Neural Networks for Shortest Path Computation and Routing in Computer Networks

Mustafa K. Mehmet Ali and Faouzi Kamoun

Abstract— **Recently neural networks have been proposed as new computational tools for solving constrained optimization problems. This paper is concerned with the application of neural networks to the optimum routing problem in packet-switched computer networks, where the goal is to minimize the network-wide average time delay. Under appropriate assumptions, the optimum routing algorithm relies heavily on shortest path computations that have to be carried out in real time. For this purpose an efficient neural network shortest path algorithm, that is an improved version of previously suggested Hopfield models, is proposed. The general principles involved in the design of the proposed neural network are discussed in detail. The computational power of the proposed neural model is demonstrated through computer simulations. One of the main features of the proposed model is that it will enable the routing algorithm to be implemented in real time and also to be adaptive to changes in link costs and network topology.**

I. Introduction

In a packet-switched computer network, routing is an important factor that has a significant impact on the network's performance. For this reason the area of network routing has been the subject of intensive research for many years. An ideal routing algorithm should strive to find the "optimum" path(s) for data transmission within a very short time so as to satisfy the users' demand for a fast service. The optimality of the routing algorithm is a relative attribute which usually implies an efficient use of the network resources so as to optimize a selected performance measure, such as network throughput or mean packet delay. This paper focuses on the optimum quasi-static bifurcated routing problem, where the goal is to minimize the network-wide average time delay. Under appropriate assumptions the optimum bifurcated minimum delay routing problem can be formulated as a nonlinear multicommodity flow problem, whose solution involves shortest path (SP) computations that have to be carried out in real time. This makes neural networks very good candidates for implementing the shortest path computations involved in the routing problem because of the potential of the neural network hardware approach for high computational speed.

In most of the current operating packet-switched networks some form of shortest path computations are required by the routing algorithm at the network layer, where a selected cost measure is assigned to each link and a shortest path algorithm is used to find the minimal cost paths. Here the link cost can be set to a constant value which, for example, can depend on the link transmission capacity or to a variable value such as the estimated average link utilization [1].

In this paper an efficient neural network shortest path algorithm, which is an improved version of the previously suggested neural algorithms, is presented. The proposed model combines many features such as a relatively low programming complexity requirement and an ability to operate in real time and to adapt to changes in link costs and network topology. In addition the solution's quality of the suggested neural Shortest Path algorithm is much better than that of the Hopfield Traveling Salesman neural algorithm and, in most cases, no random initialization for the neural state is required.

This paper is organized as follows:

The next section gives a brief review of the application of Hopfield type neural network to solve discrete combinatorial optimization problems. In Section III, a neural network architecture based on the Hopfield model is proposed for solving the shortest path problem. The main steps involved in the design of the neural network are described. The computational power of the proposed model is demonstrated through computer simulations. In Section IV, the neural network shortest path algorithm developed in Section III, will be used to solve the optimum minimum delay routing problem in packet-switched computer networks. The applicability of the neural network shortest path algorithm to the traffic routing problem will then be demonstrated through computer simulations and a distributed version of the neural based, minimum delay routing algorithm will be described.

II. Hopfield Type Neural Network

The use of neural networks to solve constrained optimization problems was initiated by Hopfield and Tank [2]–[4]. They proposed a neural network model to get good solutions to discrete combinatorial optimization problems. They demonstrate the computational power of their network by applying their model to the Traveling Salesman Problem (TSP). Since then many researchers have attempted to apply the Hopfield model to solve other types of combinatorial optimization problems.

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

Manuscript received June 20, 1991; revised June 2, 1993.

The authors are with the Department of Electrical and Computer Engineering, Concordia University, Montreal, P.Q. H3G 1M8, Canada.

IEEE Log Number 9212563.

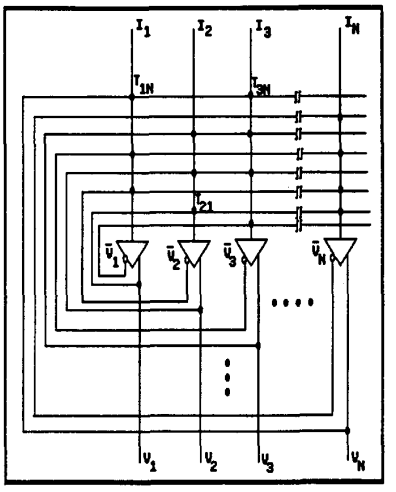


Fig. 1. Hopfield neural network.

the logistic function

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

Each neuron receives resistive connections (to model the synaptic connections) from other neurons. These synaptic connections can be fully described through the matrix $T = [T_{ij}]$, also known as the connection matrix of the network. In addition, as shown in Fig. 1, each neuron receives an external current (known also as a bias) I_i, which could represent actual data provided by the user to the neural network [2], [3]. The dynamics of the Hopfield network are described by

$$\frac{dU_i}{dt} = \sum_{j=1}^{N} T_{ij} \cdot V_j - \frac{U_i}{\tau} + I_i \tag{2}$$

where τ is the circuit's time constant.

Hopfield [5] has shown that for a symmetric connection matrix T, and if the gains of the amplifiers are sufficiently high ($\lambda_i \to \infty$) then the dynamics of the neurons (2) follow a gradient descent of the quadratic energy function

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} T_{ij} V_i V_j - \sum_{i=1}^{N} I_i V_i. \tag{3}$$

Hopfield [5] has also shown that 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 (3) occur at the 2^N corners of this space, only if $\lambda_i = \infty$. In terms of the energy function (3), the dynamics of the ith neuron are described by

$$\frac{dU_i}{dt} = -\frac{U_i}{\tau} - \frac{\partial E}{\partial V_i}. \tag{4}$$

III. A Neural Network Solution to the Shortest Path Problem

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

$$P^{sd} = (s, i, j, k, \cdots, r, d). \tag{5}$$

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

The applicability of the shortest path problem to telecommunication and to transportation networks is diverse. For instance, if C_{ij} is the cost of sending data on arc (i, j) then the shortest path from s to d is the optimum route for data transmission. Alternatively if C_{ij} is set to - ln $[p_{ij}]$, where p_{ij} is the probability that arc (i, j) is usable, then the shortest path from s to d is the most reliable path for data transmission, provided that the usability of a given arc is independent from that of the remaining arcs. In addition there is a large class of network optimization problems whose solution requires solving shortest path problems as subproblems [6].

The use of neural networks to find the shortest path between a given source–destination pair was initiated by Rauch and Winarske [7]. They proposed a neural network architecture arranged in a two-dimensional array of size $n \times m$, where m is the number of nodes forming the path. The output V_{xi} of the neuron at location (x, i) is defined as follows:

$$V_{xi} = \begin{cases} 1 & \text{if node } x \text{ is the } i\text{th node to be visited in the path} \\ 0 & \text{otherwise.} \end{cases} \tag{6a}$$

One obvious limitation of the above representation is that it requires a prior knowledge of the number of nodes forming the shortest path, however this number is usually unknown. In [7], m is fixed to the minimum number of nodes forming the path. It is obvious then that the Rauch and Winarske algorithm finds the shortest path only among those m node paths. Therefore their algorithm gives only a suboptimal solution to the SP problem; after all, the shortest path may consist of more than m nodes. To correct for this shortcoming, Zhang and Thomopoulos [8] extended m to the total number of nodes in the network, which is also the maximum number of nodes the SP may consist of. They assigned zero-cost self-loops connecting each node to itself and very large costs to nonexisting links. In this way a self loop can be included in the SP without increasing the path cost. For a given source destination pair, they also fix the state of all neurons located at the first and last column, while allowing the remaining

neurons to evolve so as to minimize the following energy function:

$$E = \frac{A}{2}\sum_{k=1}^{n-1}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}C_{ij}V_{jk+1} + \frac{B}{2}\sum_{k=1}^{n}\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ik}V_{jk} + \frac{C}{2}\left(\sum_{i=1}^{n}\sum_{j=1}^{n} V_{ij} - n\right)^2. \quad (6b)$$

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

Although the neural network formulation (6) is optimal, it still has significant drawbacks. Since each neuron belonging to the first and last column has a fixed output voltage, the neural network is then designed to find the shortest path between only a given source–destination pair. To find the SP between another pair, the neural configuration has to be changed. More importantly, since the cost term in the energy function (6b) is quadratic, the connection strengths among the neurons depend on link costs. In practice the link costs in a communication network are usually time varying since they normally depend on the amount of flow they are carrying. Consequently Zhang and Thomopoulos neural network will not be suitable for use in traffic routing since the resistances of the synaptic connections must to be changed continuously in order to adapt to changes in link costs. In what follows a new approach to the shortest path problem, also based on the Hopfield model, is proposed. This solution is more appropriate for the optimal routing application to be studied in the next section.

