A Hopfield Neural Network Model for the Outerplanar Drawing Problem

Hongmei. He, Ondrej. Sýkora *

Abstract

In the outerplanar (other alternate concepts are circular or one-page) drawing, one places vertices of a $n-$vertex $m-$edge connected graph G along a circle, and the edges are drawn as straight lines. The minimal number of crossings over all outerplanar drawings of the graph G is called the outerplanar (circular, convex, or one-page) crossing number of the graph G. To find a drawing achieving the minimum crossing number is an NP-hard problem. In this work we investigate the outerplanar crossing number problem with a Hopfield neural network model, and improve the convergence of the network by using the Hill Climbing algorithm with local movement. We use two kinds of energy functions, and compare their convergence. We also test a special kind of graphs, complete p-partite graphs. The experimental results show the neural network model can achieve crossing numbers close to the optimal values of the graphs tested.

Keywords: Outerplanar crossing number, Hopfield model, Energy function, Motion equation, Learning algorithms.

1 Introduction

Graphs that can be drawn without edge crossings (i.e. planar graphs) have a natural advantage for visualization. When visualizing nonplanar graphs, a natural approach is to draw the graph in a way as close to planarity as possible (with as few edge crossings as possible). Especially, when we want to draw a graph to make the information contained in its structure easily accessible, it is highly desirable to have a drawing with as few edge crossings as possible [6].

In the outerplanar [4](other alternate concepts are circular [8] or one-page [7]) drawing, one places vertices of a $n-$vertex $m-$edge connected graph G along a circle, and the edges are drawn as straight lines. The minimal number of crossings over all outerplanar drawings of the graph G is called outerplanar [4](circular [8] or one-page [7]) crossing number of the graph G. The task for the outerplanar drawing problem is simplified to find a good order of vertices to minimise the crossing number. The problem has been proved to be NP-hard [5]. We have the adjacency matrix adj_g of G, in which, $adj\_g[u][v]$ is 1 if there is an edge between vertex u and v; 0, otherwise. If we use an adjacency matrix adj_d to express an outerplanar drawing of G, and if the positions of u and v are i and j respectively, then $adj\_d[i][j] = adj\_g[u][v]$. The outerplanar crossing number ν_1 can be calculated with following formula:

$$v_1(G) = \sum_{i=0}^{n-4} \sum_{j=i+2}^{n-2} \sum_{k=i+1}^{j-1} \sum_{l=j+1}^{n-1} adj\_d[i][j]adj\_d[k][l]. \quad (1)$$

A Hopfield network is a fully connected recurrent single layer and unsupervised network. Hopfield and Tank [3] were the first to use a neural network model for solving optimisation problems. Takefuji and Lee [11] used the binary neural network model for the graph planarisation problem, and Takefuji [9, 10] also proved that the state of the binary model always converges to the local minimum. It is easy to parallelise a Hopfield network because of its special structure, and it is also easy to implement in hardware. Hopfield and Tank successfully solved the traveling salesman problem (TSP) with their neural network model, but they did not get good results. We investigate modeling the outerplanar drawing problem with the Hopfield neural network. A set of stochastic binary units (i.e. $n \times n$ neuron array) are used to represent possible solutions on the permutation of vertices of a graph G with n vertices and m edges for the outerplanar drawing problem. We use two kinds of energy functions for the Hopfield model. Neural network training

*This research was supported by the EPSRC grant GR/S76694/01, UK. The paper is submitted to IWAIA 2006 on 10 March 2006. Department of Computer Science, Loughborough University, Loughborough, Leicestershire, LE11 3TU, UK. Email: H.He@lboro.ac.uk.

plays an important role in the implementation of the neural network. We train the network by using the Hill Climbing algorithm with local movement to improve the convergence of the network, compare the convergence of the two kinds of energy functions, and test them on some complete p-partite graphs.

