A Parallel Algorithm for Fixed Linear Crossing Number Problem

Rong-Long Wang[1] and Zheng Tang[2]

[1]Faculty of Engineering, Fukui University, Fukui-shi, Japan 910-8507
[2]Faculty of Engineering, Toyama University, Toyama-shi, Japan 930-8555

Summary
This paper proposes a gradient ascent learning algorithm of the Hopfield neural networks for solving fixed linear crossing number problem. The fixed linear crossing number problem is an important problem in printed circuit board layout, VLSI circuit routing, and automated graph drawing. The objective of this problem which is shown to be NP-hard is to embed the edges so that the total number of crossings is minimized. The proposed algorithm uses the Hopfield neural network to get a near-minimal edge crossings, and increases the energy by modifying weights in a gradient ascent direction to help the network escape from the state of the near-minimal edge crossings to the state of the minimal edge crossings or better one. The proposed algorithm is tested on complete graph. We compare the proposed learning algorithm with some other existing algorithms. The experimental results indicate that the proposed algorithm could yield optimal or near-optimal solutions and outperforms the other algorithms.

Key words:
Fixed linear crossing number problem, Graph layout, NP-complete problem, Hopfield neural network, Gradient ascent learning

Introduction

Fixed linear crossing number problem [1] is a representative subproblem of graph layout problem. In this problem, the vertex order is predetermined and fixed along the note line and each edge is drawn as an arc in one of the two pages. The objective of this problem is to embed the edges so that the total number of crossings is minimized. This problem has important applications in printed circuit board layout, VLSL circuit routing, and automated graph drawing. In the layout of printed circuit boards, for the case of non-insulated wires, overlapping wires between electrical components may cause short circuits and thus are to be avoided as much as possible [2]. Similar considerations also hold for the design of VLSI circuits [3]. Also, automatic graph drawing systems make use of crossing reduction techniques to display graphs which are aesthetically pleasing and more comprehensible [4].

The fixed linear crossing number problem is NP-hard [1]; hence researches have focused on finding efficient heuristics or on methods for special families of graphs. Several heuristic have been presented. In 1984, Bhatt and Leighton [5] proposed a bisection heuristic using a straightforward divide-and-conquer approach. In 1996 Shahrokhi et al. [6] proposed a one-page heuristic. For solving such problems, the Hopfield neural networks [7]-[10] constitute an important avenue. Using the neural network techniques, Cimikowski et al. [11] proposed a parallel algorithm for crossing number problem. Unfortunately, due to its inherent local minimum problem and sensitivity to parameter values, the rate to get the minimal crossings number using Hopfield network is very low, and performance of the algorithm becomes poorer with large problem. This may be improved by some more sophisticated architecture, such as simulated annealing (SA) [12]. It could be described as a randomized scheme, which reduces the risk of getting trapped in local minima by allowing moves to inferior solution. Simulated annealing is a powerful method for solving local minima, but it always requires more iterations than exhaustive search to find a good solution [13].

In this paper we present a parallel algorithm for fixed linear crossing number problem based on gradient ascent learning of the Hopfield networks. The learning algorithm has two phases, the Hopfield network updating phase and the gradient ascent learning phase. The first phase uses the Hopfield network to decrease the energy in state domain and find a near-minimal crossings number. The second phase intentionally increases the energy of the Hopfield network by modifying parameters in weight domain in a gradient ascent direction, thus making the network escape from the near-minimal crossings number (i.e., a local minimum) which the network once falls into in the phase one. The proposed algorithm is tested on complete graph and compared with some other existing algorithms. The experimental results indicate that the proposed algorithm could yield optimal or near-optimal solutions and outperform the other algorithms.