A. Problem Formulation

To formulate the SP problem in terms of the Hopfield neural model, a suitable representation scheme must be found so that the shortest path can be decoded from the final stable state of the neural network. The proposed model is organized in an $(n \times n)$ 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 the row subscript x and the column subscript i denote node numbers. Therefore the computational network requires $n(n-1)$ neurons, and a neuron at location (x, i) is characterized by its output V_{xi}, defined as follows:

$$V_{xi} = \begin{cases} 1 & \text{if the arc from node } x \text{ to node } i \text{ is in the shortest path} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

We also define ρ_{xi} as

$$\rho_{xi} = \begin{cases} 1 & \text{if the arc from node } x \text{ to node } i \text{ does not exist} \\ 0 & \text{otherwise.} \end{cases} \quad (8)$$

In addition the cost of an arc from node x to node i will be denoted by C_{xi}, a finite real positive number. For nonexisting arcs this cost will be assumed to be zero, but as we will show shortly, all nonexisting arcs will be eliminated from the solution.

In order to solve the SP problem, using the Hopfield model, we first have to define an energy function whose minimization process drives the neural network into its lowest energy state. This stable state shall correspond to the shortest path solution. The energy function must favor states that correspond to valid paths between the specified origin–destination pair. Among these valid paths it must also favor the one which has the shortest length. A suitable energy function that satisfies such requirements is found and it is given by

$$E = \frac{\mu_1}{2}\sum_{\substack{x=1}}^{n}\sum_{\substack{i=1\\ i\neq x}}^{n} C_{xi}\cdot V_{xi} + \frac{\mu_2}{2}\sum_{\substack{x=1}}^{n}\sum_{\substack{i=1\\ i\neq x}}^{n} \rho_{xi}\cdot V_{xi} + \frac{\mu_3}{2}\sum_{x=1}^{n}\left\{\sum_{\substack{i=1\\ i\neq x}}^{n} V_{xi} - \sum_{\substack{i=1\\ i\neq x}}^{n} V_{ix}\right\}^2 + \frac{\mu_4}{2}\sum_{i=1}^{n}\sum_{\substack{x=1\\ x\neq i}}^{n} V_{xi}\cdot(1-V_{xi}) + \frac{\mu_5}{2}(1-V_{ds}). \quad (9)$$

In (9) the μ_1 term minimizes the total cost of a path by taking into account the cost of existing links. The μ_2 term prevents nonexistent links from being included in the chosen path. The μ_3 term is zero if for every node in the solution, the number of incoming arcs equals the number of outgoing arcs. This makes sure that if a node has been entered it will also be exited by a path. The μ_4 term pushes the state of the neural network to converge to one of the 2^{n^2-n} corners of the hypercube, defined by $V_{xi} \in \{0, 1\}$. The μ_5 term is zero when the 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 a path, which must originate at s and terminate at d. This makes sure that the final solution contains the arc from d to s and therefore both source and destination nodes will be in the solution. Thus the final solution will always be a loop, with nodes s and d included. This loop consists of two parts; a directed path from s to d and an arc from d to s. If there are no zero length loops in the network, then the μ_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. To illustrate this consider the example depicted in Fig. 2. For the network

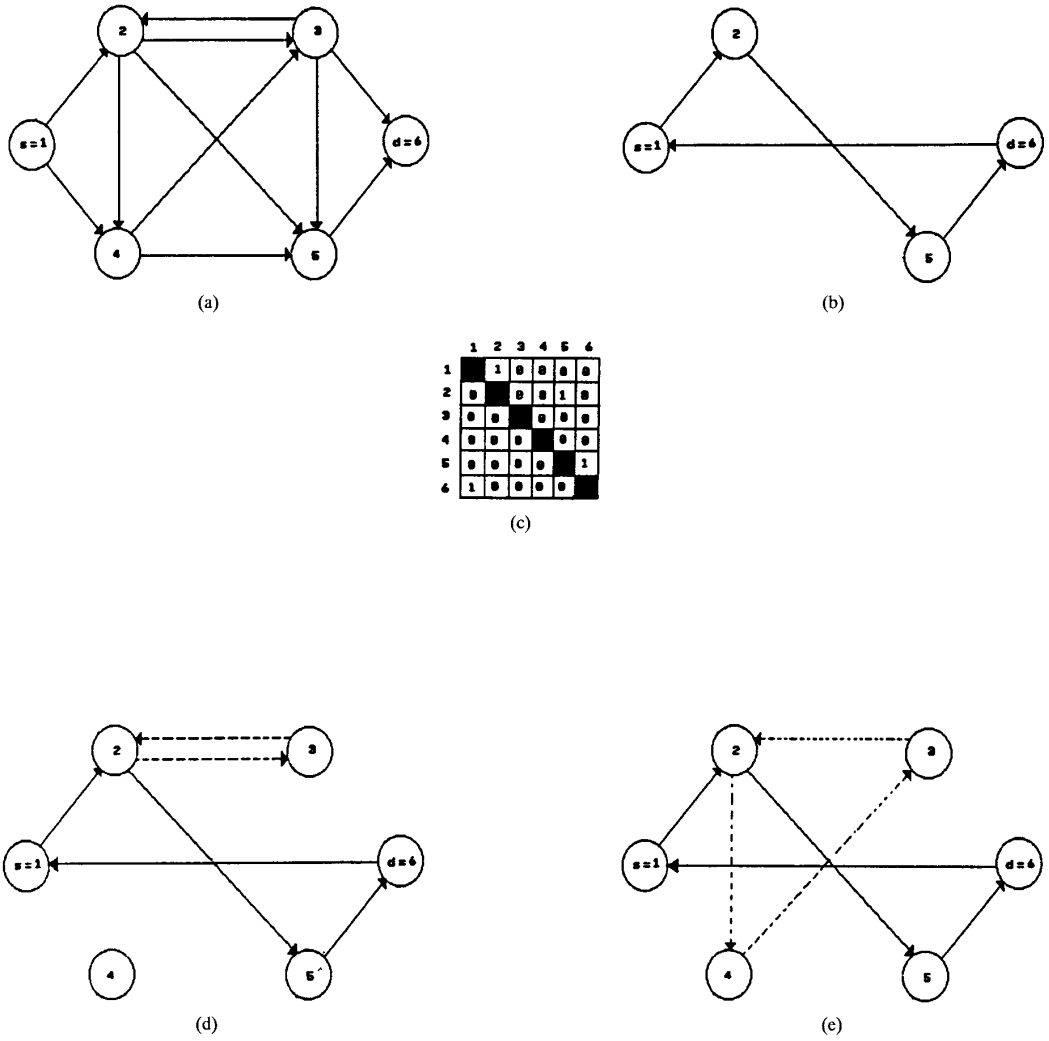


Fig. 2. An illustrating example. (a) The original network. (b) The primary loop forming the SP solution. (c) The neural output representation. (d) A secondary loop (2, 3, 2). (e) Another secondary loop (2, 4, 3, 2).

shown in Fig. 2(a), let us assume that the shortest path is P^{sd} = (1,2,5,6). The corresponding arcs which are to be included in the final solution are shown in Fig. 2(b). These arcs form a loop, thereafter referred to as a primary loop. Correspondingly, the neural output will be represented as shown in Fig. 2(c), where each node included in the shortest path has a single 1 in its corresponding row/column. Note that since nodes 3 and 4 are not part of the shortest path, they have all zero entries in their corresponding rows/columns.

