NEURAL NETWORK MODELS FOR OPTIMAL ROUTING IN COMPUTER NETWORKS

A DISSERTATION

submitted in partial fulfilment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE AND TECHNOLOGY

By

ACHUTA DEVI VENKATA KUMAR

DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
UNIVERSITY OF ROORKEE
ROORKEE-247 667 (INDIA)

FEBRUARY, 1996

CANDIDATE'S DECLARATION

I hereby declare that the work which is being presented in the dissertation entitled **"NEURAL NETWORK MODELS FOR OPTIMAL ROUTING IN COMPUTER NETWORKS"**, in partial fulfil ment of the requirement for the award of the degree of **MASTER OF TECHNOLOGY** with specialization in **COMPUTER SCIENCE AND TECHNOLOGY**, University of Roorkee, Roorkee, is an authentic record of my own work carried out from July 1995 to February 1996 under the guidance of **Dr. (Mrs.) KUMKUM GARG**, Professor, Department of Electronics and Computer Engineering, University of Roorkee, Roorkee.

The matter embodied in this dissertation has not been submitted by me for other degree.

[illegible]

Date : 12th Feb. 1996 **(ACHUTA DEVI VENKATA KUMAR)**

Place : Roorkee

This is to certify that the above statement made by the candidate is correct to the best of my knowledge and belief.

[illegible]

Date : 12. 2. 96

Place : Roorkee

(Dr. KUMKUM GARG)
Professor
Deptt. of Electronics & computer Engg.
University of Roorkee
Roorkee - 247 667 (INDIA)

ACKNOWLEDGEMENT

I am taking this opportunity to express my sincere and heartful gratitude to **Dr. (Mrs.) KUMKUM GARG,** Professor, Department of Electronics and Computer Engineering, University of Roorkee, Roorkee, for her valuable guidance and sustained encouragement through out the dissertation period.

I also owe a lot to my dearest friends for their suggestions and help during the dissertation period.

[illegible]

Date : 12th Feb, 1996

(ACHUTA DEVI VENKATA KUMAR)

Place : Roorkee

ABSTRACT

The routing of packets from source to destination is an important issue in the design of packet-switched computer networks, where the goal is to minimize the network wide average time delay. The routing algorithms rely heavily on the shortest path computations that have to be carried out in realtime.

This dissertation addresses the application of neural networks to the optimal routing problem. Three neural network models are compared. Their performance in giving optimal routes is analysed through simulation results by selecting three different communication network topologies. The neural network models compared are Lee-Chang model, Zhang-Thomopoulos model and Mustafa-Faouzi model, all based on Hopfield neural networks.

All-through Lee-change model gives multiple optimal, suboptimal routes simultaneously it is not fool proof in giving all optimal routes. But Mustafa-Faouzi model is found to be giving all optimal routes. The performance of these models in finding the multiple optimal routes simultaneously and the conditions there in are analysed through simulation results. Other factors like divergence problems, computational power requirement have also been examined.

CONTENTS

CHAPTER - 1
INTRODUCTION

1.1 ROUTING IN COMPUTER NETWORKS

The routing of packets from source node to destination node is an important issue in the design of communication network consisting of multiple nodes and links, because it affects several performance measures of interest. The objective of the routing algorithm is to optimize some performance measure such as mean packet delay or network throughput.

Routing can be done in a centralized, distributed or localized manner [1]. In centralized algorithms, all route choices are made at a central node, while in distributed algorithms the computation of routes is shared among the network nodes with information exchanged between the nodes as necessary. In localized routing algorithms, each node needs to have the most current network connectivity and computes the routes to all possible destination nodes based on connectivity information. In order to have the most current network connectivity, all network nodes broadcast their connectivity to neighboring nodes.

Centralized routing method requires a special node in the network which periodically receives information from all other network nodes and based on this global information, it sets up and updates routing tables for all nodes. This method requires high computational facilities at central control node and also high reliability of central control node since failure of the central control node results in the shutdown of the entire network.

The distributed routing approach can reduce some problems in centralized routing. In this case each node makes its own routing decisions based on the local information it receives from its neighboring nodes. Looping of packets and deadlocks might occur due to inconsistent routing paths.

In localized routing algorithm, all network nodes broadcast their network connectivity to neighbors nodes, so that each node can react quickly to changes in the network, but does incur the communication cost of broadcasting such changes.

Either centralized, distributed or localized routing algorithms can be operated in static or adaptive manner. In static routing algorithms, path used between each origin destination pair is fixed regardless of traffic conditions and network changes. In adaptive routing algorithms, the paths used to route new message between origin and destination change occasionally in response to the traffic conditions and network changes, i.e. failed links, increase or decrease of link cost.

1.2 WHY NEURAL NETWORKS FOR ROUTING

Neural networks are parallel distributed information processing systems that consists of non linear processing elements and weighted connections [2]. Each layer in a neural network consists of a collection of processing elements. Each processing element collects the value from all of its input connections, performing a predefined mathematical operation and produces a single output value.

The motivation for the neural networks is from natural neural systems. Human information processing system is composed of many neurons switching at speeds about a million times slower than computer gates. Yet humans are more efficient than computers at computationally complex tasks such as speech understanding, visual recognition etc. Neural networks are designed to exploit the unique computational power of human brain - parallel distributed nature of processing. Neural networks offer interesting alternative solutions to many problems. Routing in computer networks is one such area.

There are several routing algorithms with different levels of sophistication and efficiency. The optimality of routing algorithm is a relative attribute which usually implies efficient use of network resources so as to optimize a performance measure [Eg. finding optimal paths for data transmission within a short time so as to satisfy users demand for a faster service]. This requires shortest path computations involved in the routing problem to be carried out in real time. Neural networks are very good candidates for implementing shortest path computations involved in routing problem because of potential of the neural network hardware approach [3] for high computational speed.

1.3 STATEMENT OF THE WORK

Given the network topology, and link capacity information, optimal routing problem requires finding all possible optimal paths of each source-destination pair of the network that will minimize the network wide average delay.

This work addresses the application of neural networks to the optimal routing problem. Three neural network models are compared w.r.t. the quality of solution provided by them. Three different network topologies are selected for this purpose. Their abilities in finding multiple optimal paths simultaneously is also analysed through simulation results.

1.4 ORGANIZATION OF THE THESIS WORK

The first chapter provides an introduction to the routing problem involved in computer networks and discusses the need of neural networks for routing problem. Statement of the work is given after.

Second chapter deals with the Hopfield neural networks and their computational power demonstration. Artificial neural network and feedback neural network model are briefly discussed in this chapter as they are the fundamentals in Hopfield neural networks.

Third chapter explores the neural network models for routing, and the advantages and disadvantages of these models. Fourth chapter discusses the software implementation of the neural network models.

Finally in the fifth chapter the results obtained from simulation are discussed, concluding remarks for the work are simultaneously given. Some suggestions for further work are also given after.

CHAPTER - 2
NEURAL NETWORKS

This chapter briefly discusses the details of artificial neural networks, classification of neural networks, and continuous time feedback neural network model. Hopfield neural networks are single layer feedback network models employing batch learning. Their computational power was first demonstrated by Hopfield and Tank [4] by applying it to Travelling salesman problem (TSP). In fact, this demonstration is the motivation behind the neural network models for routing. The details of Hopfield neural network and its application to TSP are discussed in subsequent sections of this chapter.

2.1 NEURON MODELING FOR ARTIFICIAL NEURAL SYSTEM

Every neuron model consists of a processing element with synoptic input connections and a single output. This single output is copied into all the outgoing connections of the neuron. The signal flow of neuron inputs, x_i is considered to be unidirectional as indicated by arrows, as a neuron's output signal flow. A general neuron symbol is shown in fig. 2.1. This symbolic representation shows a set of weights and the neuron's processing unit. The neuron output signal is given by the following relationship.

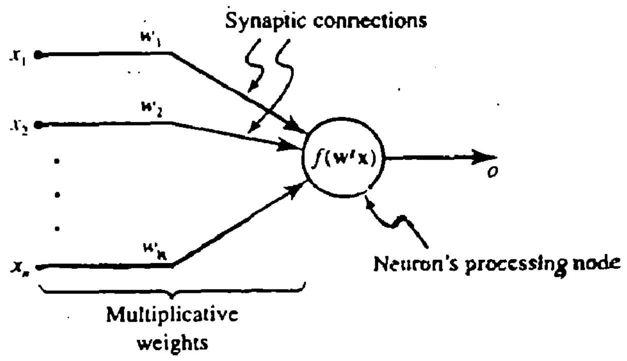


Fig. 2.1 General symbol of neuron

$$O = f(W^T X) = f\left[\sum_{i=1}^{n} w_i\ x_i\right] \qquad (2.1)$$

where, W is the weight vector defined as

$$W = [W_1\ W_2\ \dots\ W_n]^T$$

and X is the input vector defined as

$$X = [x_1\ x_2\ \dots\ x_n]^T$$

[Subscript T denotes transposition]. The function $f(W^T X)$ is referred to as an activation function. Its domain is the set of activation values, *net*, of the neuron model. The variable *net* is defined as a scalar product of the weight and input vector.

$$net = W^T X \qquad (2.2)$$

It is clear from (2.1) that the neuron or a processing node performs the operation of summation of its weight inputs to obtain *net*. Subsequently it performs the nonlinear operation f(*net*) through its activation function. Typical activation function used is

$$f(net) = \frac{2}{1 + \exp(-\lambda \, net)} - 1 \qquad (2.3)$$

Where $\lambda > 0$ in (2.3) is proportional to the neuron gain determining the steepness of the continuous function f(*net*) near *net* = 0. The continuous activation function is shown in the fig. 2.2, for various λ. Activation function (2.3) is called bipolar continuous function.

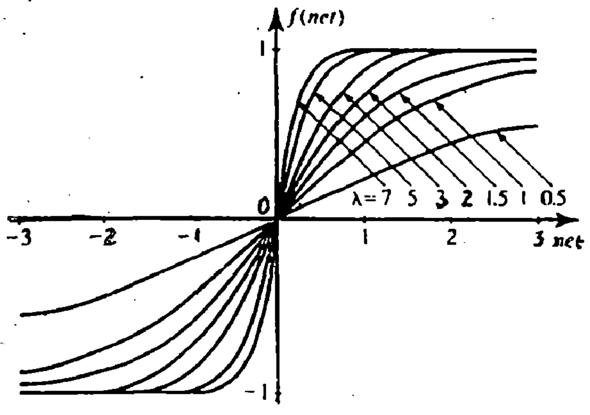


Fig.2.2 Bipolar continuous activation functions of neuron

By shifting and scaling the bipolar activation function defined by (2.3), unipolar continuous functions can be obtained as :

$$f(net) = \frac{1}{1 + \exp(-\lambda \, net)} \qquad (2.4)$$

Given a layer of m neurons, their output values $O_1, O_2, \ldots O_m$ can be arranged in a layer's output vector :

$$O = [\, O_1 \; O_2 \quad \ldots \; O_m]^T \tag{2.5}$$

Where O_i is the output signal of the *i*th neuron. The domains of vector O are defined in m-dimensional space as follows for i = 1,2, ... m.

$$(-1,\ 1)^m = \{O \in R^m\ ,\ O_i \in (-1,1)\} \tag{2.6}$$

or

$$(O,\ 1)^m = \{O \in R^m\ ,\ O_i \in (O,1)\} \tag{2.7}$$

for bipolar and unipolar continuous activations defined as in (2.3) and (2.4) respectively. It is evident that the domain of vector O is the interior of either m-dimensional cube $(-1,1)^m$ or of the cube $(O,1)^m$.

Now the artificial neural network can be defined as an inter connection of neurons as defined in (2.1) through (2.4) such that neuron outputs are connected through weights, to all other neurons including themselves.

2.2 CLASSIFICATION OF NEURAL NETWORKS

Broadly speaking, neural nets can be classified in to two types, feed forward networks, and feedback networks. In feedforward networks, recall of information is performed in the feedforward mode or from input towards output only. Such networks have no memory. Recall in such networks is instantaneous. Thus past time conditions are irrelevant, for their computation. Network responds only to its present input. Feedback network models perform recall computation with feedback operational. These networks are considered as dynamical systems and a certain time interval is needed for their recall to be computed. Feedback networks are also called recurrent. They interact with their input through the output.

Another meaning-ful basis for classification is to differentiate neural networks by their learning mode. Learning in all neural networks fall into three groups, supervised, unsupervised and batch. Learning is necessary when the information about relationship between input and output is unknown or incomplete, *a priori* so that no design of a network can be performed in advance and network requires training in a particular learning mode. In supervised learning at each instant of time when the input is applied the desired response of the system is provided by the supervisor/environment. The distance between actual and desired response results as an error measure and is used to correct network parameters (weights) externally. In unsupervised mode learning must be accomplished based on the observation of responses to inputs since the desired response is not known previously. In batch learning mode, all network weights are adjusted in a single training step. Here complete set of design data is needed to determine weights and feedback information is produced by the network it self is not involved in developing the network. This learning technique is also called recording.

2.3 FEEDBACK NEURAL NETWORK MODEL

Here all the neuron outputs are connected back to their inputs as shown in fig.2.3.

The essence of closing the feedback loops is to enable control of output O_i through output O_j, for j = 1,2,....m. Such control is especially meaningful if the present output say O(t) controls the output at the following instant O(t+Δ). The time Δ elapsed between t and t+Δ is introduced by the delay elements in the feedback loop.

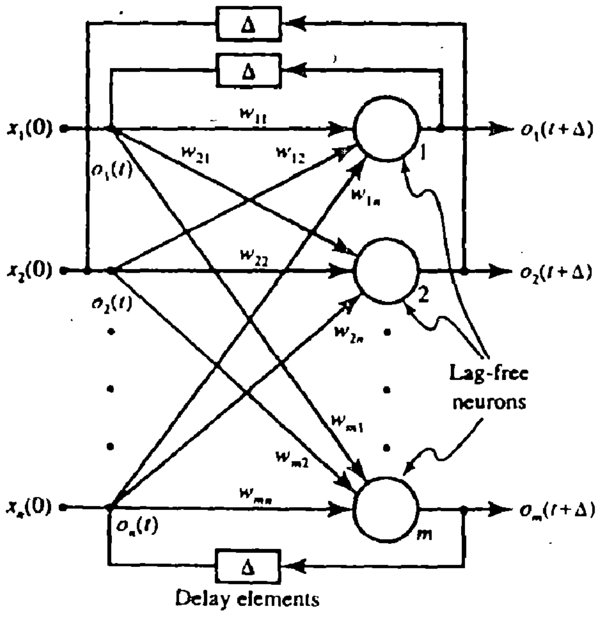


Fig 2.3 Single-layer feed back network interconnection scheme

The mapping of O(t) into O(t+Δ) can be written as :

$$O(t+\Delta) = \Gamma[W\ O(t)] \qquad (2.8)$$

where W is an n x n weight matrix also called connection matrix :

$$W = \begin{bmatrix} w_{11} & w_{2} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ w_{m1} & w_{m2} & & w_{mn} \end{bmatrix} \qquad (2.9)$$

and Γ is a non linear matrix operator :

$$\Gamma[\cdot] = \begin{bmatrix} f(\cdot) & 0 & \cdots & 0 \\ 0 & f(\cdot) & \cdots & 0 \\ \cdot & \cdot & & \cdot \\ \cdot & \cdot & & \cdot \\ 0 & 0 & & f(\cdot) \end{bmatrix} \qquad (2.10)$$

Here nonlinear activations $f(\cdot)$ on the diagonal of the matrix operator Γ operate component wise on the activation values *net* of each neuron.

The input X(t) is only needed to initialize this network so that O(0) = X(0), after that input is removed and the system remains autonomous for t>0. If the feedback concept is implemented with any infinitesimal delay between output and input introduced in the feedback loop then the output vector can be considered to be a continuous time function. As a result entire network operates in continuous time. An example of one such elementary delay network is shown in fig. 2.4.

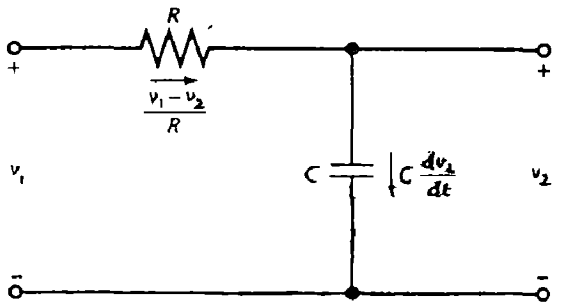


Fig. 2.4 Feedback Connection in continuous time network

It is an analogy of a simple electric network consisting of resistance and capacitance. Here the differential equation relating v_2 and v_1 the output and input voltage respectively is

$$\frac{dv_2}{dt} = \frac{v_1}{RC} - \frac{v_2}{RC} \tag{2.11}$$

Continuous time networks employ neurons with continuous activation functions. An elementary synaptic connection using delay network given in fig. 2.4 is shown in figure 2.5.