2. Hopfield Neural Network Updating Phase

We use standard graph-theoretic terminology such as that given in [14]. In fixed linear crossing number problem, the crossing number, $\upsilon(G)$, of a graph $G(V, E)$ is the minimum number of edge crossings required in any drawing of G in the plane. This problem is similar to graph planarization problem [15]. Figure 1 show the fixed linear

Manuscript received November 5, 2006.
Manuscript revised November 25 , 2006.

embedding of complete graph K_6. In 2-page drawing of this problem, edge is embedded in upper page or lower page. Then any pair of edges *ij* and *kl* cross in a drawing iff $i<k<j<l$ and both lie in the same page as shown in Fig. 2. In [11] Cimikowski et al. used 2*m* neurons to establish a parallel algorithm for this problem, where *m* is the number of edges of graph *G*(*V*, *E*). In this paper, we show a new Hopfield neural network representation for this problem. We use neuron y_{ij} express the edge between the i^{th} and j^{th} vertices. The state $y_{ij}=1$ indicates that the edge between the i^{th} and j^{th} vertices is embedded in upper page and state $y_{ij}=0$ indicates that the edge is embedded in lower page. The number of edges in a given graph determines the number of neurons required. Thus in our Hopfield neural network representation, only *m* neurons are used, where *m* is the number of graph *G*(*V*, *E*). Furthermore, the no partition and the double partition violation are avoided. Thus the fixed linear crossing number can be mathematically stated as finding the minimum of the following objective function:

$$\frac{1}{2}\sum_{ij}\sum_{kl}\left(g_{ij}g_{kl}d_{ij,kl}y_{ij}y_{kl}+g_{ij}g_{kl}d_{ij,kl}(1-y_{ij})(1-y_{kl})\right) \quad (1)$$

where $d_{ij,kl}$ is crossing condition and can be described as:

$$d_{ij,kl}=\begin{cases}1 & if \quad i<k<j<l \quad or \quad k<i<l<j \\ 0 & otherwise\end{cases} \quad (2)$$

g_{ij} represent whether edge between vertex #*i* and vertex #*j* exist or not.

$$g_{ij}=\begin{cases}1 & if \quad \#ij \quad edge \quad exist \\ 0 & otherwise\end{cases} \quad (3)$$

Then the energy function for the fixed linear crossing number problem is given by:

$$e=\frac{A}{2}\sum_{ij}\sum_{kl}\left(g_{ij}g_{kl}d_{ij,kl}y_{ij}y_{kl}+g_{ij}g_{kl}d_{ij,kl}(1-y_{ij})(1-y_{kl})\right) \quad (4)$$

where *A* is a coefficient.

Note that the standard energy function of the Hopfield network can be written as follow:

$$e=-\frac{1}{2}\sum_{ij}\sum_{kl}w_{ij,kl}y_{ij}y_{kl}-\sum_{ij}h_{ij}y_{ij} \quad (5)$$

where w_{ijkl} is weight of a synaptic connection from the *kl*# neuron to the *ij*# one, h_{ij} is external input of neuron #*ij* and is also called threshold.

For the fixed linear crossing number problem, the resulting weight and threshold can now be obtained by equating the energy specified by Eq.(5) with the energy as in Eq.(4). The weight of the Hopfield network is

$$w_{ij,kl}=-2Ag_{ij}g_{kl}d_{ij,kl} \quad (6)$$

Using Eq.(2) and Eq.(3) we can see that $w_{ij,ij}=0$ and $w_{ij,kl}=w_{kl,ij}$. The thresholds become:

$$h_{ij}=g_{ij}g_{kl}d_{ij,kl} \quad (7)$$

The equations of motion are:

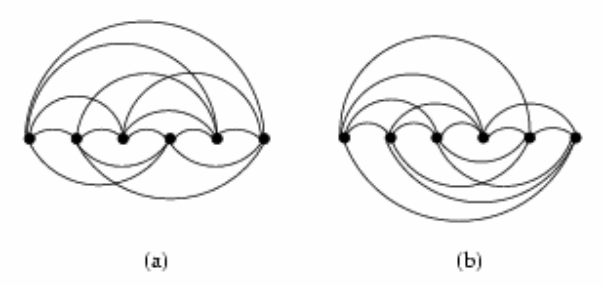

Fig. 1 Fixed linear embedding of the complete graph K_6 with: (a) 4 crossing; (b) crossing number $\nu(K_6)=3$.

