PERGAMON

Neural Networks 15 (2002) 1289–1298

Neural Networks

www.elsevier.com/locate/neunet

Neural network based optimal routing algorithm for communication networks

Pallapa Venkataram*, Sudip Ghosal, B.P. Vijay Kumar

Protocol Engineering and Technology (PET) unit, Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore 560 012, India

Received 13 January 2002; revised 17 May 2002; accepted 17 May 2002

Abstract

This paper presents the capability of the neural networks as a computational tool for solving constrained optimization problem, arising in routing algorithms for the present day communication networks. The application of neural networks in the optimum routing problem, in case of packet switched computer networks, where the goal is to minimize the average delays in the communication have been addressed.

The effectiveness of neural network is shown by the results of simulation of a neural design to solve the shortest path problem. Simulation model of neural network is shown to be utilized in an optimum routing algorithm known as *flow deviation* algorithm. It is also shown that the model will enable the routing algorithm to be implemented in real time and also to be adaptive to changes in link costs and network topology. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Communication network; Routing; Hopfield neural network

1. Introduction

In a packet switched computer network, routing is an important factor that has a significant impact on the network's performance. The main aim of the routing algorithm is to find the 'OPTIMUM' path(s) for data transmission within a very short time to satisfy the users' demand. In terms of network definitions, the optimality of the routing algorithm, that means, to optimize a selected performance such as network throughput or mean packet delay. The computations have to be carried out in real time. This makes the Neural Network (NN) as an ideal candidate for implementation because NN possesses immense capability for processing data in parallel thus speeding up the computation. In almost all of the routing algorithms in the currently operating packet switched networks some forms of shortest path (SP) computation is essentially required in the network layer (OSI stack) (Hopfield & Tank, 1985; Topkis, 1988; Wang, 1996). This paper deals with shortest path computation in real time.

1.1. Related works

Routing is a critical issue in packet-switched networks due to its significant impact on the network performance. Existing algorithms (Cantor & Gerla, 1974; Gallenger, 1977; Wang & Browning, 1991) performance measures. The routing problem is one of the important issue to be studied in computer communications. In the beginning of 1970s, many researchers (Frank & Chou, 1971; Yaged, 1973) focused on developing heuristic algorithms in order to reduce the computational complexity and thus ensure the responsiveness of the algorithms. Thereafter, research interests in searching for both rigorous and computationally efficient mathematical methods surged. However, as the transmission rate of future communication networks will be extremely high, a much faster response of routing algorithms must be guaranteed. Therefore, traditional mathematical methods may face challenges, and new faster routing algorithms will be needed to meet this requirement. The application of neural networks to the routing problem has been motivated by the idea of taking advantages of the powerful computational ability of the neural network and the fact that a hardware-implemented neural network can achieve high response speeds. In (Park & Lee, 1995; Wang & Weisseler, 1995) some neural network applications in high speed communication networks are dealt with for

* Corresponding author. Tel.: +91-80-309-2282; fax: +91-80-360-0991.
E-mail addresses: pallapa@ece.iisc.ernet.in (P. Venkataram), sudip@ece.iisc.ernet.in (S. Ghosal), vijaybp@ece.iisc.ernet.in (B.P. Vijay Kumar).

optimal routing. In (Marbach, Mihatsch, & Tsitsiklis, 2000) the use of neural network to solve the call admission control and routing problem in high speed networks was discussed. In (Pornavalai, Chakraborty, & Shiratori, 1995) a modified version of a Hopfield neural network model to solve QoS (delay) constrained multicast routing is proposed and showed that the proposed model has performed well nearer to the optimal solution and are comparable to existing heuristics. A neural network based shortest-path algorithm was developed in (Mustafa, Ali, & Kamoun, 1993), by means of which the implementation of the optimal routing problem was discussed.

1.2. Solving problems with neural networks

A neural network is a massively parallel defined distributed processor that jags a natural propensity for storing experiential knowledge and making it available for use (Yang & Dillon, 1994). The requirement from any typical neural algorithm are:

1. The computation should be carried out in real time.
2. The algorithm should adapt to the changes in the link costs.
3. The overhead (memory requirement plus CPU load) should be as low as possible.
4. The algorithm should adapt to the changes in the network topology.

There are many neural network based techniques available to solve varieties of realistic problems.

1.3. The problem

We employ neural networks to solve the problem of finding the shortest path of a given communication network between a source and destination nodes. Recurrent type of neural net is employed to solve the problem. The hardware description of the neural network architecture is also described.

Rest of the paper is organized as follows. Section 2 deals with the neural computational approaches for solving shortest paths, Section 3 discusses the SP problem formulation in neural network domain. Section 4 describes the energy function generation, parameters tuning has been given in Section 5. Section 6 elaborates the simulation carried out and discusses the results obtained. And finally, Section 7 concludes with the remarks on the neural network architecture and how far this computational tool fits into real-time application scenario.

2. Neural computation approach for solving shortest path problem

In many situations, a very good answer computed on a time scale short enough so that the solution can be used in the choice of appropriate action is more important than a normally better solution (Johannet, Personnaz, Dreyfus, Gasquel, & Weinfield, 1994; Mustafa et al., 1993). This feature is achieved in an artificial neural network by collective analog computation circuits. The speed of the computation is immense because all of the neurons continuously and simultaneously change their states in parallel. A continuous decision space and continuous computation can improve the quality of solutions obtained by a highly interconnected neural network.

The neural network architecture is discussed in Section 2.1.

