Biol. Cybern. 67, 243–251 (1992)

Biological Cybernetics

An artificial maximum neural network: a winner-take-all neuron model forcing the state of the system in a solution domain

Yoshiyasu Takefuji [1], Kuo-Chun Lee [2], and Hideo Aiso [3]

[1] Department of Electrical Engineering and Applied Physics, Case Western Reserve University, Cleveland, OH 44106, USA
[2] Research & Development Department, Cirrus Logic, Freemont, CA 94538, USA
[3] Faculty of Environmental Information, Keio University, Fujisawa, 252 Japan

Received September 19, 1991/Accepted in revised form January 29, 1992

Abstract. A maximum neuron model is proposed in order to force the state of the system to converge to the solution in neural dynamics. The state of the system is always forced in a solution domain. The artificial maximum neural network is used for the module orientation problem and the bipartite subgraph problem. The usefulness of the maximum neural network is empirically demonstrated by simulating randomly generated massive instances (examples) in both problems. In randomly generated more than one thousand instances our system always converges to the solution within one hundred iteration steps regardless of the problem size. Our simulation results show the effectiveness of our algorithms and support our claim that one class of NP-complete problems may be solvable in a polynomial time.

1 Introduction

Since Cook presented the foundations for the theory of NP-completeness in 1971 (Cook 1971), a large number of commonly encountered problems from mathematics, computer science, molecular biology, management science, seismology, communications, and operations research have been known to be NP-complete. Garey and Johnson have collected more than 300 NP-complete problems (Garey and Johnson 1979). It is widely believed that no polynomial time algorithm exists in finding the optimum solution of any NP-complete problems (Garey and Johnson 1979). Although there is no known polynomial time algorithm for solving the NP-complete problem, a polynomial time algorithm may exist on a particular instance of the problem. An algorithm for NP-complete problems must be carefully tested based on time complexity or must be tested by massive instances (examples) of the problem using empirical simulation runs. To practically cope with NP-complete problems, we have been forced to be satisfied with finding a near-optimum or a good solution within an acceptable time instead of finding an optimum solution.

The NP-completeness of the module orientation problem was proven (Ran Libeskind-Hadas and Liu 1989). Parallel algorithms for the module orientation problem and the bipartite subgraph problem are introduced and demonstrated in this paper. The empirical demonstration of the global minimum convergence of our algorithm is given by simulating massive instances in both problems as far as we could test the optimality using exhaustive search. In randomly generated more than one thousand examples in the same manner as Liu used (Ran Libeskind-Hadas and Liu 1989), the system for the module orientation problem and the bipartite subgraph problem always converges within one hundred iteration steps regardless of the problem size. Our empirical simulation result suggests that a polynomial algorithm may exist for solving NP-complete problems. The simulation result also shows the effectiveness of our approach and support our claim that one class of NP-complete problems may be solvable in a polynomial time.

2 Module orientation problems

The proposed parallel algorithms are based on the artificial neural network mathematical model where a large number of simple processing elements are used and interconnected with one another. The processing element and the interconnection between processing elements represent the simplified biological neuron and the synaptic interconnection respectively. In 1943 McCulloch and Pitts proposed a mathematical model based on the biological computation (McCulloch and Pitts 1943). The input/output function of the McCulloch-Pitts processing element is given by $V_i = f(U_1) = 1$ if $U_i \geqslant 0$ and 0 otherwise where V_i and U_i are the output and the input of the ith neuron respectively. The artificial neural network for combinatorial optimization problems was

Correspondence to: Y. Takefuji

first introduced by Hopfield and Tank (Hopfield and Tank 1985). They used the differentiable, continuous, and nondecreasing neuron model, called sigmoid function, where the input/output function of the ith sigmoid processing element is given by $V_i = g(U_i) = \frac{1}{2}(1 + \tanh(U_i/U_0))$. Note that U_0 is a constant.

The artificial neural network architecture and the interconnections between neurons are determined by the fabricated computational energy function E where it is given by considering the necessary and sufficient constraints of the problem. The goal of the proposed algorithms are not only to minimize the fabricated energy function but also to empirically find the optimum solution. Since the energy function of the NP-complete problem has many local minima so that we have been forced to be satisfied with finding a near-optimum or a good solution within an acceptance time instead of the global optimum solution. Simulated annealing attempts to give us a hope to alleviate the local minima problem (Kirkpatrick et al. 1983). However it needs careful temperature scheduling and it takes prohibitively long time or infinite time to obtain the global optimum solution (Kirkpatrick et al. 1983; Geman and Geman 1984). Liu has already stated that his neural algorithm is better than the best known algorithms including simulated annealing, backtracking, dynamic programming, and others (Libeskind-Hadas and Liu 1989).

In this paper parallel algorithms for solving one class of NP-complete problems are proposed and demonstrated. Our empirical simulation result using massive instances of the problem shows that the maximum neural network model is very promising for optimization problems. The instances are randomly generated in the same manner as Liu used (Libeskind-Hadas and Liu 1989). The solution quality of our algorithm is always much better than that of Liu's algorithm. Our approach seems to eliminate a large part of the local minima in the energy function to obtain the optimum solution.