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

B. The Connection Matrix and the Biases

Now, rewriting (2), (4), and (1) in such a way as to take into account the representation of the neurons with double indices, we get

$$\frac{dU_{xi}}{dt} = -\frac{U_{xi}}{\tau} + \sum_{y=1}^{n}\sum_{\substack{j=1 \\ j\neq y}}^{n} T_{xi,yj}\cdot V_{yj} + I_{xi} \qquad (10a)$$

$$\frac{dU_{xi}}{dt} = -\frac{U_{xi}}{\tau} - \frac{\partial E}{\partial V_{xi}} \qquad (10b)$$

$$V_{xi} = g_{xi}(U_{xi}) = \frac{1}{1 + e^{-\lambda_{xi} \cdot U_{xi}}}$$
$$\forall (x, i) \in \overline{N} \times \overline{N} / x \neq i. \quad (11)$$

By substituting (9) in (10b), the equation of motion of the neural network is readily obtained:

$$\frac{dU_{xi}}{dt} = -\frac{U_{xi}}{\tau} - \frac{\mu_1}{2} C_{xi}(1 - \delta_{xd} \cdot \delta_{is}) - \frac{\mu_2}{2} \rho_{xi}(1 - \delta_{xd} \cdot \delta_{is}) - \mu_3 \sum_{\substack{y=1 \\ y \neq x}}^{n} (V_{xy} - V_{yx}) + \mu_3 \sum_{\substack{y=1 \\ y \neq i}} (V_{iy} - V_{yi}) - \frac{\mu_4}{2}(1 - 2V_{xi}) + \frac{\mu_5}{2} \delta_{xd} \cdot \delta_{is}$$
$$\forall (x, i) \in \overline{N} \times \overline{N} / x \neq i \quad (12)$$

where δ is the Kronecker delta defined by

$$\delta_{ab} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise.} \end{cases} \quad (13)$$

By comparing the corresponding coefficients in (10a) and (12), the connection strengths and the biases are derived:

$$T_{xi,yj} = \mu_4 \delta_{xy} \delta_{ij} - \mu_3 \delta_{xy} - \mu_3 \delta_{ij} + \mu_3 \delta_{jx} + \mu_3 \delta_{iy} \quad (14)$$

$$I_{xi} = -\frac{\mu_1}{2} C_{xi}(1 - \delta_{xd} \cdot \delta_{is}) - \frac{\mu_2}{2} \rho_{xi}(1 - \delta_{xd} \cdot \delta_{is}) - \frac{\mu_4}{2} + \frac{\mu_5}{2} \delta_{xd} \cdot \delta_{is}$$
$$= \begin{cases} \frac{\mu_5}{2} - \frac{\mu_4}{2} & \text{if } (x, i) = (d, s) \\ -\frac{\mu_1}{2} C_{xi} - \frac{\mu_2}{2} \rho_{xi} - \frac{\mu_4}{2} & \text{otherwise} \end{cases}$$
$$\forall (x \neq i), \forall (y \neq i). \quad (15)$$

The first term in (14) represents excitatory self-feedbacks, and the second and third terms represent local inhibitory connections among neurons in the same row and in the same column, respectively. The last two terms represent excitatory cross-connections among neurons.

As stated earlier, an advantage of the proposed model is that it maps the data (here defined by link costs and node connectivity information) into the biases rather than into the neural interconnections. This is due to the fact that the data terms are associated with linear rather than quadratic expressions in the energy function (9). One advantage of the proposed representation scheme is a flexibility reflected by the fact that the link costs C_{xi}'s and the network topology information, embedded in the ρ_{xi}'s terms, can be changed through the biases. Since, in this case, we do not have to modify the internal parameters of the neural net architecture, then the proposed SP algorithm can adapt rapidly to changes in network topology and link costs. This will make the neural network SP algorithm very attractive to operate in real time. Another advantage is that the interconnections strengths do not depend on a particular source or destination. Hence the neural network can find the shortest path between any given two nodes by properly choosing the input biases as given in (15). In addition the connection matrix of our proposed algorithm is less complex, requiring $(n^2 - n)(4n - 6)$ connections, whereas Zhang and Thomopoulos formulation requires (n^4) connections. For a 20 node network, this results in a saving of 131 880 connections.

C. The SP Simulation Results

Given the initial neurons' input voltages U_{xi}'s at time t = 0, the time evolution of the state of the neural network is simulated by numerically solving (12). This corresponds to solving a system of $n(n-1)$ nonlinear differential equations, where the variables are the neurons' output voltages V_{xi}'s. To achieve this, we have chosen the fourth order Runge–Kutta method since it is sufficiently accurate and easy to implement. Accordingly the simulation consisted of observing and updating the neuron output voltages at incremental time steps δt. In addition, the time constant τ of each neuron is set to 1 without any loss of generality and for simplicity it is assumed that $\lambda_{xi} = \lambda$ and $g_{xi} = g$ all independent of the subscript (x, i). Simulation has shown that a good value for δt is 10^{-5}. Reducing this tolerably small value increases the simulation time without improving the results. Another important parameter in the simulation is the neuron initial input voltages U_{xi}'s. Since the neural network should have no a prior favor for a particular path, all the U_{xi}'s should be set to zero. However some initial random noise $-0.0002 \leq \delta U_{xi} \leq +0.0002$ will help to break the symmetry caused by symmetric network topologies and by the possibility of having more than one shortest path. The simulation is stopped when the system reaches a stable final state. This is assumed to occur when all neuron output voltages do not change by more than a threshold value of $\Delta V_{th} = 10^{-5}$ from one update to the next. At the stable state each neuron is either On ($V_{xi} \geq 0.5$) or Off ($V_{xi} < 0.5$).

As an example, the five node network shown in Fig. 3(a) is considered. There are six feasible paths between the source node 1 and the destination node 5, whose corresponding neural output representations are shown in Fig. 3(b). The simulation was conducted with two goals in mind: first to test the performance of the proposed neural network SP algorithm and second to investigate the impact of various parameters on the quality of the solution.

One major issue related to the efficiency of the Hopfield model in solving combinatorial optimization problems is the lack of rigorous guidelines in selecting appropriate values of the energy function coefficients. A trial and error approach, although extensively used in the literature, is not the best scheme to ensure convergence to valid as well as optimal solutions. For this reason, and in the sequel, we will give some general guidelines as how to select the μ_i's coefficients so that the neural dynamics will converge to a valid path which is also of minimum length.

As a first step, let us characterize the energy surface of the proposed SP neural network formulation. It is clear that the shape of our SP energy surface is tailored by the memory terms $T_{xi,yj}$ (14) and the analog prompt terms I_{xi} (15). Ignoring the link cost bias terms, for the moment, it can be seen that the memory and prompt terms create a set of finite local energy attractors, with equal depth ($E = 0$), each corresponding to a

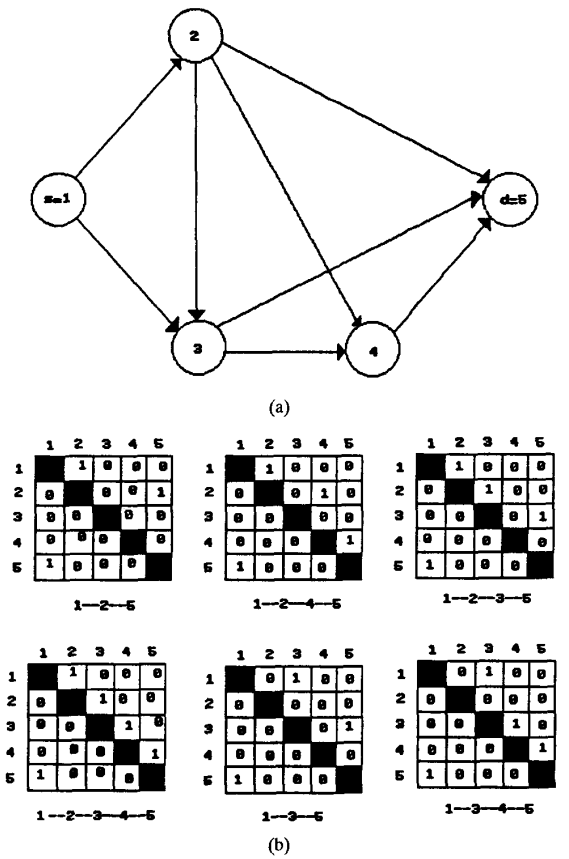


Fig. 3. An illustrating example. (a) A network example used in the simulation. (b) Output representation of the feasible paths.

feasible path. In the sequel, these attractors will be referred as valid basins of attraction. The role of the linear μ_1 cost term is then to provide a negative bias that enlarges the depth of these valid basins of attractions by varying amounts, depending on the corresponding path cost. Therefore, and from (15), the global energy minimum corresponds to that valid basin of attraction which complies with the shortest path requirement.

