An improved neural network model for the two-page crossing number problem

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Citation: HE, H., ŠÝKORA, O. and MÄKINEN, E., 2006. An improved neural network model for the two-page crossing number problem. IEEE Transactions on Neural Network, 17 (6), pp.1642-1646

Additional Information:

- This article was published in the journal, IEEE Transactions on Neural Networks [© IEEE]. It is also available at: http://ieeexplore.ieee.org/Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Metadata Record: https://dspace.lboro.ac.uk/2134/2382

Publisher: © IEEE Computational Intelligence Society

Please cite the published version.
This item was submitted to Loughborough’s Institutional Repository (https://dspace.lboro.ac.uk/) by the author and is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
An Improved Neural Network Model for the Two-Page Crossing Number Problem

Hongmei He, Ondrej Sýkora, and Erkki Mäkinen

Abstract—The simplest graph drawing method is that of putting the vertices of a graph on a line and drawing the edges as half-circles either above or below the line. Such drawings are called two-page book drawings. The smallest number of crossings over all two-page drawings of a graph \( G \) is called the two-page crossing number of \( G \). Cimikowski and Shope have solved the two-page crossing number problem for an \( n \)-vertex and \( m \)-edge graph by using a Hopfield network with \( 2m \) neurons. We present here an improved Hopfield model with \( m \) neurons. The new model achieves much better performance in the quality of solutions and is more efficient than the model of Cimikowski and Shope for all graphs tested. The parallel time complexity of the algorithm, without considering the crossing number calculations, is \( O(m) \) for the new Hopfield model with \( m \) processors clearly outperforming the previous algorithm.

Index Terms—Energy function, Hopfield model, learning algorithm, motion equation, two-page crossing number.

I. INTRODUCTION

The simplest graph drawing method is that of putting the vertices of a graph on a line (called “spine”) and drawing the edges as half-circles in \( k \) half-planes. Such drawings are called \( k \)-page book drawings, and they correspond to the linear very large-scale integration (VLSI) design. Edge crossing minimization is the most important goal in the linear VLSI design, since a smaller number of crossings means lower costs. The minimal number of edge crossings over all \( k \)-page book drawings of a graph is called the \( k \)-page book crossing number [13]. Correspondingly, if the edges are completely drawn in one of two pages (called two-page book drawings), the smallest number of crossings over all two-page drawings of \( G \) is called the two-page crossing number of \( G \), denoted by \( \nu_2(G) \). The problem is NP-hard [10].

A Hopfield network is a fully connected recurrent single layer and unsupervised network. Hopfield and Tank [8] were the first to use a neural network model for solving optimization problems. Takefuji and Lee [16] used the binary neural network model for the graph planarization problem, and Takefuji [14], [15] also proved that the state of the binary model always converges to a local minimum. It is easy to parallelize a Hopfield network because of its special structure, and it is also easy to implement in hardware.

We study the problem of modeling two-page book drawings with a Hopfield network and present a new model outperforming that of Cimikowski and Shope [2]. Another Hopfield network was presented by Wang and Okazaki [17]. But they have meant the model as a solution for the general crossing number minimization problem, and they tested their model only for complete graphs \( K_n \), \( n = 5, \ldots, 13 \).

II. PREVIOUS RESEARCH

Cimikowski and Shope’s [2] algorithm tries to minimize the number of edge crossings in a nonplanar embedding of a graph. The routing of edges either above or below the spine determines the number of crossings in a fixed linear embedding. The ordering of vertices along the spine also affects the minimum number of crossings obtainable, that is, once an ordering is fixed, the minimum number of crossings for that ordering is not necessarily the global optimum for the graph.

In the Cimikowski and Shope algorithm, each edge is associated with an “up” and a “down” neuron, representing an embedding of the edge in the upper or lower half-plane. Hence, the neurons \( V_{up}^{ij} \) and \( V_{down}^{ij} \) represent the upper and lower connections between vertices \( i \) and \( j \), respectively. The state \( (V_{up}^{ij} = 1, V_{down}^{ij} = 0) \) indicates an upper edge \( e(i, j) \), \( (V_{up}^{ij} = 0, V_{down}^{ij} = 1) \) a lower edge \( e(i, j) \). \( (V_{up}^{ij} = 1, V_{down}^{ij} = 1) \) a double-edge violation, and \( (V_{up}^{ij} = 0, V_{down}^{ij} = 0) \) nonadjacency.