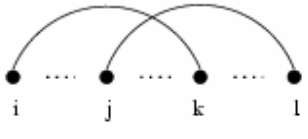

Fig. 2 Edge crossing condition $i<k<j<l$.

$$dx_{ij}/dt=\sum_{kl}w_{ij,kl}y_{kl}+h_{kl} \quad (8)$$

The sigmoid function is used as input/output function.

$$y_{ij}=1/\left(1+e^{\left(-x_{ij}/T\right)}\right) \quad (9)$$

where *T* is a parameter called temperature parameter. It is proved by Hopfield [8] that such network could be used as an approximate method for solving 0-1 optimization problems because provided the weights are symmetric ($w_{ij,kl}=w_{kl,ij}$), the network converges to a minimum of energy function. Furthermore, if there are no self-connections ($w_{ij,ij}=0$), in the high-gain limit of the activation function these minima will be at or near a vertex of $\{0, 1\}^n$.

The Hopfield network updating procedure can be viewed as seeking a minimum in a mountainous terrain. Thus, in the first phase we can find the solution to the fixed linear crossing number problem simply by observing the stable state that the Hopfield network reaches. Because in our Hopfield network representation, the constraint condition is avoided, the solution found in this phase is a feasible solution. Unfortunately, the quality of solution is not very good; since the Hopfield network will attempt to take the best path to the nearest minimum, whether global or local. If a local minimum is reached, the network will fail to update. It is usually difficult for the Hopfield network to find the minimal crossing number which corresponds to the state of global minimum. We now propose a learning method in order to help the network get out from a local minimum to the global minimum or a better local minimum.

3. The Gradient Ascent Learning Phase

In the section 2, we have described the neural network representation for fixed linear crossing number problem. Using the Hopfield network updating we can find a local minimum or a global minimum of the energy function. However, it is usually difficulty for the network to find the global minimum because of the inherent local minimum problem of the network. In this section we propose a learning algorithm which can help the network get out from a local minimum to the global minimum. In order to explain the learning method, we use a two-dimensional graph (Fig. 3) of energy function with a local minimum and a global minimum. The energy function value is reflected in the height of the graph. Each position on the energy terrain corresponds to a possible state of the network. For example, if the network is initialized onto the mountainous terrain A, the updating procedure of the Hopfield network makes the state of network move towards a minimum position and reach a steady state B (Fig. 3(a)).

Because the weights and the thresholds of the Hopfield network determine the energy terrain, we can change the weights and the thresholds to increase the energy at the point B so as to fill up the local minimum valley and finally drive the point B out of the valley. Here, suppose that a vector $\vec{v}$ corresponds to the weights and the thresholds of the Hopfield network. Since for a parameter vector $\vec{v}$, the learning requires the parameter change to be in the positive gradient direction, we take:

$$\Delta\vec{v} = \varepsilon\nabla e(\vec{v}) \tag{10}$$

where ε is a positive constant and ∇e is the gradient of energy function e with respect to the parameter vector $\vec{v}$ in the state B. Applying this learning rule (Eq.(10)) to the fixed linear crossing number problem, we then obtain:

$$\Delta w_{ij,kl} = p\frac{\partial e}{\partial w_{ij,kl}} \tag{11}$$

$$\Delta h_{ij} = q\frac{\partial e}{\partial h_{ij}} \tag{12}$$

and

$$\frac{\partial e}{\partial w_{ij,kl}} = -y_{ij}y_{kl} \tag{13}$$

$$\frac{\partial e}{\partial h_{ij}} = -y_{ij} \tag{14}$$

where, p and q are small positive constants and y_{ij}, y_{kl} correspond to the state of B.

Now we show that after we change the weights and the thresholds according to Eq.(11)-Eq.(14), point B will be on the slope of the valley. Suppose y_{bij} represent the state of point B, y_{Pij} represent the state of any point P of energy terrain, then the change of energy in point P by the learning rule (Eq.(11) - Eq.(14)) will be:

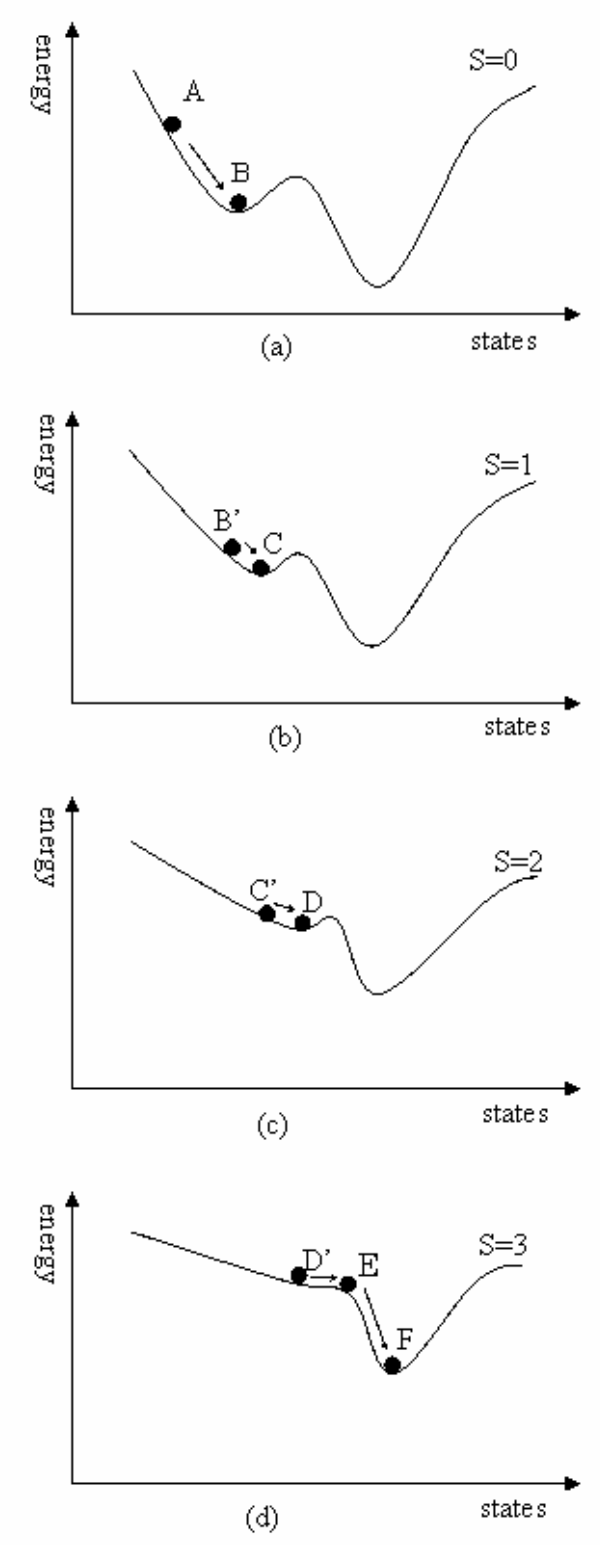

Fig. 3 The conceptual graph of the relation between energy and state transition in the learning process of the Hopfield network.

$$\begin{aligned}\Delta e_p &= \frac{1}{2}\sum_{ij}\sum_{kl} w_{ij,kl} y_{Pij} y_{Pkl} + \sum_{ij} h_{ij} y_{Pij} \\ &\quad - \frac{1}{2}\sum_{ij,kl} (w_{ij,kl} - p y_{Bij} y_{Bkl}) y_{Pij} y_{Pkl} - \sum_{ij} (h_{ij} - q y_{Bij}) y_{Pij} \\ &= \frac{p}{2}\sum_{ij,kl} (y_{Bij} y_{Bkl}) \cdot (y_{Pij} y_{Pkl}) + \sum_{ij} (y_{Bij} y_{Pij})\end{aligned} \tag{15}$$

Because point B is a minimum of energy function and the output of neuron in point B is at or near 0 or 1 [8], from Eq.(15), we can know easily that the change of energy is largest when point P is at the same point as point B, and the larger the difference of state between point P and point B is, the smaller energy changes in point P. Thus, we can see that the valley will be filled up in a most effective way. In general, point B may become a point on the slope of the valley.

Thus, the learning (the second phase) makes the previous stable state *B* becomes a point on the slope of a valley (*B'*). After updating of the Hopfield network with the new weights and the new thresholds in the Hopfield network updating phase again, point *B'* goes down the slope of the valley and reaches a new stable state *C* (Fig. 3(b)). Thus, the Hopfield network updating (Phase 1) and the gradient ascent learning (Phase 2) in turn may result in a movement out of a local minimum, and lead the network converge to a global minimum or a new local minimum (Fig. 3(c) and (d)).