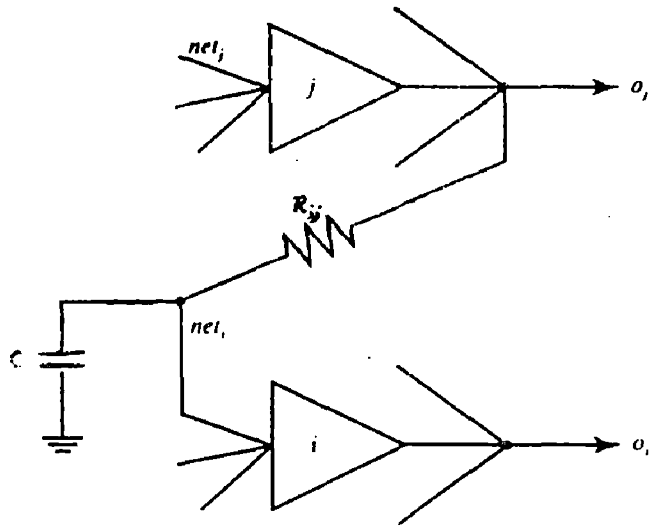


Fig. 2.5 Elementary synaptic connection in continuous-time network.

The resistance R_{ij} serves as a weight from the output of the *j*th neuron to the input of the *i*th neuron using finite time interval Δt, equation (2.12) can be discretized as

$$\frac{net_i^{k+1} - net_i^k}{\Delta t} = \frac{1}{R_{ij}C_i} (O_i^k - net_i^k) \tag{2.12}$$

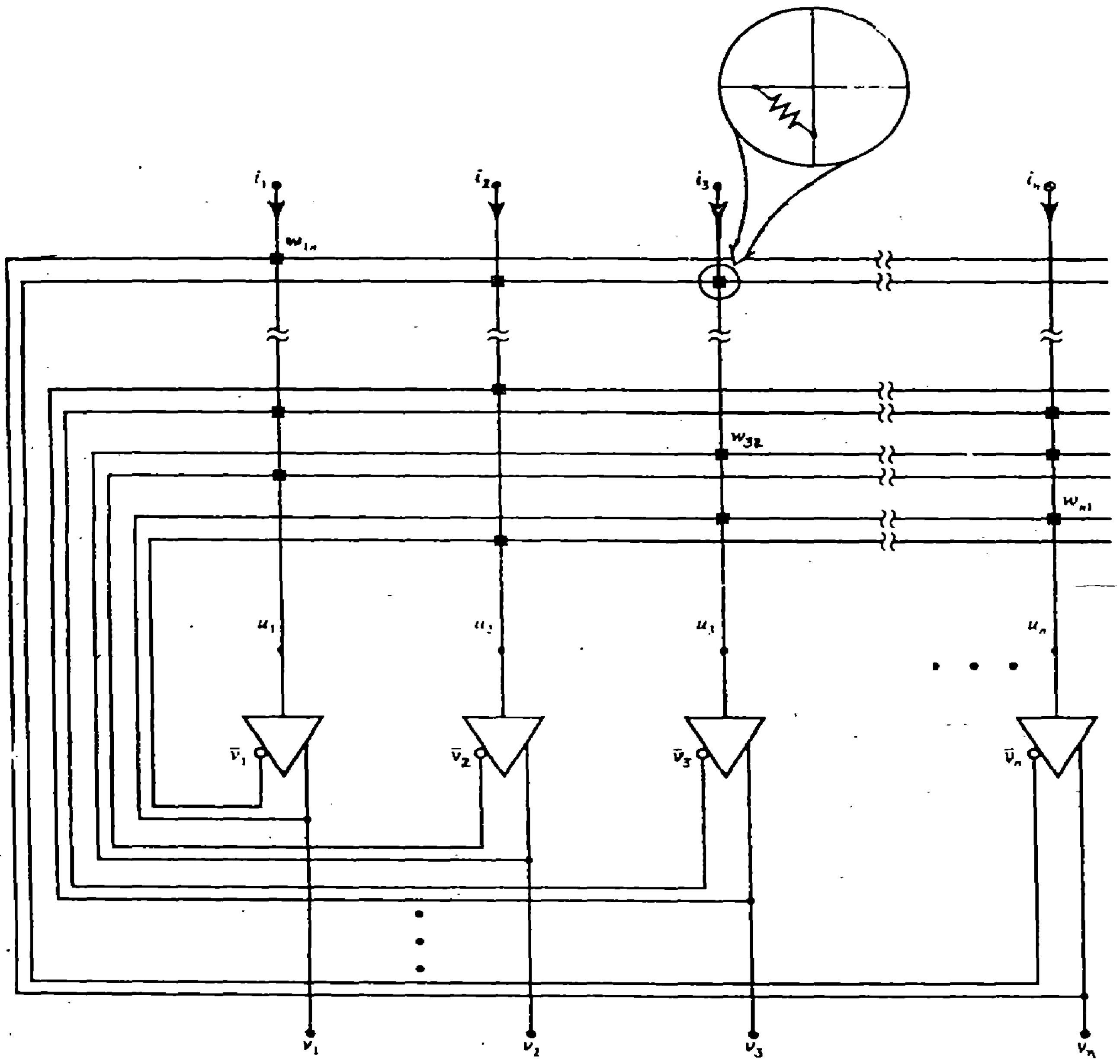


Fig.2.6 Gradient type Hopfield Neural Network

The activation of *i*th neuron at the instant k + 1 can be expressed as:

$$net_i^{k+1} = net_i^k + \frac{\Delta t}{R_{ij}C_i}(O_j^k - net_i^k) \qquad (2.13)$$

The contribution to net_i by *j*th neuron is distributed in time according to (2.13). When n neurons are connected to the input of *i*th neuron as shown in fig. (2.5) expression (2.13) needs to be computed for j=1,2...,n and summed.

2.4 HOPFIELD NEURAL NETWORK

Continuous time single layer feedback networks also called Gradient type networks are generalized Hopfield networks in which the computational energy decreases continuously in time.

Gradient type networks converge to one of the stable minima in the state space. The evolution of the system is in the general direction of negative gradient of the of an energy function. Typically network energy function is made equivalent to a certain objective (penalty) function that needs to be minimized.The search for an energy minimum performed by gradient type network corresponds to the search for a solution of optimization problem.

The single layer feedback networks can be modeled by a physical system. This modeling provides the link between the theory and implementation. The model of a gradient-type neural system using electrical components is shown in the fig.2.6. It consists of a neurons, each mapping its input voltage u_i in to output voltage v_i through the activation function $f(u_i)$, which is common static voltage transfer characteristic (VTC) of the neuron. Any high gain voltage amplifier with saturation could be used in this model as a replacement for a neuron. Conductance w_{ij} connects the output of the *j*th neuron to the input of *i*th neuron. The inverted neuron outputs $\bar{v}_i$ are usually tapped at inverting rather than noninverting output to avoid negative conductance values w_{ij} connecting in inhibitory mode, the output of *j*th neuron to the input of *i*th neuron. This network is required to be symmetric $w_{ij} = w_{ji}$ i.e. the outputs of neurons are not connected back to their own inputs. Each neuron receives an external current (known also as a bias I_i). Now denoting synaptic connection matrix $[w_{ij}]$ by W. The neuron dynamics [3] of Hopfield network are described by

$$\frac{du_i}{dt} = \sum_{j=1}^{n} w_{ij} v_i - \frac{v_i}{\tau} + I_i \qquad (2.14)$$

where τ is circuit time constant.

For a synaptic connection matrix W, if the gains of the neurons are sufficiently high then the dynamics of neurons follow a gradient descent of the quadratic energy function

$$E = -\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij} v_i v_j - \sum_{i=1}^{n} I_i v_i \qquad (2.15)$$

Also, while the state of the neural network evolves inside the N-dimensional Hypercube defined by $v_i \in \{0,1\}$, the minima of the energy function (2.15) occurs at 2^n corners of this space, only if gain of the amplifiers is very high. In terms of the energy function (2.15), the dynamics of *i*th neuron are described by

$$\frac{du_i}{dt} = -\frac{u_i}{\tau} - \frac{\partial E}{\partial v_i} \qquad (2.16)$$

2.5 TRAVELLING SALESMAN TOUR LENGTH MINIMIZATION

For solving TSP a suitable objective function has to be formulated to substitute the energy function.

Travelling salesman problem is minimization of tour length through a number of cites with only visit in each city. The network consisting of n unipolar continuous neurons arranged in an n x n array [matrix], where *i*th row in the n x n matrix corresponds to a city Y_j, and *j*th column in the matrix corresponds to a city position X_i, can be used, to solve the TSP . Thus, there will be city rows and position columns. Since each city can only be visited once and no simultaneous visits are allowed, solution matrix can contain only a single 1 in each column and a single 1 in each row. The neuron turned on or with output 1 in the square array of neurons indicates a particular position of a particular city in the tour.

The energy (objective) function to solve the problem is given as follows :

$$E = A \sum_{X=1}^{n} \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} v_{Xi} v_{Xj} + B \sum_{i=1}^{n} \sum_{X=1}^{n} \sum_{\substack{Y=1 \\ Y \neq X}}^{n} v_{Xi} v_{Yi}$$

$$+ C \left[\sum_{X=1}^{n} \sum_{i=1}^{n} v_{Xi} - n \right]^2 +$$

$$D \sum_{X=1}^{n} \sum_{\substack{Y=1 \\ Y \neq X}}^{n} \sum_{i=1}^{n} d_{XY} v_{Xi} (v_{Y,i+1} + v_{Y,i-1}) \qquad (2.17)$$

The A term becomes O if the matrix does not contain more than one 1 in each row. Similarly the B term restricts the number 1 's in each column to 1. C term is required to ensure of that the matrix simply does not contain all zeros. D term takes into account, the true goal - tour optimization. The distances between the adjacent cities are summed while computing the term, and summations are to be as modulo n. So the D term is numerically equal to the length of the path of the tour.

The resulting weight matrix and bias terms can be obtained by equating (2.16) with (2.15). The weights and bias currents given as follows :

$$w_{Xi,Yj} = -2\,A\ \delta_{XY}(1 - \delta_{ij}) - 2\,B\,\delta_{ij}\,(\,1 - \delta_{XY}\,) - 2\,C$$

$$- 2\,D\,d_{XY}\,(\delta_{i,j+1} + \delta_{j,i-1})$$

Where δ_{ij} is kronecker delta function defined as

$\delta_{ij} = 1$, for $i = j$ and $\delta_{ij} = 0$, for $i \neq j$.

and $\quad I_{Xi} = 2\,C\,N$ (2.18)

using the weight and bias currents (2.18) and equation (2.14) a system of non linear differential equations can be solved for minimization of tour length in TSP problem.

CHAPTER - 3
NEURAL NETWORK MODELS

3.1 LEE- CHANG MODEL[5],[6]

Symbols and definitions used in the model are as follows :

s : source node ,

d : destination node ,

h : maximum number of links of optimal paths from s to d ,

n : number of nodes in the network topology,

Control vector $U_k = [u_1^k \; u_2^k \; .... \; u_n^k]^T$,

where u_i^k stands for the ratio of the traffic of node i, in the *k*th position of the source to destination path. u_i^k is always between 0 and 1. If u_i^k equals 1, it means that all the traffic will be concentrated on node i. If the elements of U_k have all zeros except at certain element i, then it represents the fact that the *k*th position of the path is to be via node i.

The state is said to convergent iff given a small tolerance $\epsilon > 0$, the update of traffic remains within this ϵ-neighborhood, i.e. there are only small changes in the states of the network. If there is only one source-to-destination path then one element of U_k will converge to 1 and other elements to 0. On the other hand if there is more than one optimal source-to-destination path, then the elements of U_k will be convergent to the same values for several different nodes.

The model is arranged in multiple layers, and the number of layers is equal to the maximum number of nodes in the routing path. The connection between successive layers is dependent on the connectivity of communication network.

Neurons in the same layer are not independent. The weight of links between layers are fixed in this model. The first layer stands for source node of the path and the last layer stands for the destination node. So, the values of neurons in the first and last layers are fixed. The output values of neurons in the intermediate layers are obtained by learning. During the training, each layer gets a forward correction from the lower layer and backward correction from upper layer and self correction among neurons in the same layer.

The energy function of this model is given by

$$E = \frac{1}{2} \sum_{j=1}^{h} U_j^T W U_{j+1} + (\gamma)(\tfrac{1}{2}) \sum_{j=2}^{h} \left[\sum_i u_i^{j-1} \right]^2 \qquad (3.1)$$

The first term of energy function is the delay time of network. The nxn delay time weighting matrix W is formed by collecting the corresponding delays from each node to every other in the network. The second term restricts the sum of u_i^j $(i=1,2,\ldots,n)$ in U_i, constrained to be close to 1. The weighing factor γ which is a positive constant can be adjusted properly so that the values of U_j will converge quickly. To minimize energy function, differentiating w.r.t. U_k

$$\frac{\partial E}{\partial U_k} = \left(\frac{1}{2}\right)\left[WU_{k-1} + WU_{k+1} + 2\gamma e_n \left(\sum_i u_i^k - 1\right)\right] \tag{3.2}$$

where e_n is an n by 1 vector with all 1's. Because the gradient is in the direction of maximal change, the vector ΔU_k is set to be proportional to $-\partial E/\partial U_k$ with proportional constant $\alpha(>0)$:

$$\Delta U_k = -\alpha \left(\frac{1}{2}\right)\left[WU_{k-1} + WU_{k+1}\right] + \alpha\gamma e_n \left(1 - \sum_i u_i^k\right)$$

$$k = 2,3,\ldots,n \tag{3.3}$$

$U_k(i)$ is defined as the value of U_k after the *i*th iteration. If $i=0$, initial values of vectors $U_k(0)$'s $(k = 1,2,\ldots, h+1)$ have to be assigned. Since, each layer has different properties, the initial assignment is divided into five cases as follows :

Case-1 : When k equals 1, U_1 is the first vector in the path. If node s is picked to be the source node, then *s*th element of U_1 is set to 1, and other elements to 0.

Case-2 : When k equals h+1, U_{h+1} is the last vector in the path. If d is the destination node the d^th element of U_{h+1} is set to 1, and other elements to 0.

Case-3 : When k equals 2, U_2 is the second vector in the path. The values of nodes which are connected to the source node are assigned to be 0. The values of u_i^2 which are not connected to the source node are assigned to be 1 divided by number of links connected to it.

Case-4 : When k equals h, values of U_h are assigned similarly as in case-3.

Case-5 : When k is not in the four classes above i.e. k = 3,....,h-1. The values of every element in U_k said to be same except for source and destination nodes. Since loops are not allowed in this path, the values for source and destination are set to 0 i.e.

$$U_i^k = \begin{cases} 0 & \text{if } i = s,d \\ 1/n & \text{otherwise} \end{cases}$$

After the initialization, U_k's are updated according to the equation.

$$U_k(i+1) = U_k(i) + \Delta U_k \;,\; k = 2, 3, \ldots, h.$$

This procedure continues until vectors of U_k satisfy the convergence criterion. This network model always converges to a stable solution. Proof is given in Appendix II. The rate of convergence will be improved if the parameters α and γ, in (3.1) are chosen carefully. The information for reasonable parameters calculation is also given Appendix II.

3.2 ZHANG - THOMOPOULOS MODEL

The neural network model is arranged in a two dimensional array of size nxn, where n is the total number of nodes in the topology of the network. The output v_{xi} of neuron at location (x,i) is defined as follows :

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

The energy function, whose minimization process, moves the neural network to the stable state (corresponding to the solution) is given by

$$E = \frac{A}{2} \sum_{k=1}^{n-1} \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ik}\, w_{ij}\, v_{j,k+1} + \frac{B}{2} \sum_{k=1}^{n} \sum_{i=1}^{n} \sum_{j=1}^{n} v_{ik}\, v_{jk}$$

$$+ \frac{C}{2} \left[\sum_{i=1}^{n} \sum_{j=1}^{n} v_{ij} - n \right]^2 \qquad (3.4)$$

Where w_{ij} consists of zero cost self loops connecting each node to itself and very large costs to nonexisting links and proper costs to corresponding existing links.

In the energy function, A term represents the total cost of the path from source to destination. The B and C terms are constraints introduced to force the neural network to coverage 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. Here in this model, for a given source and destination pair (s,d) state of all neurons located in the first and last column are fixed [$v_{s1} = v_{dn} = 1$ and the remaining neuron in the first and last column are set to 0], while allowing the output values of remaining neurons to evolve so as to minimize the energy function.

The state of $(i,j)^{th}$ neuron, u_{ij} can be described by the differential equation :

$$\frac{du_{ij}}{dt} = -\frac{u_{ij}}{\tau} + \sum_{l=1}^{n} \sum_{m=1}^{n} T_{ij,lm} \cdot v_{lm} + I_{ij} \qquad (3.5)$$

where, $T_{ij,mn} = -A\ W_{im}(\delta_{n,i+1} + \delta_{n,i-1}) - B\ \delta_{in}(1 - \delta_{im}) - C$ is the connection weight between $(i,j)^{th}$ neuron and $(m,n)^{th}$ neuron in the nxn neuron array obtained by comparing the corresponding coefficients in (3.4) and (2.15).

Here δ_{ij} is the Kronecker delta function defined as before, also

$v_{ij} = g(u_{ij}) = [1 + \tanh(u_{ij}/u0)]/2$,

$I_{ij} = C * N$ (input bias term),

$u0$ = gain factor.

Energy minimization procedure involves solving n^2 nonlinear differential equations. This procedure continues until output of each v_{ij} approaches either 0 or 1, which corresponds to the steady state (it could be a local minimum).