For physical design of very large scale integrated circuits, it is important to properly place the given modules under certain constraints such as the minimum area and/or the minimum total wire length. The module orientation problem was first proposed by Libeskind-Hadas and Liu (Libeskind-Hadas and Liu 1989). The first assumption of the problem is that all the modules have already been placed according to some placement algorithm (Hanan et al. 1976) (Preas and Karger 1988). The second assumption is that the pin positions on each module are fixed. The goal of this problem is to minimize the total wire length by flipping each module with respect to its vertical and/or horizontal axes of symmetry as shown in Fig. 1. This means that there are only four possible orientations for each module. Libeskind-Hadas and Liu called the problem as module orientation problem: finding the optimal flips for a given set of modules. Another similar problem was discussed at the same time by Libeskind-Hadas and Liu: module rotation problem. Libeskind-Hadas and Liu have proved the module orientation problem and module rotation problem are both NP-complete (Libeskind-Hadas and Liu 1989).

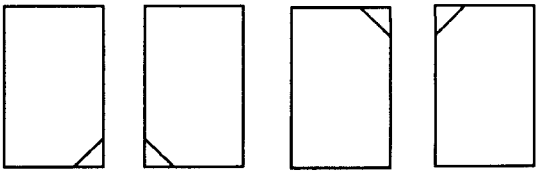

Fig. 1. Four orientations for one module

For simplicity, the Euclidean metric is used to define the distance between two pins in the module orientation problem. Let p and q be two pins which belong to the same net N. Let (x_p, y_p) and (x_q, y_q) represent the positions of p and q according to a certain orientation of the modules. The goal is to minimize the total wire length where the summation is carried out over all pairs of pins which belong to the same net. That is to minimize L where L is given by

$$L = \sum_{p,q,\in N} \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$$

$$= \frac{1}{2} \sum_{m}^{M} \sum_{m' \neq m}^{M} \sum_{i}^{4} \sum_{j}^{4} d_{mi,m'j} V_{m,i} V_{m'j} \tag{1}$$

where $d_{mi,m'j}$ denotes the total length of all wires between the mth module in the ith orientation and the m'th module in the jth orientation. Note that $V_{m,i} = 1$ if the mth module is in the ith orientation, 0 otherwise.

The problem is to determine the optimum orientation for each module in order to minimize L. Libeskind-Hadas and Liu mapped the problem into the Hopfield neural network by using the $4 \times M$ neural array where M is the number of modules and each module has 4 possible orientations. The mapping procedure is similar to that of the travelling salesman problem proposed by Hopfield and Tank (Hopfield and Tank 1985) where an $N \times N$ neural array is prepared for an N-city problem. The computational energy function E for the N-city problem is given by:

$$E = \frac{A}{2} \sum_{x}^{N} \sum_{i}^{N} \sum_{j \neq i}^{N} V_{xi} V_{xj} + \frac{B}{2} \sum_{i}^{N} \sum_{x}^{N} \sum_{y \neq x}^{N} V_{xi} V_{yi}$$

$$+ \frac{C}{2} \left(\left(\sum_{i}^{N} \sum_{x}^{N} V_{xi} - N \right) \right)^2$$

$$+ \frac{D}{2} \sum_{x}^{N} \sum_{y \neq x}^{N} \sum_{i}^{N} d_{xy} V_{xi} (V_{y,i+1} + V_{y,i-1}) \tag{2}$$

Note that V_{xi} represents the output state of the xith neuron and d_{xy} for a distance between the xth and the yth city.

For an M-module orientation problem, Libeskind-Hadas and Liu give the following energy form:

$$E = \frac{A}{2} \sum_{m}^{M} \sum_{i}^{4} \sum_{j \neq i}^{4} V_{mi} V_{mj} + \frac{B}{2} \left(\left(\sum_{m}^{M} \sum_{i}^{4} V_{mi} - N \right) \right)^2$$

$$+ \frac{C}{2} \sum_{m}^{M} \sum_{m' \neq m}^{M} \sum_{i}^{4} \sum_{j}^{4} d_{mi,m'j} V_{m,i} V_{m'j} \tag{3}$$

where $d_{mi,m'j}$ is the total length of all wires between the mth module in the ith orientation and the m'th module in the jth orientation. Note that V_{mi} shows the output state of the mith neuron and represents possibility of the mth module in the ith orientation. $V_{mi}=1$ means that the mth module is in the ith orientation. $V_{mi}=0$ means that the mth module is not in the ith orientation. The first two terms in (3) globally attempts to force their system to have a valid configuration. However they cannot guarantee it because of using the sigmoid neurons. The last term is to minimize the total wire length.

It is well known that the neural representation and the computational energy function E are not unique. The first two terms in (3) are simply replaced by local constraints such that one and only one orientation is allowed for each module. The energy function is given by:

$$E=\frac{A}{2}\sum_{m}^{M}\left(\sum_{i}^{4}V_{mi}-1\right)^2 + \frac{C}{2}\sum_{m}^{M}\sum_{m'\neq m}^{M}\sum_{i}^{4}\sum_{j}^{4}d_{mi,m'j}V_{m,i}V_{m',j} \tag{4}$$