2 Hopfield neural network for the outerplanar drawing problem

2.1 Hopfield model

Hopfield and Tank [3] first successfully used a neural network approach for the TSP. They solved the TSP with $n = 30$, and used neural network with an array of 30×30 neurons. Similarly, we can use a Hopfield model to solve the outerplanar drawing problem (ODP) of a graph G with n vertices and m edges. The model for ODP is described as below:
According to the task of finding a good vertex order minimising the crossing number, each vertex has n possible positions, so we use an $n \times n$ array, in which each element corresponds to a neuron in the neural network. In the array of the neural network model, if a neuron is fired, namely $v[x][i]$=1, this means that the vertex x is at the i-th position in the vertex order of graph G. Table 1 shows the state array of the Hopfield neural network model for a five-vertex graph. Fig. 1 (b) is the outerplanar drawing of the graph in Fig. 1 (a) corresponding to the state array in Table 1, and the vertex order $\pi = \{v_2, v_3, v_4, v_1, v_5\}$.

Table 1: State of the neural network for five-vertex graph

node order	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	0
5	0	0	0	0	1

2.2 Energy function 1

The array of neurons should satisfy the following conditions:

(1) Each vertex appears only once on the order; namely each row in the array has only one '1', other elements in this row are '0'.

(2) Each position of the order is occupied by just one vertex; namely, each column has only one '1', and other elements in each column are '0'.

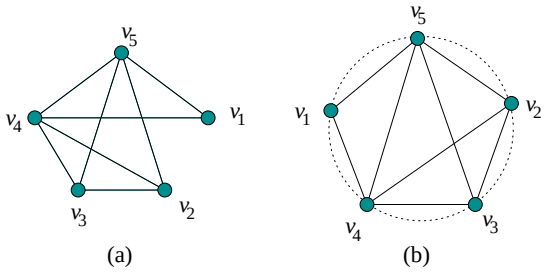


Figure 1: An random graph with five vertices and its outerplanar drawing corresponding to the state of the neural network in Table 1

(3) There are n vertices in total; namely the sum of all elements in the array is n. Therefore, a array of stochastic units v_{ij} to present possible solutions, and there are n^2 units in all.

The following functions correspond to the goal of the outerplanar drawing problem and the constrains above:

(1) The outerplanar crossing number is minimised:

$$Ea = A \sum_{x,i=1}^{n} \sum_{y,j=1}^{n} \sum_{z,k=1}^{n} \sum_{w,l=1}^{n} e(x,y)e(z,w) \quad (2)$$

$$\delta(x,y,z,w)\Delta(i,j,k,l)v_{xi}v_{yj}v_{zk}v_{wl}.$$

where

$$\delta(x,y,z,w) = \begin{cases} 1 & if\ x \neq y \neq z \neq w, \\ 0 & otherwise. \end{cases}$$

$$\Delta(i,j,k,l) = \begin{cases} 1 & if\ i < k < j < l\ or\ i > k > j > l \\ & or\ k < i < l < l\ or\ k > i > l > j \\ 0 & otherwise. \end{cases}$$

(2) One vertex has one position:

$$Eb = B \sum_{x=1}^{n} \sum_{i=1}^{n} \sum_{j=1, j\neq i}^{n} v_{xi}v_{xj}. \quad (3)$$

(3) One position has one vertex:

$$Ec = C \sum_{x=1}^{n} \sum_{y=1, y\neq x}^{n} \sum_{i=1}^{n} v_{xi}v_{yi}. \quad (4)$$

(4) Totally n neurons are fired:

$$Ed = \frac{D}{2}\sum_{x=1}(\sum_{i=1} v_{xi} - n)^2. \quad (5)$$

Therefore the energy function is the sum of the four functions.

$$E_1 = Ea + Eb + Ec + Ed. \quad (6)$$

In the energy function 1, only when $E_b = E_c = E_d = 0$, the state of neurons is a valid solution. Any invalid state, e.g. the number of neurons fired is less or more than n, will make $E_b > 0$, $E_c > 0$,or $E_d > 0$, which will be as an forbidden force acting on the neural network. E_a is always a forbidden force acting on the neural network, unless $E_a = 0$. It is possible that the neural network converge to a local minimum for which the state of the neural network does not correspond a solution. For example, there exists a state when $Eb + Ec + Ed \neq 0$, but $E_a = 0$, so that the sum of energy values $E_a + E_b + E_c + E_d \to 0$, and the neural network arrives at a local minimum.