Fig. 1 shows the four conditions for a pair of crossing edges in a linear embedding of a graph: \( V_{up}^{im} = V_{up}^{jm} = 1 \) for \( i < l < m < j \) [Fig. 1(a)] and for \( i < l < j < m \) [Fig. 1(b)]; \( V_{down}^{im} = V_{down}^{jm} = 1 \), for \( l < i < m < j \) [Fig. 1(c)] and for \( i < l < j < m \) [Fig. 1(d)].

The following sums correspond to conditions in Fig. 1(a)–(d). For \( V_{up}^{ij} \)

\[
\sum_{i < j \leq m} f(i, i, m, j) V_{up}^{im} \quad (1)
\]

and for \( V_{down}^{ij} \)

\[
\sum_{i < j \leq m} f(i, i, m, j) V_{down}^{im} \quad (3)
\]

where \( f(i, m, r) = 1 \) if \( l < m < r \), and 0 otherwise.

There are two motion equations for the up and down neurons, respectively [2]. In the motion equation of upper neurons, sums (1) and (2) are inhibitory forces, and sums (3) and (4) are excitatory forces. If the number of crossings above the line for edge \( e(i, j) \) is greater than the number below the line for the same edge, the four terms collectively act as an inhibitory force. If the reverse is true, they act as an excitatory force. Inversely, in the motion equation of down neurons, sums (1) and (2) are excitatory forces, and sums (3) and (4) are inhibitory forces. In addition to the four terms above, each motion equation uses also the terms

\[
V_{up}^{ij} + V_{down}^{ij} - 1 \quad (5)
\]

\[
\text{hull}(V_{up}^{ij}, V_{down}^{ij}) \quad (6)
\]
Equation (5) performs an excitatory force if the edge $e(i, j)$ is absent from the embedding. If only one copy of edge $e(i, j)$ exists, it has no effect, but if both the upper and lower edge $e(i, j)$ are present, it acts as an inhibitory force. The function $\text{hill}(x, y)$ has a value of one, if $x = y = 0$, and zero otherwise. Hence, it performs an excitatory force only when the edge is missing. This hill-climbing term allows the state of the system to escape from a local minimum and allows the system to converge to a feasible solution.

### III. Our Work

We present a new Hopfield model as an improvement of their model. If we use a model of binary neurons [11], the fired neurons indicate the edges which are in one page, e.g., upper side of the spine, and the unfired neurons indicate the edges in the other page (down side of the spine). Hence, we need only $m$ neurons, where $m$ is the number of edges in $G$.

#### A. The Improved Model

Suppose we have a graph $G = (V, E)$ with edges $e_i(x, y)$, $x, y \in \{0, \ldots, n - 1\}$, $x < y$, and a fixed order $(0, \ldots, n - 1)$ of vertices on the spine. Consider a model that consists of $m$ neurons $v_0, \ldots, v_{m-1}$. The condition $(v_i = 1)$ (respectively, $(v_i = 0)$) indicates that edge $e_i$ is above (respectively, below) the spine. The edge list of $G$ is sorted. Namely, for each edge $e_i(x, y)$ in the edge list, we have $x < y$. Therefore, when a pair $e_i(x, y)$ and $e_j(z, w)$ of edges cross, it fulfills one of the following conditions.

**Condition 1:**

$$v_i = 1, v_j = 1 \quad \text{and} \quad x < z < y < w.$$  

**Condition 2:**

$$v_i = 1, v_j = 1 \quad \text{and} \quad z < x < w < y.$$  

**Condition 3:**

$$v_i = 0, v_j = 0 \quad \text{and} \quad x < z < y < w.$$  

**Condition 4:**

$$v_i = 0, v_j = 0 \quad \text{and} \quad z < x < w < y.$$  

According to Conditions 1–4, we define a function $\Delta(x, y, z, w)$ to express the relationship of two edges in one page

$$\Delta(x, y, z, w) = \begin{cases} 1, & \text{if } (x < z < y < w) \lor (z < x < w < y) \\ 0, & \text{otherwise.} \end{cases}$$  

(7)

For the fired neurons (edges above the spine), the crossing number can be calculated by

$$c_{\text{up}} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \Delta(x, y, z, w)v_iv_j.$$  

(8)

Similarly, for the unfired neurons (edges below the spine), the crossing number can be calculated by

$$c_{\text{down}} = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} \Delta(x, y, z, w)v_i^rv_j.$$  