3.3 MUSTAFA - FAOUZI MODEL [8]

This model gives a suitable representation scheme, such that the shortest path is encoded in the final state of the neural network. The model is organized in an nxn matrix, with all diagonal elements removed since they are not needed. Each element in the matrix is represented by a neuron which is described by double indices (x,i) where row subscript x and column subscript i denote the node numbers. Therefore, the neural network requires n(n-1) neurons and a neuron at location (x,i) is characterized by its output v_{xi}, defined as follows:

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

Also ρ_{xi} is defined as

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

In addition, the cost of an arc from node x to node i will be denoted by w_{xi}, a finite positive number. For nonexisting arcs this cost is zero.

The suitable energy function, whose minimization process drives the neural network into its lowest energy state (corresponding to the shortest path) is given as follows :

$$E = \frac{\mu_1}{2} \sum_{x=1}^{n} \sum_{\substack{i=1 \\ i \neq x \\ (x,i) \neq (d,s)}}^{n} w_{xi} \cdot v_{xi} + \frac{\mu_2}{2} \sum_{x=1}^{n} \sum_{\substack{i=1 \\ i \neq x \\ (x,i) \neq (d,s)}}^{n} \rho_{xi} \cdot v_{xi} +$$

$$\frac{\mu_3}{2} \sum_{x=1}^{n} \left\{ \sum_{\substack{i=1 \\ i \neq x}}^{n} v_{xi} - \sum_{\substack{i=1 \\ i \neq x}}^{n} v_{xi} \right\}^2 +$$

$$\frac{\mu_4}{2} \sum_{x=1}^{n} \sum_{\substack{x=1 \\ x \neq i}}^{n} v_{xi}(1-v_{xi}) + \frac{\mu_5}{2}(1-v_{ds}) \qquad (3.6)$$

The μ_1 term minimizes the total cost of path by taking into account the cost of existing links. The μ_2 term prevents the nonexisting links being included in the chosen path. The μ_3 term is zero for every node in the solution, if the number of incoming arcs equals the number of outgoing arcs. This makes sure that if a node is entered in the solution path, it will also be exited by a path. The μ_4 term pushes the state of the neural network to converge to one of the 2^{n^2-n} corners of the Hypercube defined by $V_{xi} \varepsilon \{0,1\}$. The μ_5 term is zero when the output of the neuron at location (d,s) settles to 1. Although the link from d to s is not part of the solution, it is introduced to enforce the construction of 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 will be in the solution.

The final solution will always be a loop, with nodes d and s 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_1 and μ_3 terms will ensure that there will be at most a single 1 at each row and at each column. This guarantees that there will be one to one relationship between the paths and the neural network representations.

Rewriting (2.14), (2.15) and (2.16) in such a way as to take into account the representation of neurons with double indices, we get

$$\frac{du_{xi}}{dt} = -\frac{u_{xi}}{\tau} + \sum_{y=1}^{n} \sum_{\substack{j=1 \\ j \neq y}}^{n} T_{xi,yj} \cdot v_{yj} + I_{xi} = -\frac{u_{xi}}{\tau} - \frac{\partial E}{\partial v_{xi}} \qquad (3.7)$$

By substituting (3.6) in (3.7), the equation of the motion of the neuron (x,i) is readily obtained :

$$\frac{du_{xi}}{dt} = -\frac{u_{xi}}{\tau} - \frac{\mu_1}{2} w_{xi}(1-\delta_{xd}\,\delta_{is}) - \frac{\mu_2}{2} \rho_{xi}(1-\delta_{xd}\,\delta_{is})$$

$$- \mu_3 \sum_{\substack{y=1 \\ y \neq x}}^{n} (v_{xy} - v_{yx}) + \mu_3 \sum_{\substack{y=1 \\ y \neq i}}^{n} (v_{iy} - v_{yi})$$

$$- \frac{\mu_4}{2} (1-2v_{xi}) + \frac{\mu_5}{2} \delta_{xd}\,\delta_{is}, \ \forall (x,i) \in n \times n \mid x \neq i \qquad (3.8)$$

where δ is the Kronecker delta defined as before .

By comparing the corresponding coefficients in (3.6) and (2.15) the connection strengths and biases can be derived. They are given as follows :

$$T_{xi,yj} = \mu_4 \delta_{xy} \delta_{ij} - \mu_3\delta_{xy} - \mu_3\delta_{ij} + \mu_3\delta_{jx} + \mu_3 \delta_{iy} \tag{3.9}$$

$$I_{xi} = - \frac{\mu_1}{2} w_{xi}(1-\delta_{xd} \delta_{is}) - \frac{\mu_2}{2} \rho_{xi}(1-\delta_{xd} \delta_{is})$$

$$- \frac{\mu_4}{2} + \frac{\mu_5}{2} \delta_{xd} \delta_{is} \tag{3.10}$$

The first term in (3.9) represents excitatory self-feedbacks, and the second and third terms represents local inhibitory connections among the neurons in the same row and in the same column, respectively. The last two terms represent excitatory cross-connections among neurons.

This neural network model maps the data represented by link costs and node connectivity information into biases rather than into neural interconnections. This is due to the fact that data terms are associated with linear rather than quadratic expressions in the energy function. Here the minimization corresponds to solving a system of n(n-1) nonlinear differential equations, where the variables are neurons output voltages v_{xi}'s. The efficiency of the model in solving the problem, requires selection of appropriate values of energy function coefficients. The general guidelines to select these coefficients are given in Appendix-II.

The neural network models discussed do not have specific names. In this work the models are referred to, with author names prefixed to them. The relative merits and demerits of these models are as follows :

The limitation of Lee-Chang model is that it is to be supplied with number of links between the source destination pair. Also every time, neural network configuration has to be initialized before starting it. This model performance is limited, only by its own drawback, but this model can be extended successfully to incorporate reliability of nodes and dependency between nodes in computing the routes. Appendix III discusses the details.

Zhang-Thomopoulos model is formulated to give optimal solution, but it inherits the inherent drawback in Hopfield network i.e. local minima problem. In this model also the values of neurons in first and last column have to be fixed before starting it. In this model, cost of links is reflected in connection weights between neurons. This becomes a problem, when model, is implemented in hardware because cost of links may change in real time.

In Mustafa Faouzi model formulation, cost of links is reflected in bias currents. This proves to be a big advantage for the model. But the parameters in this model have to be chosen very carefully.

CHAPTER - 4
DESIGN AND IMPLEMENTATION

The simulation program is written in 'C' language and run under UNIX environment on TATA ELXSI POWER SERIES 3200 SYSTEM. The operating system is IRIX version of UNIX.

The data structures and subroutines used in the simulation program are discussed in the following sections.

4.1 Simulation Program Data Structures

SCE : Specifies the source node

DST : Specifies the destination node

SIZE : Specifies total number of nodes in the communication network topology.

C[ ][ ]: is the link capacity matrix, each element C[i][j] represents the capacity of the link between node i & node j.

W[ ][ ],

c[ ][ ],

cost[ ][ ]: are the delay weighting matrices, each of them represents the delay on the links of the communication network.

Result :

This structure is used to store route(s) information for a given source-destination pair. The declaration of the structure is

```
struct res {
        int  *ROUTE [    ];
        int  RTCNT;
        int  RTLINKS [ ];
        float *RTVAL [   ];
        float PREF [    ];
        float RTCOST [   ];
        }
struct res result;
```

RTCNT : specifies the number of route(s) existing between a given source-destination pair.

ROUTE : is the list of routes found between a given source destination pair.

RTLINKS : specifies the number of links in the route(s)

RTCOST : specifies the cost of the route(s)

RTVAL : is used to store the output values of neurons representing the route(s).

PREF : specifies the preference of route(s).

The data structures used in Lee-Chang model are as follows:

lks : specifies the number of links between source and destination.

vec :

This data structure is used to represent the control vector. Its declaration is as follows:

```
Struct vect {
     float ele [ ];
          }
Struct vect     *vec[ ];
```

ele : specifies the value of a neuron in the control vector.

The data structures used in the Zhang-Thomopoulos model and Mustafa-Faouzi model respectively are as follows:

nnvec [ ][ ], no[ ][ ]	: are the neuron output arrays, each element specifies the output value of the neuron in a particular position in the neuron array.
coeff [ ][ ], ni[ ][ ]	: are the neuron input arrays, each element specifies the input activation value present at a particular neuron in the neuron array.
Y0, LAMDA	: Specifies the gain value of neurons.
DT,IT	: Specifies the difference interval between successive instants overwhich neuron output values are computed.

Routines

main ( ) : main function is used to select a particular neural network model depending on the settings of constants MODEL1, MODEL2, and MODEL3.

takeip ( ) : This function receives source, destination and link capacity matrix as inputs and computes the delays over links.

genrand ( ) : This function generates a random number between any number x $(1>x>0)$ and zero.

nnw_energy( ): This function computes energy associated with the neural network model .

The functions used for Lee-Chang model are as follows:

detlinks( ): This function determines the number of links between source and destination. The link capacity matrix is used for this purpose. Matrix C^m is checked whether element in source row and destination column is greater than zero, if so, m will be the minimum number of links between source and destination. Here C^m denotes, multiplication of C itself m times.

creatmem( ): This function creates the control vectors. The number of control vectors created are equal to lks plus 1.

initvect( ): This function initializes the control vectors, representing the neural network for a particular source and destination pair.

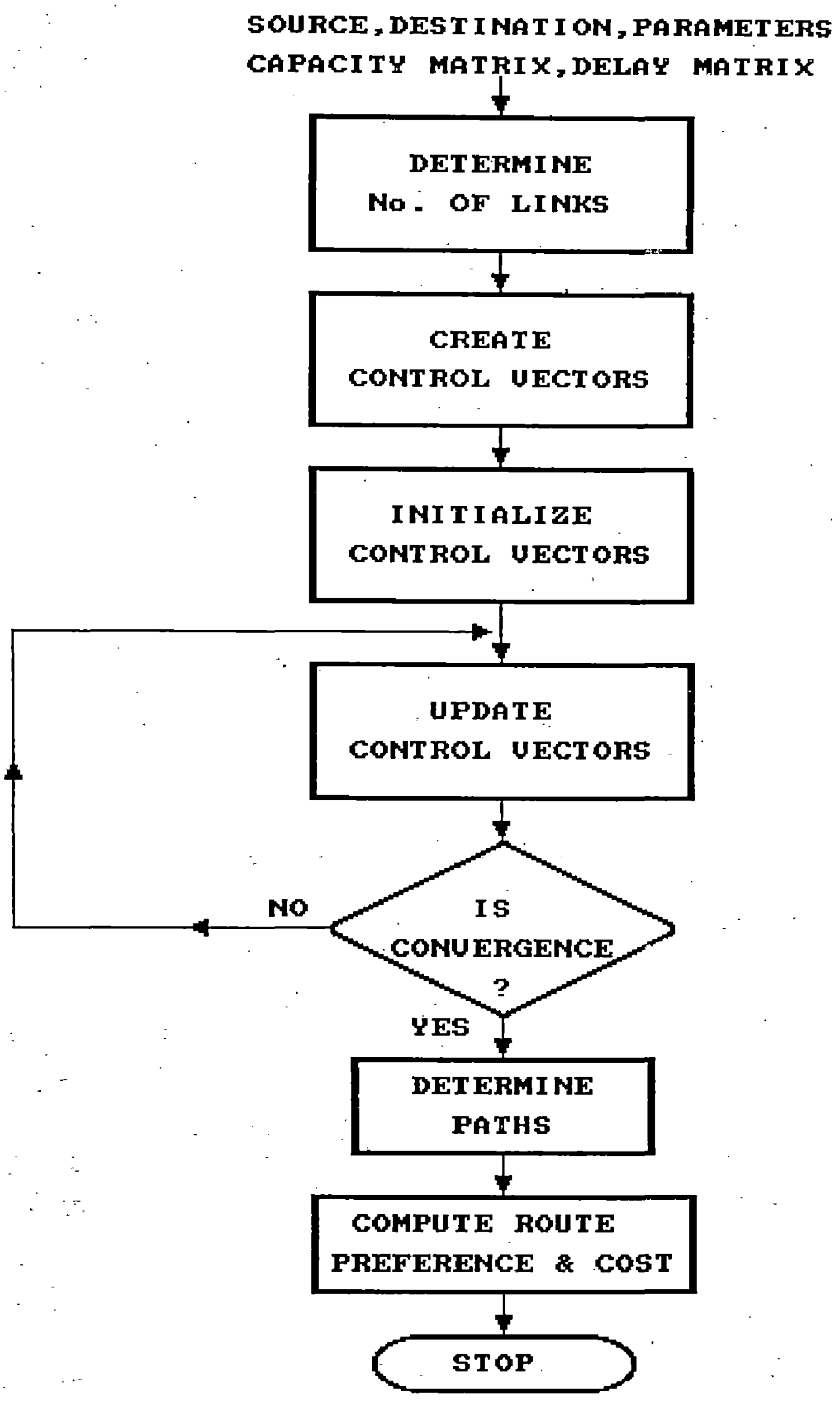

SOURCE,DESTINATION,PARAMETERS
CAPACITY MATRIX,DELAY MATRIX
DETERMINE No. OF LINKS
CREATE CONTROL VECTORS
INITIALIZE CONTROL VECTORS
UPDATE CONTROL VECTORS
IS CONVERGENCE ?
NO
YES
DETERMINE PATHS
COMPUTE ROUTE PREFERENCE & COST
STOP

Fig. 4.1 Flow chart for implementation of Lee-Chang model.

diffcalc( ): This function computes changes in control vector at successive iterations.

update( ) : This function uses diffcalc ( ) function to compute changes in control vectors to update them at successive iterations. The first and last control vectors are not updated because changes are not needed in them.

findroutes( ): This function determines the routes, from the informat-ion coded in the converged neural network. It generates proper combinations using position of elements in control vectors which are greater than a certain value to determine the routes.

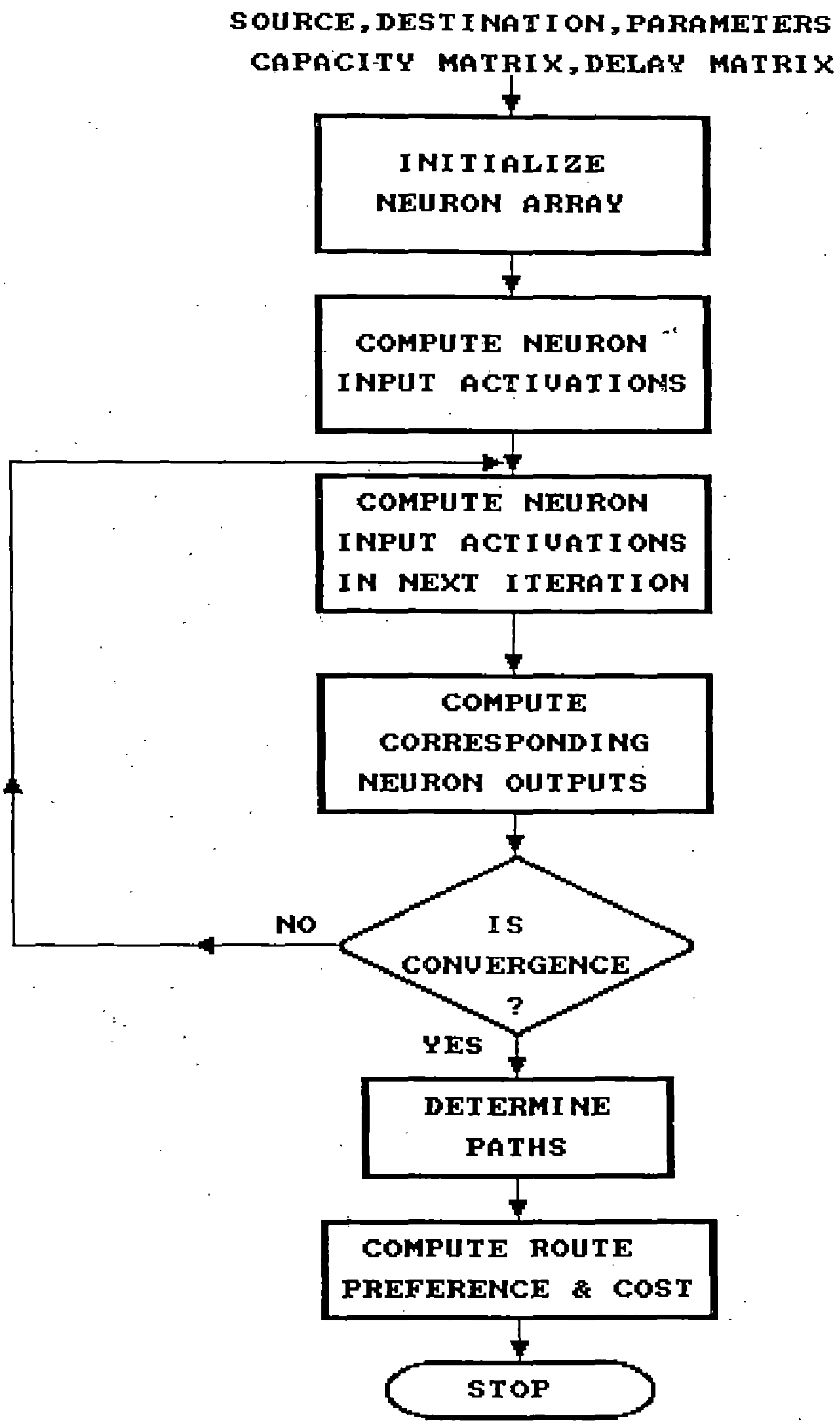
router1( ): This function performs the implementation of Lee-Chang model using the above functions of Lee-Chang model as shown in fig. 4.1.