2.3 Energy function 2

We can also describe the constraints above with two terms as below:

(1) $\sum_i v_{xi} = 1$ (for each vertex x)

(2) $\sum_x v_{xi} = 1$ (for each position i)

Correspondingly, the energy function can be constructed by adding to the crossing number Ea two penalty terms which are minimised when the constrains are satisfied:

$$E_2 = A\sum_{x,i=1}^{n}\sum_{y,j=1}^{n}\sum_{z,k=1}^{n}\sum_{w,l=1}^{n} e(x,y)e(z,w) \quad (7)$$

$$\delta(x,y,z,w)\Delta(i,j,k,l)v_{xi}v_{yj}v_{zk}v_{wl}$$

$$+\frac{B}{2}\sum_{x=1}^{n}(\sum_{i=1}^{n} v_{xi} - 1)^2 + \frac{C}{2}\sum_{i=1}^{n}(\sum_{x=1}^{n} v_{xi} - 1)^2.$$

In the similar way with energy function 1, it is possible that the energy value (E_2) would converge to a local minimum, but the state of the neural network will not map to a solution.

2.4 Motion function

According to Neurodynamics, assuming the dynamic system with $n \times n$ state variables $v_{11}, v_{12}, ..., v_{nn}$, the network motion equation is $du_{xi}/dt = -\partial E/\partial v_{xi}$, where u_{xi} and v_{xi} are the input and output of the xi-th neuron.

For energy function E_1, we have the motion function as follows:

$$\frac{du_{xi}}{dt} = -A\sum_{y,j=1}^{n}\sum_{z,k=1}^{n}\sum_{w,l=1}^{n} e(x,y)e(z,w) \quad (8)$$

$$\delta(x,y,z,w)\Delta(i,j,k,l)v_{yj}v_{zk}v_{wl}$$

$$-B\sum_{j=1,j\neq i}^{n} v_{xj} - C\sum_{y=1,y\neq x}^{n} v_{yi} - D(\sum_{y=1}^{n}\sum_{j=1}^{n} v_{yj} - n) - \xi,$$

where, ξ is the threshold of input.

For the energy function E_2, we can write the motion function as below:

$$\frac{du_{xi}}{dt} = -A\sum_{y,j=1}^{n}\sum_{z,k=1}^{n}\sum_{w,l=1}^{n} e(x,y)e(z,w) \quad (9)$$

$$\delta(x,y,z,w)\Delta(i,j,k,l)v_{yj}v_{zk}v_{wl}$$

$$-B\sum_{x=1}^{n}(\sum_{i=1}^{n} v_{xi} - 1) - C\sum_{i=1}^{n}(\sum_{x=1}^{n} v_{xi} - 1) - \xi.$$

2.5 Learning algorithms

Takefuji [9, 10] proved that the state of the binary model always converges to a local minimum. For a normal Hopfield neural network,

$$u_i = \sum_{j=1,j\neq i} w_{ij}v_j - \theta_i,$$

where w_{ij} is the synaptic weight from neuron j to neuron i, and θ_i is the threshold. The convergence can be formalised as follows:

Theorem 2.1 *[12] 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 $O(p)$ computational cycles, where $p = \frac{1}{2}\Sigma_{i=1}^{m}\Sigma_{j=1}^{m}|w_{ij}| + \Sigma_{i=1}^{m}|\theta_i|$; moreover this stable state represents a local minimum of E.*

We can simply apply a Hill Climbing approach to find out the best solution from "the local minimum" described in Theorem 2.1.