(9)

Therefore, the energy function can be written as

$$E = A \sum_{i=0}^{m-1} \sum_{j=0, j\neq i}^{m-1} \Delta(x, y, z, w)(v_i v_j + v_i^r v_j^r).$$  

(10)

For a binary neuron, it holds $v_i^r = 1 - v_i$. Equation (10) can be written as

$$E = A \sum_{j=0}^{m-1} \sum_{j=0, j\neq i}^{m-1} \Delta(x, y, z, w)(v_i v_j + (1 - v_i)(1 - v_j))$$

$$= A \sum_{j=0}^{m-1} \sum_{j=0, j\neq i}^{m-1} \Delta(x, y, z, w) \times (2v_i v_j - v_i - v_j + 1).$$  

(11)

Hirsch [6] viewed a neural network as a nonlinear dynamic system called neurodynamics, which presents a conceptual and eclectic methodological approach to understanding neural network activity. Assuming the dynamic system with $m$ state variables $v_0, v_1, \ldots, v_{m-1}$, the network motion equation is $du_i/dt = -\partial E/\partial v_i$, where $u_i$ and $v_i$ are the input and output of the $i$th neuron. Therefore, we can write the motion equation as

$$\frac{du_i}{dt} = -A \sum_{j=0, j\neq i}^{m-1} \Delta(x, y, z, w)(2v_i - 1).$$  

(12)

#### B. Learning Algorithms

Takefuji [14], [15] proved that the state of the binary model always converges to a local minimum. This can be formalized as follows.

**Theorem 3.1:** [18] Starting from any initial configuration, any symmetric neural network with energy function $E$ computing in a sequential mode will achieve a stable state after at most $O(p^2)$ computational cycles, where $p = (1)/(2)\Sigma_{m-1}^{m-1}|\Sigma_{m-1}^{m-1}|w_{ij}| + \Sigma_{m-1}^{m-1}$. Moreover, this stable state represents a local minimum of $E$.

Equation (12) shows that the input from feedback is linear to the sum of states of the neural network model. Therefore, we can simply apply a hill-climbing approach to find out the best solution from "the local minimum" described in Theorem 3.1.

We suppose that an order of vertices $\pi = 0, 1, \ldots, n - 1$ has been obtained by some heuristic algorithm. We refer to the vertices by their number in this order. Moreover, we suppose that an ordered edge list $e_0, e_1, \ldots, e_{m-1}$ is created. At the startup, the network is randomly given an initial input $u_i \in \{-1, 1\}$. Correspondingly, the neuron states of the network are randomly initialized with 0 or 1, i.e., all edges are placed in the upper page or in the lower page. According to the value of the motion equation in each iteration, the state of each neuron is updated, and consequently the edge corresponding to the updated neuron is moved between page 1 and page 2. When the neural network arrives at a stable state, the two-page crossing number is calculated and the current embedding is output. The learning procedure is repeated with different initial states of the neurons for several times, and the best embedding is output. The learning algorithm is described as Algorithm 1.

**Learning algorithm of Hopfield neural network**

1: $k = 0$; $v_2 = \text{MAX_INT}$
2: While $(k < \text{max\_iteration})$ do
3: \hspace{1cm} $t = 0$;
4: \hspace{1cm} Randomly initialize the state of neurons $v[0 \ldots m - 1]$;
5: \hspace{1cm} Calculate energy $E$ with (10);
6: \hspace{1cm} repeat
7: \hspace{1.2cm} for $(i = 0$ to $m - 1)$ do
8: \hspace{2cm} Compute $\Delta u_i$ for $\Delta t = 1$ with (12);
9: \hspace{1cm} end for
10: \hspace{1cm} $t = t + \Delta t$;
11: \hspace{1cm} end repeat
12: \hspace{1cm} Compute $\Delta u_i$ for $\Delta t = 1$ with (12);
13: \hspace{1cm} end while
14: Compute $\Delta u_i$ for $\Delta t = 1$ with (12);
9: Compute $u_i(t + 1)$ with $\Delta u_i$;
10: if $(u_i > 0)$ then
11: $v_i = 1$;
12: else
13: $v_i = 0$;
14: end if
15: end for
16: Calculate energy $E$ with (10);
17: $t = t + 1$;
18: until $(\Delta E = 0)$;
19: $local_{v_2} = calculate\_crossings (G, v[0 \ldots m - 1])$;
20: if ($local_{v_2} < v_2$)
21: $v_2 = local_{v_2}$;
22: end if
23: $k = k + 1$;
24: end while

IV. COMPARISON OF THE NEW MODEL WITH THE PREVIOUS MODEL

We compare the new model with the model of Cimikowski and Shope [2] in the following aspects.

