A Neural Network for Shortest Path Computation

Filipe Araújo, *Associate Member, IEEE*, Bernardete Ribeiro, *Member, IEEE*, and Luís Rodrigues, *Member, IEEE*

***Abstract*—This paper presents a new neural network to solve the shortest path problem for internetwork routing. The proposed solution extends the traditional single-layer recurrent Hopfield architecture introducing a two-layer architecture that automatically guarantees an entire set of constraints held by any valid solution to the shortest path problem. This new method addresses some of the limitations of previous solutions, in particular the lack of reliability in what concerns successful and valid convergence. Experimental results show that an improvement in successful convergence can be achieved in certain classes of graphs. Additionally, computation performance is also improved at the expense of slightly worse results.**

***Index Terms*—Neural networks, shortest path computation problem, two-layer Hopfield neural network.**

I. Introduction

THE PROBLEM of finding the shortest path (SP) from a single source to a single destination in a graph arises as a subproblem to many broader problems, including routing problems in computer networks. This problem has some well-known polynomial algorithmic solutions, namely Bellman–Ford's [2], [5] or Dijkstra's.

Problems that require multiple or extremely fast computations of SP, such as the *quasistatic bifurcated routing problem* in packet switched computer networks [8], can benefit from more efficient methods of finding the SP. Motivated by this class of problems, a number of attempts using neural networks (NNs), namely Hopfield NN [4], were made to solve or provide an approximate solution to the SP problem faster than would be possible with any algorithmic solution, relying on the NN parallel architecture.

This idea was first presented by Rauch and Winarske [7]. To solve some limitations of this first method an extension was introduced by Zhang and Thomopoulos [9] which made possible to find a path with an arbitrary number of hops.

Ali and Kamoun [1] proposed a new method that aimed at NN adaptability to external varying conditions as in a computer network where links or nodes may go up and down easily. This method has two major drawbacks: first, the NN often fails to converge toward a valid solution and this problem worsens when the number of nodes in the graph increases. This makes Ali and Kamoun's method nearly useless in certain classes of graphs, when the number of nodes approaches 40; second, the method finds poor solutions when compared to optimum solutions found by Dijkstra's algorithm.

Manuscript received April 20, 2000; revised January 24, 2001. This work was supported in party by Praxis/C/EEI/12202/1998, TOPCOM.

F. Araújo and L. Rodrigues are with the Faculdade de Ciências of Universidade de Lisboa, 1749–016 Lisboa, Portugal.

B. Ribeiro is with the Centro de Informática e Sistemas, Departamento de Engenharia Informática of Universidade de Coimbra, Coimbra, Portugal.

Publisher Item Identifier S 1045-9227(01)05535-7.

An evolution of the Ali and Kamoun's method, thought for the multidestination routing problem, was introduced by Park and Choi [6]. When used in a single-destination version it extends the range of operation of the former method, achieving noticeable improved solutions even with a bigger number of nodes. In spite of these advantages, Park and Choi's NN still fails to converge too many times and presents poorer behavior with increasing number of graph nodes in certain classes of graphs. All these solutions demand a number of neurons that squares the number of graph nodes.

This paper presents a new Hopfield NN that aims to improve the reliability of the solutions, where reliability stands for successful and valid convergence. To achieve this, a new architecture, named dependent variables (DVs), which consists of a two-layer Hopfield NN is presented. This architecture automatically guarantees an entire class of restrictions, considerably increasing the reliability of the method. At the same time, the number of neurons is equal to the number of arcs in the graph rather than being equal to the squared number of nodes as it is the case in Ali–Kamoun's and Park–Choi's NN. Thus, in general, the proposed architecture needs fewer neurons and neurons' connections. Only in the worst-case scenario, where all graph nodes are connected to each other, the number of neurons is equal. The price to pay for this reduced number of neurons is that the NN is harder to adapt to topology changes such as new nodes or connections. However, our NN retains the main advantage of the Ali and Kamoun's NN, namely the ability to adapt to changes in arc costs, since these are, once again, coded in neurons biases.

This paper is organized as follows. Section II defines the single source–destination SP problem. The new dependent variable Hopfield NN (DVHNN) is described in Section III and its performance is evaluated in Section IV. Section V concludes this paper.

II. Problem Definition

Consider a directed graph $G = (V, A)$ composed of a set of N vertices—V—and a set of M directed arcs—A. Associated with each arc (r, s) is a nonnegative number C_{rs} that stands for the cost from node r to node s. Nonexisting arc costs are set to infinity (∞). Often, for clarity of exposition, namely when referring to figures, we will label each existing arc with a unique index and denote its cost simply by C_i.