4. Algorithm

The following procedure describes the proposed algorithm. Note that there are two kinds of conditions for end of the learning. One has a very clear condition, for example, the *N*-queen problem in which the energy is zero if the solution is the optimal. Another one has not a clear condition, for example, the travelling salesman problem and the fixed linear crossing number problem in which the energy is not zero even the solution is the optimal. For the latter case, we have to set a maximum number of the learning (*learn_limit*) in advance. Learning stops if the maximum number of learning is performed. In general, we can determine the value of *learn_limit* according to the allowable computation time and the complexity of the problem. For the fixed linear crossing number problem, we found that the network can always find good solutions within 5 learning times; therefore, we selected 10 as the maximum number of learning time in our simulations. If the *learn_limit* is supposed to be the maximum number of learning times for the system termination condition, we have,

1. Set *learn_time*=0 and set *A* and *learn_limit.*
2. Randomize the initial values of each neuron y_{ij} in the range of 0.0 to 1.0.
3. The updating procedure is performed on the Hopfield network with original weights and thresholds until the network reaches a steady state (Phase 1).
4. Use Eq. (11)-Eq.(14) to computer the new weights and the new thresholds (Phase 2).
5. The updating procedure is taken on the Hopfield network with the new weights and thresholds until the network reaches a steady state.
6. In order to avoid the shift of the state of the global minimum to a specific problem, the updating procedure on the Hopfield network may be re-performed with original weights and thresholds until the network reaches a new steady state.
7. If the new steady state is better than the old one then the old state is replaced by the new state using the steady state obtained from step 6.

Increment the *learn_time* by 1. If *learn_time=learn_limit* then terminate this procedure, otherwise using the new steady state obtained from step 5, go to the step 4

Table 1: Simulation results on complete graphs

graph	Number of Crossing found by different algorithm				
	Opt	1-page	Bisect	Neural	Learning
K5	1	1	1	1	1
K6	3	3	4	3	3
K7	9	9	11	9	9
K8	18	19	19	18	18
K9	36	36	42	36	36
K10	60	62	75	60	60
K11	100	100	128	100	100
K12	150	154	174	150	150
K13	Unknown	265	277	255	255

5. Simulation and Results

The proposed learning algorithm was experimented on PC Station (Pentium4, 1500MHz) to several complete graphs. Simulations refer to parameter set at A=1.0. In the experiments *learn_limit* was set to 10. The initial values of neurons were randomized in the range of 0.0 to 1.0. To evaluate our results, bisection heuristic [5], one-page heuristic [6] and Cimikowski et al's parallel algorithm [11] were also executed for comparison.

The simulation results were shown in Table 1. From this table, we can know that both Cimikowski et al's algorithm and the proposed learning method found optimal solution or near-optimal solution. But in our simulations we found that the rate to find good solution of Cimikowski et al's algorithm is very low. On the other hand, the proposed learning algorithm can find one hundred percent good solution.

Table 2: Simulation results for complete graph in 100 runs with different initial value

Graph	Crossing Number found by Cimikowski et al's Algorithm				Crossing Number found by the proposed Learning Algorithm					
	Worst	Avg	Best	Rate	Worst	Avg	Best	Rate	Average Learning Times	CPU Time(s)
K5	1	1	1	100%	1	1	1	100%	0	0.01
K6	4	3.58	3	42%	3	3	3	100%	0	0.01
K7	11	10.01	9	37%	9	9	9	100%	0	0.01
K8	24	19.77	18	19%	18	18	18	100%	0.65	0.01
K9	46	38.91	36	44%	36	36	36	100%	0.45	0.01
K10	72	63.6	60	12%	60	60	60	100%	1.11	0.01
K11	118	108.48	100	29%	100	100	100	100%	1.02	0.01
K12	184	165.76	150	13%	150	150	150	100%	1.05	0.01
K13	267	239.23	225	18%	225	225	225	100%	1.36	0.02