A. Structure of the Neural Networks

The Cimikowski and Shope (CS) model uses two neurons ($V_{\text{down}}$ and $V_{\text{up}}$) to correspond to one edge in a graph. Therefore, the model is an $n \times n$ neuron array. The new model uses $m$ neurons to map $m$ edges, as the fired neurons indicate the edges in one page, and the unfired neurons indicate the edges in the other page. For the design of a parallel algorithm, the CS model needs $n^2$ processors, but the new model needs just $m$ processors, if each processor is dispatched for one neuron.

B. Time Complexity

According to the motion function of the new model, the calculation in each neuron takes time $O(m)$ in the worst case. Therefore, a sequential traversal of all $m$ neurons takes time $O(m^2)$. Cimikowski and Shope [2] reported that the sequential time of the CS model is $O(n^2)$.

C. Parameter Robust

For the CS model, setting the penalty parameters $A$ and $B$ to small values can delay convergence considerably, and setting them to large values can adversely affect the solution quality. Generally, values in the range 1, ..., 3 are suggested. To encourage missing edges to enter the embedding, a $C$-value in the same range is recommended. Values above this range can lead to the appearance of too many duplicate edges. Ideally, to further improve the solution quality, these constants should be adjusted to fit each problem instance. Typically, this requires extensive experimentation before a good set of values can be found [2]. However, (12) shows that in the new model, the input from feedback is linear to the sum of states of the neural network model. So there is only one parameter $A$ in (12). This greatly simplifies the complexity of experiments. The following experiments verify this.

D. Learning Procedure

For the CS model, during the learning procedure of the neural network there must be redundant states, such as $(V_{\text{up}} = 1, V_{\text{down}} = 1)$, for a duplicate-edge violation, according to the parameter robustness of the CS model described above. For a missing-edge violation $(V_{\text{up}} = 0, V_{\text{down}} = 0)$, Cimikowski and Shope use a hill climbing to compensate the problem. Namely, if both neurons are zero, then perform the excitatory force hill($V_{\text{up}} = V_{\text{down}} = 0$), which returns one [not $(V_{\text{up}} = 0$ or $V_{\text{down}} = 0$)]. Any violation of duplicate edges or missing edges will lead to the next iteration. For the new model, there is no redundant state to be produced during the learning procedure, and input from feedback is an inhibitory force. Namely, any state will correspond to a solution, as the edges corresponding to the fired neurons are in one side of the spine, and then the remaining edges must be on the other side of the spine. Finally, the network arrives at a stable state (a local minimum). Obviously, the convergence of the new model is much faster than that of the CS model.

V. Experiments

As in the work of Cimikowski and Shope [2], a sequential version of the parallel algorithm was designed in C language for testing purposes. The program runs on a Dell platform with Intel Pentium 4 CPU3.00 GHZ and 1 GB RAM. The test suites include complete graphs, some of which were used by Cimikowski and Shope [2], three-row and four-row meshes, hypercubes tested by Cimikowski [1], complete $p$-partite graphs, and circulant graphs. For these graphs, it is easy to get the vertex order of the graphs, for some of which we know the optimal one-page solutions and their optimal two-page crossing number. We compare our results with those of Cimikowski and Shope or with the known optimal crossing numbers.

A. Test on Complete Graphs

For complete graphs $K_n$, the vertex order does not affect the crossing number. We use complete graphs $K_5$, ..., $K_{10}$ tested also by the Cimikowski and Shope [2]. We denote the new model as HSM. In order to select the parameter $A$, we changed the initialization of the program. The hill climbing iteration runs for a range of $A$ from 0.01 to 1, and the initial input $u_i$ is fixed as 0.5, namely, each neuron state is initialized with 1. The results indicate that no matter how parameter $A$ was selected, the results are the same.

For all graphs $K_n$, $n = 5$, ..., 10, CS and HSM achieve the same crossing numbers. However, as shown in Table I, HSM needs much smaller numbers of iterations for reaching a stable state than CS. Moreover, the time needed for each iteration is much shorter in HSM than in CS. All of the following data indicate that HSM converges quickly, and the states of neurons in each hill climbing application arrive at a stable state in less than ten iterations. Each test run uses hill climbing at most 20 times.

