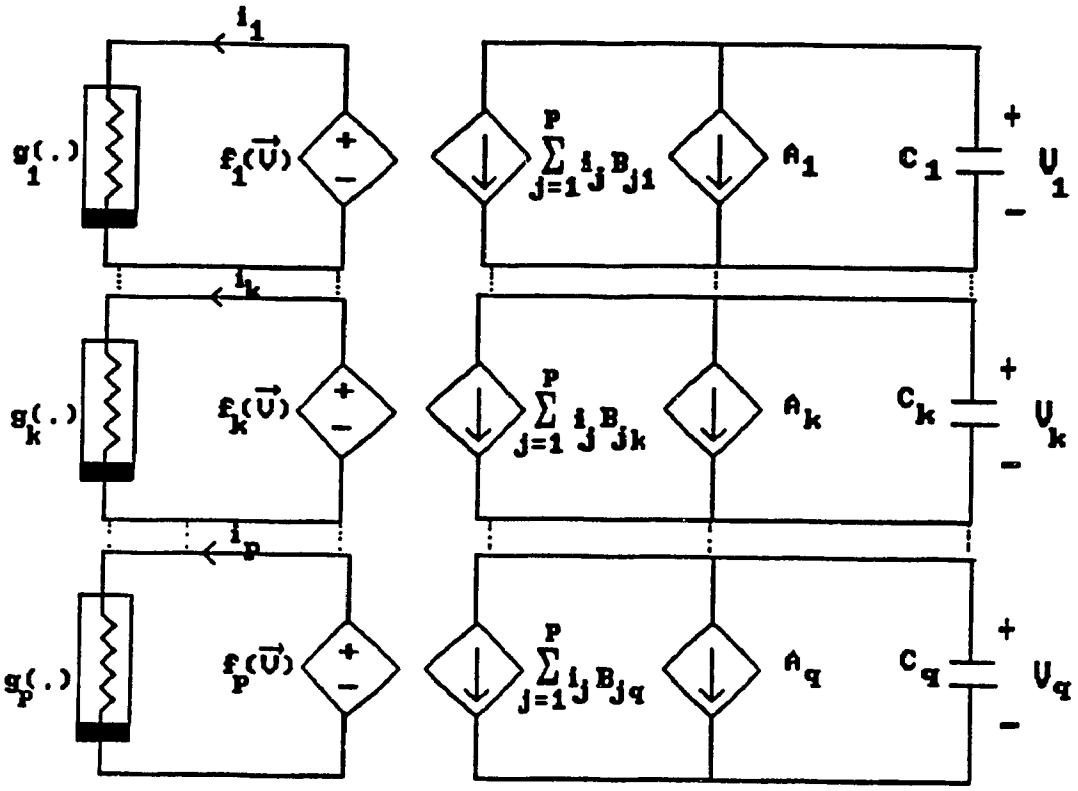


Figure a.1. Chua and Lin Linear Programming Network

The dynamic neural model shown in figure a.1 consists of controlled current and voltage sources, nonlinear resistors and linear capacitors. Each symbol on the left side of figure a.1 represents a voltage controlled nonlinear resistor whose characteristic (shown in figure a.2) is governed by the following equation:

$$g_j(V)=\begin{cases}0 & \text{if } V>0\\ \dfrac{V}{R_j} & \text{if } V\le 0\end{cases} \qquad (a.5)$$

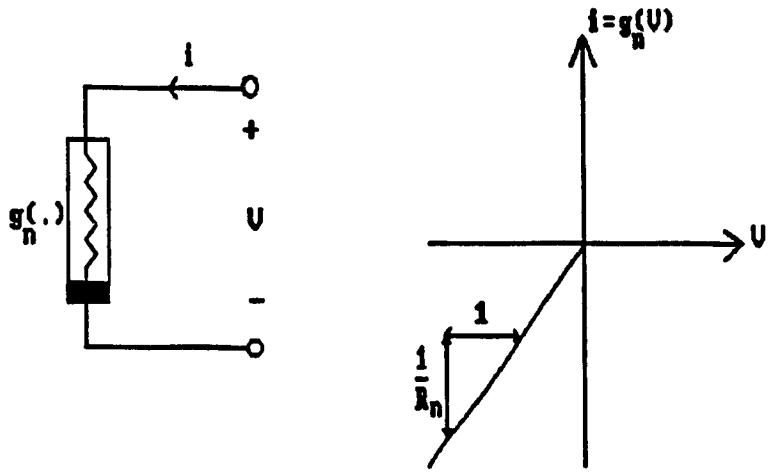


Figure a.2. Chua and Lin Nonlinear Resistor Characteristic

The circuit equation for the k^{th} row of the network shown in figure a.1 is given by:

$$C_k \frac{dV_k}{dt} = -A_k - \sum_{j=1}^{p} B_{jk} i_j \qquad (a.6.1)$$

$$= -A_k - \sum_{j=1}^{p} B_{jk} g_j(f_j(\vec{V})) \qquad (a.6.2)$$

Next substituting from (a.2) for $f_j(\vec{V})$ we get:

$$C_k \frac{dV_k}{dt} = -A_k - \sum_{j=1}^{p} B_{jk} g_j(\vec{B_j} \cdot \vec{V} - E_j) \qquad (a.7)$$

$$\forall k \in \{1,2,\ldots q\}$$

To show how the model shown in figure a.1 solves the linear programming problem formulated in (a.1-a.4), Chua and Lin consider the scalar function:

$$P : \mathbb{R}^q \longrightarrow \mathbb{R}$$

$$\vec{V} \longrightarrow P(\vec{V}) = \vec{A} \cdot \vec{V} + \sum_{j=1}^{p} G_j(f_j(\vec{V})) \qquad (a.8)$$

where :

$$G_j(V) = \int_0^V g_j(x)\,dx = \begin{cases} 0 & \text{if } V > 0 \\ \dfrac{V^2}{2R_j} & \text{if } V \le 0 \end{cases} \qquad (a.9)$$

The time derivative of P is:

$$\frac{dP}{dt} = \sum_{k=1}^{q} \frac{\partial P}{\partial V_k} \cdot \frac{dV_k}{dt} \qquad (a.10.1)$$

$$\frac{dP}{dt} = \sum_{k=1}^{q} \frac{\partial P}{\partial V_k} \cdot \frac{dV_k}{dt} \qquad (a.10.1)$$

$$= \sum_{k=1}^{q} \left(A_k + \sum_{j=1}^{p} B_{jk} g_j(\vec{B_j} \cdot \vec{V} - E_j) \right) \cdot \frac{dV_k}{dt} \qquad (a.10.2)$$

The equation inside the parentheses is nothing but $-C_k \frac{dV_k}{dt}$, therefore:

$$C_k \frac{dV_k}{dt} = -\frac{\partial P}{\partial V_k} \qquad (a.11)$$

and

$$\frac{dP}{dt} = -\sum_{k=1}^{q} C_k \left(\frac{dV_k}{dt} \right)^2 \le 0 \qquad (a.12)$$

In addition

$$\frac{dP}{dt} = 0 \Leftrightarrow \frac{dV_k}{dt} = 0 \quad \forall k \in \{1, 2, \ldots, q\} \qquad (a.13)$$

Therefore Chua and Lin conclude that the state of the network described by (a.6) follows a gradient descent of the scalar function P , and that starting from some initial condition, this state evolves towards a minima of P and stabilizes towards a stationary point when

$$\frac{dV_k}{dt} = 0 \,, \forall k \in \{1,2,\ldots q\}.$$

In order to relate the solution of the Linear Programming problem to the minimum of the scalar function P we note that P represents a global penalty function which takes care of the p inequality constraints implicitly . This function consists of the objective function Φ plus a sequence of quadratic penalty functions for constraint violations so that the solution of the new unconstrained problem, formulated in (a.8), approaches that of the original constrained problem (a.1-a.3), provided that the slope of the resistor characteristic in the third quadrant of the v-i plane (figure a.2) is sufficiently high.

A.2 Network Implementation

Kennedy and Chua [9] proposed a circuit implementation to the Linear Programming circuit of figure a.1, using solid-state devices, as shown in figure a.3.

The circuit implementation shown in figure a.3 is a special case of the circuit shown in figure 2.6, which was proposed to implement the more general Linear Programming problem. To see how the network of figure a.3 actually implements the Linear Programming model shown in figure a.1 , consider an arbitrary neuron. Its input current is:

$$I_k = A_k + \sum_{j=1}^{p} B_{jk} O_j \qquad (a.14.1)$$

$$= A_k + \sum_{j=1}^{p} B_{jk} g_j(f_j(\vec{V})) \qquad (a.14.2)$$

$$= A_k + \sum_{j=1}^{p} B_{jk} g_j(B_j \cdot \vec{V} - E_j) \qquad (a.14.3)$$

Its terminal equation is given by :

$$I_k = -C_k \frac{dV_k}{dt} \qquad (a.15)$$

hence:

$$C_k \frac{dV_k}{dt} = -A_k - \sum_{j=1}^{p} B_{jk} g_j(\vec{B_j} \cdot \vec{V} - E_j) \qquad (a.16)$$

Equation a.16 is readily identified as the circuit equation (a.7) of the Linear Programming circuit to be implemented.