The functions used in Zhang-Thomopoulos model are as follows:

initnnw ( ): This function initializes the output values of neuron array.

initcoeff( ):This function computes the input activation to the neuron corresponding to the initial output value of neurons.

cal-neuronop( ): This function computes the neuron output values corresponding to the input activations present at their inputs.



SOURCE,DESTINATION,PARAMETERS
CAPACITY MATRIX,DELAY MATRIX
INITIALIZE NEURON ARRAY
COMPUTE NEURON INPUT ACTIVATIONS
COMPUTE NEURON INPUT ACTIVATIONS IN NEXT ITERATION
COMPUTE CORRESPONDING NEURON OUTPUTS
IS CONVERGENCE ?
NO
YES
DETERMINE PATHS
COMPUTE ROUTE PREFERENCE & COST
STOP

Fig. 4.2 Flow chart for implementation of Mustafa-Faouzi and Zhang-Thomopoulos models.

doupdate( ): This function implements the relaxation algorithm to
compute neuron input activations at successive time instants.

findroute( ):This function picks out the route from source to
destination from the state of converged neural network.

ztrouter( ): This function performs the implementation of Zhang-Thomopoulos model using the above functions for Zhang-Thomopoulos model as shown in fig.4.2.

Functionsused for Mustafa-Faouzi model:

iccal( ) : This function computes the net input activation of neuron.

findpath( ):This function determines the route(s) from the
information encoded in the state of converge neural network.

rtfinder( ):This function performs the implementation of Mustafa-Faouzi model using the above function for Mustafa-Faouzi model as shown in fig.4.2.

pathpref( ):This function computes the preference and cost of
route(s) found between source- destination pair.

Paracal( ),

Paracheck( ):These functionscheck the validity of parameters chosen
for Lee-Chang model and Mustafa-Faouzi model respectively.

CHAPTER - 5
CONCLUSION

5.1 DISCUSSION OF RESULTS :

The three communication network topologies chosen are shown in figs. 5.1, 5.2 and 5.3. In figs 5.1 and 5.2 the number on the arcs indicates the link capacity. For simulation purpose link cost (delay)[1] is assumed to be inversely proportional to the link capacity. For communication network topology shown in fig. 5.3 the number on the arcs indicates the link cost chosen randomly..

The reason for selecting different sizes of communication networks is to bring out the divergence problem if there exists any. The reason for selecting the variable cost conditions is to compare the performance of Lee-Chang model and Mustafa-Faouzi model, in giving multiple optimal routes simultaneously.

The parameter set selected [9], [10] for each model is shown in table 5.1. How the link cost (delay) is chosen, in accordance with link capacity is also shown in table 5.1.

1: Functions for delay calculations are given in APPENDIX-IV

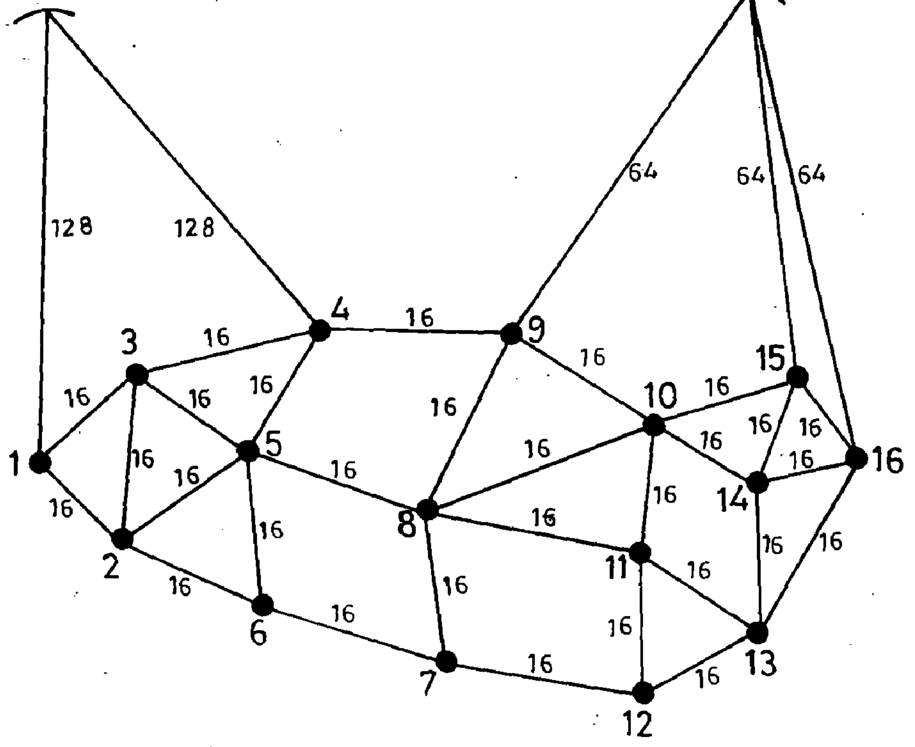


FIG. 5·1–NETWORK TOPOLOGY 1 WITH VARIABLE LINK CAPACITIES

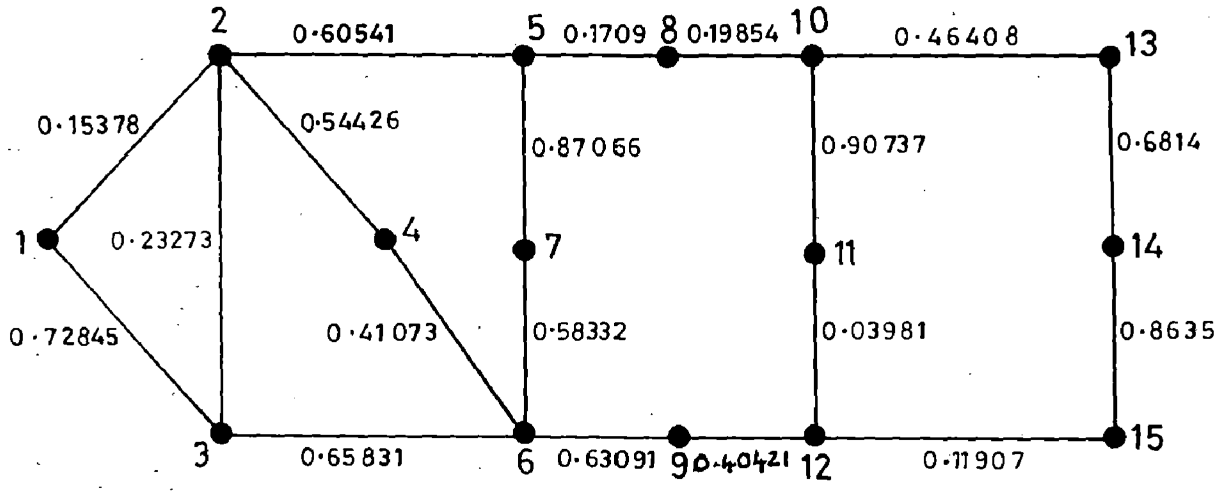


FIG. 5·2–NETWORK TOPOLOGY–2 WITH RANDOM LINK COSTS

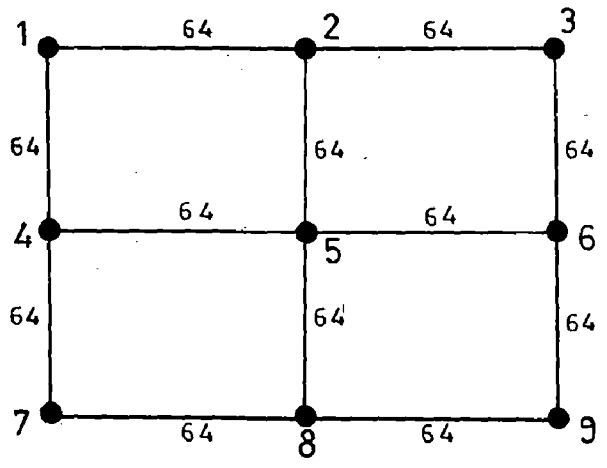


FIG.5·3–NETWORK TOPOLOGY–3 WITH FIXED LINK CAPACITIES

In all the three models, the route(s) information will be encoded in the state of converged neural network. For the topology shown in fig. 5.3, he state of neural network models for source - destination pair (1, 8) is shown in figs. 5.4 (a), 5.4 (b) and 5.4 (c).

Table 5.1

Lee-Chang model		**Zhang-Thomopoulos Model**		**Mustafa-Faouzi Model**	
Parameter set		parameter set		parameter set	
$\alpha = 0.005$		$A = 25$		$\mu = 950$	
$\beta = 30$		$B = 500$		$\mu_2 = 3500$	
		$C = 500$		$\mu_3 = 3200$	
		$N = 9.5$		$\mu_4 = 3500$	
		$\Delta t = 2X10^{-6}$		$\mu_5 = 85$	
		$u0 = 0.02$		$\Delta t = 10^{-5}$	
				$\lambda = 1$	
Link Capacity	**Link cost Chosen**	**Link Capacity**	**Link cost Chosen**	**Link capacity**	**Link cost Chosen**
128	0.2	128	2.2	128	0.2
64	0.4	64	2.4	64	0.4
32	0.6	32	2.6	32	0.6
16	0.8	16	2.8	16	0.8

SOURCE 1 -- DESTINATION 8

MODEL : Lee-Chang

ROUTE(s) : 1 - 2 - 5 - 8

1 - 4 - 5 - 8

NUMBER OF LINKS : 3

Fig. 5.4(a) STATUS OF CONVERGED NEURAL NETWORK :

	1	2	3	4
1	1.000000	0.000000	0.000000	0.000000
2	0.000000	0.172443	0.000000	0.000000
3	0.000000	0.000000	0.000000	0.000000
4	0.000000	0.814312	0.000000	0.000000
5	0.000000	0.000000	0.986755	0.000000
6	0.000000	0.000000	0.000000	0.000000
7	0.000000	0.000000	0.000000	0.000000
8	0.000000	0.000000	0.000000	1.000000
9	0.000000	0.000000	0.000000	0.000000

MODEL : Zhang-Thamapoulos

ROUTE(s) : 1 - 4 - 5 - 8

NUMBER OF LINKS : 3

Fig. 5.4(b) STATUS OF CONVERGED NEURAL NETWORK :

	1	2	3	4	5	6	7	8	9
1	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.9998	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
5	0.0000	0.0000	1.0000	1.0000	1.0000	1.0000	0.0000	0.0000	0.0000
6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
7	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
8	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.9999	1.0000	1.0000
9	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

MODEL : Mustafa-Faouzi

ROUTE(s) : 1 - 2 - 5 - 8

1 - 4 - 5 - 8

1 - 4 - 7 - 8

NUMBER OF LINKS : 3

Fig. 5.4(c) STATUS OF CONVERGED NEURAL NETWORK :

	1	2	3	4	5	6	7	8	9
1	0.0000	0.2755	0.0000	0.6352	0.0000	0.0000	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000	0.0000	0.2450	0.0000	0.0000	0.0000	0.0000
3	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
4	0.0000	0.0000	0.0000	0.0000	0.3541	0.0000	0.2465	0.0000	0.0000
5	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.6280	0.0000
6	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
7	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.2712	0.0000
8	1.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
9	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0171	0.0000

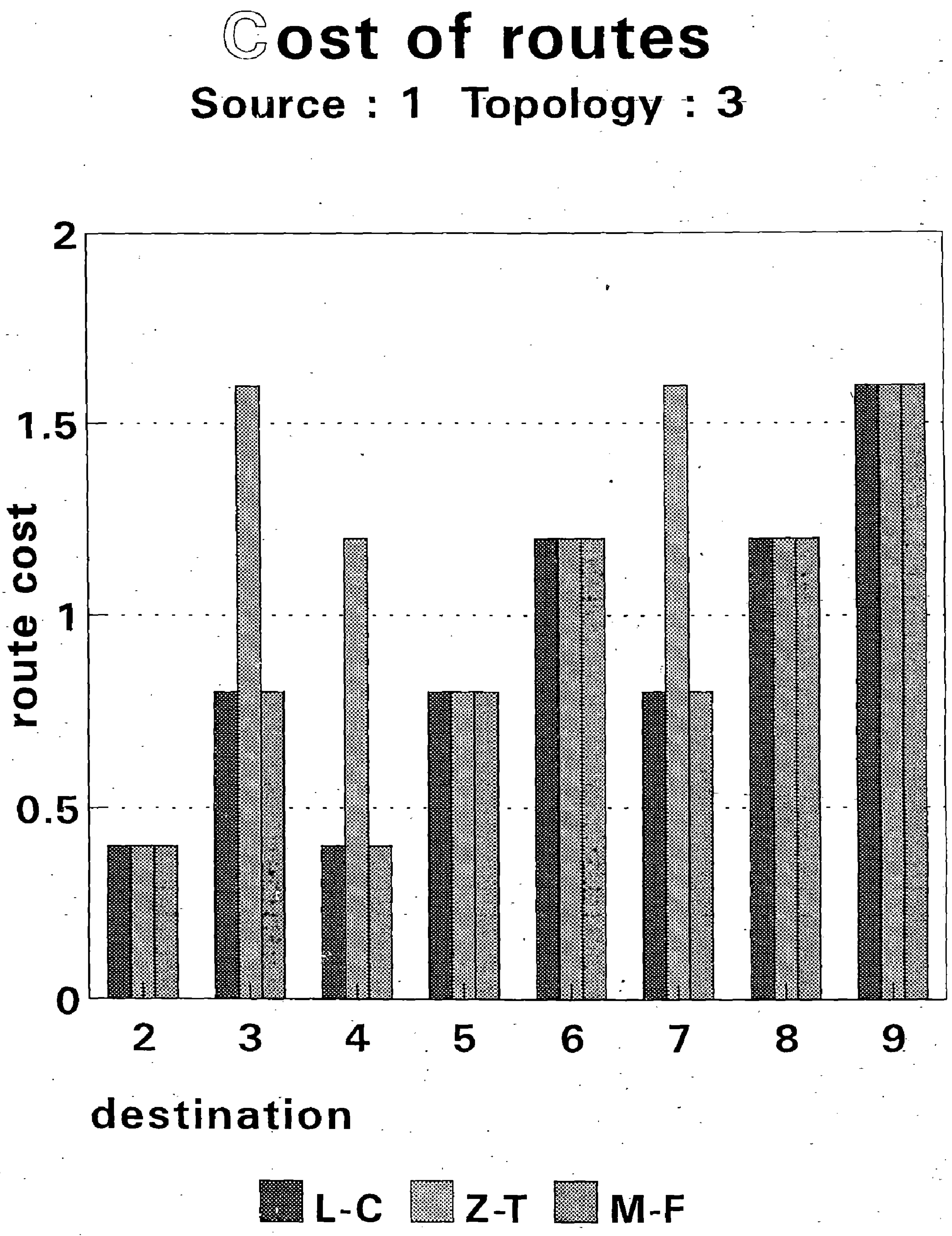


Fig. 5.5 Performance comparision
L-C,Z-T,M-F models
Measure : route cost

In Lee-Chang model and Zhang-Thomopoulos model, it can be easily seen that at any position in path sum of values of neurons representing that position is equal to one. But this is also satisfied in Mustafa-Faouzi model.

For all combinations of source-destination pairs in communication network topology shown in fig. 5.3, all three models are applied. The following are based on the results obtained.

MODEL	**% OF CASES IN WHICH OPTIMAL ROUTES ARE FOUND**
Lee-Chang	100
Zhang-Thomopoulos	69.44
Mustafa-Faouzi	100

Taking source as 1 and for all possible destinations the cost of routes found by the three models are shown in fig. 5.5.

The inherent problem found in Zhang-Thomopoulos model is, the cost of links is reflected in interconnection weights. So the neurons belonging to the row which represents the node, having highest number of links associated with other nodes in the communication network topology are very highly activated, always. This won't become a problem for small networks.

But because of this reason neural network diverges when applied to communication networks of big size.

It is found that, almost all routes given by Zhang-Thomopoulos model go through the node 5, [node 5 has more number of links, i.e. (4) connected to it]. The above mentioned reason is found to be true, when Zhang-Thomopoulos model is applied to network topology shown in fig. 5.1, in many cases network is found to be diverging.

For all combinations of source- destination pairs in communication network topologies shown in fig. 5.1 and fig. 5.2 Lee-Chang model and Mustafa-Faouzi model are applied. The following are based on the results obtained.

In all the cases both models converged to valid solutions.

It is found, Lee-Chang model can give simultaneously multiple optimal, suboptimal routes, differing in their route cost by approximately 5%. But Mustafa-Faouzi model gives multiple optimal routes, only when all the routes are of equal cost. Because of this performance of Lee-Chang model is slightly better in some cases when applied to topology shown in fig. 5.2.