Let P_{sd} be a path from a source node s to a destination node d, defined as a set of consecutive nodes, connected by arcs in A

$$P_{sd} = \{s, n_1, n_2, \ldots, d\}.$$

There is a cost associated with each path P_{sd} which consists of the sum of all partial arc costs participating in the path. The shortest path problem consists in finding the path connecting a

given source–destination pair, (s, d), such that the cost associated with that path is minimum. Stated as an integer linear programming problem this is [3]

$$\text{Minimize} \quad \sum_{(i,j)\in A} C_{ij} v_{ij} \tag{1}$$

$$\text{subject to} \quad \sum_{\{j:\,(i,j)\in A\}} v_{ij} - \sum_{\{j:\,(j,i)\in A\}} v_{ji} = \phi_i \qquad i = 1, \ldots, N \tag{2}$$

$$\text{and} \quad v_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A. \tag{3}$$

[Here double indexes are used because (2) is thus easier to state.]

Where

- v_{ij} is the participation of the arc (i, j) in the path which can only be 0 or 1, i.e., the arc whether participates entirely or does not participate at all in the path. Nonexisting arcs are not to be considered.
-

$$\phi_i = \begin{cases} 1 & \text{if } i = s \\ -1 & \text{if } i = d \\ 0 & \text{otherwise.} \end{cases}$$

The set of N equations (2) can be stated in matrix form, though only $N - 1$ equations are linearly independent (thus, one of them is omitted). In addition, the double indexes in v_{ij} are replaced by corresponding single index variables. The resulting equation is then

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & & & \\ a_{N-1,1} & a_{N-1,2} & \cdots & a_{N-1,M} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_M \end{bmatrix} = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{N-1} \end{bmatrix} \Longleftrightarrow \mathbf{Av} = \phi \qquad v_i \in \{0, 1\}. \tag{4}$$

In this equation, $\mathbf{A}$ is an $(N-1) \times M$ matrix which depends on graph configuration, ϕ is a vector with $(N - 1)$ elements, which enables path source and destination specification and $\mathbf{v}$ is a vector with M elements, where each one represents the participation of a single arc of the directed graph in the selected path.

The constraints expressed in (4), that we will call Kirchoff's constraints, are of considerable importance, because they are held by any valid solution to the SP problem (though the converse is not necessarily true).

III. Dependent Variables Hopfield Neural Network

A. Kirchoff's Constraints

To build an NN that guarantees constraints (4), we will first rewrite them to the form presented in (5), using Gauss–Jordan elimination. The computational order of Gauss–Jordan elimination is not relevant and a method to achieve an equivalent result will be presented later in the paper.

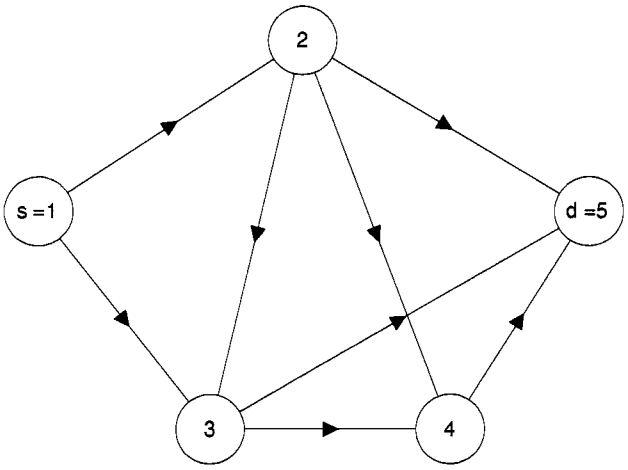

Fig. 1. Graph example.

dependent variables

$$\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{bmatrix} = \begin{bmatrix} \alpha_{1N} & \alpha_{1,N+1} & \cdots & \alpha_{1M} \\ \alpha_{2N} & \alpha_{2,N+1} & \cdots & \alpha_{2M} \\ \vdots & & & \\ \alpha_{N-1,N} & \alpha_{N-1,N+1} & \cdots & \alpha_{N-1,M} \end{bmatrix}$$

independent variables

$$\cdot \begin{bmatrix} v_N \\ v_{N+1} \\ \vdots \\ v_{M-1} \\ v_M \end{bmatrix} + \begin{bmatrix} o_1 \\ o_2 \\ \vdots \\ o_{N-1} \end{bmatrix}. \tag{5}$$

The variables $v_1, \ldots, v_{N-1}$ are dependent variables and will be represented by dependent neurons in the NN to be presented next, while variables $v_N, \ldots, v_M$ are independent variables, that will be represented by independent neurons.

Only independent neurons have internal motion equation depending on the problem data and the present NN state, capable of guiding the entire NN toward the SP problem solution. On the other hand, dependent neurons are simpler devices that perform the linear computation determined in (5). However, both independent and dependent neurons need activation functions, to be presented later. Any variable v_j may be considered as dependent or independent; it is only a question of solving equation (4) for v_j or for other $N - 1$ variables, but not for v_j, respectively. The algorithm used to choose a variable to be dependent or independent is presented in Section III-F.