Guy [4] presented an upper bound of the crossing number for the complete graphs.

<table>
<thead>
<tr>
<th>Graphs</th>
<th>$n$</th>
<th>$i$ (CS)</th>
<th>$i$ (HSM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_5$</td>
<td>16</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>$K_6$</td>
<td>15</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$K_7$</td>
<td>19</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$K_8$</td>
<td>22</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$K_9$</td>
<td>13</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>$K_{10}$</td>
<td>21</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>
Theorem 5.1: [4] The crossing number of the complete graph satisfies the inequality
\[ \text{cr}(K_n) \leq \frac{1}{4} \cdot \frac{n}{2} \cdot \left( \frac{n-1}{2} \cdot \frac{n-2}{2} \cdot \frac{n-3}{2} \right) \]
For a graph \( G \), \( \nu_2(G) \geq \text{cr}(G) \). We test larger complete graphs \( K_{11} \) – \( K_{200} \). All results of the tested complete graphs are the same as the upper bound of the standard crossing numbers. Table II presents some of the results, where \( m \) stands for the number of edges, e.g., the number of neurons.

### B. Test on Three-Row and Four-Row Meshes
The optimal one-page drawings are known for three-row meshes [3] and conjectured for four-row meshes [5]. We have the following theorem for the two-page crossing numbers of three-row and four-row meshes.

Theorem 5.2: [5] For any three-row mesh and four-row mesh, the two-page crossing number is zero.

It is easy to get the optimal vertex order of a three-row or four-row mesh graph in linear time. The experiments were done based on the optimal vertex order of the tested meshes. Table III presents the two-page crossing numbers of three-row and four-row meshes tested with CS and HSM, respectively, and their running time in each test. HSM achieves the optimal results in every case, while CS does not, and the running time of HSM is much shorter than that of CS.

### C. Test on Hypercubes
The hypercube \( Q_d \) of dimension \( d \) is a \( d \)-regular graph with \( 2^d \) vertices and \( 2^{d-1} \) edges [1], [9]. Each vertex is labelled by a distinct \( d \)-bit binary string, and two vertices are adjacent if they differ in exactly one bit.

For the \( d \)-dimensional hypercube \( Q_d \), there are up to \( 2^{d-3}d! \) cycles [1], [12]. A \( d \)-bit Gray code corresponds to a Hamiltonian cycle on an \( d \)-dimensional hypercube. We generate the \( d \)-bit Gray code to obtain a Hamiltonian cycle of \( Q_d \) and test hypercubes based on the Hamiltonian cycle with the new model. We compare the results with those presented by Cimikowski [1]. Table IV shows the results by HSM and CS models. It can be seen that HSM achieves better results than CS.

HSM achieves very good performance in running time, because there is no redundant state to be produced during the learning procedure. It can be seen that the running time of HSM on \( Q_8 \) (Table IV) is even less than the running time of CS on \( P_3 \times P_3 \) (Table III). Table IV shows that the \( n_i \) is at most eight. Namely, in each hill climbing, HSM converges to a stable state very quickly.

The experimental results for all tested complete \( p \)-partite graphs and circulant graphs also show that HSM is better than CS in both solution quality and efficiency.

### VI. Conclusion
We designed an improved Hopfield model with \( m \) neurons rather than \( 2m \) neurons for the two-page crossing number problem of a graph. The model exactly reflects the property of the two-page book drawings and avoids redundant states to be produced. So the model achieves much better performance in the quality of solutions and efficiency for all tested graphs than the previous model by Cimikowski and Shope.

### ACKNOWLEDGMENT
The authors would like to thank Prof. R. Cimikowski for providing them with the CS programs for the tests.