Table 2 shows the detail simulation results found by Cimikowski et al's algorithm and the proposed learning algorithm in 100 runs with different initial value of neuron, where the worst solution, the average solution, the best solution and the rates of best solution were summarized. We also showed the average learning times and the CPU time of proposed algorithm to find the minimal crossing number. From this table, we can say that the proposed learning algorithm works better than Cimikowski et al's algorithm. The reason why the proposed algorithm works better than Cimikowski et al's algorithm are: (1) In [11] Cimikowski et al. used 2*m* neurons to establish their parallel algorithm, where *m* is number of edges of graph *G*(*V*, *E*). But in the proposed algorithm, we used only *m* neurons actually. Furthermore, in the proposed algorithm the no partition and the double partition violation were avoided. (2) In the proposed algorithm, we used the gradient ascent learning on the Hopfield network to help the network get out from the local minima. But in Cimikowski et al's algorithm, if a local minimum is reached, their network will fail to update.

6. Conclusions

We have proposed a Hopfield network learning algorithm for fixed linear crossing number problem and showed its effectiveness by simulation experiments. The learning algorithm which is designed to minimize the crossing number, has two phases, the Hopfield network updating phase and the gradient ascent learning phase. In the first phase we implemented the Hopfield network for optimizing the energy function in state space. In the second phase we intentionally increased the energy of the Hopfield network by modifying parameters in weight domain in a gradient ascent direction, thus making the network get out from the near-minimal edge crossings (i.e., a local minimum). The proposed algorithm was tested on complete graph and compared with some other best existing algorithms. The experimental results indicated that the proposed algorithm could yield optimal or near-optimal solutions and outperform the other algorithms.

References

[1] S. Masuda, K. Nakajima, T. Kashiwabara, T. Fujisawa, "Crossing minimization in linear embeddings of graphs," IEEE Trans. Comput. Vol.39, no.1, pp.124-127, 1990.
[2] F. W. Sinden, "Topology of thin film circuits," Bell Sys. Tech. J. XLV, pp.1639-1666, 1966.
[3] F. T. Leighton, "New lower bound techniques for VLSI," Math. Sys. Theory, vol.17, pp.47-70, 1984.
[4] R. Tamassia, G. Di Battista and C. Batini, "Automatic graph drawing and readability of diagrams," IEEE Trans. Sys., Man, and Cyber., vol.18, pp.61-79, 1988.
[5] S. N. Bhatt and F. T. Leighton, "A framework for solving VLSI graph layout problem," J. Comput. & Sys. Sci. vol.28, pp.300-343, 1984.
[6] F. Shahrokhi, L. A. Szekely, O. Sykora and I. Vrto, "The book crossing number of a graph," J. Graph Theory, vol.21, no.4, pp.413-424, 1996.
[7] J.J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities." Proc. Nat. Acad. Sci. U.S., vol.79, pp.2554-2558, 1982.
[8] J.J. Hopfield, "Neurons with graded response have collective computation properties like those of two-state neurons." Proc. Nat. Acad. Sci. U.S., vol.81, pp.3088-3092, 1982.
[9] J.J. Hopfield and D.W.Tank, " 'Neural' computation of decisions in optimization problems," Bio. Cybern., no.52, pp.141-152, 1985.
[10] J.J. Hopfield and D.W.Tank, "Computing with neural circuits: A model." Science, vol.233, pp.625-633, Aug.8, 1986.

[11] R. Cimikowski and P. Shope, "A neural network algorithm for a graph layout problem," IEEE Trans on Neural Network, vol.7, no.2, pp.341-345, 1996.
[12] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimisation by Simulated Annealing," Science, Vol.220, pp.671-680, 1983.
[13] P. J. M. van Laarhoven and E. H. L. Aarts, Simulated Annealing: Theory and Applications. Kluwer, Dordrecht, 1988.
[14] F. Harary, Graph Theory, Reading, MA: Addison-wesley, 1969.
[15] R.Jayakumar, K.Thulasiraman, and M.N.S.Swamy, "$O(n^2)$ algorithms for graph planarization," IEEE Trans. Computer Aided Design, Vol.8, No.3, pp.257-267, (1989).