Consider as an example the graph represented in Fig. 1. Here, node 1 is the source and node 5 is the destination of the path. Arcs (1, 3), (2, 4), (2, 5) and (3, 5) may be represented by independent neurons while neurons representing arcs (1, 2), (2, 3), (3, 4) and (4, 5) will then depend linearly on former neurons values, for this selection of dependent and independent variables, as shown in (6)

$$\begin{aligned} &(1) \quad v_{12} = 1 - v_{13} \\ &(2) \quad v_{23} = 1 - v_{13} - v_{24} - v_{25} \\ &(3) \quad v_{34} = 1 - v_{24} - v_{25} - v_{35} \\ &(4) \quad v_{45} = 1 - v_{25} - v_{35}. \end{aligned} \tag{6}$$

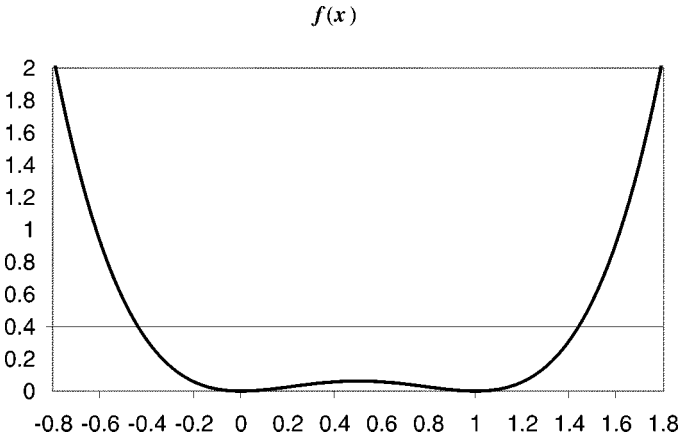

Fig. 2. Neuron energy function.

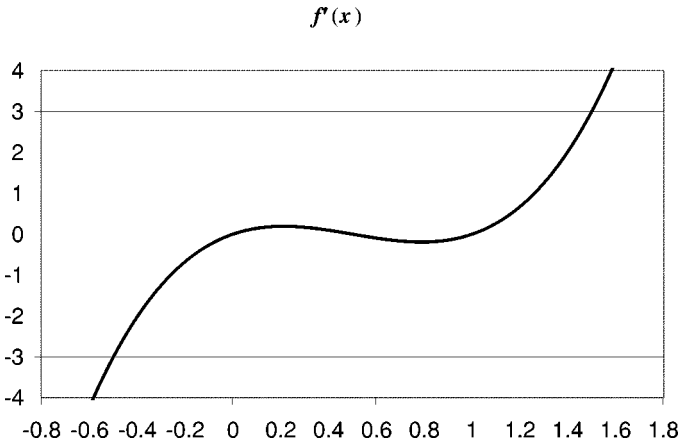

Fig. 3. Neuron activation function (energy function derivative).

B. Binary Outputs

Even if Kirchoff's constraints are held, as in equations (6), it is still necessary to ensure that $v_i \in \{0, 1\}$, for $i = 1, \ldots, M$. In order to do that, a Lyapunov energy function is defined for the NN

$$E = \sum_{i=1}^{M} \rho_1 f(v_i). \tag{7}$$

In (7), ρ_1 is a positive constant and $f(x)$ is a function with zeros at $x = 0$ and $x = 1$. In (8) a function with these properties is presented. This function is differentiable, which is an important characteristic for DVHNN method as it will be seen shortly. $f'(x)$ is presented in (9). These functions are depicted in Figs. 2 and 3. The energy function (7) will be minimum when all the v_is have binary values of 0 or 1.

If (4) is met and all the variables v_i have binary output values then the solution is valid, though not necessarily optimal.

$$f(x) = (x-0)^2(x-1)^2 = x^4 - 2x^3 + x^2 \tag{8}$$

$$f'(x) = 4x^3 - 6x^2 + 2x. \tag{9}$$

C. Neuron Motion Equation

Only the motion equation of independent neurons (v_i, with $N \leq i \leq M$) needs to be defined since dependent neurons have a totally conditioned behavior. The neuron to be used comprises a summation unit and a capacitor, as depicted in Fig. 4. The corresponding electrical equation is expressed in (10) without any loss of generality.

$$\frac{dv_i(t)}{dt} = \sum_{j=1}^{N} w_{ij} x_j(t) + I_i. \tag{10}$$

For the NN to follow a gradient-descent of the energy function, the neuron motion equation is defined in (11) similarly to an Hopfield NN (k is a positive constant). Then, by (12) and since $E \geq 0$, the DVHNN should evolve to an energy minimum (though, this does not ensure by itself that the NN may not cycle between equivalent energy states with $E > 0$)

$$\frac{dv_i}{dt} = -k \frac{\partial E}{\partial v_i} \tag{11}$$

$$\frac{dE}{dt} = -k \sum_{i=N}^{M} \left(\frac{\partial E}{\partial v_i} \right)^2 \leq 0. \tag{12}$$