2.1. Neural network architecture

The architecture chosen is Hopfield Tank type of neural network as shown in Fig. 1. These networks have three major forms of parallel organization found in neural systems (Mortara & Vottoz, 1994; Waurzynek, Asannovic, & Morgan, 1993): parallel input channels, parallel output channels, and a large amount of interconnectivity between the neural processing elements. The processing elements or the *neurons* are modeled as amplifiers in conjunction with feedback circuits comprised of wires, resistors and capacitors so as to model the most basic computational features of neurons, namely axons, dendrites and synapses. The amplifiers have sigmoid monotonic input/output relations (Massengil & Mundie, 1992). From the simple circuit theory analysis the dynamics of the circuit is shown to be:

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{u_i}{\tau_i} + \sum_{j=1}^{n} T_{ij}v_j + I_i \tag{1}$$

where $T_{ij} = \sum_{j=1}^{n} 1/R_{ij}C$, $I_i =$ Input bias current/C, $R_i = 1/\rho_i + \sum_{j=1}^{n} 1/R_{ij}$, $\tau_i = R_iC_i$ and $v = g(u) = 1/(1+\mathrm{e}^{-\lambda u})$, u's are the inputs to the amplifiers and v's are the outputs of the amplifiers. R_{ij} is the value of the resistance connecting the opamps j and i. ρ is the resistance connected with the opamp and the ground. C is the capacitance connected with the amplifiers and the ground. Both of them are included to simulate the effect of *cell impedance*.

Hopfield showed that the computational circuit shown in Fig. 1 minimizes a function, which he coined *energy function*:

$$\mathbf{E} = -\frac{1}{2}\mathbf{V}^t\mathbf{T}\mathbf{V} - \mathbf{I}^t\mathbf{V} + \frac{1}{\tau}\sum_{i=1}^{n}\int_0^{v_i} g^{-1}(x)\mathrm{d}x \tag{2}$$

It was also shown that when the amplifiers are operated in high gain limit the energy function reduces to (Hopfield & Tank, 1985):

$$\mathbf{E} = -\frac{1}{2}\mathbf{V}^t\mathbf{T}\mathbf{V} - \mathbf{I}^t\mathbf{V}. \tag{3}$$

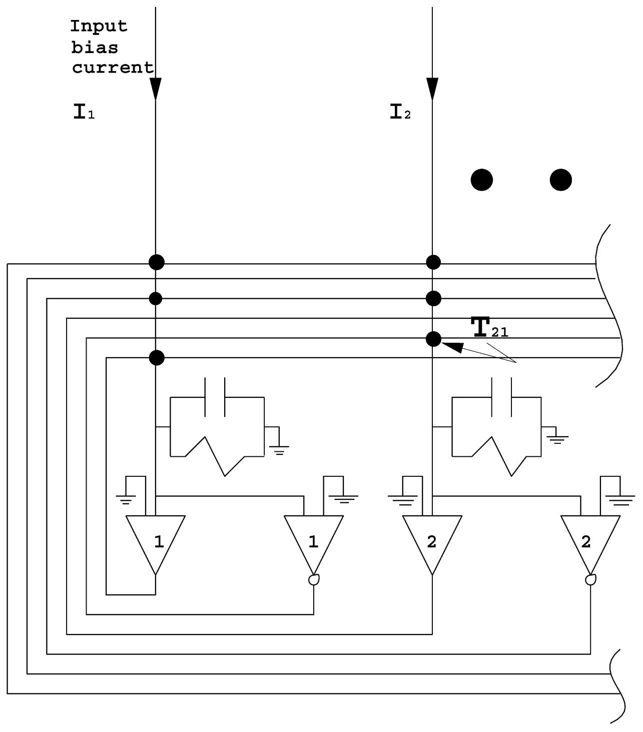


Fig. 1. Model of Hopfield tank neural network.

3. SP problem formulation in neural network domain

To formulate the SP problem in terms of the architecture shown in Fig. 1, a suitable representation scheme must be found so that the shortest path can be decoded from the final stable state of the neural network. If there are total n nodes existing then the neurons are arranged in $n \times n$ matrix with all diagonal elements removed. Each element in the matrix is represented by double indices (x, i), where the row subscript, x and column subscript, i denotes node numbers. As the diagonal elements are removed hence the total number of neurons required is $n(n-1)$. A neuron at location (x, i) is characterized by its output V_{xi} defined as follows:

$$V_{xi} = \begin{cases} 1 & \text{if the arc from the node } x \text{ to node } i \text{ is on the shortest path,} \\ 0 & \text{Otherwise.} \end{cases} \tag{4}$$

To characterize a partially connected network, we define γ_{xi} as:

$$\gamma_{xi} = \begin{cases} 1 & \text{if the arc from the node } x \text{ to node } i \text{ does not exist,} \\ 0 & \text{Otherwise.} \end{cases} \tag{5}$$

A capacity matrix is defined. The cost of an arc from node x to node i will be denoted by C_{xi}. This number is assumed to be real non-negative number. For non-existing arcs, the corresponding entries will be zero. This capacity matrix is actually the cost of each link and will remain constant during execution of the shortest path finding algorithm. Along with the capacity matrix we also defined a weight vector B_x, the weight vector is actually the cost of each node and cost of each node will remain constant during execution of the algorithm. The energy function formulated is shown below with the elements defined earlier:

$$E = \frac{\alpha_1}{2}\sum_{x=1}^{n}\sum_{\beta}^{n} C_{xi}V_{xi} + \frac{\alpha_2}{2}\sum_{x=1}^{n}\sum_{\beta}^{n} \gamma_{xi}V_{xi} + \frac{\alpha_3}{2}\sum_{x=1}^{n}\sum_{\beta}^{n} B_x V_{xi}$$
$$+ \frac{\alpha_4}{2}\sum_{x=1}^{n}\left\{\sum_{i=1, i\neq x} V_{xi} - \sum_{i=1, i\neq x}^{n} V_{ix}\right\}^2$$
$$+ \frac{\alpha_5}{2}\sum_{i=1}^{n}\sum_{x=1, x\neq i}^{n} V_{xi}(1 - V_{xi}) + \frac{\alpha_6}{2}(1 - V_{ds}) \tag{6}$$

where β is defined:

$\beta \triangleq= i = 1,\ i \neq x,\ (x, i) \neq (d, s)$

The justification of choosing this energy function is given below:

1. The α_1 term minimizes the total cost of a path by taking into account the cost of the existing links. The cost can be defined according to the specific application.
2. The α_2 term prohibits non-existent links to be included in the chosen path.
3. The α_3 term minimizes the total cost of a path by taking into account the cost of the existing nodes. Here cost can be defined according to the specific applications, like buffer availability, processor speed, etc.
4. If a node is entered during traversal it should be exited otherwise the path will not be a valid one. The energy function is defined to find a circuit with a hypothetical link connecting the destination and the source rather than a path from the source to destination. This eases the formulation. Hence with this assumption the α_4 term makes sure that if a node has been entered it will also be exited by a path. This is evident because this term will be zero if for every node in the solution, the number of incoming arcs equals the number of outgoing arcs.
5. In order to get a meaningful result from the neural network it is desired that the stable state should be such that it can be decoded. Hence, we require each of the neurons to be either in ON or OFF state. α_5 does exactly that by pushing the state of the neural network to converge to one of the 2^{n^2-n} corners of the hypercube, defined by $V_{xi} \in \{0, 1\}$.
6. It is required that the source (s) and destination (d) be always in solution. The link from destination to source is a hypothetical one. The term α_6 is zero when the output of the neuron at the location (d, s) settles to one. Thus, the final solution will always be a loop, with nodes s and d

included. This loop will consist of two parts:

- a directed path from source to destination,
- an arc from d to s.

4. Energy function transformation

In this section the interconnection strengths and the bias currents are obtained from the energy function formulated in Section 3. The energy function is to be mapped into the connection matrix and the input bias currents. Rewriting the dynamics of the neurons with double indices (as in our case each neuron is represented by a double index (x, i)), we have:

$$\frac{\mathrm{d}u_{xi}}{\mathrm{d}t} = -\frac{u_{xi}}{\tau} + \sum_{y=1}^{n} \sum_{j=1, j\neq y}^{n} T_{xi,yj} V_{yj} + I_{xi} \tag{7}$$

$$\frac{\mathrm{d}u_{xi}}{\mathrm{d}t} = -\frac{u_{xi}}{\tau} - \frac{\partial E}{\partial V_{xi}} \tag{8}$$

The activation function is given by:

$$V_{xi} = g_{xi}(u_{xi}) = \frac{1}{1 + \mathrm{e}^{-\lambda_{xi} u_{xi}}} \qquad \forall (x, i) \in \bar{N} \times \bar{N} / x \neq i \tag{9}$$

Substituting the energy function formulated in Eq. (6) into Eq. (8) the equation of motion becomes:

$$\begin{aligned} \frac{\mathrm{d}u_{xi}}{\mathrm{d}t} = {} & -\frac{u_{xi}}{\tau} - \frac{\alpha_1}{2} C_{xi}(1 - \delta_{xd}\delta_{is}) - \frac{\alpha_2}{2} \gamma_{xi}(1 - \delta_{xd}\delta_{is}) \\ & - \frac{\alpha_3}{2} B_x (1 - \delta_{xd}\delta_{is}) - \alpha_4 \sum_{y=1, y\neq x}^{n} (V_{xy} - V_{yx}) \\ & + \alpha_4 \sum_{y=1, y\neq i} (V_{iy} - V_{yi}) - \frac{\alpha_5}{2}(1 - 2V_{xi}) + \frac{\alpha_6}{2} \delta_{xd}\delta_{is} \\ & \forall (x, i) \in \bar{N} \times \bar{N} / x \neq i \end{aligned} \tag{10}$$

In the earlier equation δ is the Kronekar delta.

In order to get the connection matrix terms Eqs. (7) and (10) are compared. The coefficients are actually compared. The connection strengths and the bias terms are derived through simple comparison from the earlier two equations. They are:

$$T_{xi,yj} = \alpha_5 \delta_{xy} \delta_{ij} - \alpha_4 \delta_{xy} - \alpha_4 \delta_{ij} + \alpha_4 \delta_{jx} + \alpha_4 \delta_{iy} \tag{11}$$

$$\begin{aligned} I_{xi} = {} & -\frac{\alpha_1}{2} C_{xi}(1 - \delta_{xd}\delta_{is}) - \frac{\alpha_2}{2} \gamma_{xi}(1 - \delta_{xd}\delta_{is}) - \frac{\alpha_3}{2} B_x \\ & \times (1 - \delta_{xd}\delta_{is}) - \frac{\alpha_5}{2} + \frac{\alpha_6}{2} \delta_{xd}\delta_{is} \end{aligned} \tag{12}$$

The earlier expression of the bias term can also be written concisely as:

$$I_{xi} = \begin{cases} \dfrac{\alpha_6}{2} - \dfrac{\alpha_5}{2} & \text{if } (x, i) = (d, s) \\ -\dfrac{\alpha_1}{2} C_{xi} - \dfrac{\alpha_2}{2} \gamma_{xi} - \dfrac{\alpha_3}{2} B_x - \dfrac{\alpha_5}{2} & \text{Otherwise } \forall (x \neq i), \forall (y \neq i). \end{cases} \tag{13}$$

There are some important features associated with these equations. There are no terms in the connection matrix which depends on the cost of the links and nodes. Thus the connection matrix is independent of the changes in the flow of the network. The costs terms are mapped into the biases. The advantage is immense because in this scheme the link costs C_{xi}'s, node costs B_x and the network topology information embedded in the γ_{xi}'s terms can be changed to adapt to the current operating situation by simply changing the bias currents. One need not change the internal parameters of the neural network (the connections between different neurons) to adapt to the changes in the environment. This is a major advantage because these changes are done in real time. Thus the chip is a general one, not dependent on the network topology and link costs. One more advantage is that it is not algorithm dependent, i.e. any routing algorithm can incorporate this chip to compute the shortest path in real time.