### REFERENCES

---

**TABLE II**

<table>
<thead>
<tr>
<th>Graphs</th>
<th>( \nu_2 ) (CS)</th>
<th>( \nu_2 ) (HSM)</th>
<th>time (CS)</th>
<th>time (HSM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>0</td>
<td>0</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>1</td>
<td>0</td>
<td>625</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>6</td>
<td>0</td>
<td>1360</td>
<td>31</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>14</td>
<td>0</td>
<td>2250</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>27</td>
<td>0</td>
<td>3375</td>
<td>47</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>54</td>
<td>0</td>
<td>4954</td>
<td>47</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>90</td>
<td>0</td>
<td>7234</td>
<td>15</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>38</td>
<td>0</td>
<td>2960</td>
<td>32</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>90</td>
<td>0</td>
<td>5296</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>128</td>
<td>0</td>
<td>7740</td>
<td>16</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>149</td>
<td>0</td>
<td>9765</td>
<td>16</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>192</td>
<td>0</td>
<td>12969</td>
<td>15</td>
</tr>
<tr>
<td>( T_{30} \times T_{30} )</td>
<td>327</td>
<td>0</td>
<td>18953</td>
<td>15</td>
</tr>
</tbody>
</table>

---

**TABLE III**

<table>
<thead>
<tr>
<th>Graphs</th>
<th>( \nu_2 ) (CS)</th>
<th>( \nu_2 ) (HSM)</th>
<th>time (CS)</th>
<th>time (HSM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( K_{20} )</td>
<td>190</td>
<td>1620</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>( K_{50} )</td>
<td>435</td>
<td>9555</td>
<td>625</td>
<td>16</td>
</tr>
<tr>
<td>( K_{50} )</td>
<td>780</td>
<td>32490</td>
<td>2047</td>
<td>2047</td>
</tr>
<tr>
<td>( K_{100} )</td>
<td>1225</td>
<td>82800</td>
<td>5422</td>
<td>5422</td>
</tr>
<tr>
<td>( K_{150} )</td>
<td>4950</td>
<td>1440600</td>
<td>114129</td>
<td>114129</td>
</tr>
<tr>
<td>( K_{150} )</td>
<td>11175</td>
<td>7405275</td>
<td>623708</td>
<td>623708</td>
</tr>
<tr>
<td>( K_{200} )</td>
<td>19900</td>
<td>24012450</td>
<td>1913866</td>
<td>1913866</td>
</tr>
</tbody>
</table>

---

**TABLE IV**

<table>
<thead>
<tr>
<th>Graphs</th>
<th>( \nu_2 ) (CS)</th>
<th>( \nu_2 ) (HSM)</th>
<th>( n_i ) (HSM)</th>
<th>time (CS)</th>
<th>time (HSM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( Q_4 )</td>
<td>8</td>
<td>8</td>
<td>2</td>
<td>&lt; 1</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>( Q_5 )</td>
<td>62</td>
<td>60</td>
<td>4</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>( Q_6 )</td>
<td>376</td>
<td>370</td>
<td>4</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>( Q_7 )</td>
<td>1874</td>
<td>1858</td>
<td>6</td>
<td>2063</td>
<td>2063</td>
</tr>
<tr>
<td>( Q_8 )</td>
<td>N/A</td>
<td>8550</td>
<td>8</td>
<td>16153</td>
<td>16153</td>
</tr>
</tbody>
</table>

---

**ACKNOWLEDGMENT**

The authors would like to thank Prof. R. Cimikowski for providing them with the CS programs for the tests.
Global Asymptotical Stability of Recurrent Neural Networks With Multiple Discrete Delays and Distributed Delays

Jinde Cao, Kun Yuan, and Han-Xiong Li

Abstract—By employing the Lyapunov-Krasovskii functional and linear matrix inequality (LMI) approach, the problem of global asymptotical stability is studied for recurrent neural networks with both discrete time-varying delays and distributed time-varying delays. Some sufficient conditions are given for checking the global asymptotical stability of recurrent neural networks with mixed time-varying delays. The proposed LMI result is computationally efficient as it can be solved numerically using standard commercial software. Two examples are given to show the usefulness of the results.

Index Terms—Discrete delays, distributed delays, global asymptotical stability, linear matrix inequality (LMI), recurrent neural networks (RNNs), time-varying delays.

I. INTRODUCTION

Recurrent neural networks (RNNs) have found successful applications in many areas such as image processing, signal processing, pattern recognition, and optimization problems. Recently, there has been a rapidly growing research interest on the dynamical properties of RNNs, including stability, attractiveness, oscillation, synchronization, bifurcation, and chaos, and so on; see [1]–[18]. It has been recognized that time delay is often one of the main sources of instability, oscillation, bifurcation, chaos, and poor performance of neural networks [1], [11]. In [2], Marcus and Westervelt incorporated a single time delay into neural networks and observed the sustained oscillation resulting from time delay. Further detailed investigation on dynamics of some classes of neural network models with discrete time-delays can be found in [3]–[15] and some references cited therein.