Since $\partial v_j / \partial v_i = 0$, $i, j = N, \ldots, M$, $i \neq j$ (v_i, v_j are both independent neurons) and $\partial v_j / \partial v_i = \alpha_{ji}$, for $j = 1, \ldots, N-1$, $i = N, \ldots, M$, (v_j is a dependent neuron and v_i independent) (13) and (14) yield, where $\mu_1 = k\rho_1$

$$\frac{\partial E}{\partial v_i} = \rho_1 f'(v_i) + \rho_1 \sum_{j=1}^{N-1} \alpha_{ji} f'(v_j) \tag{13}$$

$$\frac{dv_i(t)}{dt} = -\mu_1 f'(v_i(t)) - \mu_1 \sum_{j=1}^{N-1} \alpha_{ji} f'(v_j(t)). \tag{14}$$

Comparing (10) with (14) the following holds, for any neuron l and for an independent neuron i

$$x_l = f'(v_l) \tag{15}$$

$$w_{il} = \begin{cases} -\mu_1 & \text{if } l = i \\ 0 & \text{if } l \neq i,\ l = N, \ldots, M \\ -\mu_1 \alpha_{li} & \text{if } l = 1, \ldots, N-1 \end{cases} \tag{16}$$

$$I_i = 0. \tag{17}$$

A relevant fact that stems from this result is that feedback to the NN is done through independent neurons with neuron outputs affected by individual derivative functions $f'(x)$ which play the role of activation functions.

For the graph presented in Fig. 1 the DVHNN depicted in Fig. 5 would be used, where independent neurons are at the bottom, while dependent neurons are at the top. Recurrent connections (top to down in the picture) are all affected by the activation function $f'(x)$.

D. Cost Consideration

The NN presented until this point considers only the validity of the solution but ignores arc costs. In order to take arc costs into consideration, the following method is used:

- change the energy function to consider the costs;
- start NN and let it converge to a final result which may not be valid but that considers arc costs;

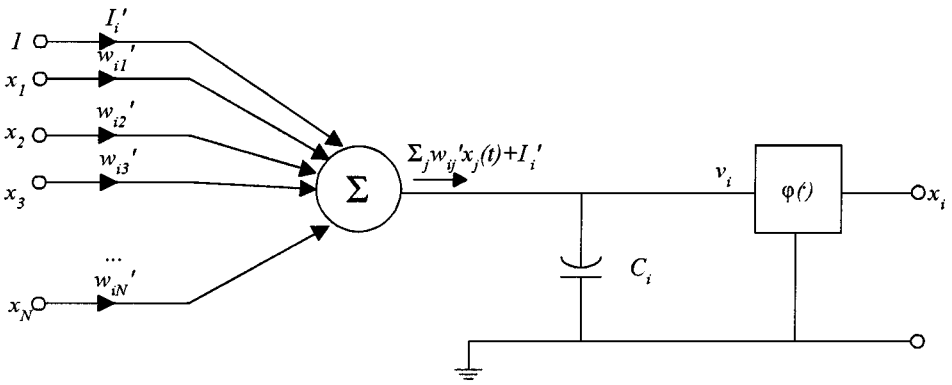

Fig. 4. Neuron model.

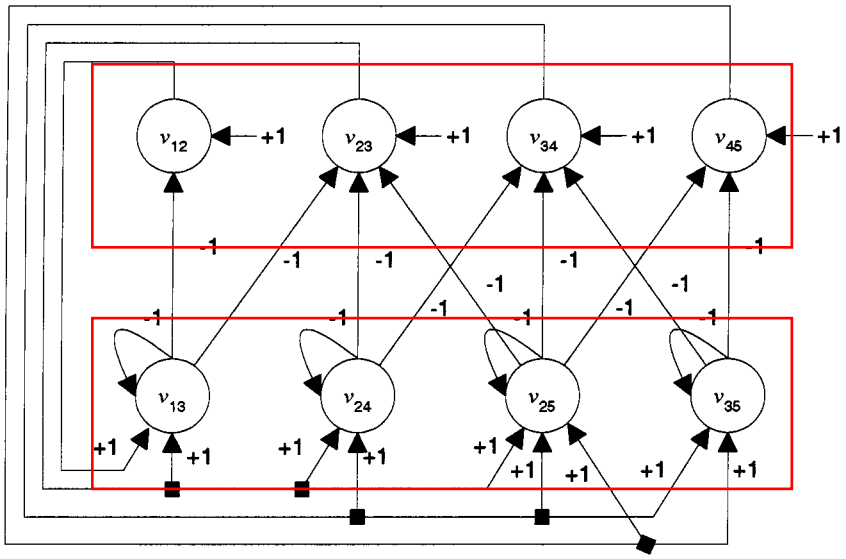

Fig. 5. Example of dependent variables Hopfield neural network.

- from this point of convergence, eliminate the costs from energy function and let NN converge again, this time to a valid solution.

Cost consideration is easy to implement because costs will be introduced in neuron biases. Therefore it is easy to take it into account or ignore it just by controlling the neuron biases.