This ability to reshape the energy landscape through the link cost bias terms, while keeping the interactions among the neurons relatively simple (since the memory terms do not incorporate the link costs) is a salient feature that is attributed to our formulation, which embodies the problem-dependent link costs in a linear rather than a quadratic energy term [9]. Although there is a possibility for the neural state to get trapped in a local energy attractor, it will be shown shortly that, by proper tuning of the μ_i's weighting coefficients, such a possibility could be minimized if not excluded. In addition since the quadratic energy function (9) is continuous and differentiable then the energy surface is characterized by the presence of valleys, where among all the points forming each valley, we can identify some "low points," each corresponding to a local minimal state. Further, in order to ensure that every valley has only one low point and hence to provide a graceful descent along the energy surface, we require

$$\frac{\partial^2 E}{\partial V_{xi}^2} > 0 \qquad \forall (x,i).$$

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. To see how this actually happens in our case, let us look at the initial state of the neural network, with all of the neurons' inputs being set to zero. For the neuron at location (d, s), the input voltage increases at a rate

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

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

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

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

$$R_3 = -\frac{\mu_1}{2} C_{xi}.$$

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

$$\mu_5 \gg \mu_1 \cdot (C_{xi})_{\max}.$$

In addition, since an equally important requirement is to prevent non existing arcs from being part of the solution, then it is reasonable to require $R_1 = R_2$, or

$$\mu_2 = \mu_5. \qquad \text{(large)}$$

For the μ_1 coefficient, it is clear that it should be kept at a value much lower than ($\mu_2 = \mu_5$) and greater than μ_4. What is left then is to determine the relationship between μ_1 and μ_3, so that the neural network converges to a valid path that is also of minimum length.

From (9) it is clear (and this was also observed through simulation), that by increasing the μ_1 term (with remaining μ_i's being unchanged) the proposed 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 this μ_1 cannot be increased indefinitely, since once it exceeds a threshold value, the simulated neural algorithm starts to diverge and gives invalid solutions, as the effect of the μ_3 energy term will be shaded by the stiff cost requirement. Therefore we need to maximize μ_1 and at the same time obtain a solution which satisfies the constraints. For this purpose let us assume that 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 weighting coefficient μ_3 will increase by μ_3, while the energy term associated with μ_1 will decrease by a maximum value of $\mu_1/2 \cdot (C_{xi})_{\max}$. Hence, for the net to reach a valid path, μ_1 should satisfy

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

By taking into account all these requirements for the μ_i's values, we have chosen the weighting coefficients as follows:

$$\mu_1 = 950; \quad \mu_2 = 2500; \quad \mu_3 = 1500;$$
$$\mu_4 = 475; \quad \mu_5 = 2500.$$

Experimentally we have found that there is a compromise between choosing a small or a large value for the neural transfer parameter λ. While a large λ gives rise to a fast neural response for which the solution is not always a global minimum, a small λ yields a slower response which can guarantee an optimum solution. In fact, it was observed that the ability of the SP neural algorithm to separate between an optimum and a very good solution is also enforced by the smoothness of the neural transfer function g (which corresponds to a low neural transfer parameter λ). This is explained by the fact that a large λ (such as 10 or 100) may not allow enough time for a neuron to optimize its performance, as the neural response time becomes very fast and hence less accurate. Therefore, in order to allow the neurons' dynamics to wander freely in their state space, in the search for the global minima, we have chosen $\lambda = 1$. For the network of Fig. 3(a), the simulated neural algorithm is run 100 times using different randomly generated link costs, between 0 and 1. In all runs the simulated algorithm has converged to states corresponding to valid solutions within 3000 to 8000 iterations. In addition the global optimum was obtained in all runs. A typical example is illustrated in Fig. 4.

These results show that the performance of the proposed shortest path neural algorithm is superior to that of Zhang and Thomopoulos. In our case the neural network can be properly designed (through a careful choice of the μ_i's coefficients, as outlined before) to guaranty convergence to valid solutions which are often global optima. This is also supported by earlier results reported in [10]–[12] where it was found that neural networks with linear cost functions perform much better than those neural networks with quadratic cost functions. Further, although it is intended to solve a problem which is by far simpler than the NP complete TSP, the Zhang and Thomopoulos SP energy function (6b) is very similar to the Hopfield TSP energy function, and therefore it inherits a key problem associated with the Hopfield TSP network, namely the growing tendency to converge towards an invalid final stable sate as the network size gets larger (see for example [13], [14]). In addition, recall from (6b) that the use of the B and C terms does not constitute a very strong constraint in ensuring a single 1 per column, as the C term may favor more than a 1 in a column, which contends with the requirement set by the B term. One way to overcome this problem, and which might help the neural net to better converge towards a valid solution, would be to combine the B and C terms in (6b) into one term, namely $D/2 \sum_{k=1}^{n} (\sum_{i=1}^{n} V_{ik} - 1)^2$, and to add a self-connection term $G/2 \sum_{i=1}^{n} \sum_{j=1}^{n} V_{ij} \cdot (1 - V_{ij})$.

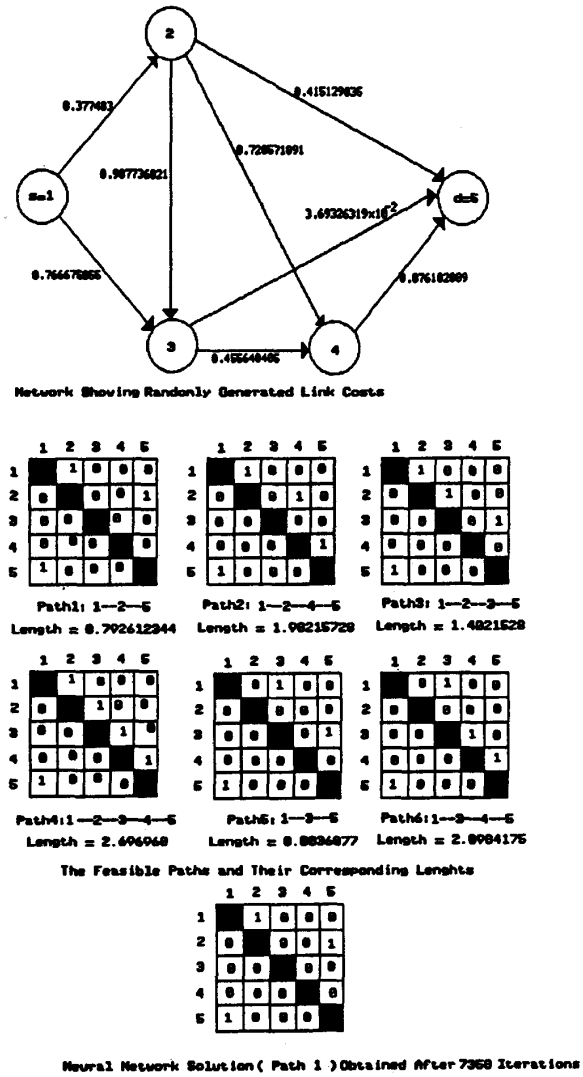


Fig. 4. Typical results for the SP problem.