Since a neural network usually has a spatial nature due to the presence of an amount of parallel pathways of a variety of axon sizes and lengths, it is desired to model them by introducing continuously distributed delays over a certain duration of time, such that the distant past has less influence compared to the recent behavior of the state [15], [16]. Therefore, when modelling neural networks, both the discrete and distributed time delays should be taken into account [17]. To the best of our knowledge, the stability problem for general RNNs with both discrete and distributed delays has received little research attention [16], [17], mainly due to the mathematical difficulties in dealing with discrete and distributed delays simultaneously. Hence, it is our intention in this paper to tackle such an important yet challenging problem. In [16] and [17], the authors only considered the cases with constant delays.

Motivated by the above discussions, in this paper we will consider the following generalized neural networks with both multiple time-varying discrete delays and distributed delays:

\[
\frac{du(t)}{dt} = -Au(t) + Bf(u(t)) + \sum_{k=1}^{n} B^{(k)} f(u(t - \tau_k(t))) + C \int_{-\tau(t)}^{0} g(u(s)) \, ds + I
\]

where \( u(t) = [u_1(t), u_2(t), \ldots, u_n(t)]^T \) is the neuron state vector; \( A = diag(a_1, a_2, \ldots, a_n) \) is a positive diagonal matrix; and \( B = [b_{ij}]_{i=1}^{n}, B^{(k)} = [b_{ij}^{(k)}]_{i=1}^{n}, C = [c_{ij}]_{i=1}^{n} \) denote, respectively, the connection weights matrix, the discretely delayed connection weights matrix and the distributively delayed connection weights matrix. \( f(u) = [f_1(u_1), f_2(u_2), \ldots, f_n(u_n)]^T \) and \( g(u) = [g_1(u_1), g_2(u_2), \ldots, g_n(u_n)]^T \) denote the neuron activation functions with \( f(0) = g(0) = 0: I = [I_1, I_2, \ldots, I_n] \) a constant input vector and \( \tau(t) \) and \( \tau_k(t), k = 1, 2, \ldots, r \) are the time-varying delays.

Throughout this paper, the activation functions and the time-varying delays are assumed to satisfy the following assumptions:

\( (H_1) \) \( f_i(\cdot) \) and \( g_i(\cdot), i = 1, 2, \ldots, n \) are bounded and satisfy the following condition:

\[
0 \leq \frac{f_i(x) - f_i(y)}{x - y} \leq \sigma_i, \quad 0 \leq \frac{g_i(x) - g_i(y)}{x - y} \leq \sigma_i,
\]

\( \forall x, y \in \mathbb{R}^n, x \neq y \quad i = 1, 2, \ldots, n \).

\( (H_2) \) \( \tau(t) \) and \( \tau_k(t) \) are positive, bounded, and differentiable functions with \( 0 \leq \tau(t) \leq \tau, 0 \leq \tau_k(t) \leq \tau, \) and \( \tau(t) \leq \eta < 1, \tau_k(t) \leq \eta_k < 1 \).

The assumption \( (H_1) \) always ensures the existence of the equilibrium point \( u^* \). We shift the equilibrium point \( u^* \) of the system (1) to the origin by the transformation \( x^* = u - u^* \), which changes system (1) to

\[
\frac{dx(t)}{dt} = -Ax(t) + B \tilde{f}(x(t)) + \sum_{k=1}^{n} B^{(k)} \tilde{f}(x(t - \tau_k(t))) + C \int_{-\tau(t)}^{0} \tilde{g}(x(s)) \, ds
\]

where \( x(t) = [x_1(t), x_2(t), \ldots, x_n(t)]^T \) is the state vector of the transformed system and the transformed neuron activation functions are \( \tilde{f}(x) = (\tilde{f}_1(x_1), \ldots, \tilde{f}_n(x_n)) = f(u(t) - f(u^*) \) and \( \tilde{g}(x(s)) = (\tilde{g}_1(x_1(s)), \ldots, \tilde{g}_n(x_n(s))) = g(u(t)) - g(u^*) \) for \( \tilde{g}_i(\cdot) \) and \( \tilde{g}_i(\cdot) \) satisfy \( (H_1) \) and we can derive that

\[
\tilde{g}_i(x) - \tilde{g}_i(u^*) - \tilde{\sigma}_i x_i \leq 0 \quad \forall x \in \mathbb{R}^n, i = 1, 2, \ldots, n.
\]