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 initialised with 0 or 1. Obviously, initially the neural network is in an invalid state. According to the values of the motion equation, the state of each neuron is updated sequentially in each iteration. Consequently the vertex order will be formed gradually. Here a traversal of n^2 neurons is viewed an iteration. When the neural network arrives a stable state, the energy of the neural network arrives at a local minimum. Sometimes, the stable state is not a solution, but the stable state must be around a local minimum solution. Therefore we can do a local movement described below. The program repeats the training procedure several times. The learning algorithm is described as Algorithm 1.

In Algorithm 1, the termination criterion is to see if the current state is equal to last state of the neuron array. Moreover, if the number of neurons fired is more than n or less than $n-1$, then the program will run another iteration. This guarantees that the energy has arrived at a small enough value. Therefore, we can use the following rules to do the local movement:

(1) if $v[i][j]$ is a redundant neuron fired in the i-th row of neuron array, then look for another row k of the neuron array, where there is no neuron fired, make $v[i][j] = 0$ and $v[k][j] = 1$;

(2) if $v[i][j]$ is a redundant neuron fired in the j-th column of the neuron array, then look for another column l of the neuron array, where there is no neuron fired, make $v[i][j] = 0$ and $v[i][l] = 1$.

(3) if $v[i][j] = 0$, and there is no neuron fired in both the i-th row and the j-th column, then make $v[i][j]$=1.

3 Experiments

The sequential algorithm of the neural network model was designed in C language for tests, we denote the neural network algorithm as NN. The program runs on the platform of a DELL desktop with Intel Pentium (R)4 CPU3.00GHZ and 1MB RAM. We examine the convergence of the two energy functions of the Hopfield model. We use p-partite graphs as test suites, and compare the results with the optimal values [1].

Algorithm 1 Learning algorithm of the Hopfield neural network

```
1:  t = 0;
2:  Randomly initialise the state of each neuron v_ij
    with 0 or 1;
3:  while t<10 do
4:    k = 0;
5:    repeat
6:      copy the state array of neurons v to last_v;
7:      nn ← number of neurons fired;
8:      for (l from 1 to n^2) do
9:        i = rand() mod n; j = rand() mod n;
10:       Compute Δu_ij at Δt = 0.1 with Equation
          8 or 9;
11:       Compute u_ij(t + 1) with Δu_ij;
12:       if (u_ij > ξ) then
13:         v_ij = 1;
14:       else
15:         v_ij = 0;
16:       end if
17:     end for
18:     k = k + 1;
19:   until v! = last_v or nn > n or nn < n − 1
20:   if (v is invalid) then
21:     local_movement(v);
22:   end if
23:   cr = calculate_crossings(G, v);
24:   if (ν_1 > cr) then
25:     ν_1 = cr;
26:   end if
27:   t=t+1;
28: end while
```

3.1 Test of convergence

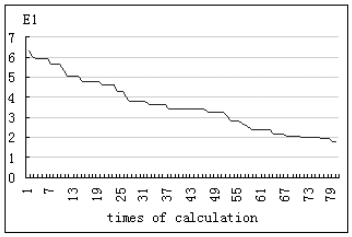
The parameters will affect the convergence of the neural network significantly. After plenty of preliminary experiments, we selected the parameters for the tests shown in Table 2.