In order to eliminate the first term in (4), the maximum neuron function is introduced. The maximum neuron model was successfully used for solving tiling problems (Takefuji and Lee 1990). The maximum neural network is composed of M clusters where each cluster consists of n neurons. One and only one neuron among n neurons with the maximum input per cluster is encouraged to fire in the maximum neural network. The input/output function of the ith maximum neuron in the mth cluster is given by: $V_{mi}=1$ if $U_{mi}=\max\{U_{m1},\ldots,V_{mn}\}$ and $U_{mi}\geqslant U_{mj}$ for $i>j$, 0 otherwise. In the maximum neuron model it is always guaranteed to keep one and only one neuron to fire per cluster. The proof of the local minimum convergence of the maximum neural network is given in Lemma 2 of Appendix for the details. The proposed parallel algorithm uses a $4\times M$ maximum neural network array where M is the number of modules (clusters). The output state of the ith neuron in the mth cluster represents one of four possible module orientations. For example, $V_{m1}=V_{m2}=V_{m3}=0$ and $V_{m4}=1$ indicate that the mth module is in the fourth orientation. If the maximum neuron function is used for the module orientation problem, then the first term in (4) will be completely eliminated. Because the condition of firing one and only one neuron per module is always satisfied. Therefore, the computational energy E is finally given by:

$$E=\frac{C}{2}\sum_{m}^{M}\sum_{m'\neq m}^{M}\sum_{i}^{4}\sum_{j}^{r}d_{mi,m'j}V_{m,i}V_{m',j} \tag{5}$$

The motion equation of the mi-th neuron is given from (5) and Lemma 2 by:

$$\frac{dU_{mi}}{dt}=-C\sum_{m'\neq m}^{M}\sum_{j}^{4}d_{mi,m'j}V_{m',j} \tag{6}$$

Note that the energy function E in (5) is exactly equal to L in (1) with $C=1$. It is guaranteed that the maximum neural network always generates a valid configuration for the module orientation problem. It is not required to tune the coefficient parameters in the computational energy function E, while in (2), (3), and (4) we must suffer from tuning the coefficients. Whenever our system converges, the corresponding configuration is always forced to be a valid solution, while none of the existing neural networks can guarantee it.

Equation (5) follows the quadratic form where symmetry of $d_{mi,m'j}$ is always satisfied and the diagonal elements are all zeros. Based on our empirical simulation (5) seems to have no local minima or less local minima than (3), and (4). An open question is that we need the mathematical proof of the global minimum convergence and how fast the system can converge to the global minimum state. We randomly generated more than one thousand instances including up to 300-module problems. We followed the same procedure to generate the instances as Liu used (Libeskind-Hadas and Liu 1989) and compared our algorithm with the Liu's algorithm. The simulator is developed on a Macintosh SE/30 and a DEC3100 machine where it numerically solves the motion equaitons based on (6). Remember that no parameters adjustment is needed in our algorithm since (6) consists of one and only one term, while Liu and others must suffer from tuning the coefficient parameters in (3).

The termination condition is given by the convergence state of the system. As long as the system reaches a stable point or an equilibrium state, the procedure will be terminated. The equilibrium state is defined that all firing neurons have the smallest change rate of the input per cluster. The condition of the equilibrium state is given by:

$$V_{m,i}(t)=2 \quad \text{and} \quad \frac{dU_{m,i}(t)}{dt} = \min\left\{\frac{dU_{m,1}(t)}{dt},\ \frac{dU_{m,2}(t)}{dt},\ \frac{dU_{m,3}(t)}{dt},\ \frac{dU_{m,r}(t)}{dt}\right\} \quad \text{for} \quad m=1,\ldots,M.$$

In the existing Hopfield neural networks the condition of the system convergence has never been clearly defined.

3 Parallel algorithm for module orientation problems

The following procedure describes the developed simulator for module orientation problems:

0. Set $t=0$

1. The same number is assigned to the initial values of $U_{m,i}(t)$ for $m=1,\ldots,M$ and $i=1,\ldots,4$ where M is the number of modules.

2. Evaluate the values of $V_{m,i}$ based on the maximum neuron function for $m=1,\ldots,M$ and $i=1,\ldots,4$.

$V_{m,i}(t)=1$ if $U_{m,i}(t)=\max\{U_{m,1}(t), U_{m,2}(t), U_{m,3}(t), U_{m,4}(t)\}$

and $U_{m,i}(t) \geqslant U_{m,j}(t)$ for $i > j$,

0 otherwise

3. Use the motion equation in (6) to compute $\Delta U_{m,i}(t)$

$$\Delta U_{m,i}(t) = -\sum_{m' \neq m}^{M} \sum_{j=1}^{4} d_{mi,m'j} V_{m',j}(t)$$

4. Compute $U_{m,i}(t)$ based on the first order Euler method:

$$U_{m,i}(t) = U_{m,i}(t) + \Delta U_{m,i}(t) \text{ for } m = 1, \ldots, M \text{ and } i = 1, \ldots, 4.$$

5. Increment t by 1. If the state of the system reaches the equilibrium state then stop this procedure else go to step 2.