IV. Neural Networks for Optimum Routing in Packet- Switched Computer Networks

The choice of an optimum routing policy to forward packets from sources to destinations in a packet-switched computer network is a significant factor that has to be dealt with care in order to optimize performance measures such as mean packet delay or network throughput. This section focuses on the optimum quasi-static bifurcated routing problem in a packet-switched computer network, where the goal is to minimize the network-wide average time delay. Bifurcated or multiple path routing arises in situations where the traffic of a given commodity is allowed to be difstributed over several paths. Here the term commodity refers to the traffic offered to a

given source node S, and destined to a given destination node D. In the next section, and under appropriate assumptions, the optimum bifurcated minimum delay routing problem will be formulated as a classical nonlinear multicommodity flow problem, whose solution is well known using nonlinear mathematical programming techniques. However, direct application of these mathematical programming techniques to the routing problem in computer networks is not very efficient because of the need to execute the computations in real time and usually in a distributed fashion. This makes neural networks very good candidates for implementing most of the computations involved in the routing problem.

A. Problem Formulation of the Optimum Routing Algorithm

Consider a packet-switched computer network with N nodes and L directed links (channels) connecting them. A packet, belonging to a particular message, experiences delay at various stages during its journey from its source node to its corresponding destination node. First there is a processing delay at each node due to packet header processing, routing computations and error checking. There is also a delay due to error control retransmission and most importantly there is a significant delay due to buffering of packets at each node. This section focuses on this node buffering delay.

Under the Kleinrock [15] independence assumption, the minimization of the average time delay (here the delay is averaged over time and over all commodities) is equivalent to minimizing the well known formula

$$T^* = \sum_{l=1}^{L} D_l(f_l) = \sum_{l=1}^{L} \frac{f_l}{C_l - f_l} \tag{16}$$

where

f_l = Average flow on link l in data units/s.
C_l = Capacity of link l in data units/s.

It is also convenient to formulate the routing optimization problem in terms of path rather than link flows as above, in which case the routing problem can be formulated as follows:

Given a directed network of N nodes and L directed links. Let $NC = \{1, 2, \cdots, K\}$ be the set of K commodities ($S-D$ pairs) in the network. Following a similar approach as in [6], the corresponding paths will be numbered sequentially, so that the sets of directed paths corresponding to the K commodities are given by:

$$P_1 = 1.2, \cdots, n_1; \quad P_2 = n_1 + 1, n_1 + 1, n_1 + 2, \ , n_2; \cdots; \quad P_K = n_{K-1} + 1, \cdots, n_K \tag{17}$$

where P_i denotes the set of directed paths available to route commodity i, and the integer n_K is the total number of paths corresponding to all K commodities.

Let

$f(n)$ = traffic flow (data units/s) carried on path n.
$\vec{f}$ = $[f(1)\, f(2) \cdots f(n_K)]^T$ = vector of path flows.
$\lambda(i)$ = Offered external traffic (data units/s) to commodity $i; i \in NC$.

Then the average link flow f_l can be expressed in terms of path flow as follows:

$$f_l = \sum_{n=1}^{n_K} f(n)\zeta(n, l) \tag{18}$$

where

$$\zeta(n, l) = \begin{cases} 1 & \text{if path } n \text{ contains link } l \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

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

Given:

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

Object:

Minimize:

$$F(\vec{f}) = \sum_{l=1}^{L} D_l(f_l) = \sum_{l=1}^{L} \frac{\sum_{n=1}^{n_K} f(n)\zeta(n, l)}{C_l - \sum_{n=1}^{n_K} f(n)\zeta(n, l)}. \tag{20}$$

Variables: Path flow vector $\vec{f}$.

Constraints:

$$\lambda(i) = \sum_{n \in P_i} f(n) \quad \forall i \in NC \tag{21}$$

$$f(n) \geq 0 \quad \forall n \in P_1 \cup P_2 \cup \cdots P_K \tag{22}$$

Note that the link capacity constraints, $f_l < C_l$, are not included here since, starting from a feasible solution, the objective function will implicitly take care of these constraints, as $F(\vec{f}) \to \infty$ when $f_l \to C_l$ [6].

B. Characterization of Optimum Routing

The solution to the optimum routing problem, formulated in (20)–(22) can be found using many iterative algorithms such as the Frank–Wolfe or flow deviation (FD) method [16], the extremal flow (EF) method [17], and the gradient projection [GP] method [18]. In this paper the FD method is chosen for solving the minimum delay routing problem. A neural network algorithm is incorporated into the FD method in order to enable its real time implementation. The main steps of the routing algorithm, based on the FD method can be found in Appendix 1. It should be noted, at this stage, that while the applicability of the SP neural network algorithm to traffic routing is demonstrated here in conjunction with the FD method, other routing algorithms can as well benefit from the SP neural implementation. In fact most of the current operating packet-switched networks use some form of shortest path computation, where a cost measure (fixed or variable) is assigned to each link and a least-cost path between each source-destination pair is sought. Most of today's routing algorithms, however, differ in the way the link costs are defined and computed and they also differ in the way the routing computations are performed.

For our case, and as may be seen from Appendix 1, shortest path computations which use link costs that are based on first derivative of link delays ($A1$), play a key role in the optimal routing algorithm, especially that they have to be carried out at each iteration. Therefore the choice of an efficient shortest path algorithm is a crucial factor in the performance of the routing algorithm, especially for large networks where thousands of iterations (hence thousands of shortest path computations) might be required by the FD algorithm. In a packet-switched computer network, routing and flow allocation decisions are to be made very fast, otherwise the network performance may be the subject of severe degradation, to the customers' dissatisfaction. As a result it is suggested that the use of the neural network SP algorithm, described in Section III, is recommended as it will reduce the execution time per iteration required by the FD method. The incorporation of the neural network SP algorithm into the optimum routing method requires that at every iteration, each node computes the MFDL path for all commodities originating from itself. Note that if the link from node x to node i is labeled l, then from ($A1$) its corresponding cost will be

$$C_{xi} = \frac{C_l}{(C_l - f_l)^2}. \tag{23}$$

Here the link costs are time varying since they depend on link flows, which change from one iteration to another. Along with the fixed network connectivity terms ρ_{xi}'s, the link costs (23) constitute the inputs to the neural network.

As mentioned earlier one of the features of the proposed SP energy function is that it will enable the routing algorithm to be adaptive to changes in network topology. For instance, when node i detects a link failure on one of its outgoing links (let us say the one that connects it to node j) then it will send control messages to all adjacent nodes. These nodes will then reset the value of ρ_{ij} from 0 to 1. The exchange of control messages and updating of ρ_{ij} move forward until all the nodes are informed. As a result the proposed SP energy function, through the μ_2 term, will make sure that the routing algorithm will steer the traffic away from all paths passing through the faulty link as the proposed neural network shortest path algorithm will treat this defective link as being nonexistent. The same adaptivity can be achieved in the case where a link is recovered. In this case the corresponding ρ value will be changed from 1 to 0.

The minimum delay routing algorithm, using the neural network SP algorithm, can also be implemented in a distributed fashion, where the computational task required at each iteration by the FD algorithm is shared among all the nodes of the network [19]. The details can be found in Appendix 2.

C. Computational Versus Programming Complexity

Computational Complexity: The computational complexity of the FD method, when applied to the optimum routing problem, depends on the network topology and on the way the routing computations are performed. Although we have formulated the optimal solution of the routing problem in terms of path flow variables, in practice, when the routing computations are implemented in a distributed fashion among all the nodes (Appendix 2), the routing algorithm solves for the individual commodity link (rather than path) flow variables. This is due to the fact that in large packet-switched computer networks, the source nodes often do not keep a detailed record of all the possible paths which are available to each destination node, since this will require additional CPU storage (core) requirement.

Therefore, in the worst situation, and assuming a fully connected network with $K = n(n-1)$ commodities, the computational complexity per iteration of the FD method has been estimated [18]. In this case, the flow assignment calculations require $O(n^2)$ computations and the updating of the individual commodity link flows require $O(K \cdot L) = O(n^4)$ calculations. For the SP computations, two types of Graph theoretic algorithms have been proposed: global (centralized) algorithms which compute the shortest paths for all commodities at once and partial (distributed) algorithms where each node computes the SP to all other nodes. In practice, distributed SP algorithms are preferred because of their lower memory requirement and lower computation time [19], [20]. Therefore, in our case, the computational complexity (per iteration) of a highly efficient distributed (graph theoretic) algorithm is $O(n^3)$ [18], which reflects the importance of the SP computations in the FD method.