5. Parameter tuning

The computation of the neural network is heavily dependent on the parameters. The parameters should be chosen in such a way that the neural network approaches towards a valid solution. For parameter tuning the formulated energy function is analyzed. The shape of the energy surface is characterized by the connection matrix terms and the input bias terms. First the convergence to a valid path is analyzed, ignoring the cost terms of the energy function which serves to minimize the tour length. Without the cost term it is easy to visualize that the energy landscape will have a set of finite local energy attractors (local minima) with equal depth, $E = 0$, each corresponding to a feasible path. Second the cost term will provide a bias that will enlarge the depth of these local minima by varying amounts, depending on the corresponding cost terms. Hence, the global energy minimum corresponds to the shortest path requirement.

The energy function is quadratic and continuous. Hence it is differentiable. The energy surface is characterized by the presence of valleys, where among all point forming the valley, some 'low' points can be identified, each of which corresponds to a local minima. In order to ensure that every valley has only one low point and hence to provide a descent along the surface of the energy function, we require:

$$\frac{\partial^2 E}{\partial V^2} > 0 \qquad \forall (x, i). \tag{14}$$

When this condition is put into the energy function, the corresponding constraint obtained is:

$$2\alpha_4 - \alpha_5 > 0 \tag{15}$$

It is already explained in Section 3 that the α_5 term serves to keep the source and destination term in the solution. This α_5 term should be large enough so that from the early stages of the neural computation a unity output for the neuron at location (d,s) will be enforced. To visualize the initial dynamics of the neural network the initial derivatives of the input function is to be analyzed. Let all the inputs of the neurons are set to zero. For a neuron at location (d,s) the input increases at the rate (initially):

$$R_1 = \frac{\mathrm{d}u_{xi}}{\mathrm{d}t}\Big|_{(x,i)=(d,s)} = \frac{\alpha_6}{2} > 0. \tag{16}$$

Among the remaining neurons, those corresponding to non-existing arcs will have their inputs decreasing at a rate (initially):

$$R_2 = \frac{\mathrm{d}u_{xi}}{\mathrm{d}t}\Big|_{(x,i)\neq(d,s)} = -\frac{\alpha_2}{2} \tag{17}$$

while those corresponding to existing arcs will have their inputs decreasing at a rate proportional to their corresponding link and node cost:

$$R_3 = -\frac{\alpha_1}{2}C_{xi} - \frac{\alpha_3}{2}B_x. \tag{18}$$

Hence to speed up the construction of the valid path, it is required:

$$\alpha_6 \gg \alpha_1(C_{xi})_{\max} + \alpha_3(B_x)_{\max}. \tag{19}$$

Also it is an equal requirement that the non-existing arcs should not enter into the solution. So it is assumed to be equally important that $R_1 = R_2$, in terms of parameters:

$$\alpha_2 = \alpha_6 \tag{20}$$

To determine a relation between the other parameters some qualitative analysis is to be done. If the α_1 term is increased the SP algorithm will gradually refine the quality of the solution. Hence the neural network will get trapped in better minima by increasing the α_1 term. This value cannot be increased indefinitely. If a threshold value is crossed then the neural network will diverge and will start to give invalid solutions. This happens because the α_1 term will override the α_4 term, which is necessary for a valid solution. To find a compromise between these two terms we consider the following scenario:

- For a valid neural output, one neuron corresponding to an existing arc, changes its output from 1 to 0. In this case the energy term associated with the weighing coefficient α_4 will increase by α_4, while the energy term associated with α_1 will decrease by a maximum value of $\alpha_1/2(C_{xi})_{\max}$ and α_3 will decrease by a maximum value of $\alpha_3/2(B_x)_{\max}$. Hence, for the neural network to reach a valid path α_1 and α_3 should satisfy:

$$\alpha_1 < 2\frac{\alpha_4}{(C_{xi})_{\max}} \text{ and } \alpha_3 < 2\frac{\alpha_4}{(B_x)_{\max}} \tag{21}$$

The remaining parameter is the activation function. It is shown in (Mortara & Vottoz, 1994) (Cybenko's Theorem) that there always exists a monotonic function that can solve the function approximation problem. In the simulation the activation function utilized (advocated in neural network literature) is given in equation Eq. (9). The parameter λ_{xi} are all set to be equal. It is evident that a larger value of λ will ensure fast convergence but the problem with this large value is that the solution will not converge to global minimum as the value of this parameter is increased. In a nut shell, the value of λ calls for a compromise between fast convergence and the quality of the solution in terms of the tour length. The parameters actually utilized is given in the next section and the simulation results are also reported.