Figure 2a,b shows the initial configuration of the 4-module problem and the final global minimum solution respectively. We have examined our algorithm using more than one thousand examples including up to 300-module problems. The 300-module problem belongs to the large size problems based on the current VLSI technology. For 4-module to 14-module problems, we were able to verify the global optimality of our solutions by exhaustive search to substantiate our algorithm. The searching complexity of the 9-module problem is 4^9 where it took 15 h to find the global minimum on a Macintosh SE/30 machine. Exhaustive search for the 14-module problem took more than 9 h on a Dec3100 machine. More than one hundred simulation runs were performed for each instance of a problem where the initial state of the system is randomly generated for every simulation run. Figure 3a–f shows the initial states and the converged states of the system for 4-module through 9-module problems respectively. Figure 4 shows the simulation result of a 100-module problem. Figure 5a,b shows the relation between the energy and the number of iteration steps for a 100-module and a 200-module problem respectively. Figure 6 shows the simulation result of the 200-module problem. Table 1 shows comparisons of the solution quality and the number of iteration steps to converge between our algorithm and Liu's algorithm. Each element except the last column in Table 1 is represented by the average value and standard deviation of 100 simulation runs. The last column represents the best solutions found by

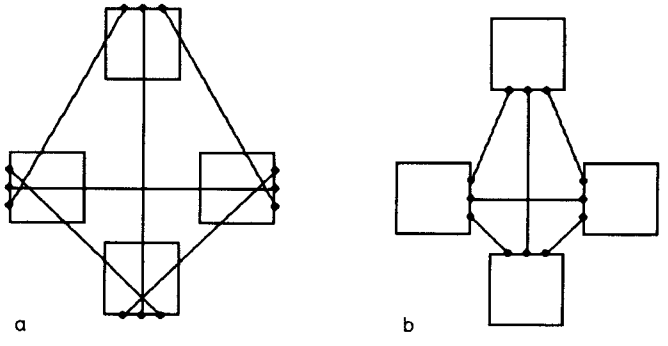

Fig. 2. A 4-module orientation problem. *a* The initial configuration, *b* the final solution

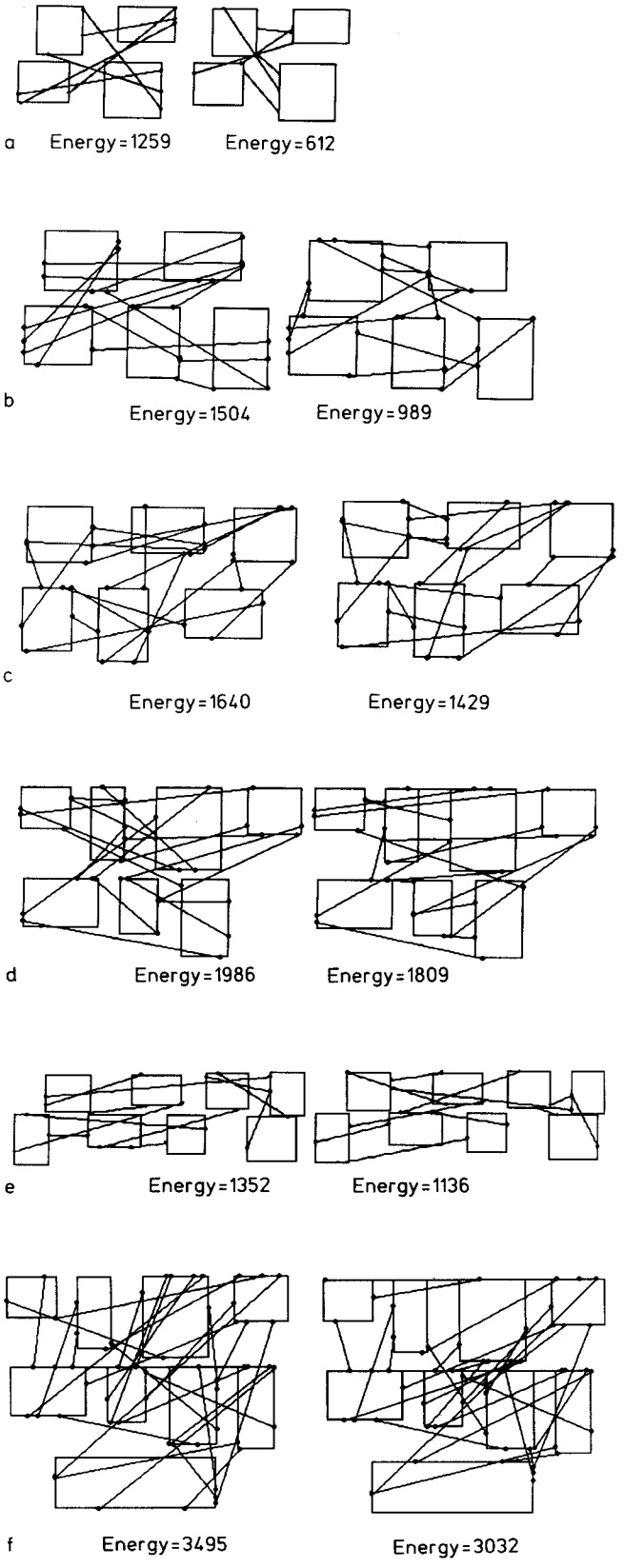

Fig. 3a–f. Module orientation problems with module size 4 through 9 The left picture and the right picture describe the initial configuration and the final converged solution respectively. **a** Module size 4, **b** module size 5, **c** module size 6, **d** module size 7, **e** module size 8, **f** module size 9