2). Programming Complexity: Since the computation time of neural networks is expected to be very short, then the efficiency of our neural network approach to the SP computations is best assessed in terms of the programming complexity, which is defined as the number of arithmetic operations required to redetermine the proper interconnection strengths and the biases each time a new data is fed to the neural net [21]. By assuming a distributed SP implementation, whereby each node is assigned a neural network to compute the SP to all other nodes, the programming complexity of the proposed approach will be determined by two factors.

First, since the interconnection strengths are fixed, then a node has to redetermine $O(n^2)$ biases for each new set of link costs. Further, since each bias can be determined by a constant number of operations, then the programming complexity is $O(n^2)$.

More importantly, once a node computes the SP to its first destination then the computation of the SP to any other node requires the redetermination of two input biases only, as given in (15). This enables each node to compute the SP to all other nodes with a programming complexity still on the order of $O(n^2)$. A comparison of this programming complexity with the computational complexity ($O(n^3)$) of the best existing distributed SP algorithm, reveals a net gain in implementing the SP computations using the proposed neural network approach. This gain has a particular significance when considering large computer networks.

D. Simulation Results

1) A Five Commodity Network Example: Here we consider the five commodity network shown in Fig. 5, where the goal is to route the traffic inputs $\lambda(i)$'s for all the commodities specified in Table I so as to minimize the average traffic delay. For this case the set of commodities is $NC = \{1, 2, 3, 4, 5\}$

TABLE I
TRAFFIC REQUIREMENTS OF THE FIVE COMMODITY NETWORK

Commodity i	Source	Destination	Traffic Input $\lambda(i)$
1	1	5	50
2	1	3	45
3	2	4	35
4	4	3	15
5	5	1	30

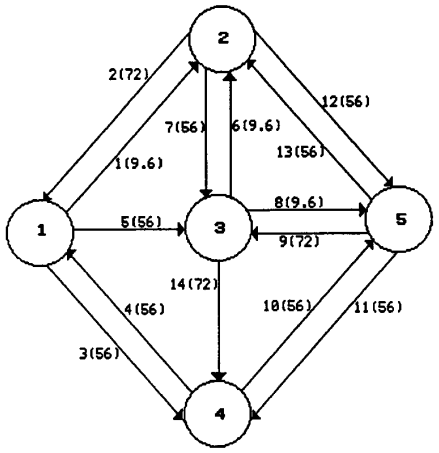


Fig. 5. The five commodity network. [Numbers inside (outside) brackets denote link capacities (link indexes).]

and the corresponding sets of directed paths are

$$P_1 = \{1, 2, 3, 4, 5, 6, 7\};$$
$$P_2 = \{8, 9, 10, 11, 12\};$$
$$P_3 = \{13, 14, 15, 16, 17, 18, 19\};$$
$$P_4 = \{20, 21, 22, 23, 24, 25\};$$
$$P_5 = \{26, 27, 28, 29, 30\}$$

where each path is defined by its node sequence as specified in Table II.

The path flow vector is initially set to the feasible flow $\vec{f}^{(0)} = [f^0(1) f^0(2) \cdots f^0(30)]^T$ where

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

with all remaining path flow components being set to zero. This corresponds to the initial flow allocation shown in Fig. 6(a) and which gives an initial cost of 78.434.

For this (n = 5) nodes network, we require $n(n-1)$ = 20 neurons. The dynamics of the resulting neural network are then simulated using (12) and with the following parameters:

$$\mu_1 = 950; \quad \mu_2 = 2500; \quad \mu_3 = 2000; \quad \mu_4 = 85$$
$$\mu_5 = 2500; \quad \lambda = 1; \quad \delta t = 10^{-5}; \quad \Delta V_{th} = 10^{-5} V.$$

In all runs, the neural network SP computations have converged to optimum solutions. In addition, with a tolerance value (ϵ) of 10^{-2}, the simulated minimum delay routing algorithm has reached the optimum solution after 38 iterations.

TABLE II
CHARACTERIZATION OF THE PATHS OF THE FIVE COMMODITY NETWORK

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

The components of the optimum path flow vector $\vec{f}^{(38)}$ have converged to the following values:

$$f^{38}(1) = 1.890; \quad f^{38}(4) = 0.270; \quad f^{38}(5) = 1.293$$
$$f^{38}(7) = 46.547; \quad f^{38}(8) = 42.188; \quad f^{38}(10) = 1.701$$
$$f^{38}(11) = 1.111; \quad f^{38}(13) = 27.100; \quad f^{38}(15) = 7.900$$
$$f^{38}(20) = 7.743; \quad f^{38}(21) = 3.871; \quad f^{38}(22) = 0.294$$
$$f^{38}(23) = 3.092; \quad f^{38}(26) = 23.229; \quad f^{38}(30) = 6.771$$

with all remaining components set to zero. This corresponds to the optimum flow allocation shown in Fig. 6(b) and whose cost is 35.448. In Fig. 7 the objective function (20) is also plotted as a function of number of iterations.

2) A Twenty Commodity Network Example: In this example we investigate the performance of the neural network SP algorithm when the network size is increased. Consider the network shown in Fig. 8, with the following parameters:

Number of nodes: n = 15.
Number of arcs: L = 38 (19 full duplex arcs).
Arc capacities: C_l = 50.
Number of commodities: K = 20.
Traffic requirements $\lambda(i)$'s: as specified in Table III.
Corresponding total number of directed paths: n_K = 290.
The path flow vector is initially set to $\vec{f}^{(0)}$ =

$[f^0(1)f^0(2)\cdots f^0(290)]$ where:

$$f^0(1)=10;\quad f^0(27)=8;\quad f^0(48)=8;\quad f^0(54)=10;$$
$$f^0(67)=5;\quad f^0(82)=12;\quad f^0(105)=9;\quad f^0(112)=8;$$
$$f^0(129)=6;\quad f^0(145)=5\quad f^0(157)=5;\quad f^0(169)=10;$$
$$f^0(181)=12;\quad f^0(191)=5;\quad f^0(207)=7\quad f^0(218)=8;$$
$$f^0(229)=6;\quad f^0(256)=12;\quad f^0(273)=10;\quad f^0(286)=10;$$

with all remaining components set to zero. This corresponds to an initial cost of 30.386.

For this case, we require (15 × 14) 210 neurons, and the corresponding neural network dynamics are simulated as before, with the same parameters. Once again and in all runs the neural network solution has converged to shortest paths. The optimal flow allocation is obtained after 91 iterations and it corresponds to a cost of 24.849. The components of the optimal path flow vector, $\vec{f}^{(91)}$, have converged to the following values:

$$f^9(1)=4.161;\ f^{91}(9)=2.702;\ f^{91}(13)=0.473;$$
$$f^{91}(21)=1.852;\ f^{91}(25)=0.812;\ f^{91}(27)=6.441$$
$$f^{91}(34)=0.796;\ f^{91}(39)=0.763;\ f^{91}(46)=5.066;$$
$$f^{91}(48)=2.733;\ f^{91}(52)=0.201;\ f^{91}(54)=10.000;$$
$$f^{91}(67)=2.405;\ f^{91}(70)=2.595;\ f^{91}(82)=11.237;$$
$$f^{91}(84)=0.178;\ f^{91}(86)=0.585;\ f^{91}(102)=5.925;$$
$$f^{91}(105)=3.075;\ f^{91}(112)=7.594;\ f^{91}(113)=0.406;$$
$$f^{91}(121)=0.358;\ f^{91}(129)=5.642;\ f^{91}(145)=5.000;$$
$$f^{91}(157)=5.000;\ f^{91}(169)=3.416;\ f^{91}(173)=1.718;$$
$$f^{91}(175)=4.866;\ f^{91}(181)=12.000;\ f^{91}(191)=5.000$$
$$f^{91}(207)=6.824;\ f^{91}(208)=0.176;\ f^{91}(216)=4.946;$$
$$f^{91}(218)=2.733;\ f^{91}(222)=0.321;\ f^{91}(229)=6.000;$$
$$f^{91}(255)=7.900;\ f^{91}(256)=4.100;\ f^{91}(265)=0.746;$$
$$f^{91}(266)=2.201;\ f^{91}(273)=7.053;\ f^{91}(281)=6.584;$$
$$f^{91}(286)=3.416$$

with all remaining components set to zero. The objective function (20) is also plotted as a function of number of iterations, as shown in Fig. 9.