The energy function presented in (18) differs from (7) in the term affected by ρ_2 (C_i stands for the cost of arc i)

$$E = \sum_{i=1}^{M} (\rho_1 f(v_i) + \rho_2 C_i v_i). \tag{18}$$

Using this energy function, (15) and (16) still hold, but (17) must be replaced by (19), for $i = N, \ldots, M$

$$I_i = -\mu_2 C_i - \mu_2 \sum_{j=1}^{N-1} \alpha_{ji} C_j. \tag{19}$$

So, it is simple to either consider costs or not: just set I_i as in (19) or $I_i = 0$, respectively, as in (17). It is important to note that dependent variables cost is coded in independent variables, which are the only ones represented by neurons that can vary freely.

E. NN Convergence

The convergence of the algorithm is affected by the existence of equivalent cost paths. These may lead the NN to an indefinite final state where it cannot decide for a single alternative. To solve this, some random noise is added to the independent neurons biases. If b was the bias to apply to some neuron, the real bias should be randomly selected in the interval $[(1-\gamma/2)b, (1+\gamma/2)b]$ around b. The experimental results shown in Section IV consider $\gamma = 5\%$. This procedure enables symmetry breaking.

Alternative techniques that first consider and then eliminate costs can also be used to improve convergence. For instance, independent neurons biases could be halved each time the NN converged. Although possibly more accurate, this method consumes much more time, because instead of converging only twice, the NN must converge a predefined number of times. Thus, there is a tradeoff between convergence time and solution quality.

In the experimental results presented in this paper, the first method described in this section (eliminate the biases at once) is used.

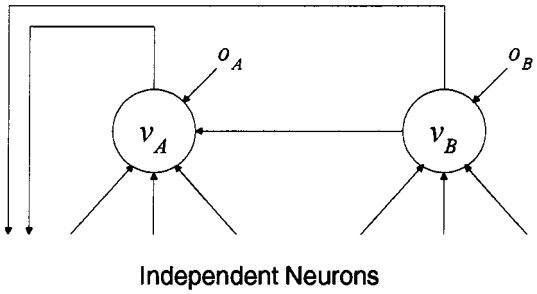

Fig. 6. Dependent variable v_A depends on another dependent variable, v_B.

F. Dependent Variable Selection

As stated before, to find an equation in form (5) as needed by a DVHNN, it is not efficient to use Gauss–Jordan elimination for each source–destination pair.

To solve the problem, (5) is first extended to let dependent variables depend on other dependent variables. This still enables NN to converge as long as there are no closed cycles dependencies (as we shall ensure below), i.e., given dependent variables v_A and v_B, v_A cannot depend on v_B if v_B previously depended on v_A. Note that v_A is also said to depend on v_B if v_A depends on v_C and v_C depends on v_B, and this definition is transitive. Equation (21) shows dependent variable v_A depending on dependent variable v_B defined in (20)

$$v_B = \sum_{k=N}^{M} \alpha_{Bk} v_k + o_B \tag{20}$$

$$v_A = \alpha_{AB}\left(\sum_{k=N}^{M} \alpha_{Bk} v_k + o_B\right) + \sum_{k=N}^{M} \alpha_{Ak} v_k + o_A. \tag{21}$$

The connection between these two neurons is represented in Fig. 6. The advantage of doing this is that the o_is, from (5), represented again in (20) and (21) (o_A and o_B) should correspond to ϕ_i in (2) thus making it possible to switch source–destination pair instantaneously as described below.

To find all the dependent variable equations such as (21) (without any cycles between DV) an algorithm is presented in Fig. 7. The algorithm keeps three sets of nodes: T—treated nodes; V—neighbors of treated nodes; N—all other nodes. In the beginning T and V are empty and N contains all the nodes.

In the end, all nodes will belong to T, though all but the first one which enters there directly must pass in V before entering T. When a node s passes from V to T, the respective restriction equation (2) is used and a dependent variable is chosen from this equation. A node enters V whenever one of its neighbors enters T. So, for s to be in V, there must be at least one neighbor r which is already in T.

Then, the DV to choose is exactly rs if it exists or sr otherwise (one of them must exist, otherwise r and s would not be neighbors). Nor rs, neither sr may participate in any other equation to be written in a further step of the algorithm since s is added to T and r has already been added to T in a previous step. Now, rs/sr (which one was used does not matter) may depend on another dependent variable to be chosen later but not on a dependent variable already chosen and the same applies to each

```
Initialize set N to contain all the nodes
Initialize two empty sets: V (neighbors) and T (treated)
For  number of nodes - 1 times
    if T = ∅ then
        Select any node s from N
        N ← N\{s} ; T ← T ∪ {s}
        Forall n; n is neighbor of s
            N ← N\{n}; V ← V ∪ {n}
    else
        if V = ∅ then
            Stop (with an error if N is not empty)
        else
            Select any node s from V
            Select r: r is neighbor of s and r ∈ T
            if arc rs exists then
                rs is the dependent variable
            else
                sr is the dependent variable
            Write the restriction equation of node s taking into
                consideration the source and destination of the path
            Forall n; n is neighbor of s
                N ← N\{n}; V ← V ∪ {n}
            V ← V\{s}; T ← T ∪ {s}
```

Fig. 7. Algorithm that determines equations representing graph restrictions.

one of the other dependent variables. Therefore cycles cannot occur.