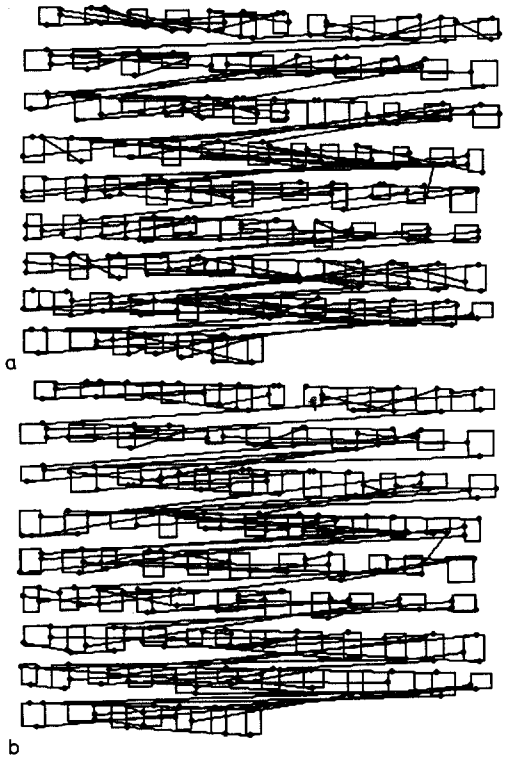

Fig. 4a,b. The initial state and final solution for the module size 100. **a** Initial state, **b** final solution

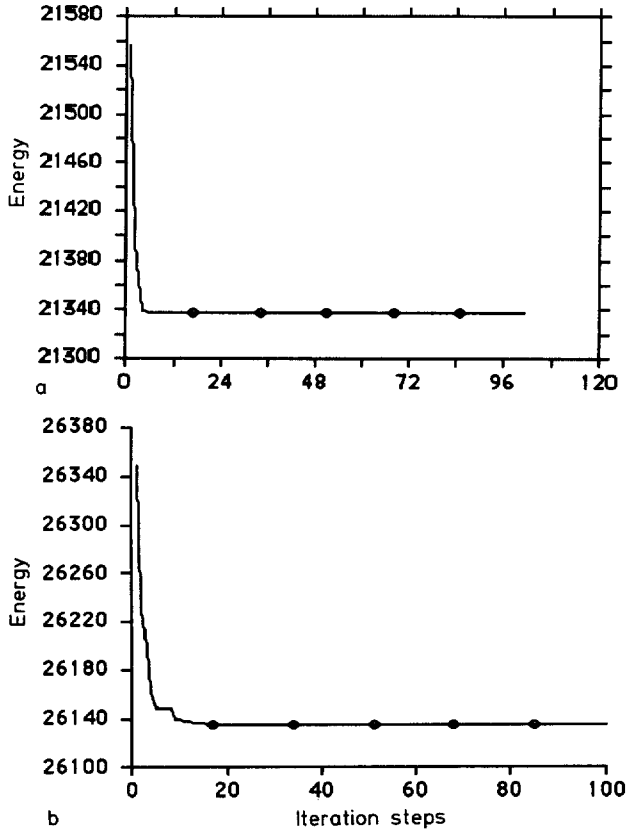

Fig. 5. **a** The relation between the energy and the number of iteration steps for the module size 100. **b** The relation between the energy and the number of iteration steps for the modules size 200

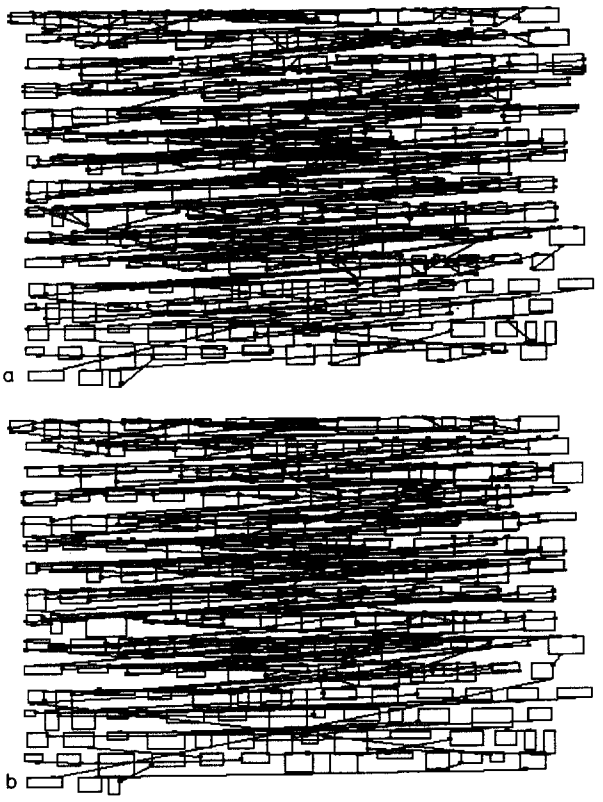

Fig. 6a,b. The simulation of the 200-module problem. **a** Initial configuration, **b** final configuration.

Liu's algorithm (Libeskind-Hadas and Liu 1989). We did not show the number of average iteration steps in the Liu's algorithm because the Liu's algorithm cannot always guarantee the valid solution. It usually takes several thousand iteration steps in the Liu's algorithm (Libeskind-Hadas and Liu 1989). In our massive instances of the simulation runs, the proposed system converges within one hundred iteration steps in all examples. All of small module orientation problems were solved by exhaustive search and our algorithm. The solutions of exhaustive search always match with that of our algorithm. As far as we could test the optimality of the solutions for the problem with up to 14 modules they are found to be optimum. The simulation results suggests that a polynomial algorithm may exist for solving certain NP-complete problems.

4 Bipartite subgraph problems

We have also investigated another parallel algorithm for the bipartite subgraph problem which belongs to NP-complete problems (Garey and Johnson 1979).
Instance: Graph $G = (V, E)$, positive $K \leqslant |E|$.
Question: Is there a subset $E' \subseteq E$ with $|E'| \geqslant K$ such $G' = (V, E')$ is bipartite?