Figs. 5.6 and 5.7 show the cost of route (s) found by Lee-Chang model and Mustafa-Faouzi, when applied to a particular set of source-destination pairs.

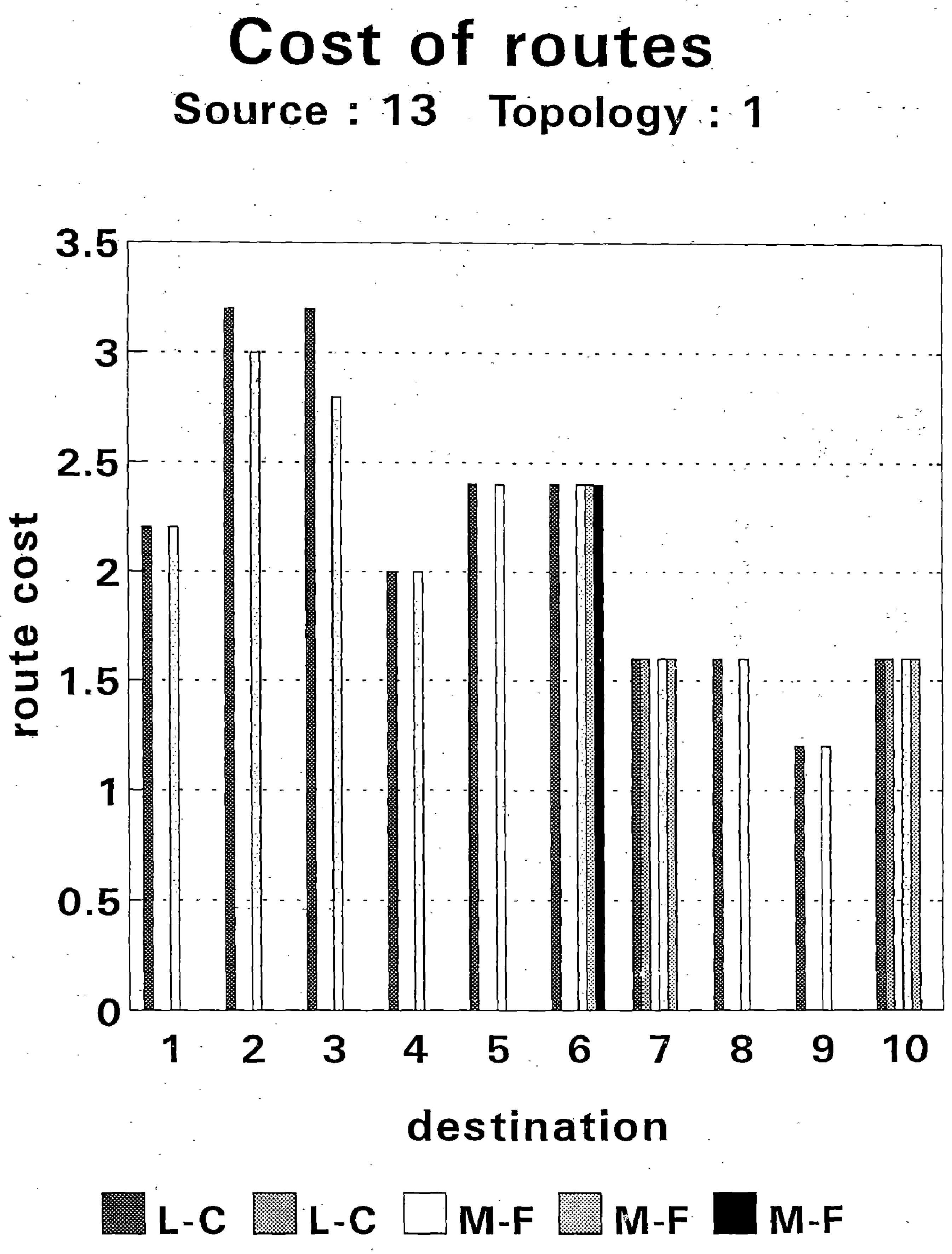


Fig. 5.6 Performance comparision
L-C and M-F models
Measure : route cost,multiple routes

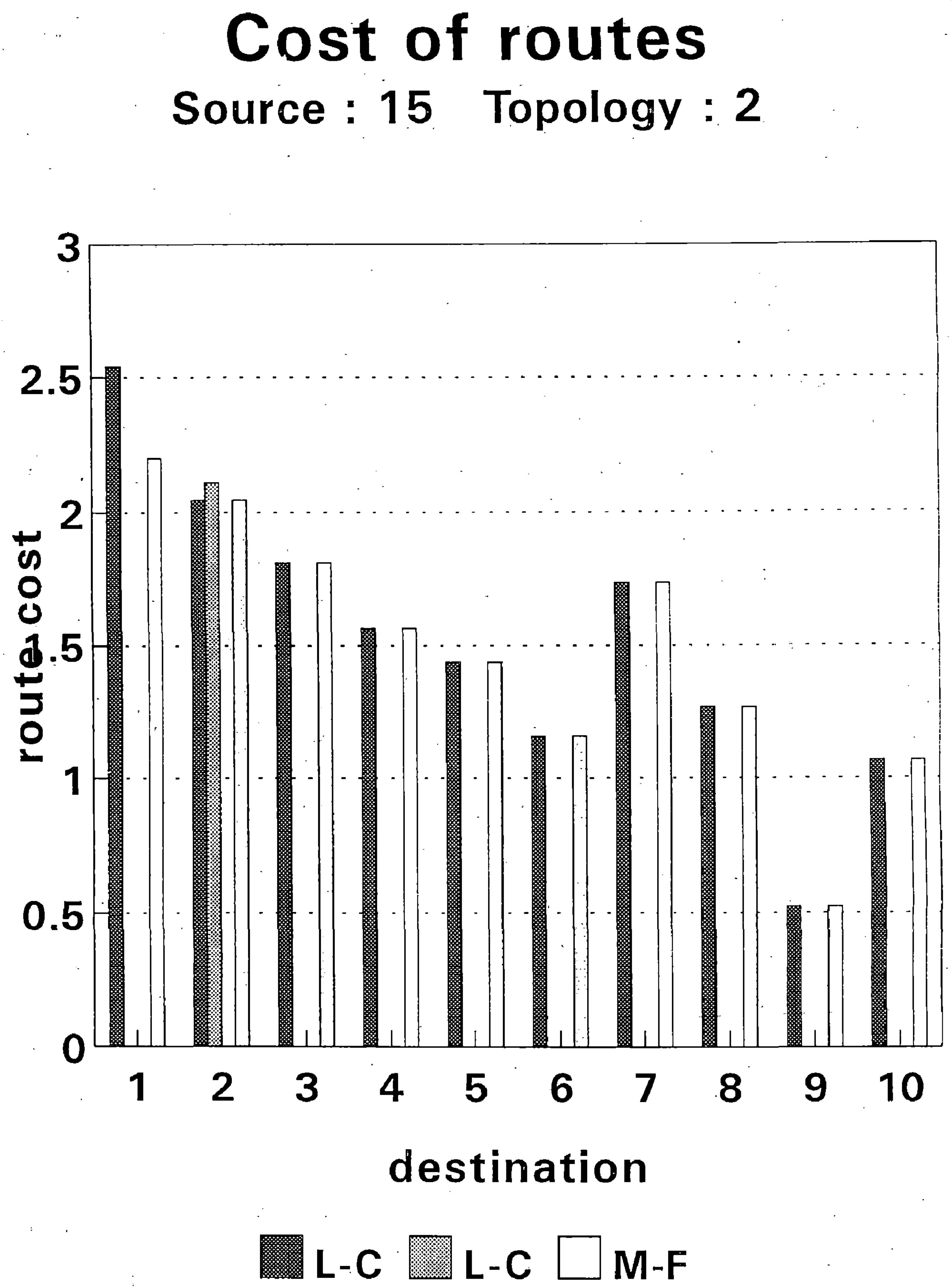


Fig. 5.7 Performance comparision
L-C and M-F models
Measure : route cost,multiple routes

In the Lee-Chang model, at any particular position in the path, a node is selected such that the sum of distance to the node in the previous position in the path and the distance to the node in the next position in the path is minimum. Because of this formulation, if the routes, differ in the i^{th} position in the path, they won't differ in $(i-1)^{th}$, $(i+1)^{th}$ positions in the path. Hence, Lee-Chang model in general not giving all existing optimal routes. But Mustafa-Faouzi model gives all the existing, optimal routes. These statements can be observed in fig. 5.6 and fig. 5.7 .

Other Observations :

- Lee-Chang model is converging in between 75 to 400 iterations. On average it is taking 225 iterations, 1 sec. of processing power in TATA ELXSI POWER SERIES 3200 SYSTEM.
- Mustafa-Faouzi model is converging in between 4000-12000 iterations. On average it is taking 3 minutes of processing power in TATA ELXSI POWER SERIES 3200 SYSTEM, 9000 iterations to produce route(s).
- Lee-Chang model can be easily extended to take reliability of nodes, dependency between nodes in the communication network, in to consideration while computing the route. But, number of iterations for convergence, becoming large (approximately 3 to 4 times i.e.. 800-900 iterations).

CONCLUDING REMARKS :

Zhang-Thomopoulos model formulation is not suitable for large communication networks.

Lee-Chang model is more or less suited for software implementation because the number of neurons in the model changes depending on the number of links. Computational power requirement of this model is low. Because of the drawback of this model there may be a chance of crowding over some links.

Mustafa-Faouzi model computational power requirement is very high in software implementation. This model is highly suitable for hardware approach. Also this model has the characteristics of ideal routing algorithm. Parameter selection is a critical matter in this model. The attractive feature of this model is extraction of all existing optimal routes simultaneously.

5.2 SUGGESTIONS FOR FURTHER WORK

* Neural network models in this work are implemented(tested) for 16 node and 15 node communication network topologies,and maximum number of links up to 6 or 7 in the routing path. Real life implementations need testing in much larger communication network topologies of sizes around or more than 50 , and number of links in the routing path up to 20 or more. With the help of high computational facilities, the models can be tested for large communication networks.

* Hopfield neural networks can be easily modeled by simple electrical components, and the real potential of the Hopfield neural networks is in hardware approach. Work towards this direction can only exploit the abilities of Hopfield neural networks for routing problem.

REFERENCES

1. D. Bertsekas and R. Gallager, Data networks, Prentice-Hall of India, second edition, 1992.
2. Robert Hecht-Nielsen, Neuro Computing, Addison Wesley Publishing company, 1990.
3. Jacek M. Zurada, Introduction to Artificial Neural Systems, Jaico Publishers, 1992.
4. 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. on Circuits and Systems Vol. CCAS-33, No. 5, pp. 533-541, 1986.
5. Herbert E. Rauch and Theo Winarske, "Neural Networks for Routing Communication Traffic "IEEE Control systems magazine, pp. 26-31, April, 1988.
6. Su-Ling Lee and Shyang Chang "Neural Networks for Routing of communication networks with unreliable components", Vol. 4, No. 5, pp. 854-863, September, 1993.
7. L. Zhang and S.C.A. Thomopoulos , "Neural network implementation of the shortest path algorithm for traffic routing in communication networks", Proceedings Int. Joint Conf.on Neural Networks, pp. 2693-2702, vol. 3, November, 1991.

8. Mustafa K. Mehmet Ali and Faouzi Kamoun, "Neural Networks for shortest path computation and routing in computer Networks Vol. 4, No. 6, pp. 941-953, November, 1993.

9. Ouyang Y.C. and Bhatti A.A., "Neural network based routing in computer communication networks ", IEEE Int. Conf. on system Engineering, pp. 621-624, August, 1990.

10. Davis Jr., Gerald W., "Sensitivity analysis in Neural net solutions", IEEE Trans. System, Man and Cybernetics, Vol. 19, No. 5, September, 1989.