With the algorithm presented above, the o_is are easily determined to be equal to ϕ_i for any node i. This is possible because all the node equations are written directly without any algebraic transformation.

IV. Experimental Results

In this section the behavior of the DVHNN is compared with other methods of finding SP, namely with Dijkstra's algorithm and with Park and Choi's NN (PCNN) [6].

To compare these methods three graphs have been used: Graph 1 and Graph 2 with 40 nodes and identical configurations, but different arc costs all set to 1 in Graph 2 and Graph 3 with 70 nodes. For each graph and for each of the routing methods, some random source–destination pairs (20 to 40) were simulated in order to achieve more accurate comparisons.

The following configurations have been used in the simulated networks. PCNN constants where assigned with the values proposed by its authors [6]: $A = 550$, $B = 2550$, $C = 2150$, $D = 250$ and $F = 1350$. The respective energy function is restated here for self-containment:

$$E = \frac{A}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n} C_{mi}x_{mi} + \frac{B}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n}\rho_{mi}x_{mi} + \frac{C}{2}\sum_{m=1}^{n}\left\{\sum_{\substack{i=1\\ i\neq m}}^{n} x_{mi} - \sum_{\substack{i=1\\ i\neq m}}^{n} x_{im} - \phi_m\right\}^2 + \frac{D}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n} x_{mi}(1-x_{mi}) + \frac{F}{2}\sum_{m=1}^{n}\sum_{\substack{i=1\\ i\neq m}}^{n} x_{mi}x_{im}.$$

Results achieved by both NN in Graph 1, and in particular by PCNN, are considerably improved in terms of successful and valid convergence if graph costs are bounded at small values. Therefore a proportional transformation was applied to all graphs, limiting the highest arc cost to 2. The impact of this bounding can slightly benefit DVHNN and is most significant when poor choices of constants μ_1 and μ_2 are made.

To simulate by software both NN, the fourth-order Runge–Kutta method was applied as described in [1] and [6]. The time step considered was $\Delta t = 10^{-5}$s for PCNN, $\Delta t = 10^{-5}$s for DVHNN in Graph 1 and $\Delta t = 10^{-6}$ for DVHNN in Graphs 2 and 3. This last NN is very sensitive to this parameter and may diverge if the step is too high. This problem would not, of course, occur in a real physical system. Symmetry breaking was introduced in DVHNN in the neurons biases representing the costs, as explained in Section III-E. In PCNN symmetry breaking was introduced in neurons initial activity, randomly set between 0 and 0.0002.

Lacking a method to analytically determine constants μ_1 and μ_2, exhaustive experiments where made in the space defined by μ_1 and μ_2 to determine which is the best possible combination of the two and how results are affected by proportionally changing μ_1 and μ_2. The chosen values for μ_1 and μ_2 used for simulation, result from the intersection of lines $\mu_1 + \mu_2 = 100$, $\mu_1 + \mu_2 = 1000$ and $\mu_1 + \mu_2 = 10\,000$ with several lines crossing the origin: $\mu_1 = \alpha\mu_2$, where $\alpha \in \{0.05, 0.11, 0.18, 0.25, 0.33, 0.45, 0.54, 0.67, 0.82, 1.00, 1.22, 1.5, 1.86, 2.33, 3.0, 4.0, 5.67, 9.0, 19.0\}$,[1] and with line $\mu_2 = 0$ (i.e., $\mu_1/\mu_2 = \infty$). These lines and respective intersections are depicted in Fig. 8.[2] For instance, the line defined by $\alpha \approx 1.22$ intersects the first three lines at points $(\mu_1, \mu_2) = (55, 45)$, $(550, 450)$ and $(5500, 4500)$.

Before presenting results, the concept of failure is defined as a source–destination pair for which the SP method is unable to find a solution. This case only happens with NN, but never with Dijkstra's algorithm (unless no path exists), when the NN gets locked in a local minimum which does not lead to a valid solution whether optimum or not.

Results concerning Graph 1 are presented in Figs. 9 and 10 where the relation μ_1/μ_2 is used as the x-axis. Fig. 9 presents the failure rate for each algorithm, while Fig. 10 presents the average cost of chosen paths. Best results in terms of cost average are achieved with $\mu_1/\mu_2 \approx 1.22$, although costs do not seem to depend much on this relation, except for very small values of μ_1 (little emphasis on convergence success) and μ_2 (little emphasis on solution quality). Note that when μ_2 approaches 0 the results degrade severely, thus proving that the first phase of convergence when the energy function includes the costs is effective.