Optimization description: minimize $\Sigma_{x=1}^{N} \Sigma_{x \neq y}^{N} \Sigma_{i=1}^{2}$ Connection(x, y) $V_{x,i} V_{y,i}$ subject $\Sigma_{i=1}^{2} V_{x,i} = 1$ for $x = 1$ to N where N is the number of vertices.

Table 1. Comparisons of the solution quality and the number of iteration steps between the maximumneural network and Liu's algorithm

size	initial length	maximum neural network		Libeskind-Hadas and Liu's method
		solution	iteration steps	best solution
10	2101.73 ± 116.44	1835.41 ± 1.47	21.76 ± 6.70	1862.14
20	3881.83 ± 121.65	3441.36 ± 8.33	20.00 ± 0.00	3615.03
30	8077.71 ± 123.42	7572.56 ± 38.89	26.99 ± 8.71	7787.16
40	11653.56 ± 237.04	10915.48 ± 19.70	20.15 ± 0.36	11098.94
50	14465.39 ± 166.13	13613.96 ± 18.88	20.00 ± 0.00	13997.05
60	17355.84 ± 213.49	16252.41 ± 5.68	28.13 ± 2.03	17025.57
70	20732.71 ± 251.92	19536.33 ± 6.42	35.06 ± 6.23	20317.16
80	23343.07 ± 262.78	21987.59 ± 10.45	20.88 ± 2.32	22863.13
90	28500.73 ± 147.09	27286.13 ± 8.74	20.12 ± 0.55	28220.61
100	29202.87 ± 166.83	27786.89 ± 22.61	21.68 ± 4.85	28872.39
110	32428.37 ± 332.73	30500.13 ± 5.33	21.84 ± 3.28	31820.94
120	36310.35 ± 268.47	34313.00 ± 12.62	26.60 ± 3.68	35710.07
130	38712.22 ± 211.43	36739.19 ± 5.10	22.92 ± 9.08	38458.68
140	39833.82 ± 169.51	38017.12 ± 12.04	27.40 ± 6.43	39490.21
150	41936.09 ± 247.46	39797.70 ± 10.68	45.31 ± 8.53	41432.39
160	47461.52 ± 322.05	45004.91 ± 9.51	21.33 ± 5.63	46753.53
170	48908.64 ± 339.95	46306.76 ± 19.05	42.49 ± 7.71	48583.02
180	53067.66 ± 404.72	50215.30 ± 20.92	20.24 ± 0.51	52262.82
190	57300.09 ± 654.85	53939.43 ± 18.81	21.14 ± 2.06	56009.94
200	61660.71 ± 624.13	58011.41 ± 15.50	20.66 ± 1.63	60533.42
210	63037.32 ± 550.81	59674.39 ± 18.90	31.38 ± 15.12	62340.82
220	66242.21 ± 438.35	62810.16 ± 8.84	20.25 ± 0.48	65517.85
230	70237.66 ± 404.48	66649.87 ± 18.96	24.97 ± 7.83	69416.02
240	76063.41 ± 403.92	72461.73 ± 12.85	44.70 ± 6.16	75178.57
250	77558.16 ± 336.19	73982.45 ± 12.66	32.22 ± 3.86	76856.05
260	81739.41 ± 485.51	77883.85 ± 18.53	39.15 ± 9.08	80752.46
270	82243.45 ± 325.69	78258.86 ± 9.85	25.00 ± 12.84	81635.57
280	86999.08 ± 423.88	82805.05 ± 13.91	31.25 ± 9.86	86288.80
290	91282.96 ± 310.18	87366.05 ± 17.80	36.14 ± 6.08	89890.12
300	91871.88 ± 351.93	87672.34 ± 19.70	20.46 ± 1.48	90107.05

$V_{x,i} \in \{1, 0\}$

Note that Connection$(x, y) = 1$ if there exists an edge between vertex x and vertex y, 0 otherwise. $V_{x,i} = 1$ means that the xth vertex belongs to the ith partition. The goal of the problem is to divide a graph into two subsets so as to minimize the number of removed edges where edges in the same subset are only removed from the given graph. Note that edges bridging between two subsets are not removed. Based on the described energy function, the motion equation is given by:

$$\frac{dU_{x,i}}{dt} = -\sum_{y \neq x}^{N} \text{Connection}(x, y) V_{y,i} \qquad (7)$$

The simulator based on (7) has been developed in the similar manner as shown in the module orientation problem. We have simulated a massive number of instances of the bipartite subgraph problem including up to 1000-vertex problems. The simulation result shows the consistency of our algorithm for the bipartite subgraph problem and supports our claim. Because of the limitation of the current printing technology, more than 100-vertex bipartite subgraph problems could not be shown in this paper. Figure 7 shows the simulation result of the 50-vertex 175-edge bipartite subgraph problem where 44 edges are removed from the original graph. Figure 8 shows the relation between the energy and the number of iteration steps for the 50-vertex 175-edge problem. Figure 9 and 10 depict the simulation result of the 100-vertex 692-edge bipartite subgraph problem and the relation between the energy and the number of iteration steps respectively. In Fig. 9, 235 edges are removed from the original graph. In Fig. 7 and 9 black and grey vertices represent two subsets.