11. A.S. Tannenbaum, computer networks, Prentice-Hall of India, Second edition, 1988.

```
#include <stdio.h>
#include <curses.h>
#include <math.h>
#include <signal.h>

     /********************************/
     /*  GLOBAL   VARIABLES          */
     /********************************/

#define MODEL1  0
#define MODEL2  0
#define MODEL3  1

#define MAX 20              /* Maximum number of nodes in network */
#define NET_SIZE 16         /* Total number of nodes in the network */
#define RTMAX 8             /*  Maximum number of  routes          */

     /* Actions in comments will be activated if constant is 1      */
#define STAT_ON_FILE 0      /* Status of NN ,Route information on file*/
#define STAT_RESDISP 1      /*  Result display on screen            */
#define SIGCTCH  1          /*  Interrupt signal catch              */
#define AID_PARACAL  0      /*  Aid to parameter calculation        */
#define INC_DEPD 0          /* Inclusion of dependencies in L-C model */
#define INPUTMODE 0         /* Takes topology information from screen */
#define OPERMODE 1          /* Takes topology information from file   */

int SIZE;              /* Size of the network topology  */
int C[MAX][MAX];       /* Link capacity information     */
int SCE,DST;           /*  Source  and Destination      */

          /* Structure to store  result  */
struct  res{
     int *ROUTE[RTMAX];        /*  Route(s)                    */
     int RTCNT;                /*  Number of route(s)          */
     int RTLINKS[RTMAX];       /*  Number of links in the route */
     float *RTVAL[RTMAX];      /*  Neuron output values        */
     float  PREF[RTMAX];       /*  Preference of the route(s)   */
     float  RTCOST[RTMAX];     /*  Cost of the route(s)         */
          };
struct res result;

#define HVAL 10000

int ICNT;                      /* Number of iterations */

int ran[3] = {9000,866,30}; /* Seeds for random number generation */
```

```
      /* Mustafa-Faouzi model  Global Variables */

        /* Energy function coefficients   */
#define MU1  950
#define MU2  3500
#define MU3  3200
#define MU4  85
#define MU5  3500

/* Minimum change in neuron output required to continue iterations   */
#define DELTAV 0.000001
#define  IT  0.00001       /* Difference time interval  */
#define LAMDA  1            /*  Gain                        */

/* Arrays to store values of Neuron output,input and change in output */
float no[MAX][MAX],ni[MAX][MAX],dno[MAX][MAX];

float cost[MAX][MAX];      /* Delay weighting matrix  */

      /* Lee-Chang model Global Variables */

#define DELTARW 0.00000001

float w[MAX][MAX];      /* Delay nformation          */
float d[MAX][MAX];     /* Dependency information     */
float q[MAX];          /* Reliability information   */
float  *dbprod;
int NLIST[MAX];

      /*  Control Vectors    */
struct vect{
      float ele[MAX];
};
struct vect  *vec[MAX], *newvec[MAX], *dvec[MAX];

          /* Energy function coefficients   */
#define THETA 0.8
#define MU     0.5
#define BETA  0.5
#define ALPHA 0.005
#define GAMMA 30

int lks ;       /* Number of links between source and destination */


      /* Zhang-Thomopoulos model Global Variables */

      /* Energy function coefficients                 */
#define A 25
#define B 500
#define DC 500
#define NN 9.5
```

```
        /****************************************/
        /* Mustafa-Faouzi model functions      */
        /****************************************/

void  rtfinder();
void create_costmat();
float iccal(int , int);
void findpath();
void sigcatch_intr();
void convrg_details(int );
void paracheck();


main()
{
#if MODEL1
        takeip();
        router1();
#endif

#if MODEL2
        takeip();
        ztrouter();
#endif

#if MODEL3
        takeip();
rtfinder();
#endif
}


        /*********************************************************/
        /*      Function reads the input information in to array      */
        /*********************************************************/

void readmatp(float ab[MAX][MAX],int ba[MAX][MAX], int S)
{
int i,j;
for(i=0;i<SIZE ;i++)
{
        printf("row %d  elements \n",i+1);
        for(j=0;j<SIZE;j++)
        {
                printf("%d    :",j+1);
                switch(S)
                {
                case 1 :
                        scanf("%f", &ab[i][j]);
                        break;
                case 2 :
                        scanf("%d",&ba[i][j]);
                        break;
                }
```

```
   /* Minimum change in energy required to continue iterations */
#define DELTAZT 0.0001

#define DT 0.000002 /*  Difference time interval   */

      /* Input,output values of neurons       */
float nnvec[MAX][MAX],coeff[MAX][MAX];

float Y0;           /*   Gain                          */
float c[MAX][MAX];  /* Link cost  information        */


      /****************************************/
      /**               FUNCTIONS           **/
      /****************************************/

void takeip();
void readmatp(float x[][MAX],int y[][MAX], int );
void storematf(char *, int , int );
void readmatf(char *, int , int);
float genrand(int , float );
void pathpref(int );
float nnw_energy(int ,int);
void fres_display();

      /****************************************/
      /*    Lee-Chang model functions        */
      /****************************************/

void router1();
void createw(int);
int detlinks(int , int ,int);
int detadj(int , int x[][MAX], int);
void createmem(int );
void init();
void initvect(struct vect *v[],int , int , int);
void vectdisp(struct vect *v[], int , int ,int);
void dbcalc();
float *matmult(float l[][MAX],float b[],float *,int , int ,int);
int diffcalc(int );
void update();
void findroutes();

      /****************************************/
      /*   Zhang-Thomopoulos model functions   */
      /****************************************/
void ztrouter();
void initnnw(int ,int );
void initcoeff();
float icval(int ,int ,int ,int );
float calk(int  , int );
void cal_neuronop();
void doupdate(int);
void findroute();
void create_cstmat();
```

```
          puts("    ");
     }
}
}

     /*************************************************************/
     /*    Function stores array in a file,                       */
     /*    always activated after function readmatp()             */
     /*************************************************************/

void storematf(char *fname, int S,int code)
{
int i, j;
FILE  *fptr;
char ch='s';

if((fptr = fopen(fname, "w")) != NULL)
{
     for(i=0; i<S; i++)
          for(j=0; j<S; j++)
               switch(code)
               {
               case 0 :fprintf(fptr, "%d%c",C[i][j],ch);
                         break;
               case 1 :fprintf(fptr, "%f%c",d[i][j],ch);
                         break;
               case 2 :fprintf(fptr, "%f%c",c[i][j],ch);
                         break;
               case 4 :fprintf(fptr, "%f%c",cost[i][j],ch);
                         break;
               }
     if(code == 3)
          for(i=0;i<SIZE;i++)
               fprintf(fptr,"%f%c",q[i],ch);
     fclose(fptr);
}
else
     puts("ERROR IN OPENING FILE\n ");
}

     /*************************************************************/
     /*    Function reads  array from a file                      */
     /*************************************************************/

void readmatf(char *fname, int  S,int code)
{
int i,j;
FILE  *fptr;
char ch=' ';

if((fptr = fopen(fname, "r"))!= NULL )
{
     for(i=0; i<=S-1; i++)
          for(j =0; j<= S-1; j++)
               switch(code)
```

```
                {
                case 0 : fscanf(fptr, "%d%c", &C[i][j],&ch);
                              break;
                case 1:  fscanf(fptr,"%f%c",&d[i][j],&ch);
                              break;
                case 2:  fscanf(fptr,"%f%c",&c[i][j],&ch);
                              break;
                case 4 : fscanf(fptr, "%f%c", &cost[i][j],&ch);
                              break;
                 }
        if(code == 3)
            for(i =0;i< S ; i++)
                fscanf(fptr, "%f%c",&q[i],&ch);

        fclose(fptr);
    }
    else
    {
        printf("ERROR IN OPENING THE FILE \n\n");
        exit(0);
    }
    }



        /*********************************************************/
        /**    Function takes Source,Destination,link capacity  **/
        /**    as inputs and computes delay information         **/
        /*********************************************************/

    void takeip()
    {
    int type;

    SIZE = NET_SIZE;

    #if INPUTMODE
    readmatp(w,C,type = 1);
    storematf("MAT",SIZE,type = 0);
    #endif

    #if OPERMODE
    readmatf("MAT" ,SIZE,type);
    #endif

    #if MODEL1
        createw(SIZE);
    #endif

    #if MODEL2
        create_costmat();
        #if AID_PARACAL
            paracheck();
            exit(0);
        #endif
```

```
#endif

#if MODEL3
     create_cstmat();
#endif

printf(" ENTER SOURCE  :");
scanf("%d", &SCE);
if(SCE > SIZE || SCE < 1)
{
     printf("Sorry Source does'nt Exist!!\n");
     exit(1);
}
printf(" ENTER DESTINATION :");
scanf("%d", &DST);
if(DST > SIZE || DST < 1)
{
      printf("Sorry Destination does'nt Exist!!\n");
      exit(1);
}
return;
}

/*****************************************************************/
/*      Function implements Lee-Chang model                     */
/*   Inputs : Capacity matrix, Source and Destination Nodes    */
/*   outputs : status of NN at different intervals during     */
/*     convergence, route(s), cost of the route(s)           */
/*****************************************************************/

void router1()
{
int count=0,i,k,j;
float ENRGYN,ENRGYO;
FILE *fpt,*f1;

     /* Determining number of links between SCE and DST   */
lks = detlinks(SIZE,SCE,DST);
printf("SOURCE-%d....DESTINATION-%d\n",SCE, DST);
printf("NO. OF LINKS-%d",lks);

     /*  Creating  control  vectors                       */
creatmem(lks);
init();

#if INC_DEPD
     /* Dependency,reliability information inclusion      */
     takedq();
     dbcalc();
#endif

     /* Initializing  control vectors                     */
initvect(vec,lks,SCE, DST);
```

```
#if AID_PARACAL
      paracal();
      exit(1);
#endif

#if  STAT_ON_FILE
      f1  = fopen("PFILE","w");
      fprintf(f1,"SOURCE : %d      DESTINATION  : %d\n\n",SCE,DST);
#endif

ENRGYO  =  nnw_energy(lks,i=1);
printf("\nENTER NO. OF ITERATIONS :");
scanf("%d",&ICNT);
while(count  <=  ICNT)
{
      /* Updating control vectors                              */
      update();
      ENRGYN  =  nnw_energy(lks,i=1);

      #if STAT_ON_FILE
      if(count  %  100   ==  0)
      {
      fprintf(f1,"\n ITERATION NUMBER : %d\n",count);
      for(j =0; j< SIZE; j++)
            for(k =0;k  <=  lks; k++)
            {
            if(k  ==  0) fprintf(f1,"\n");
            fprintf(f1,"%6f   ", vec[k]->ele[j]);
            }
      }
      #endif

      /* Checking for convergence                              */
      if (count  >  100)
      if(ENRGYO - ENRGYN   <   DELTARW) break;
      ENRGYO  =  ENRGYN;

      count++;
}

#if STAT_ON_FILE
      fclose(f1);
#endif

/* Determining route(s)   information   from converged NN   */
findroutes();
pathpref(i=1);

return;
}
```

```
void takedq()
{
/* Function fills the reliability,dependency information */
int i;

readmatf("DEPD",SIZE,i=1);
readmatf("REL",SIZE,i=3);
system("clear");
return;

}

    /***********************************************************/
    /* Function determines the number of links in the path */
    /***********************************************************/

int detlinks(int X , int S, int D)
{
int c1[MAX][MAX],c2[MAX][MAX],c3[MAX][MAX];
int count , i, j,k;

S-=1; D-=1;
for (i =0; i<= X-1; i++)
for(j=0; j <=X-1; j++)
    {   if(C[i][j] > 0)    c1[i][j] = c2[i][j] = 1;
             else           c1[i][j] = c2[i][j] = 0; }

count = 1;
if(S == D) return 0;
    else if(c1[S][D] == 1) return count;
while(count <= X)
{
for(i=0; i< SIZE; i++)
    for(j=0; j< SIZE ; j++)
    {   c3[i][j] = 0;
        for(k =0; k < SIZE; k++)
        c3[i][j] = c3[i][j] | (c1[i][k] & c2[k][j]);
    }

for(i=0; i< SIZE; i++)
    for(j=0; j< SIZE; j++)
        c2[i][j] = c3[i][j];
if(c2[S][D] == 1)  break;
count++;
}

if(count == X) return -1;
    else return ++count;
}

int detadj(int N, int cx[MAX][MAX], int S)
{
/* Function determines the number of adjecencies to node N */
int i,j;
```

```
i=0;
N-=1;
for(j=0; j< S ; j++)
     if((cx[N][j] >0) &&(j!= N))
     {
          i++;
          NLIST[i-1] =j;
     }
return i;
}

     /**************************************************/
     /*     Function creates the control vectors      */
     /**************************************************/

void creatmem (int links)
{
int i;

for(i=0; i<= links;i++)
{
     vec[i] = (struct vect *) malloc(sizeof(struct vect));
     newvec[i] = (struct vect *) malloc(sizeof(struct vect));
     dvec[i] = (struct vect *) malloc(sizeof(struct vect));
}
return;
}

     /**************************************************/
     /*     Function initializes control vectors      */
     /**************************************************/

void initvect(struct vect *vp[],int vcnt,int S, int D)
{
int i,j,k;
int adcnt;
char ch;

i=0;
while(i< =vcnt)
{
     for(j=0;j< =SIZE-1;j++)
          vp[i]->ele[j] =0;

    if(i==0)
     {
          vp[i]->ele[S-1] =1;
     }
     else if(i==vcnt)
     {
          vp[i]->ele[D-1] =1;
     }
          else if (i == vcnt -1)
          {
                adcnt = detadj(D,C,SIZE);
                for(j=0;j< adcnt; j++)
                {
                     k = NLIST[j];
                     vp[i]->ele[k] = 1/(float)adcnt;
                }
           }
                else if(i ==1)
                {
                     adcnt = detadj(S,C,SIZE);
                     for(j=0; j<adcnt;j++)
                     {
                          k = NLIST[j];
                        vp[i]->ele[k] = 1/(float)adcnt;
                     }
                }
                     else
                     {
                      for(j=0; j< SIZE; j++)
                      if(j!= S-1 && j != D-1)
                vp[i]->ele[j] = 1/(float)SIZE;
                     }
i++;
}
return;
}

void createw(int S)
{
/* Function computes the delays corresponding to existing links */
int i,j;

for(i=0;i<S;i++)
     for(j=0; j<S ;++j)
     {
          if(C[i][j]  == 0)    w[i][j] =10;
          else   switch(C[i][j])
                     {
                     case  128  : w[i][j] = 1;
                                  break;
                     case   64  : w[i][j] = 1.3;
                                  break;
                     case   32  : w[i][j] = 1.6;
                                  break;
                     case   16  : w[i][j] = 1.9;
                                  break;
                     }
     }
return;
}

void dbcalc()
{
/* Function computes dbprod                         */
int i;
```

```
dbprod = (float *) malloc(sizeof(float)  * SIZE);
matmult(d,q,dbprod,SIZE,SIZE,i=1);
return;
}

      /*****************************************************/
      /*  Function computes the change in control vector   */
      /*****************************************************/

int diffcalc(int k )
{
int i,j,n=1;
float *d1, *d2;
float l,m,sum,q;

if(k == 0)  return  -1;

d1 = (float *)malloc((sizeof(float))*SIZE);
matmult( w ,newvec[k-1]->ele,d1,SIZE,SIZE,n);
d2 = (float *) malloc((sizeof(float))*SIZE);
matmult( w , newvec[k+1]->ele,d2,SIZE,SIZE,n);
sum =0;
for(i=0;i<SIZE; i++)
      sum = sum + newvec[k]->ele[i];
m = ALPHA * GAMMA * (1-sum);
for(i=0;i<SIZE; i++)
{
      l = -(0.5) * ALPHA * (d1[i] + d2[i]);
      #if INC_DEPD
            q = -(ALPHA) * BETA * dbprod[i] ;
            dvec[k]->ele[i] = l+m+q ;
      #else
            dvec[k]->ele[i] = l+m ;
      #endif
}

return 1;
}

      /*****************************************************/
      /*  Function updates the control vectors             */
      /*****************************************************/

void update()
{
int i,j;

for(i=0;i< =lks;i++)
      for(j=0;j<SIZE;j++)
            newvec[i]->ele[j] = vec[i]->ele[j];
for(i =1; i< lks; i++)
   diffcalc(i);
for(i =1; i < lks; i++)
      for(j =0; j< SIZE; j++)
      {
        if(newvec[i]->ele[j] > 0 && j != SCE-1  && j!= DST-1)
           if(dvec[i]->ele[j] > 0)
           newvec[i]->ele[j] = vec[i]->ele[j] + dvec[i]->ele[j];
           else
           newvec[i]->ele[j] = vec[i]->ele[j] +  dvec[i]->ele[j];
           if(newvec[i]->ele[j] < 0)   newvec[i]->ele[j] =0;
     }
for(i=1; i<lks;i++)
     for(j=0;j<SIZE;j++)
           vec[i]->ele[j] = newvec[i]->ele[j];
return;
}

void init()
{
/* Function initializes control vectors to zero  */
int i,j;
for(i=0; i<=lks; i++)
     for(j=0;j<SIZE; j++)
     newvec[i]->ele[j] = vec[i]->ele[j] = dvec[i]->ele[j] = 0;
return;
}

     /*****************************************************/
     /*  Function checks the validity of parameters,      */
     /*  suggests valid ranges of parameters              */
     /*****************************************************/

paracal()
{
int i,j;
float limit1,limit2,limit3,limit4;
float gma,alfa,teta,beta,mu,omegal,omegah,qmax,qmin,dqmax;
char ch;
WINDOW *win;

initscr();
win = newwin(18, 70,2,5);
wclear(win);
wattrset(win, A_ALTCHARSET);
box(win, ACS_VLINE, ACS_HLINE);
gma = GAMMA;
teta = THETA;
alfa = ALPHA;
mu = MU;
beta = BETA;
mvwprintw(win,2,10,"PARAMETERS : ");
mvwprintw(win,2,30,"GAMMA : %f    THETA  : %f",gma,teta);
mvwprintw(win,3,30,"ALPHA : %f    MU     : %f",alfa,mu);
mvwprintw(win,4,30,"BETA  : %f    ",beta);

omegal = 50;
omegah =0;
for(i = 0; i < SIZE;i++)
     for(j =0; j< SIZE;j++)
```

```
    if(w[i][j] < omegal) omegal = w[i][j];
    else if (w[i][j] > omegah) omegah = w[i][j];

qmin = qmax = 0;
for(i =0; i<SIZE; i++)
{
    if(q[i] < qmin) qmax = q[i];
        else if(q[i] > qmax) qmax = q[i];
}
dqmax = 0;
for(i =0; i<SIZE; i++)
{
    if(dbprod[i] > dqmax) dqmax = dbprod[i];
}
limit1 = mu/((1-teta)*SIZE);
limit2 = (teta * omegal)/(1-teta);
limit3 = (omegah)/(1-teta);
limit4 = ((SIZE - 1) * dqmax + qmax)/(1 - teta);

mvwprintw(win,8,2,"RANGES");
mvwprintw(win,8,45,"SATISFIED/NOT");
mvwprintw(win,9,45,"    (Y/N)");

mvwprintw(win,11,2,"(ALPHA)*(GAMMA) < %f",limit1);
mvwprintw(win,13,2,"%f <= ALPHA ",limit2);
mvwprintw(win,14,14,"<= %f + (BETA) * %f",limit3,limit4);
if((alfa * gma) < limit1) ch = 'Y';
    else ch = 'N';
mvwprintw(win,11,55,"%c",ch);
if((alfa > limit2) || (alfa <= limit3 + beta * limit4))  ch = 'Y';
    else ch = 'N';
mvwprintw(win,13,55,"%c",ch);

wrefresh(win);
wgetch(win);
endwin();
}

    /*********************************************************/
    /*  Function implements the Zhang-Thomopoulos model      */
    /*********************************************************/

void ztrouter()
{
int i,j,k,cnt;
float engy,p,ENRGYN,ENRGYO;
FILE *fpt,*f1;

Y0 = 0.02;

/* Initializing the NN model  */
initnnw(SCE,DST);
ENRGYO = nnw_energy(SIZE,k = 2);
```

```
/* Computing the initial neuron activations     */
initcoeff();

cnt = 1;
printf("Initial Energy :%f\n",ENRGYO);
printf("ENTER NO. OF ITERATIONS");
scanf("%d", &ICNT);
printf("ITERATION    --    ENERGY \n");
printf("NUMBER    \n");

while (cnt <= ICNT)
{
     /* Computing neuron outputs at successive intervals */
     doupdate(cnt);
     cal_neuronop();

     ENRGYN = nnw_energy(SIZE,k = 2);
     if(cnt > 100)
     if(fabs(ENRGYO - ENRGYN) < DELTAZT) break;
     ENRGYO = ENRGYN;
     cnt++;
}

#if STAT_ON_FILE
f1 = fopen("OPFILE","w");
fprintf(f1,"SOURCE %d  --  DESTINATION %d\n\n",SCE,DST);
for(i = 0; i < SIZE;i++)
     for(j =0; j< SIZE; j++)
          {
               if(j == 0) fprintf(f1,"\n");
               fprintf(f1,"%6.4f  ",nnvec[i][j]);
          }
fclose(f1);
#endif

/* Determining the route information     */
findroute(i=2);
pathpref(i=2);

return;
}

     /*********************************************************/
     /* Function generates a random number between 0 and 1 */
     /*********************************************************/

float genrand(int S,float ded)
{
int i,k;
float j,temp;

switch(S)
{
```

```
case 1 :
     j = drand48();
     if(ded > 0)
     while(j > 0)
          j = j - ded;
     j = j+ ded;
     return j;
case 2 :
     ran[0] = 171 * (ran[0] / 177) - 2 *(ran[0] % 177);
     if( ran[0] < 0 )  ran[0] = ran[0] + 30269;
     ran[1] = 172 * (ran[1] / 176) - 35 * (ran[1]  % 176);
     if ( ran[1] < 0 ) ran[1] = ran[1] + 30307;
     ran[2] = 170 * (ran[2]/ 178) - 63 * (ran[2] % 178);
     if(ran[2] < 0) ran[2] = ran[2] + 30323;
     temp = ran[0]/30269.0  + ran[1]/30307.0  + ran[2]/30323.0;
     k = temp;
     temp = temp - k ;
     if( ded > 0)
          while(temp > 0)
               temp = temp - ded;
     if (temp < 0) temp  += ded;
     return temp;
}
return;
}


     /*********************************************************/
     /* Function initializes the NN configuration           */
     /*********************************************************/

void initnnw(int S , int D)
{
int i,j,k;
float xk;

i = 0;
while(i < SIZE)
{
     if(i == 0 || i == SIZE -1)
     {
          for(j = 0; j < SIZE; j++)
               nnvec[j][i] = 0;
          switch(i)
          {
          case 0 :
               nnvec[S-1][i] = 1.0;
               break;
          default :
               nnvec[D-1][i] = 1.0;
               break;
          }
     }
     else
     {
          xk = 0.1 * Y0;
          for(j = 0; j < SIZE ; j ++)
               nnvec[j][i] = 1/(float)SIZE + genrand(k=2,xk);
     }
     i++;
}

xk = 0;
for(i = 0; i < SIZE ; i++)
     for(j =1;j < SIZE-1; j++)
          xk = xk + nnvec[i][j];
xk = xk - SIZE +1;
xk = xk / ((SIZE - 1) *SIZE);
for(i = 0; i < SIZE ; i++)
     for(j = 1; j < SIZE-1; j++)
     {
          if(nnvec[i][j] > 0.05) nnvec[i][j]  -= xk;