On the other hand the number of failures is strongly affected not only by the relation μ_1/μ_2, but also by their absolute value, so these should be carefully selected. The relation $\mu_1/\mu_2 \approx 1.22$ seems satisfactory and the best overall results in terms of convergence success are achieved with $\mu_1 + \mu_2 = 1000$, so the constants are set to $(\mu_1, \mu_2) = (550, 450)$. These values were selected to perform the tests presented next. Note, however, that with $(\mu_1, \mu_2) = (5500, 4500)$ almost ten times faster convergence would have been possible with a greater chance of failing convergence.

Comparisons results between different methods are shown in Table I. As it can be seen in Graph 1 and Graph 2 results, the PCNN performs slightly better (3% to 6% smaller costs) than DVHNN whenever it reaches a successful convergence, but

[1]The values in the set are only approximations.

[2]This figure is not scaled.

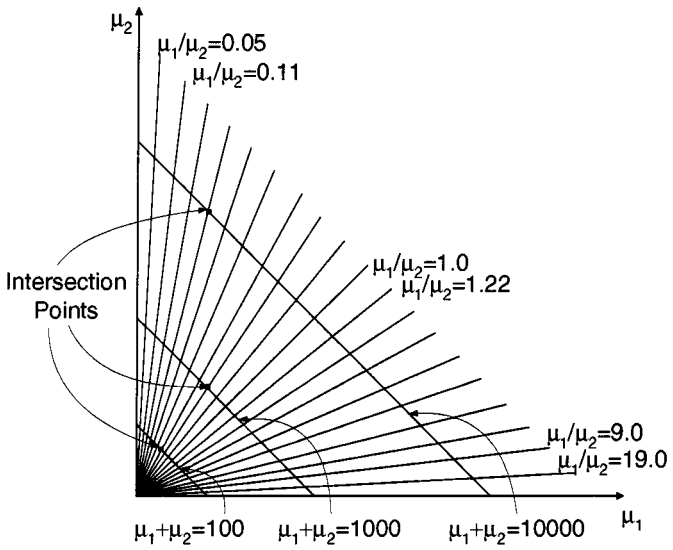

Fig. 8. Selection of μ_1 and μ_2.

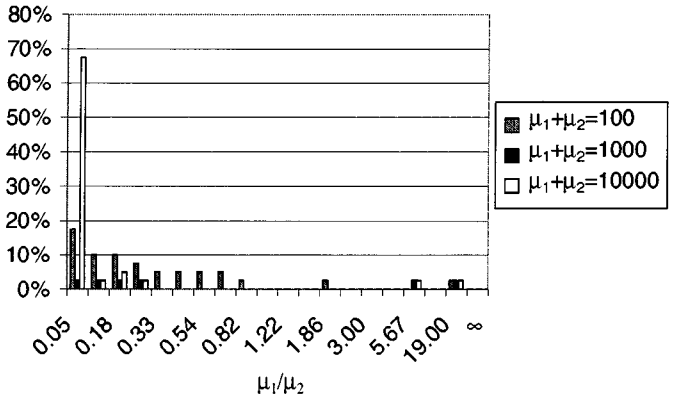

Fig. 9. Failure rate as a function of μ_1/μ_2.

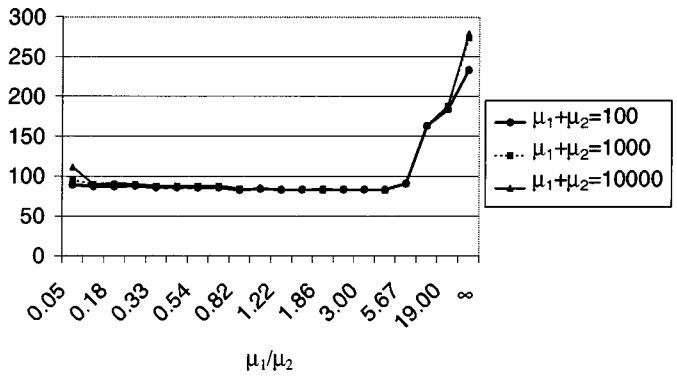

Fig. 10. Average cost as a function of μ_1/μ_2.

TABLE I
SIMULATION RESULTS

Graph 1			
	Dijkstra	PC	DV
Total cost (average)	2603 (72.3)	2613 (72.6)	2702 (75.1)
Failure rate	0	10%	0
Graph 2			
	Dijkstra	PC	DV
Total cost (average)	94 (3.1)	94 (3.1)	100 (3.3)
Failure rate	0	25%	0
Graph 3			
	Dijkstra	PC	DV
Total cost (average)	1132 (87.1)	1132 (87.1)	1243 (95.6)
Failure rate	0	20%	20%

TABLE II
NUMBER OF ITERATIONS AND TIME AVERAGE

	PC	DV
Graph 1 iterations avg. (Time avg.)	7489 (74.89 ms)	1628 (16.28 ms)
Graph 2 iterations avg. (Time avg.)	8485 (84.85 ms)	1226 (12.26 ms)
Graph 3 iterations avg. (Time avg.)	7579 (75.79 ms)	2022 (20.22 ms)

DVHNN successfully converges much more often, proving itself to be a more reliable solution, achieving 100% reliability with these graphs. An interesting fact is that PCNN performs worse with equal arc costs (failing 25% of the times in Graph 2). In Graph 3 both NN have a failure rate of 20% while the advantage in terms of solution quality goes once again to PCNN (10% smaller costs).