5 Conclusion

In this paper we have proposed a new approach to empirically solving certain NP-complete problems in a polynomial time. The simulator was developed for solving massive instances of the module orientation problem and the bipartite subgraph problem. The simulation result empirically substantiates our claim such that one class of NP-complete problems may be solvable in a polynomial time. The relationship between some important NP-complete problems is illustrated in Fig. 11 where an arrow represents the exact transformation. In other words, no exact-polynomial transformation algorithm is needed so that they are mathematically equivalent to each other (Garey and

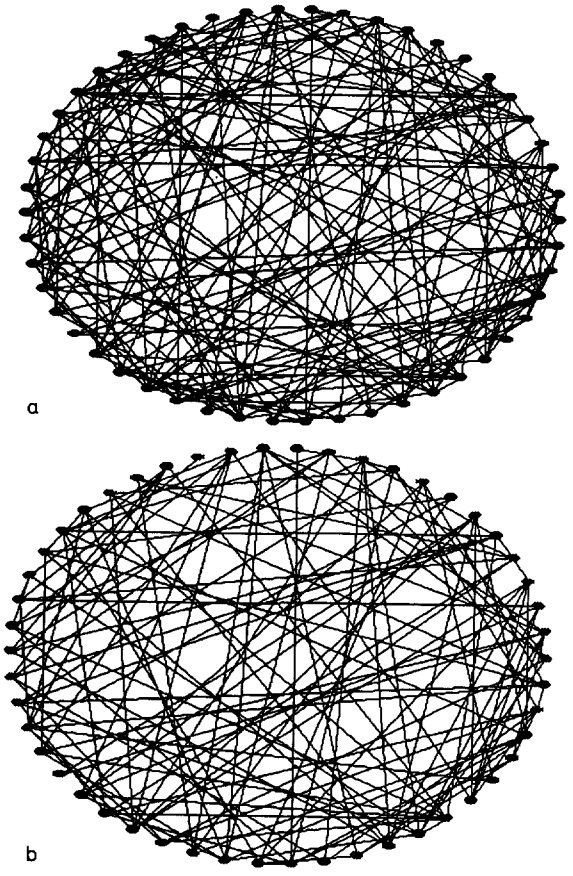

Fig. 7a,b. 50-vertex 175-edge bipartite subgraph problem. **a** 50-vertex 175-edge graph, **b** final solution with 131-edge

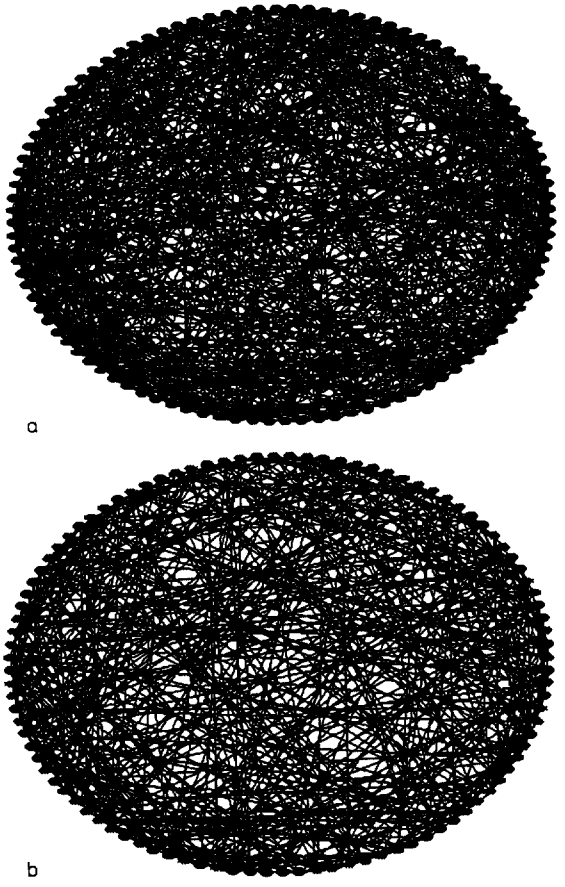

Fig. 9a,b. 100-vertex 692-edge bipartite subgraph problem. **a** 100-vertex 692-edge graph, **b** final solution with 452-edge

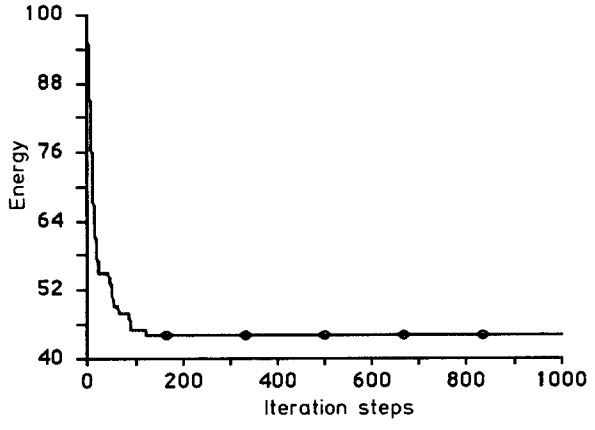

Fig. 8. The relation of the energy and the number of iteration steps for the 50-vertex 175-edge bipartite subgraph problem