          if(nnvec[i][j]  < 0)  nnvec[i][j] += xk;
     }
return;
}

void cal_neuronop()
{
/*  Function computes the neuron ouputs        */
/*   corresponding to input activations      */
int i,j;
float temp,val;

i = 0;
for(i = 0; i < SIZE ; i++)
     for(j = 1; j < SIZE -1 ; j++)
     {
          val = coeff[i][j] ;
          val = val/Y0;
          temp =   1 + ftanh(val) ;
          nnvec[i][j] = temp / 2;
     }
return;
}

float icval(int i,int j,int m,int n)
{
/* Calculating the inter connection value between ij'th and */
/* mn'th  neurons.                                           */

float temp;

temp = -DC;
if(n == j+1) temp -= A * c[i][m];
if(n == j -1) temp -= A * c[i][m];
if(j == n  && i != m)  temp -=  B * 1.0;
return temp;
}
```

```
float calk(int i , int j)
{
/* Function computes the net input activation          */
int m,n;
float temp,k;

k = 0;
for(m = 0; m < SIZE ; m++)
     for(n = 0; n < SIZE; n++)
     {
          temp = icval(i,j,m,n);
          k += temp  *  nnvec[m][n];
     }
return  k;
}

     /*******************************************************/
     /*   Function computes neuron outputs at succesive      */
     /*      time instants                                  */
     /*******************************************************/

void doupdate(int cnt )
{
int i,j,k;
float c1;

for(i = 0; i < SIZE ; i++)
     for(j = 1; j < SIZE-1; j++)
     {
          c1 =  calk(i,j);
          c1 += DC * NN;
          coeff[i][j] = coeff[i][j] * (1 -DT) + DT * c1;
     }
return;
}

void initcoeff()
{
/* Function comppputes the initial input activations of neurons */
int i,j;
float  temp,lambda;

lambda = 2 / Y0;
for(i = 0; i < SIZE ; i++)
     for(j = 1; j < SIZE -1 ; j++)
     {
          temp = (1 - nnvec[i][j])/ nnvec[i][j] ;
          temp = flog(temp);
          temp =(-1 / lambda) * temp ;
          coeff[i][j] = temp;
     }
return;
}
```

```
void create_cstmat()
{
/* Function computes the delay information      */

int i,j;

/* Cost is in relation to the capacity of the link */
for(i=0;i<SIZE;i++)
      for(j=0; j<SIZE ;++j)
            if(i != j)
            if(C[i][j] == 0)  c[i][j] = 20;
            else
            switch(C[i][j]) {
                  case  128  : c[i][j] = 2.2;
                               break;
                  case   64  : c[i][j] = 2.4;
                               break;
                  case   32  : c[i][j] = 2.6;
                               break;
                  case   16  : c[i][j] = 2.8;
                               break;
                        }
return;
}

void  fres_display()
{
/* Function stores route(s) information in file  */
int i,j,cnt;
FILE  *resf;

resf = fopen("RESU","a");
fprintf(resf,"\nSOURCE : %d   DESTINATION : %d\n",SCE,DST);
fprintf(resf,"NO. OF ROUTES : %3d\n",result.RTCNT);
fprintf(resf,"\tROUTE                       ");
fprintf(resf,"   NO. OF LINKS  ");
fprintf(resf,"  COST   ");
fprintf(resf," PROBABILITY \n ");

i = 0;
while(i < result.RTCNT)
{
      cnt = 0;
      for(j = 0; j < result.RTLINKS[i];j++)
      { cnt+= 4;
      fprintf(resf,"%2d  ",result.ROUTE[i][j] + 1);
      }
      for(j = 0; j < 38-cnt;j++)
      fprintf(resf," ");
      fprintf(resf,"%3d          ",result.RTLINKS[i] -1);
      fprintf(resf,"%7.4f      ",result.RTCOST[i]);
      fprintf(resf,"%7.4f\n",result.PREF[i]);
      i++;
}
fprintf(resf,"ITERATION COUNT : %5d\n",ICNT);
```

```
fflush(resf);
fclose(resf);
return;
}

/*******************************************************************/
/*        This function implements the Mustafa-Faouzi model        */
/*   Inputs : Capacity matrix, Source and Destination Nodes   */
/*   outputs : status of NN at different intervals during     */
/*      convergence, route(s), cost of the route(s)          */
/*******************************************************************/

void rtfinder()
{
int i,j,k,ch,x,cnt,ds,sc;
float l,temp,ic[MAX][MAX];
FILE *flpt;

#if STAT_ON_FILE
flpt = fopen("OPFILE","w");
#endif

#if SIGCTCH
      signal(SIGINT,sigcatch_intr);
      signal(SIGQUIT,sigcatch_intr);
      signal(SIGHUP,sigcatch_intr);
#endif

/* Initializing the  state of Neural Network  */

for(i =0; i < SIZE; i++)
      for(j =0; j< SIZE; j++)
      if(i!=j)no[i][j] = 1/(float)(SIZE * 10) + genrand(k=2,l=0.0002);

/* Calculating the corresponding input activations of Neurons  */
/* of the Neural Network                                        */
temp = LAMDA;
for(i =0; i < SIZE; i++)
      for(j =0; j< SIZE; j++)
      if(i != j) ni[i][j] = -(1/temp) * flog((1 - no[i][j])/no[i][j]);
ch = 1;
cnt = 0;
while(ch)
{
      /* Calculating the incremental changes in the input      */
      /* activations of Neurons of Neural Network using         */
      /* Relaxation method                                      */

      for(i = 0; i < SIZE; i++)
            for(j =0; j < SIZE; j++)
            if(i != j)
            {
                  l = ic[i][j] = iccal(i,j);
                  ni[i][j] = (ni[i][j] + l *IT) / (1 + IT);
            }
```

```
    /* Calculating the corresponding output values of neurons  */
    /* of Neurons for the input activations calculated above   */

    for(i =0; i < SIZE; i++)
        for(j =0; j < SIZE; j++)
        if(i != j)
        {
            temp = 0;
            dno[j][i] = no[j][i];
            temp = fexp( - LAMDA * ni[j][i]);
            no[j][i] = 1/(1+temp);
            dno[j][i] -= no[j][i];
        }

    if(cnt % 2000 == 0)
    {
        #if STAT_ON_FILE
        fprintf(flpt,"\nITERATION NUMBER %d \n",cnt);
        for (i = 0; i < SIZE; i++)
            for(j = 0; j < SIZE; j++)
            {
                if(j == 0) fprintf(flpt,"\n");
                fprintf(flpt,"%6.4f  ",no[i][j]);
            }
        #endif

    }
    x = 1;
    for(i =0; i < SIZE && x ; i++)
        for(j =0; j < SIZE && x ; j++)
        if(i != j)
            if(dno[i][j]  < -DELTAV) x =0;

    if(x !=0 || cnt > 15000)
    {
        convrg_details(cnt);
        ch = 0;
    }
    cnt++;
}

#if STAT_ON_FILE
    fclose(flpt);
#endif

findpath();
pathpref(i = 3);

return;
}
```

```
/**********************************************************************/
/*Function  Input : Neuron location in the array                     */
/*          Output : Calculates the net activation of neuron         */
/**********************************************************************/

float iccal(int x, int i)
{
int y,j;
float ic,t,i1,i2,i3,i4,i5;

i1 = 0;
if(x == DST-1 || i == SCE-1) ;
else      i1 = -MU1  * 0.5 * cost[x][i];

i2 = 0;
if(x == DST-1 || i == SCE-1) ;
else      if(cost[x][i] <= 0)
           i2 = -MU2 * 0.5;

t = 0;
i3 = 0;
for(j =0; j< SIZE; j++)
if(x != j)  t += no[x][j] - no[j][x];
i3 = -MU3 * t;
t = 0;
for(j =0; j < SIZE; j++)
if(j != i) t += no[i][j]  - no[j][i];
i3 +=  MU3 * t;

i4 = i5 = 0;
i4  = -MU4  * 0.5 * (1 - 2 * no[x][i]);

if(x == DST-1  &&  i == SCE -1)
     i5 =  MU5  * 0.5;

ic = i1+i2+i3+i4+i5;
return ic;
}

     /*****************************************************/
     /* Input is Capacity Matrix and creates Cost matrix */
     /*****************************************************/

void create_costmat()
{
int i,j;

/* Cost is in relation to the capacity of the link */
for(i=0;i<SIZE;i++)
     for(j=0; j<SIZE ;++j)
          if(C[i][j] == 0)  cost[i][j] = 0;
          else
          switch(C[i][j]) {
               case  128  : cost[i][j] = 0.2;
                          break;
```

```
                case   64  : cost[i][j] = 0.4;
                            break;
                case   32  : cost[i][j] = 0.6;
                            break;
                case   16  : cost[i][j] = 0.8;
                            break;
                default     : ;
                        }
return;
}

/*****************************************************************************/
/* Function will be activated when there is an interrupt signal      */
/*    during the convergence of the neural network                   */
/*****************************************************************************/

void sigcatch_intr()
{

WINDOW *win;
int i;

initscr();
win = newwin(16,60,2,10);
wattrset(win, A_ALTCHARSET);
box(win,ACS_VLINE,ACS_HLINE);

mvwprintw(win,4,10," INTERRUPT RECEIVED ");
findpath();

/*        Checks the RTCNT to determine whether present            */
/*    neural network status gives any valid route             */

if(result.RTCNT <= 0)
{
     mvwprintw(win,7,10,"SORRY!!!  NETWORK NOT CONVERGED");
     wrefresh(win);
     sleep(2);
     endwin();
     exit(0);
}
else
{
     mvwprintw(win,7,10,"SHOWING  ROUTES");
     wrefresh(win);
     sleep(2);
     endwin();
     pathpref(i = 3);
     exit(1);
}
}
```

```
void convrg_details(int cnt)
{
/* Function informs whether about convergence        */
WINDOW *win;

initscr();
win = newwin(10,60,5,10);
wattrset(win,A_ALTCHARSET);
box(win,ACS_VLINE,ACS_HLINE);
mvwprintw(win,3,10,"CONVERGENCE AFTER ITERATION %d",cnt);
mvwprintw(win,5,10,"STATUS OF NNW IS AS FOLLOWS :-");
wrefresh(win);
sleep(3);
endwin();
}

     /*******************************************************/
     /*   Function checks the validity of parameters,        */
     /*     suggests valid ranges of parameters              */
     /*******************************************************/

void paracheck()
{
WINDOW *win;
float m1,m2,m3,m4,m5,cmax;
int i,j;
char ch;

initscr();
win = newwin(20,60,2,10);
wattrset(win,A_ALTCHARSET);
box(win,ACS_VLINE,ACS_HLINE);

m1 = MU1;
m2 = MU2;
m3 = MU3;
m4 = MU4;
m5 = MU5;

mvwprintw(win,2,10,"PARAMETERS :-          MU1 : %f",m1);
mvwprintw(win,3,30,"MU2 : %f",m2);
mvwprintw(win,4,30,"MU3 : %f",m3);
mvwprintw(win,5,30,"MU4 : %f",m4);
mvwprintw(win,6,30,"MU5 : %f",m5);
mvwprintw(win,8,10,"CONDITIONS                         SATISFIED/NOT");
mvwprintw(win,9,10,"                                     (Y/N)  ");
mvwprintw(win,11,10,"MU5 = MU2");

cmax = 0;
for(i = 0; i < SIZE;i++)
     for(j =0; j< SIZE;j++)
     if(cost[i][j] > cmax)  cmax = cost[i][j];
mvwprintw(win,13,10,"MU5  >> MU1 * %f",cmax);
mvwprintw(win,15,10,"2 * MU3  - MU4 > 0");
mvwprintw(win,17,10,"MU1 < 2 * MU3/ %f ",cmax);
```

```
if(m5 == m2) ch = 'Y';
     else ch = 'N';
mvwprintw(win,11, 47,"%c",ch);
if(m5 > 3 * m1* cmax) ch = 'Y';
     else ch = 'N';
mvwprintw(win,13, 47,"%c",ch);
if(2 * m3 - m4 > 0) ch = 'Y';
     else ch = 'N';
mvwprintw(win,15, 47,"%c",ch);
if(m1 < 2 * m3/cmax)  ch = 'Y';
     else ch = 'N';
mvwprintw(win,17, 47,"%c",ch);

wrefresh(win);
sleep(5);
wgetch(win);
endwin();
return;
}

      /**************************************************************/
      /* Function extracts the route(s) from converged M-F model  */
      /**************************************************************/

void findpath()
{
int *ls[MAX];
int lcnt,elcnt,ch,plcnt,pele,proc;
int i,j,k,l,m,n,y,ins;
int RTE[RTMAX],lknt,crt;

lcnt = 0;
ls[0] = (int *)malloc(sizeof(int) * RTMAX);
for(i = 0; i < RTMAX; i++)
      ls[lcnt][i] = -1;
ls[0][0] = SCE -1;
lcnt = 1;
ch = 1;
/* Picking out the neuron outputs related to routes  */
while(ch)
{
      ls[lcnt] = (int *)malloc(sizeof(int) * RTMAX);
      for(i = 0; i < RTMAX; i++)
            ls[lcnt][i] = -1;
      plcnt = 0;
      elcnt = 0;
      while((y = ls[lcnt-1][plcnt++] ) >= 0)
      {
            for(i = 0; i < SIZE; i++)
            if(no[y][i] > 0.15)
            {
                  ins = 1;
                  for(j = 0; j < elcnt; j++)
                  if(ls[lcnt][j] == i) ins  = 0;
```

```
                if(ins != 0) ls[lcnt][elcnt++] = i;
            }
        }
        for(i = 0; i < elcnt ; i++)
            if(ls[lcnt][i] == DST-1) ch = 0;
        lcnt++;
}

do
{
/* Generating combinations                          */
        RTE[0] = SCE ;
        lknt = 1;
        pele = SCE - 1;
        ls[0][0] += HVAL;
        for(i = 1; i < lcnt ; i++)
        {
            for(j  = 0; j < RTMAX; j++)
            if(ls[i][j] >= 0 && ls[i][j] < HVAL)
            {
                if(cost[pele][ls[i][j]] > 0)
                {
                    RTE[lknt++] = ls[i][j] + 1;
                    pele = ls[i][j];
                    ls[i][j] += HVAL;
                    break;
                }
                else ls[i][j] += HVAL;
            }
        }
        crt = 0;
        for(i = 0; i <lknt;i++)
        {
            j = RTE[i];
            if(j == DST ) crt = 1;
        }
        proc = 1;
        for(i = lcnt -1; i >= 0 && proc ; i--)
            for(j =0; j<RTMAX && proc ; j++)
            if(ls[i][j] >= 0 && ls[i][j] < HVAL)
            {
                for( m =  i+1; m < lcnt; m++)
                for(n = 0; n < RTMAX; n++)
                if(ls[m][n] >= HVAL) ls[m][n] -= HVAL;
                for(m = i -1; m>=0; m--)
                    for(n = RTMAX -1; n >= 0; n--)
                    if(ls[m][n] >= HVAL)
                    {
                        ls[m][n] -= HVAL;
                        break;
                    }
                proc = 0;
            }
        if(crt == 1)
        {
```

```
     /* Result is accumulated if combination is a valid route */

     result.ROUTE[result.RTCNT] = (int *)malloc(sizeof(int)*(lknt));
     for(i = 0; i < lknt;i++)
          result.ROUTE[result.RTCNT][i] =  RTE[i] - 1;
     result.RTLINKS[result.RTCNT] = lknt;
     result.RTVAL[result.RTCNT] = (float *)malloc(sizeof(float)*(lknt));
     i = 0;
     while(i < lknt - 1)
     {
          m = RTE[i] -1;
          n = RTE[++i] -1;
          result.RTVAL[result.RTCNT][i-1] = no[m][n];
     }
     (result.RTCNT)++;
     }

     elcnt = 0;
     for(i =0;i < lcnt; i++)
          for(j =0; j< RTMAX; j++)
          if(ls[i][j] > 0  && ls[i][j] < HVAL)
          elcnt++;
}
while(elcnt != 0);
return;
}

     /*************************************************************/
     /* Function picks the route(s) from converged L-C model      */
     /*************************************************************/

void findroutes()
{
int *path;
int prev,pres;
int i,j,k,proc,m,n,cnt,RTE[RTMAX];

do
{
     for(i =0; i <= lks;i++)
     {
          for(j =0; j< SIZE; j++)
          if(vec[i]->ele[j] > 0 && vec[i]->ele[j] < HVAL)
          {
               if(vec[i]->ele[j] > 0.25)
               {
                    RTE[i] = j;
                    vec[i]->ele[j] += HVAL;
                    j = SIZE;
               }
          }
     }
     proc = 1;
     for(i = lks ; i > 0 && proc ; i--)
          for(j =0; j < SIZE && proc ; j++)
```

```
            if(vec[i]->ele[j] > 0  && vec[i]->ele[j] < HVAL)
            {
                 if(vec[i]->ele[j] > 0.25)
                 {
                 for(m = i+1; m <= lks;m++)
                      for(n=0; n< SIZE;n++)
                      if(vec[m]->ele[n] > HVAL)vec[m]->ele[n] -= HVAL;
                 for(m = i-1; m >= 0;m--)
                 {
                      cnt  =0 ;
                      for(n =0; n<SIZE; n++)
                      if(vec[m]->ele[n] > HVAL) cnt++;
                      if(cnt > 1)
                      {
                       for(n= SIZE -1; n>=0 ; n--)
                       if(vec[m]->ele[n] > HVAL)
                       {
                            vec[m]->ele[n] -= HVAL;
                            n = -1;
                       }
                      }
                      else
                      for(n=0; n< SIZE;n++)
                      if(vec[m]->ele[n] > HVAL)vec[m]->ele[n] -= HVAL;
                 }
                 proc = 0;
                 }
            }
       result.ROUTE[result.RTCNT] = (int *)malloc(sizeof(int)*(lks+1));
       for(i = 0; i <= lks;i++)
            result.ROUTE[result.RTCNT][i] =  RTE[i] ;
       result.RTLINKS[result.RTCNT] = lks + 1;
       result.RTVAL[result.RTCNT] = (float *)malloc(sizeof(float)*(lks +1));
       i = 0;
       while(i < lks )
       {
            m = RTE[i] ;
            if(vec[i]->ele[m] > HVAL)
            result.RTVAL[result.RTCNT][i] = vec[i]->ele[m] - HVAL;
            else
            result.RTVAL[result.RTCNT][i] = vec[i]->ele[m] ;
            i++;
       }
       (result.RTCNT)++;

       cnt =0;
       for(i =0; i<= lks ; i++)
            for(j =0; j< SIZE; j++)
            if(vec[i]->ele[j] > 0 && vec[i]->ele[j] < HVAL)
                 if(vec[i]->ele[j] > 0.25) cnt++;
  }while(cnt != 0);
```

```
for(i =0; i<= lks ; i++)
      for(j =0; j< SIZE; j++)
      if(vec[i]->ele[j] > HVAL) vec[i]->ele[j] -= HVAL;
return;
}

      /***************************************************************/
      /* Function determines the route from converged Z-T model     */
      /***************************************************************/

void findroute()
{
int *path;
int prev,pres;
int i,j,RTE[RTMAX],lknt;

path = (int *)malloc(sizeof(int) * SIZE);
for(j = 0; j < SIZE; j++)
{
      path[j] = 0;
      for(i =0; i < SIZE;i++)
      if(nnvec[i][j] > 0.45) path[j] = i+1;
}

i =0;
prev = 0;
pres = path[i];
lknt = 0;
while(i < SIZE)
{
      if(prev != pres)
            RTE[lknt++] = pres - 1;
      prev = pres;
      pres = path[++i];
}

result.ROUTE[result.RTCNT] = (int *)malloc(sizeof(int) * lknt);
for(i = 0; i < lknt;i++)
      result.ROUTE[result.RTCNT][i] =  RTE[i] ;
result.RTLINKS[result.RTCNT] = lknt;
result.RTVAL[result.RTCNT] = (float *)malloc(sizeof(float) * lknt);
i = 0;
while(i < lknt )
{
      result.RTVAL[result.RTCNT][i] = 1;
      i++;
}
(result.RTCNT)++;

return;
}
```

```
/********************************************************/
/*  Function computes the energy of the NN model        */
/********************************************************/

float nnw_energy(int lkcnt,int cse)
{
float E,temp,e1,e2,e3,e4,e5;
float *PA,temp1,temp2;
int i,j,k,x;

PA = (float *)malloc(sizeof(float) * MAX);
switch(cse)
{

case 1 :
      e1 = 0;
      for(j =0; j< lkcnt;j++)
      {
          matmult(w,vec[j+1]->ele,PA,SIZE,SIZE,i=1);
          for(i =0; i< SIZE; i++)
          e1 = e1 + vec[j] ->ele[i] * PA[i];
      }
      e1 *= 0.5;

      e2 = 0;
      for(j = 1; j < lkcnt ; j++)
      {
          for(i = 0; i < SIZE; i++)
          temp = temp + vec[j]->ele[i];
          temp = temp -1;
          temp *= temp;
          e2 = e2 +temp;
      }
      e2 *= ALPHA * 0.5;

      e3 = 0;
      for(j =0; j<=lkcnt ; j++)
      {
          matmult(d,q,PA,SIZE,SIZE,1);
          for(i =0; i < SIZE; i++)
          e3 = e3 + vec[j]->ele[i] * PA[i];
      }
      e3 *= BETA;

      E = e1 + e2 + e3;
      free(PA);
      break;

case 2 :
     e1 = 0;
     for(k =0; k < SIZE; k++)
          for(i =0; i < SIZE ; i++)
               for(j =0; j < SIZE; j++)
               e1 = e1 + nnvec[i][k] * c[i][j] * nnvec[j][k+1];
     e1 *= 0.5 * A;
```

```
          e2 = 0;
          for(k =0; k < SIZE ; k++)
               for(i =0; i < SIZE; i++)
                    for(j=0; j < SIZE ; j++)
                    e2 = e2 + nnvec[i][k] * nnvec[j][k];
          e2 *= 0.5 * B;

          e3 =0;
          for(i =0; i< SIZE; i++)
               for(j =0; j< SIZE; j++)
               e3 = e3 + nnvec[i][j];
          e3 -= SIZE;
          e3 *= e3;
          e3 *= 0.5 * e3 ;

          E = e1 + e2 + e3 ;
          break;

     case 3 :
          e1 =0;
          for(x =0; x < SIZE; x++)
               for(i =0; i < SIZE; i++)
               if(x != SCE  || i != DST)
               if(x != i) e1 +=  cost[x][i] * no[x][i];
          e1  *= MU1  * 0.5;

          e2 =0;
          for(x =0; x < SIZE; x++)
               for(i =0; i < SIZE; i++)
               if(x != SCE  || i != DST)
                    if(x != i)
                    if(cost[x][i] > 0) e2 +=  no[x][i];
          e2  *= MU2  * 0.5;

          e3 =0;
          for(x =0; x < SIZE; x++)
               for(i =0; i < SIZE; i++)
               {
                    if(i != x)
                    {
                         temp1 +=  no[x][i];
                         temp2 +=  no[i][x];
                    }
               temp1 = temp1 - temp2;
               e3 += temp1  * temp1;
               }
          e3  *= MU3  * 0.5;

          e4 =0;
          for(i =0; i < SIZE; i++)
               for(x =0; x < SIZE; x++)
               if(x != i) e4 += no[x][i] * (1 - no[x][i]);
          e4 *= MU4  * 0.5;
```

```
        e5 = MU5 * 0.5 * (1 - no[SCE][DST]);

        E = e1 + e2 + e3 + e4 + e5;
        break;

    }
    return E;
    }

    float *matmult(float a[][MAX],float b[MAX],float *ci,int S,int R,int T)
    {
    /* Function performs the Multiplication of vector and matrix  */
    int i,j,k;

    for(i=0; i<S; i++)
        for(j=0; j<T; j++)
        {
            ci[i] = 0;
            for(k =0; k < R; k++)
                ci[i] = ci[i] + a[i][k] * b[k] ;
        }
    return ci;
    }

        /************************************************************/
        /* Function computes preferences of route(s),their costs  */
        /************************************************************/

    void  pathpref(int cse)
    {
    float total[RTMAX],ctotal;
    int i,j,ond,nnd;
    WINDOW *win;

    ctotal = 0;
    for(i = 0; i < result.RTCNT; i++)
    {
        total[i] = 1;
        j = 0;

        while( j < result.RTLINKS[i] - 1)
            total[i]  *=  result.RTVAL[i][j++];
        ctotal += total[i];
    }

    for(i = 0; i < result.RTCNT; i++)
        result.PREF[i] = total[i] / ctotal;
    for(i = 0; i < result.RTCNT;i++)
    {
        j = 0;
        result.RTCOST[i] = 0;
        while(j < result.RTLINKS[i] - 1)
        {
            ond = result.ROUTE[i][j] ;
            nnd = result.ROUTE[i][++j] ;
            switch(cse)
            {
            case 1 :
                 result.RTCOST[i] += w[ond][nnd];
                 break;
            case 2 :
                 result.RTCOST[i] += (c[ond][nnd] - 2);
                 break;
            case 3 :
                 result.RTCOST[i] += cost[ond][nnd];
                 break;
            }
       }
  }
  }
```