6. Simulation and results

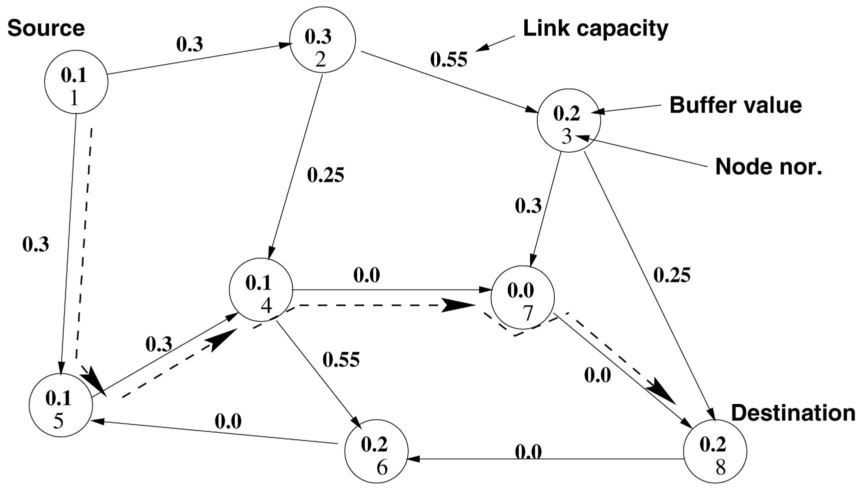
To discuss the optimal path computation consider an example network given in Fig. 2, where each node with a node cost (normalized between 0 and 1) whose value indicates the buffers allocated to the running applications (buffer matrix B). The arc represents the link between two nodes with link cost representing the normalized bandwidth value that is allocated to the running applications (capacity matrix C). Buffer matrix B and capacity matrix C for the example network in Fig. 2 are as shown in Fig. 3. Now the goal is to find an optimal path (a path with minimum allocated network resources, i.e. minimum cost) between the requested source and destination node. This optimal path problem is to be mapped into neural network domain for computation. The equation governing the dynamics of the neural network and formulation of the optimal path problem in terms of the NN architecture are already discussed.

6.1. Optimal path algorithm

The algorithm used in simulation for optimal path finding is as follows:

The NN_Algorithm designed uses four basic functions:

- *Int_act_vector*( ) generates a random input vector and presents this vector to the network as an initial inputs to the neuron, and then determines the initial activations of the neurons.
- *Get_wt_bias*( ) takes capacity matrix C and buffer matrix B as an input data, and calculates the weights on the connections between the neurons and bias values for each neurons. Values of different tuning parameters



(Dotted line shows the path computed by the neural network)

Fig. 2. Simulation example with eight nodes.

chosen to satisfy the constraints are:

$\alpha_1 = 100 \quad \alpha_2 = 3000 \quad \alpha_3 = 500$

$\alpha_4 = 550 \quad \alpha_5 = 825 \quad \alpha_6 = 3000$

- *Eq_motion*( ) updates the neuron inputs in steps, with a step size $\delta t = 0.00001$.
- *Get_outs*( ) function finally computes the output activation values and determines the optimal path.

The minimum path vector is taken as the final optimal path.

NN_Algorithm (input: C, B matrix, iterate; Output: optimal path)

$$\begin{bmatrix} 999 & 0.3 & 999 & 999 & 0.3 & 999 & 999 & 999 \\ 999 & 999 & 0.55 & 0.25 & 999 & 999 & 999 & 999 \\ 999 & 999 & 999 & 999 & 999 & 999 & 0.3 & 0.25 \\ 999 & 999 & 999 & 999 & 999 & 0.55 & 0.0 & 999 \\ 999 & 999 & 999 & 0.3 & 999 & 999 & 999 & 999 \\ 999 & 999 & 999 & 999 & 0.0 & 999 & 999 & 999 \\ 999 & 999 & 999 & 999 & 999 & 999 & 999 & 0.0 \\ 999 & 999 & 999 & 999 & 999 & 0.0 & 999 & 999 \end{bmatrix}$$

Capacity Matrix (C)

$$\begin{bmatrix} 0.1 & 0.3 & 0.2 & 0.1 & 0.1 & 0.2 & 0.0 & 0.2 \end{bmatrix}$$

Buffer Matrix (B)

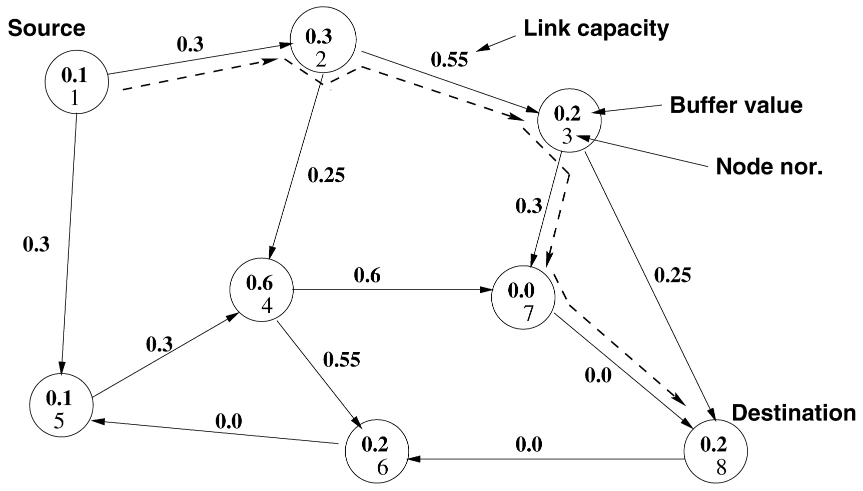
Fig. 3. Capacity matrix (C) and buffer matrix (B) for the example network in Fig. 2.

```
{
/* $C$ = Capacity Matrix */
/* $B$ = Buffer matrix as in Fig. 3 */
begin
    $i$ = iterate;
    initialize count = 0; /* count = No. of iteration */
    repeat
        begin
            Increment the count;
            Int_act_vector( );
            Get_wt_bias($C$, $B$);
            iterate Eq_motion( ) $i$ times;
            Get_outs( );
        end
    until(count = 10000 or neuron update threshold value
    ($10^{-3}$) is reached);
    output the optimal path vector;
end.
}
```

6.2. Results for optimal path

The simulation is carried out on eight node network shown in Fig. 2, along with the obtained result. The source and destination are the nodes labeled 1 and 8, respectively. The obtained optimal path based on buffer and link capacity is 1–5–4–7–8. Another example with the same network with different load (network resources) is shown in Fig. 4, the obtained optimal path is 1–2–3–7–8. For the network of Fig. 2, the simulated neural network algorithm is run 100 times using randomly generated link costs and node costs. In all the cases, the results have converged to states with valid solutions with in 4000–8000 iterations.