These results lead us to infer that the solution quality of the proposed algorithm is scalable, in the sense that the computational power of the proposed algorithm, in identifying the shortest path, is not degraded when we consider networks with a much larger number of nodes. This leads us to get some optimistic feelings for the appropriateness of the proposed neural algorithm when applied to more realistic problems, involving much larger networks.

The fact that the quality of the neural network solution is not downgraded, when the problem size is increased, is mainly attributed to the proposed energy function formulation (9) which has a linear rather than a quadratic cost (μ_1) term. Our proposed formulation keeps the neural interconnection matrix T constant, and this seems to favor convergence even for relatively large scale problems. Compared to the Hopfield TSP network or to the Zhang and Thomopoulos SP network, our proposed SP neural architecture is less complex, because in the first two networks the incorporation of the internodal distances in the interconnection matrix increases the interactions among the neurons. In addition, as mentioned before, in our energy function formulation, all valid paths correspond to energy minima of equal depth. With the help of the μ_4 term, these minima will be located around the corners of the hypercube. The cost-dependent bias terms, which are fed to each neuron, push the neural state to converge towards the corner corresponding to the shortest path. This is achieved by enforcing strong inhibitions (negative bias cost terms) on those neurons which are associated with higher link costs so that they will be turned Off.

TABLE III
TRAFFIC REQUIREMENTS OF THE TWENTY COMMODITY NETWORK

Commodity i	Source	Destination	Traffic Input $\lambda(i)$
1	1	13	10
2	1	5	8
3	2	9	8
4	2	10	10
5	3	8	5
6	4	11	12
7	5	14	9
8	6	12	8
9	7	15	6
10	7	11	5
11	8	3	5
12	9	2	10
13	10	13	12
14	11	4	5
15	12	6	7
16	12	1	8
17	13	1	6
18	14	5	12
19	15	7	10
20	15	9	10

V. Conclusion

A new solution to the shortest path problem was proposed, using a Hopfield type neural network. The general principles involved in the design of the proposed neural network were discussed. The proposed model combines many features, such as a very good convergence and scaling properties, a relatively low programming complexity and an ability to operate in real time and to adapt to changes in network topology and link costs. The proposed model was then applied to the optimum minimum delay routing problem in computer networks and a distributed implementation of the neural network-based routing algorithm was also considered. From the simulation results, obtained under different network topologies and link costs, it can be concluded that the proposed model is both efficient and effective in identifying the shortest path.

Appendix 1

Main Steps of the Flow Deviation Method Applied to the Optimal Routing Problem

Step 1: Find an initial feasible path flow vector $\vec{f}^{(0)}$. Here it will be assumed that a feasible starting flow is available; although there are methods to find it (see for example [20]).

Step 2: Compute the first derivatives D_l' at the current path flow vector $\vec{f}^{(k)}$, where

$$\vec{f}^{(k)} = [f^k(1) \quad f^k(2) \quad \cdots \quad f^k(n_K)]^T$$
$$f^k(n) = \text{flow on path } n \text{ at the } k\text{th iteration}$$
$$D_l'(f_l) = \frac{C_l}{(C_l - f_l)^2} \quad \forall l \in \{1, 2, \cdots, L\}. \tag{A1}$$

Under this metric, each node computes the shortest path (known also as the minimum first derivative length (MFDL) path) for all commodities originating from itself.

Step 3: Find the path flow vector $\tilde{\vec{f}}^{(k)} = [\tilde{f}^k(1) \quad \tilde{f}^k(2) \quad \cdots \quad \tilde{f}^k(n_K)]^T$, obtained by routing all input traffic $\lambda(i)$, for each commodity along its corresponding shortest path.

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

$$\vec{f}^{(k+1)} = \vec{f}^{(k)} + \alpha_k \left(\tilde{\vec{f}}^{(k)} - \vec{f}^{(k)} \right); \quad \alpha_k \in [0, 1] \tag{A2}$$

where

$$\alpha_k = \min \left\{ 1, -\frac{\sum_{l=1}^{L} (\tilde{f}_l^k - f_l^k) D_l'(f_l^k)}{\sum_{l=1}^{L} (\tilde{f}_l^k - f_l^k) D_l''(f_l^k)} \right\} \tag{A3}$$

and

$$f_l^k = \sum_{n=1}^{n_K} f^k(n) \zeta(n, l) \tag{A4}$$
$$\tilde{f}_l^k = \sum_{n=1}^{n_K} \tilde{f}^k(n) \zeta(n, l) \tag{A5}$$

are the total link flows corresponding to $\vec{f}^{(k)}$ and $\tilde{\vec{f}}^{(k)}$, respectively. In (A3), the first derivative D_l' is as given in (A1), while the second derivative D'' is given by

$$D_l''(f_l) = \frac{2 \cdot C_l}{(C_l - f_l)^3}. \tag{A6}$$

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

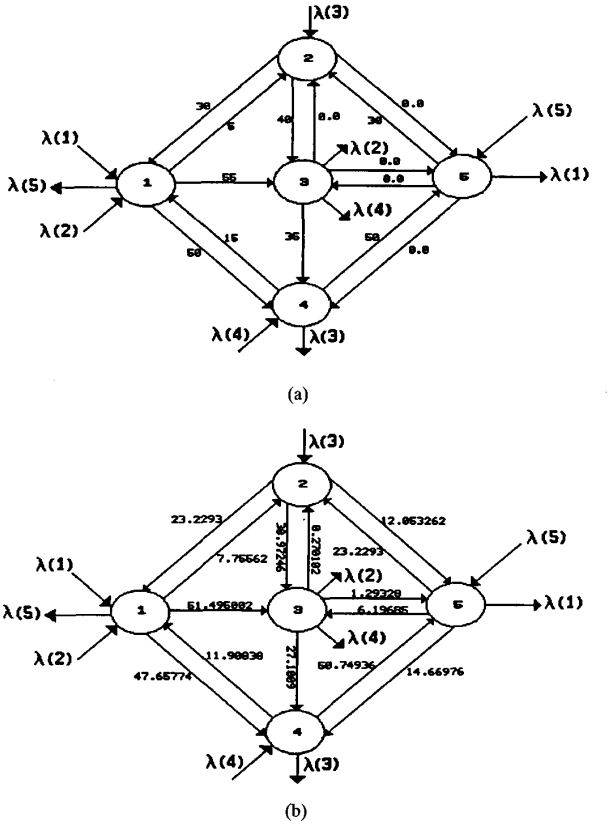


Fig. 6. Initial and final flow allocation for the traffic routing example. (a) Initial flow allocation: Cost = 78.4336. (b) Optimal flow allocation: Cost = 35.4485.

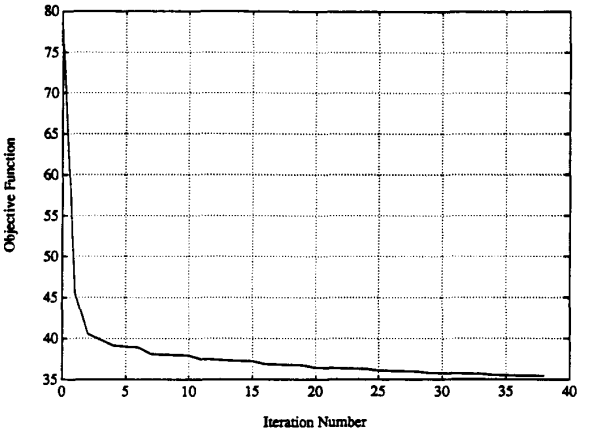


Fig. 7. Simulation results for the five commodity example.