APPENDIX - II

A2.1 PARAMETER CALCULATION IN LEE-CHANG MODEL

The neural network model with the energy function (3.1) has a stable solution. All the elements of matrix W in E are positive clearly first term is positive. Second term is summation of square terms, it is certainly positive. Therefore energy function E is bounded below by 0. The change in energy can be written as

$$\overset{\Delta}{\nabla} E = E(U(i+1)) - E(U(i))$$
$$= -\frac{1}{\alpha} \sum_{j=1}^{h} [\Delta U_j^T \, \Delta U_j] < 0 \qquad \text{(A2.1)}$$

Therefore the state changes result in a decrease in the value of the energy function and also energy function is bounded below. Hence the system has a stable solution. To improve the convergence speed, parameters should be chosen carefully.

There are three terms in ΔU_k of (3.3). The third term of ΔU_k is always positive, for the assumption that the sum of all elements in vector U_k is close to 1 but cannot be greater than 1. Let θ be the minimum value of $\Sigma_i \mu_i^k$ since $\Sigma_i \mu_i^k < 1$

$$\left|\alpha\gamma e_n \left[\sum_i u_i^k -1\right]\right| = \alpha\gamma n \left| \sum_i u_i^k -1\right| \le \alpha\gamma n\ (1-\theta) \qquad \text{(A2.2)}$$

If ($\theta \ne 1$), in order to keep the values of U_k within range, i.e. close to 1,

$$\alpha\gamma < \mu/(1-\theta)n \qquad \text{(A2.3)}$$

Where $\mu$is a positive constant. If the value of μ is small, then the size of correction step is small so that the speed of convergence is slow. On the other hand, if μ is large then the convergence speed will be fast. However if the value is too large i.e. the correction step is too large then it has a high probability that U_k to go out of range. Typical range of μ is [0.2,1).

The first two term of ΔU_k are always negative. The bounds of the two terms are same. They can be written as

$$\alpha \frac{1}{2}\theta\, w_{min} \le \alpha \frac{1}{2}\theta\, \psi_i \le \alpha \frac{1}{2}\, w_{max} \qquad \text{(A2.4)}$$

where w_{min} and w_{max} are minimal and maximal elements in W, and ψ_i is the element in column vector WU_{k-1} or WU_{k+1}. Since elements in Δu_k cannot be all positive or negative each element of must lie between the upper and lower bounds of (A2.4). Hence the following inequality

$$\theta\, w_{min} \le \gamma\ (1-\theta) \le \alpha\, w_{max} \qquad \text{(A2.5)}$$

The constraint (A2.3) and (A2.5) can be used as upper and lower bounds for parameters α and γ.

A2.2 PARAMETER CALCULATION IN MUSTAFA-FAOUZI MODEL

General guidelines to select the coefficients of energy function so that neural dynamics will converge to a valid path which is also of minimum length are given here for Mustafa-Faouzi model.

The shape of the energy surface is tailored by the memory terms $T_{xi,yi}$(3.9) and analog prompt terms I_{xi}(3.10). The memory and prompt terms create a set of finite local energy attractors, (valid basins of attraction) with equal depth (E=0) each corresponding to a feasible path. The role of linear μ_1 cost term is then to provide a negative bias that enlarges the depth of these valid basins of attraction by varying amounts, depending on the corresponding path cost. Therefore from (3.6) the global minimum corresponds to that valid basin of attraction which complies with the shortest path requirement. The ability to reshape the energy landscape through the link cost bias terms, while keeping the interactions among neurons relatively simple (since memory terms do not incorporate the link costs) is the salient feature of this neural network model.

The quadratic energy function is characterized by the presence of valleys, where among all points forming a valley some are low points corresponding to a local minimal state. To ensure that every valley has only one low point and hence to provide a graceful descent along the energy surface, we require.

$$\frac{\partial^2 E}{\partial v^2_{xi}} > 0$$

This corresponds to having $2\mu_3 - \mu_4 > 0$

The μ_5 term should be relatively large so that from the early stages of the neural computation a unity output for the neuron at location (d,s) will be enforced and hence the construction of the shortest path will be initiated. In the initial state of the neural network, with all neuron inputs being set to zero, for the neuron at location (d,s), the input activation increases at a rate

$$R_1 = \left.\frac{du_{xi}}{dt}\right|_{(x,i) = (d,s),\ \text{intial state}} = \frac{\mu_5}{2} > 0$$

Among the remaining neurons those corresponding to non existing arcs will have their input decreasing at a rate

$$R_2 = \left.\frac{du_{xi}}{dt}\right|_{(x,i) \neq (d,s),\ \text{intial state}} = -\frac{\mu_2}{2}$$

While those corresponding to existing arcs will have their inputs decreasing at a rate proportional to their corresponding link costs, namely

$$R_3 = \frac{\mu_3}{2} \cdot w_{xi}$$

Therefore, in order to speed up the construction of the valid path

$\mu_5 >> \mu_1 \cdot (w_{xi})_{max}$ is required.

An equally important requirement is to prevent nonexisting arc from being part of the solution, it is reasonable to require $R_1 = R_2$ or $\mu_2 = \mu_5$

From (3.4) it is clear, that by increasing the μ_1 term (with remaining μ_is being unchanged) shortest path algorithm gradually refines the quality of the solution, hence minimizing the chance of getting the neural network state trapped in an "attractive" unfavorable local minimum. However μ_1 cannot be increased indefinitely, since once it exceeds a threshold value, the neural algorithm starts to diverge and gives invalid solutions, as the effect of μ_3 energy term will be shaded by stiff cost requirement. Therefore, μ_1 should be maximized at the same time solution must be obtained. Assuming that for a valid neural output one neuron corresponding to an existing arc, changes output from 1 to 0. In this case, the energy term associated with the weighting coefficient μ_3 will increase by μ_3, while the energy term associated with μ_1 will decrease by a maximum value of $(\mu_1/2)\cdot(w_{xi})_{max}$. Hence for the net to reach a valid path μ_1 should satisfy

$$\mu_1 < 2 \frac{\mu_3}{(w_{xi})_{max}} .$$

APPENDIX - III

Incorporating reliability of nodes and dependency between nodes in neural network algorithm can be done as follows :

The failure probability of every node in the network can be represented by an nx1 vector Q.

$Q = [q_1\ q_2\ \dots\ q_n]^T$

where, q_i represents the probability of failure of node i,

The dependency between any two nodes can be represented by any number d ($0 \le d \le 1$), where $d=1$ indicates complete dependency between nodes. The dependency relationships of entire communication network can be represented by an nxn symmetric matrix D, where each entry d_{ij} stands for probability of dependence of node j on node i. Assuming the node failures are mutually dependent, the term of the energy function that includes the uncertainties of each node can be represented as follows :

$$E3 = \beta \sum_{j=1}^{h+1} U_j^T D\, Q \qquad \text{(A3.1)}$$

Now the resultant energy function will be the summation of (3.1) and (A3.1). The change in vector U_k, ΔU_k now becomes.

$$\Delta U_k = -\alpha(1/2)[WU_{k-1} + WU_{k+1}] - \alpha\beta DQ + \alpha\gamma e_n(1 - \sum U_i^k) \quad \text{(A3.2)}$$

The second condition for calculating the lower and upper bounds of parameters becomes

$$(\theta w_{min} + \beta q_{min}) \le \gamma(1 - \theta) \le \alpha(w_{max} + \beta(n-1)dq_{max} + \beta q_{max}) \text{ (A3.3)}$$

APPENDIX - IV

The routing algorithm minimizing expected delay usually assumes some knowledge of overall traffic pattern.

Expected delay across a link is usually considered [M/M/1 queueing systems] as a function of capacity C_{ij} of the link and the actual traffic a_{ij} on the link. Some of the functions [1] are

$$* \quad F_1 (a_{ij} , C_{ij}) = \rho/(1-\rho),$$

$$\rho = (a_{ij}/C_{ij})$$

The loss function is proportional to the average delay per message across a link, when the message arrival rate and required transmission time can be represented as poisson [exponential distribution] processes.

When Kleinrock independence approximation [11] is used, the sums of delay across all links will be assumed proportional to the sum of loss over whole network.

$$* \quad F_2(a_{ij}, C_{ij}) = F_o + \rho \exp(p)$$

where F_o indicates transmission time over the link.

Here the loss increases exponentially as the traffic increases.