Given the initial neurons' input voltages u_{xi}'s at time $t = 0$, the time evolution of the state of the neural network is also simulated by solving numerically Eq. (10). Put in mathematical terms the simulation corresponds to solving a



(Dotted line shows the new path computed by the neural network)

(for change in link [4-7] capacity and buffer value in node 4, as compared to figure 2)

Fig. 4. Simulation example with eight nodes.

system of $n(n-1)$ non-linear differential equations, where the variables are the neurons output voltages V_{xi}s. To solve this system of equations fourth order Runge Kutta method is applied. The simulation consisted of observing and updating the neuron output voltages at incremental time steps Δt. The time constant of each neuron (τ) is set to one, without loss of generality. The value of Δt is set to 10^{-5}. The initial input voltages are not set to zero at the start of the computation. Instead some initial random noise is incorporated:

$$-0.002 \le \delta u_{xi} \le +0.002 \tag{22}$$

This helps to break the symmetry. This is always advocated in the neural network literature to add some random noise. The simulation is stopped when the system reaches a stable state. This stable state is assumed to occur when the output of all neurons do not change by more than a threshold value of $\Delta V_{\text{th}} = 10^{-3}$ from one update to another. At the stable state a neuron is termed ON if its output voltage is more than 0.7 and OFF if its output voltage is less than 0.3. If value of the neuron output voltage comes out in between these two values, the iterations are again started after addition of a small random noise equivalent to the noise added at the start of the computation in the input.

This simulation is carried out on eight node partially connected network with different topology. The corresponding result obtained is shown in Fig. 5. The neuronal state is also shown which is decoded to get the path. The arc from destination to source is deliberately inserted to achieve the purpose of satisfying the requirement that the source and destination should always be incorporated into the solution. Hence, instead of a path we get a tour with the last link being a hypothetical link. Also the requirement of exiting a node whenever entered is also satisfied. The source node in this case is node labeled 1 and the destination is the node labeled 8. Another example with the same network with a different destination is shown in Fig. 6.

The cost of the links and nodes are generated randomly with a uniform distribution between 0 and 1. Hence the maximum cost of a link can be 1, which is an essential parameter for fixing up the value of α_1 and α_3. The values of different parameters chosen to satisfy the constraints discussed in Section 5 are given below:

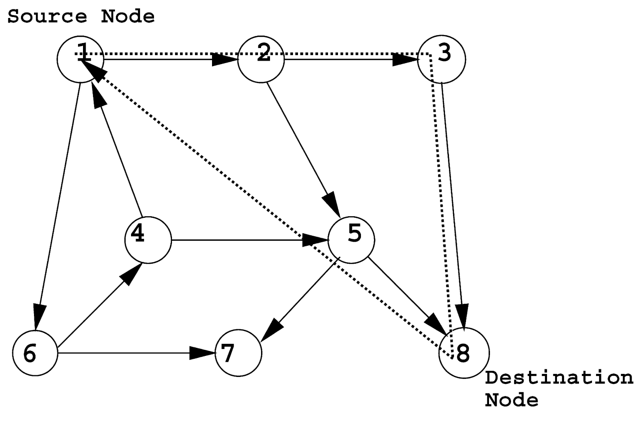
- $\alpha_1 = 100$ $\alpha_2 = 3000$ $\alpha_3 = 500$
- $\alpha_4 = 550$ $\alpha_5 = 825$ $\alpha_6 = 3000$

The number of iterations required for valid solutions varied for this 8-node network between 7000 and 10,000 (Fig. 6). Several experiments are done with different source destination pairs (two examples are shown in Figs. 5 and 6).

Experiment is also done with a formidable network of 30-nodes. The cost of the links are chosen between 0 and 1, distributed randomly with uniform distribution. The parameters chosen to satisfy the constraints discussed in Section 5 is shown below:

- $\alpha_1 = 70$ $\alpha_2 = 10{,}000$ $\alpha_3 = 300$
- $\alpha_4 = 1750$ $\alpha_5 = 2080$ $\alpha_6 = 10{,}000$

Fig. 7 shows one example with node 1 as source node and node 30 being the destination node. The result matches well with the actual shortest path (validated via Dijkstra's algorithm). Another example with 30 nodes is also illustrated in Fig. 8. The number of iterations required in each case was $\sim 10 \times 10^3$ and $\sim 15 \times 10^3$, respectively. The task is computationally expensive when simulated. The algorithm seems to work well even with 30 node network. Hence, the performance tuning i.e. parameter selection is optimum. The number of iteration increases considerably as the number of nodes increases in the network. For solving the problem of 30 nodes a set of 900 equations was required to be solved with the constraint that the neural net reaches