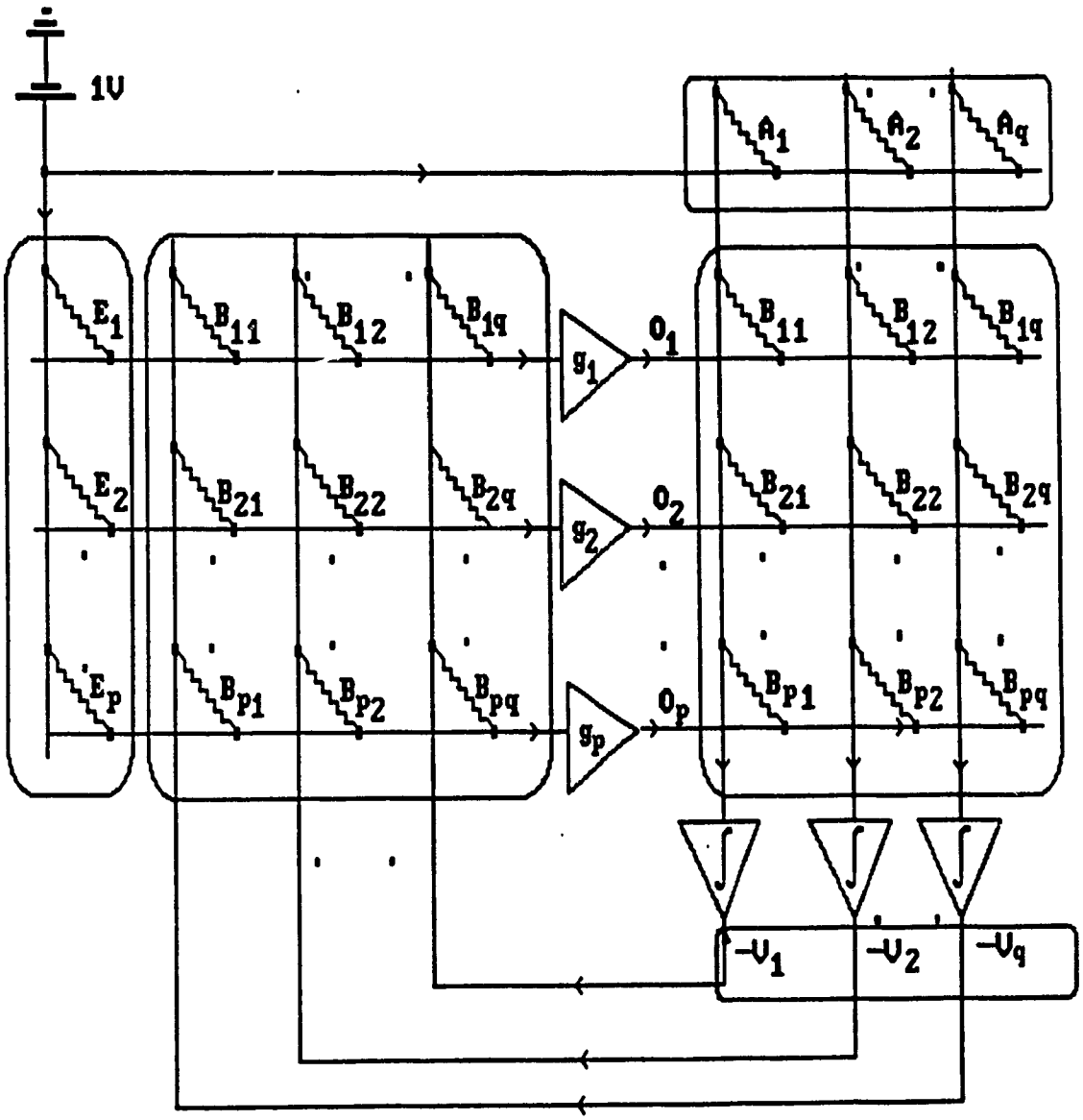


Figure a.3. A Circuit Implementation of Chua and Lin Linear Programming Network