Regarding the number of iterations needed, DVHNN shows the best results, needing only 22% of the iterations, which means, 22% of the time in Graph 1, 14% of the time in Graph 2 and 27% of the time in Graph 3, as can be seen in Table II. The number of iterations was normalized for a time step of $\Delta t = 10^{-5}$ (note that an iteration with step $\Delta t = 10^{-5}$ is worth ten iterations with step $\Delta t = 10^{-6}$).

V. CONCLUSION

A new method to solve the shortest path problem was proposed using a two-layer Hopfield NN. This solution aims to achieve an increased number of successful and valid convergences, which is one of the main limitations of previous solutions based on NNs. Additionally, in general, it requires less neurons and less time to converge.

In all experiments, the DVHNN offers a significantly better computational performance when compared to other NN solutions, at the expense of a slight degradation in the quality of the solutions. Additionally, the reliability of the DVHNN has also shown improvements in several configurations, exhibiting successful convergence in all experiments with smaller graphs.

Some open issues deserve further work: the number of failures in larger graphs; the convergence to suboptimal results; and finally, the adaptability to external varying conditions, in particular, to different graph topologies, which may not be trivial to achieve with the proposed architecture.

REFERENCES

[1] K. Mustafa, M. Ali, and F. Kamoun, "Neural networks for shortest path computation and routing in computer networks," *IEEE Trans. Neural Networks*, vol. 4, pp. 941–953, Nov. 1993.

[2] R. E. Bellman, *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press, 1957.

[3] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models*. Belmont, MA: Athena, 1998.

[4] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," *Biol. Cybern.*, pp. 533–541, 1986.

[5] L. R. Ford, Jr. and D. R. Fulkerson, *Flows in Networks*. Princeton, NJ: Princeton Univ. Press, 1962.

[6] D.-C. Park and S.-E. Choi, "A neural-network-based multidestination routing algorithm for communication network," in *Proc. 1998 IEEE Int. Joint Conf. Neural Networks*, vol. 2, May 1998, pp. 1673–1678.

[7] H. E. Rauch and T. Winarske, "Neural networks for routing communication traffic," *IEEE Contr. Syst. Mag.*, pp. 26–31, Apr. 1988.

[8] M. Scharwtz, *Telecommun. Networks*: Addison-Wesley, 1987.

[9] L. Zhang and S. C. A. Thomopoulos, "Neural network implementation of the shortrest path algorithm for traffic routing in communication networks," in *Proc. Int. Conf. Neural Networks*, 1989, p. 591.

Filipe Araújo (A'01) received the Bachelor's degree in 1996 and the M.Sc. degree in 1999 from the Department of Informatics Engineering, Faculty of Science and Technology, University of Coimbra, Coimbra, Portugal.

He was at Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Leiria. He is currently teaching at Department of Informatics, Faculty of Sciences, University of Lisboa, Lisboa, Portugal. He is a researcher of the Navigators group and a member of the LASIGE laboratory at University of Lisboa.

Bernardete Ribeiro (M'01) received the Bachelor's degree in chemical engineering degree from the Department of Chemical Engineering, University of Coimbra, Coimbra, Portugal, in 1975 and the M.Sc. degree in computer science and the Ph.D. degree in electrical engineering both from the Electrical Engineering Department, University of Coimbra, in 1987 and 1995, respectively.

She is Assistant Professor at the Department of Informatics Engineering, Faculty of Science and Technology, University of Coimbra, Portugal. Her present research interests are soft computing techniques, specifically neural networks and their applications to engineering systems, fault detection and diagnosis, and intelligent process control.

Luís Rodrigues (M'96) received the Bachelor's degree in 1986, the M.Sc. degree in 1991, and the Ph.D. degree in 1996, in electric and computers engineering, from the Instituto Superior Técnico de Lisboa (IST), Lisboa, Portugal.

From 1986 to 1996, he was a member of the Distributed Systems and Industrial Automation Group at INESC. Since 1996, he has been a Senior Researcher of the Navigators group and a founding member of the LASIGE laboratory at University of Lisboa. He is Assistant Professor at Department of Informatics, Faculty of Sciences, University of Lisboa. Previously he was at the Electric and Computers Engineering Department of Instituto Superior Técnico de Lisboa (he joined IST in 1989). His current interests include fault-tolerant and real-time distributed systems, group membership and communication, and replicated data management. He has more than 50 publications in these areas. He is coauthor of a book in distributed systems.

Dr. Rodrigues is a member of the Ordem dos Engenheiros and ACM.