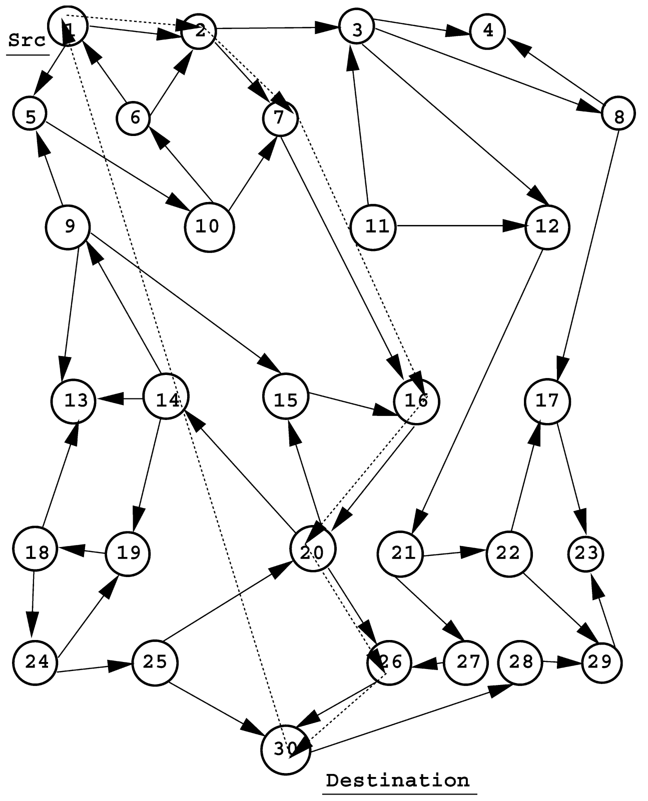


Dotted line shows the path computed by the Hopfield Model.

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

The neuronal state which is decoded to obtain the path.

Fig. 5. First example of eight node SP problem solved numerically.



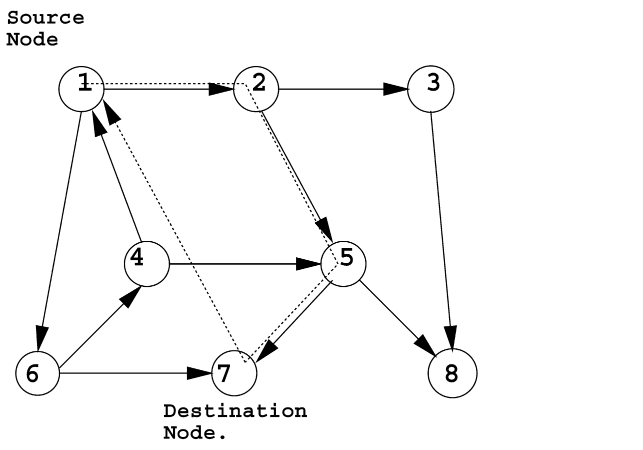
The dotted path shows the path computed by the Hopfield Model . The arc 30->1 is inserted deliberately to facilitate computation.

Fig. 7. First example of 30 node SP problem.

the stable state when none of the neuron's output voltage changes by 10^{-3}.

Here the application of neural networks to the routing problem has been motivated by the idea of taking advantage of the powerful computational ability of the neural network and the fact that a hard-ware implemented neural network achieve high response speeds. Hence the maximum number of nodes to obtain feasible solutions cannot be determined exactly unless neural networks are hardware implemented and tested. In our simulation experiment, when the number of nodes are more then 30, the execution time for the simulation to converge to feasible solution is high. The reason is that the neural network simulation program run on a sequential machine, which otherwise will take less time, if neural networks are implemented using hard-ware, which is beyond the scope of this work.

The value of λ is varied to see the effect on the network computation time and the quality of the solution. In case of 8-node network a value of 1 was adequate. In case of 30-node network it was found that for a compromise between the speed and the quality of the solution λ should be in the range [10,…,30]. Any value outside this range degraded the performance or the speed of computation. Thus the value of λ is dependent on the number of nodes. The value must be tuned by the application requirement in terms of the quality of the solution and the speed of the application.



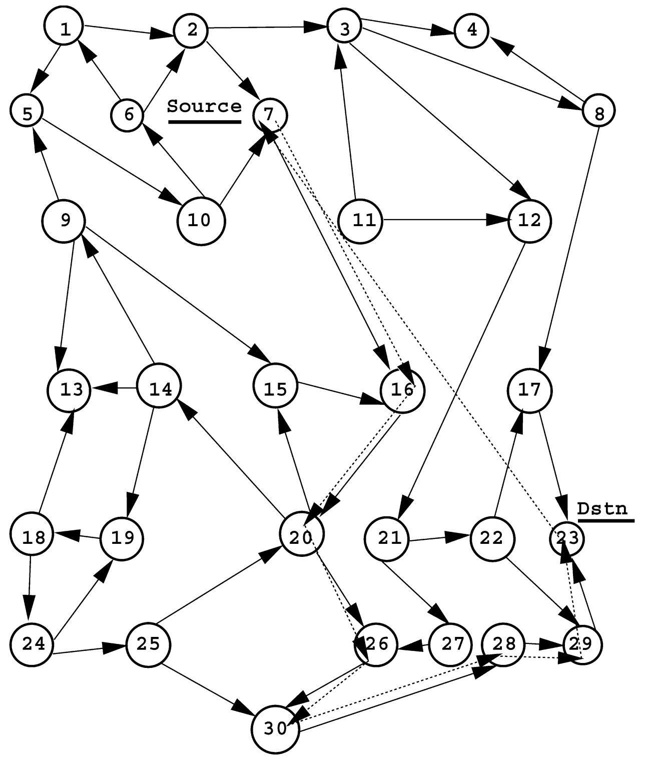
The dotted path shows the SHORTEST path computed by the NEURAL NETWORK .

The last arc : 7 to 1 is inserted deliberately into the solution , the significance is explained in this section.

Fig. 6. Second example of eight node SP problem solved numerically.

7. Conclusion

The shortest path algorithm working on the well defined neural net architecture of Hopfield and Tank is simulated, with a different energy function formulation. The guidelines



```
The dotted path shows the shortest path computed
by the  Hopfield Model.The arc 23->7 is inserted
deliberately to facilitate computation.
```

Fig. 8. Second example of 30 node SP problem.

for the tuning of different parameters are also shown. The hardware details remain the same as in case of Hopfield and Tank network. The use of neural network for shortest path computation is given in (Rauch & Winarske, 1988). The architecture proposed in this work do have some limitation. Knowledge of number of links to be incorporated into the shortest path is required before the start of the iterations. Some more inherent limitations are:

- the connection matrix is topology dependent,
- the source and destination cannot be changed.