APPENDIX 2

IMPLEMENTATION OF THE ROUTING ALGORITHM IN A DISTRIBUTED MANNER

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

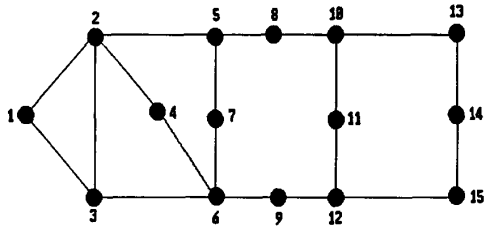


Fig. 8. The 20 commodity network.

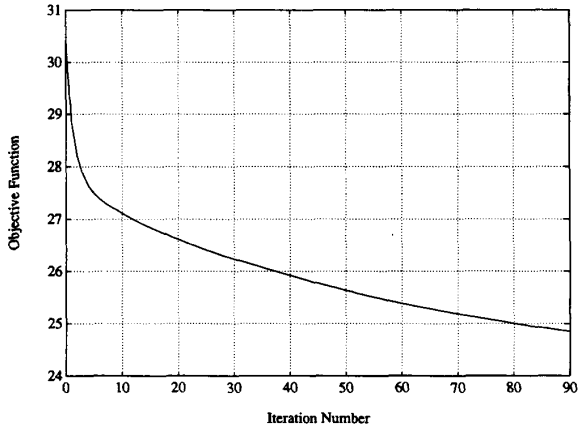


Fig. 9. Simulation results for the 20 commodity example.

calculate the first and second derivatives $D_l^{'}(f_l^k)$ and $D_l^{''}(f_l^k)$ for all links l. Subsequently each node computes the MFDL path to each of its destinations, using the metric $D_l^{'}(f_l^k)$ as link costs. This requires repeated or preferably parallel application of the neural network SP algorithm for each destination node. Each node then broadcasts to each destination the current value $\lambda(i)$ of the offered traffic along the corresponding shortest path. Then every node computes the flow value $\tilde{f}_l^k$ for each outgoing link and transmits the difference $(\tilde{f}_l^k - f_l^k)$ to all other nodes. At this stage each node computes α_k (A3) and updates the link flows f_l^{k+1} according to

$$f_l^{k+1} = f_l^k + \alpha_k \cdot (\tilde{f}_l^k - f_l^k). \tag{A7}$$

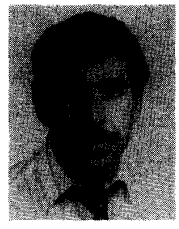
The process is then repeated until the optimum path flow vector is reached. It should also be noted that the minimum delay routing algorithm can also be implemented for time-varying external flows $\lambda(i)$'s. In this case, at the k th iteration, each node keeps track of the fraction of flows:

$$\pi^k(n) = \frac{f^k(n)}{\lambda(i)}; \quad \forall n \in P_i \tag{A8}$$

for all destinations and subsequently routes each commodity flow according to these fractions [6].

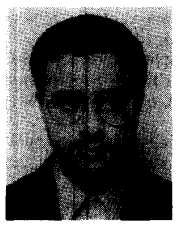
REFERENCES

[1] M. Schwartz, *Telecommunication Networks: Protocols, Modeling and Analysis.* Reading, MA: Addison-Wesley, 1987.
[2] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proc. Nat. Acad. Sci.,* vol. 79, pp. 2554–2558, 1982.
[3] J. J. Hopfield and D. W. Tank, "'Neural' computations of decisions in optimization problems," *Biol. Cybern.,* vol. 52, pp. 141–152, 1986.
[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. Circuits. Syst.,* vol. CAS-33, no. 5, pp. 533–541, 1986.
[5] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," *Proc. Nat. Acad. Sci.,* vol. 81, pp. 3088–3092, 1984.
[6] D. P. Berteskas and R. G. Gallager, *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1987.
[7] H. E. Rauch and T. Winarske, "Neural networks for routing communication traffic," *IEEE Cont. Syst. Mag.,* pp. 26–30, Apr. 1988.
[8] L. Zhang and S. C. A. Thomopoulos, "Neural network implementation of the shortest path algorithm for traffic routing in communication networks," in *Proc. Int. Joint Conf. Neural Networks,* June 1989, p. II. 591.
[9] A. Moopenn, A. P. Thakoor, and T. Duong, "A neural network for Euclidean distance minimization," in *Proc. IEEE Int. Conf. Neural Networks,* 1988, pp. II.349–356.
[10] P. W. Protzel, "Comparative performance measure for neural networks solving optimization problems," in *Proc. Int. Joint Conf. Neural Networks,* 1990, pp. II.523–526.
[11] R. D. Brandt *et al.,* "Alternative networks for solving the traveling salesman problem and the list-matching problem," in *Proc. IEEE Int. Conf. Neural Networks,* 1988, pp. II. 333–340.
[12] A. Moopenn, T. Duong, and A. P. Thakoor, "Digital–analog hybrid synapse chips for electronic neural networks," in *Advances in Neural Information Processing Systems,* vol. 2, D. S. Touretzky, Ed. Morgan Kaufmann, 1990, pp. 769–776.
[13] V. Wilson and G. S. Pawley, "On the stability of the TSP problem algorithm of Hopfield and Tank," *Biol. Cybern.,* vol. 58, pp. 63–70, 1988.
[14] A. Kahng, "Traveling salesman heuristics and embedding dimension in the Hopfield model," in *Proc. IJCNN,* vol. I, 1989, pp. 513–520.
[15] L. Kleinrock, *Communication Networks: Stochastic Message Flow and Delay.* New York: McGraw-Hill, 1964.
[16] L. Fratta, M. Gerla, and L. Kleinrock, "The flow deviation method: An approach to store-and-forward communication network design," *Networks,* vol. 3, pp. 97–133, 1973.
[17] D. G. Cantor and M. Gerla, "Optimum routing in a packet switch computer network," *IEEE Trans. Commun.,* vol. COM-23, no. 10, pp. 1062–1069, 1974.
[18] M. Schwartz and C. K. Cheung, "The gradient projection algorithm for multiple routing in message switched systems," *IEEE Trans. Commun.,* vol. COM-24, no. 4, pp. 449–456, 1976.
[19] J. F. Hayes, *Modeling and Analysis of Computer Communications Networks.* New York: Plenum, 1984.
[20] P. J. Courtois and P. Semal, "A flow assignment algorithm based on the flow deviation method," in *Proc. ICCC 1980,* pp. 77–83.
[21] M. Takeda and J. W. Goodman, "Neural networks for computation: Number representations and programming complexity," *Appl. Opt.,* vol. 25, no. 18, pp. 3033–3046, 1986.



Mustafa K. Mehmet Ali received the B.Sc. and M.Sc. degrees from Bogazichi University, Istanbul, Turkey, in 1977 and 1979, respectively, and the Ph.D. degree from Carleton University, Ottawa, Ont., Canada, in 1983, all in electrical engineering.

He worked as a Research Engineer at TELESAT Canada until the end of 1984. Since then he has been with the Department of Electrical and Computer Engineering at Concordia University, Montreal, P.Q., Canada, where at present he is an Associate Professor. His current research interests are applications of neural networks in communications and performance analysis of computer networks.



Faouzi Kamoun was born in Sfax, Tunisia, on March 30, 1965. He received the B.Eng. (with distinction) and M.A.Sc. degrees in electrical engineering from Concordia University, Montreal, P.Q., Canada, in 1988 and 1990, respectively.

He is currently working toward the Ph.D. degree in electrical engineering at the same university. His research interests are in the areas of teletraffic theory, modeling and performance analysis of communication networks, and the application of neural networks to telecommunication.

In 1988, Mr. Kamoun was awarded the Electrical Engineering Medal for most outstanding graduating student in electrical engineering at Concordia. From 1985 to 1990 he was awarded a scholarship from the Ministry of Higher Education and Scientific Research, Tunisia. He has been holding a Concordia University Graduate Fellowship since 1991.