Table 2: Experimental parameters

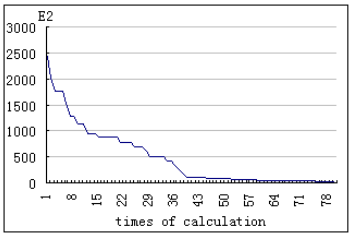
EnergyNo.	A	B	C	D	ξ
1	0.068	0.24	0.24	0.36	0.26
2	0.08	0.24	0.24	N/A	0.38

In order to observe the energy trend, we record the energy value of each calculation in the first iteration. Fig.2 (a) and (b) show the convergence curves for the two energy functions in the first iteration. It can be seen that energy function 1 converges slowly, while energy function 2 converges very quickly. Fig.3 (a) and

(b) show the values of the two energy functions for all iterations. Both energy functions keep a downtrend in the first climbing. However the energy function 1 has large fluctuation after the first climbing, while the energy function 2 keeps almost the same value after the first climbing, and occasionally there is some fluctuation. On the other hand, it indicates that there are more invalid states to be produced by the model with energy function 1 than by that with energy function 2.



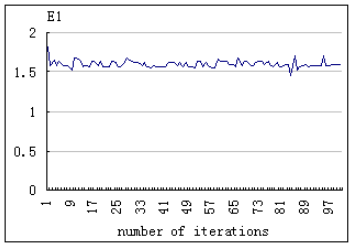
(a) Energy function 1



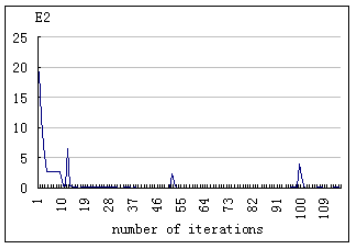
(b) Energy function 2

Figure 2: Energy convergence trend in the first iteration

According to dynamics, an object will move along the direction of the composition of forces that act on the object. Fig. 4 shows three forces (f_a,f_b,and f_c) act on an objects. If the direction of f_a is changed dynamically, and $|f_a| \neq 0$, then the vertical composition and horizontal composition of f_a also are changed dynamically. In order to keep the balance of the vertical composition and horizontal composition acting on the object, f_b and f_c must be changed dynamically. In the motion functions, each term can be viewed as a force that acts on a neuron. For motion function 9 as an example, similarly, there are also three terms. Regardless of the crossing number minimised, the second and the third term should be symmetrical. However, the balance of row and column in the neuron array will be broken, because of the changing crossing number in different states of neural network. It means that it is hard to find out a group of parameters A, B, C to make the energy of the neural network decrease smoothly.



(a) Energy function 1



(b) Energy function 2

Figure 3: Energy convergence trend for all iterations

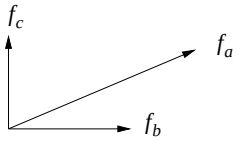


Figure 4: The composition forces act on an object

3.2 Test on complete p-partite graphs

We denote a complete p-partite graph with equal size (n) of the partite sets as:

$$K_n(p) = K_{\underbrace{n,n,...,n}_{p}}.$$

The 1-page crossing numbers for $K_n(p)$ graphs are known:

Theorem 3.1 *[1] For a complete p-partite graph with n vertices in each partite set,*

$$\nu_1(K_n(p)) = n^4\binom{p}{4} + \frac{1}{2}n^2(n-1)(2n-1)\binom{p}{3} + n\binom{n}{3}\binom{p}{2}.$$

Table 3 show the outerplanar crossing number of some complete p-partite graphs. It can be seen that the results of the graphs tested by the model with both energy functions are close to the optimal values, but can still be improved.

Table 3: Outerplanar crossing numbers of $K_n(p)$ tested with the neural network model with different energy functions

Graphs	$\nu_1(E_1)$	$\nu_1(E_2)$	Opt.
$K_3(2)$	3	3	3
$K_4(2)$	16	19	16
$K_5(2)$	54	54	50
$K_3(3)$	54	56	54
$K_4(3)$	224	226	216
$K_5(3)$	628	617	600
$K_3(4)$	290	286	279
$K_4(4)$	1045	1066	1024