The energy function formulated in the previous section do not contain any such constraint. The requirements from the neural net formulated in Section 3, are all satisfied.

1. The neural net when implemented in hardware will be very fast and will adhere to the requirement of real time computation. For speeding up computation of the Hopfield Neural Network, please refer to (DeYong, Findley, & Fields, 1992; Johannet et al., 1994; Massengil & Mundie, 1992; Mortara & Vottoz, 1994; Waurzynek et al., 1993).
2. The neural network can adapt to the changes in the network link costs which varies with the load by just a change in the input bias current. This is a significant advantage. Hence this algorithm will be useful in the scenario when the flow of the links will vary with time. When implemented in hardware there is no need to have a storage greater than the $n \times n$ matrix. This is required in order to find out when the neural net reaches the stable state. In terms of hardware complexity this is not much demanding.
3. There is no concept of a CPU running the whole neural network. Instead the network calculates the solution in a fully distributed fashion. Though the advent of neuro-processor is at the verge of induction (Waurzynek et al., 1993), a study is yet to be made to find out the feasibility of putting the whole Hopfield architecture into digital domain and solve it with a NeuroMicroprocessor. Even with the Hopfield's analog model the computation will converge within a multiple of time constants. The problem while simulation is due to the fact that it was run on a sequential machine and hence took a long time when the number of nodes were more than 30.
4. The network topology change can be incorporated by just changing the input bias currents. The connection matrix is independent of the source and destination indices and hence the neural net architecture is not to be changed dynamically, instead a change in the bias current supplied from the outside environment is sufficient enough.

The parameter which is left out is the gain parameter λ. This parameter is tuned in due course of the simulation. Though the performance of the neural network is not heavily dependent on this parameter but surely tuning of this parameter is required for effective computation. Stability of the neural is also dependent on this parameter. This parameter is always found out, in the experiments conducted, via hit and trial. As already specified, the range in which the value of λ is acceptable is wide enough and roughly lies near unity, thus hitting a right value while designing the neural network is not a very difficult task. Moreover the value of λ when changed gives graceful degradation, thus finding out an optimum value for this parameter is not a bottleneck for the SP algorithm.

References

Cantor, D. G., & Gerla, M. (1974). Optimal routing in a packet-switched computer network. *IEEE Transactions on Computers*, *C23*, 1062–1069.

DeYong, M. R., Findley, R. I., & Fields, C. (1992). The design fabrication and test of a new VLSI hybrid analog—Digital neural processing element. *IEEE Transactions on Neural Networks*, *3*(3), 363–374.

Frank, H., & Chou, W. (1971). Routing in computer networks. *Networks*, *1*, 99–122.

Gallenger, R. (1977). Minimum delay routing algorithm using distributed computation. *IEEE Transactions on Communication*, *COM-25*, 73–85.

Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, *52*, 1–25.

Johannet, A., Personnaz, L., Dreyfus, G., Gasquel, J., & Weinfield, M. (1994). Specification and implementation of a digital Hopfield type associative memory with on chip training. *IEEE Transactions on Neural Networks*, *3*(4), 529–539.

Marbach, P., Mihatsch, O., & Tsitsiklis, J. N. (2000). Call admission control and routing in integrated service networks using neuro-dynamic programming. *IEEE Journal of Selected Areas of Communication*, *February*.

Massengil, L. W., & Mundie, D. B. (1992). An analog neural hardware implementation using charge injection multipliers and neuron specific gain control. *IEEE Transactions on Neural Networks*, *3*(3), 354–362.

Mortara, A., & Vottoz, E. A. (1994). A communication architecture tailored for analog VLSI ANN; intrinsic performance and limitation. *IEEE Transactions on Neural Networks*, *5*(3), 459–466.

Mustafa, K., Ali, M., & Kamoun, F. (1993). Neural networks for shortest path computation and routing in computer networks. *IEEE Transactions on Neural Networks*, *4*(6), 941–954.

Park, Y.-K., & Lee, G. (1995). Applications of neural networks in high-speed communication networks. *IEEE Communications Magazine*, *33*(10), 68–74.

Pornavalai, C., Chakraborty, G., & Shiratori, N. (1995). A neural network approach to multicast routing in real-time communication networks. *Proceedings of the 1995 International Conference on Network Protocols*.

Rauch, H. E., & Winarske, T. (1988). Neural networks for routing communication traffic. *IEEE Control Systems Magazine*, 26–31.

Topkis, D. M. (1988). A K-shortest algorithm for adaptive routing in communication networks. *IEEE Transactions on Communication*, *36*, 855–859.

Wang, J. (1996). A recurrent neural network for solving the shortest path problem. *IEEE Transactions on Circuits and Systems-1: Fundamental Theory and Applications*, *43*(6).

Wang, Z., & Browning, D. W. (1991). An optimal distributed routing algorithm. *IEEE Transactions on Communication*, *39*, 1379–1387.

Wang, C. J., & Weisseler, P. N. (1995). The use of artificial neural networks for optimal message routing. *IEEE Network*, *March/April*, 16–24.

Waurzynek, J., Asannovic, K., & Morgan, N. (1993). The design of a neuro microprocessor. *IEEE Transactions on Neural Networks*, *4*(3), 394–399.

Yaged, B., Jr (1973). Minimum cost routing for dynamic network models. *Networks*, *3*, 193–224.

Yang, H., & Dillon, T. S. (1994). Exponential stability and oscillation of Hopfield graded response neural network. *IEEE Transactions on Neural Networks*, *5*(5), 719–729.