For energy function 1 (E_1), the number of iterations ($\mathcal{N}_1$) varies with different tests, while the number of iterations ($\mathcal{N}_2$) for energy function 2 (E_2) keeps nearly same for every test, but usually $\mathcal{N}_1 < \mathcal{N}_2$. The calculation time t_1 in each iteration for E_1 is longer than t_2 for E_2. Therefore, T_{E1} ($= \mathcal{N}_1 \times t_1$) varies, but T_{E2} keeps nearly the same time for different tests. For an example of testing $K_3(3)$, T_{E1} varies from $1s$ to $3s$, while $T_{E2} \approx 1.5s$ for 20 tests. However, the model with E_1 has more potential to improve the results than that with E_2.

4 Conclusions

We successfully solve the outerplanar crossing number problem with Hopfield neural network, and achieve the results close to the optimal values of complete p-partite graphs tested. We examine the convergence of the neural network by using different energy functions. The running time for energy function 1 varies with different tests, while energy function 2 keeps nearly the same running time for different tests. However, there are more invalid states to be produced by the model with energy function 1 than by that with energy function 2, and energy function 1 has more potential to improve results than energy function 2. Our experiments have demonstrated the parameters of a neural network are vital for the convergence of the neural network. We have done a lot of experiments for selecting a group of proper parameters. Our further work is to improve the performance of the network by using different approach to train the network parameters, and to compare the model with other kind of neural networks.

Acknowledgment

We would like to thank Prof. G. Andrejková, for helping in creating the Hopfield neural network model.

References

[1] Fulek, R., He, H., Sýkora, O. and Vrt'o, I. (2005), Outerplanar Crossing Numbers of 3-Row Meshes, Halin Graphs and Complete p-Partite Graphs. In Proc. of the 31st Annual Conference on Current Trends in Theory and Practic of Informatics (SOFSEM'05), LNCS **3381**, Springer, pp. 376-379.

[2] Hirsch, M. W. (1989), Convergent activation dynamics in continuous time networks. In Proc. Nat. Acad. Sci., USA, **2**, pp. 331-349.

[3] Hopfield, J. and Tank, D. (1985), Neural computation of decisions in optimization problems. *Biol. Cybern.* Vol. **52**, pp. 141-152.

[4] Kainen, P.C. (1990), The book thickness of a graph II, *Congr. Numerantium*, **71**, pp 121-132.

[5] Masuda, S., Kashiwabara, T., Nakajima, K. and Fujisawa, T. (1987), On the NP-completeness of a computer network layout problem, in Proc. of IEEE Intl. Symposium on Circuits and Systems, IEEE Computer Society Press, Los Alamitos, pp. 292-295.

[6] Purchase, H. (1997), Which aesthetic has the greatest effect on human understanding? In Proc. of Symposium on Graph Drawing, GD'97, LNCS **1353**, Springer, pp. 248-261.

[7] Shahrokhi,F., Sýkora, O., Székely, L. A. and Vrt'o, I. (1996), The book crossing number of graphs, *Journal Graph Theory*, **21**, pp. 413-424.

[8] Six, J. M. and Tollis, I.G. (1990), Circular drawings of biconnected graphs, In Proc. of ALENEX'99, LNCS **1619**, Springer, pp. 57-73.

[9] Takefuji, Y. (1989), Design of parallel distributed Cauchy machines. In Proc. of the International Joint Conference Neural Networks, pp. 529-536.

[10] Takefuji, Y. (1992), Design of parallel distributed Cauchy machines. *Neural Network Parallel Computing.* Boston: Kluwer.

[11] Takefuji, Y. and Lee, K. C. (1989), A near-optimum parallel planarization algorithm. *Science* Vol. **245**, pp. 1221-1223.

[12] Wiedermann, J. (1990), Complexity issues in discrete neurocomputing. Aspects and Prospects of Theoretical Computer Science, in Proc. of IMYCS'90, LNCS Vol. **464**, Springer Verlag, Berlin, pp. 93-108.