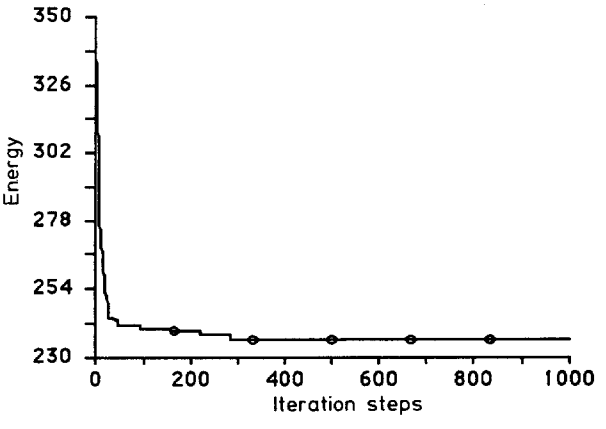

Fig. 10. The relation between the energy and the number of iteration steps for the 100-vertex 692-edge graph problem

Johnson 1979). Our algorithms are able to solve the NP-complete problems described in Fig. 11 without any additional transformation algorithm (Lee and Takefuji 1991). We have successfully applied the maximum neural model to solve the problems listed in Fig. 11 (Lee 1991). It can be concluded that the maximum neural network has the following advantages over the conventional neural network models: *1)* the maximum neural network model has the exact termination condition to terminate the procedure, while in the existing neural network models the condition is not clearly defined; *2)* the maximum neural network always guarantees the

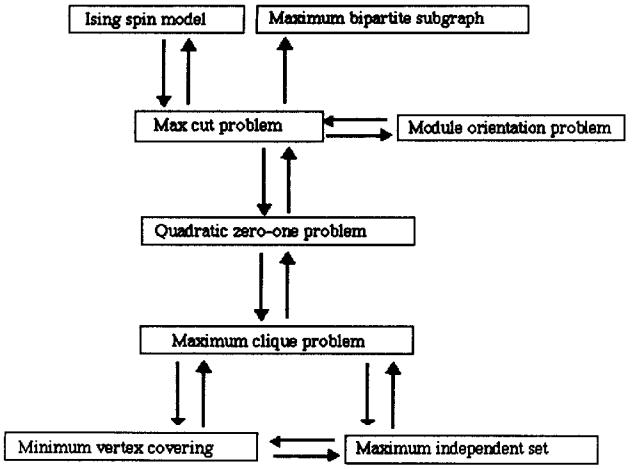

Fig. 11. The relationship between certain NP-complete problems

feasible solution while the existing neural network models generate invalid solutions; and *3)* coefficient-parameter adjustment or turing is not needed in the maximum neural network while the existing neural network models must suffer from it.

Appendix The proof of the local minimum convergence

Lemma 1 and lemma 2 are introduced to prove that the proposed system is always allowed to converge to the local minimum.

Lemma 1. $dE/dt \leqslant 0$ *is satisfied under two conditions such as* $(dU_i/dt) = -(\partial E/\partial V_i)$ *and* $V_i = f(U_i)$ *where* $f(U_i)$ *is a nondecreasing function.*

Proof. $dE/dt = \Sigma_i (dU_i/dt) \quad (dV_i/dU_i) \quad (\partial E/\partial V_i) = -\Sigma_i (dU_i/dt)^2 (dV_i/dU_i)$ where $\partial E/\partial V_i$ is replaced by $-dU_i/dt$ (condition 1) $\leqslant 0$ where $dV_i/dU_i > 0$ (condition 2) Q.E.D.

In Lemma 2, the local minimum convergence of the maximum neural network is given.

Lemma 2. $dE/dt \leqslant 0$ *is satisfied under two conditions such as (1)* $dU_{m,i}/dt = -\partial E/\partial V_{m,i}$ *and (2)* $V_{m,i} = 1$ *if* $U_{m,i} = \max\{U_{m,1}, U_{m,2}, U_{m,3}, U_{m,4}\}$ *and* $U_{m,i} \geqslant U_{m,j}$ *for* $i > j$ 0 *otherwise*

Proof. Consider the derivatives of the computational energy E with respect to time t.

$$\frac{dE}{dt} = \sum_m \sum_i \frac{dU_{m,i}}{dt} \frac{dV_{m,i}}{dU_{m,i}} \frac{\partial E}{\partial V_{m,i}}$$

$$= -\sum_m \sum_i \left(\frac{dU_{m,i}}{dt}\right)^2 \frac{dV_{m,i}}{dU_{m,i}}$$

where $\partial e/\partial V_{m,i}$ is replaced by $-dU_{m,i}/dt$ (condition 1). Let $dU_{m,i}/dt$ be $U_{m,i}(t + dt) - U_{m,i}(t)/(dt)$. Let $dV_{m,i}/dU_{m,i}$ be $(V_{m,i}(t + dt) - V_{m,i}(t))(U_{m,i}(t + dt) - U_{m,i}(t))$. Let us consider the term $\Sigma_i (dU_{m,i}/dt)^2 \; dV_{m,i}/dU_{m,i}$ for each module separately. Let $U_{m,a}(t + dt)$ be the maximum at time $t + dt$ and $U_{m,b}(t)$ be the maximum at time t for the module m. $U_{m,a}(t + dt) = \max\{U_{m,1}(t + dt), U_{m,2}(t + dt), U_{m,3}(t + d), U_{m,4}(t + dt)\}$ $U_{m,b}(t) = \max\{U_{m,1}(t), U_{m,2}(t), U_{m,3}(t), U_{m,4}(t)\}$. It is necessary and sufficient to consider the following two cases:

1) $a = b$

2) $a \neq b$

If the condition *1)* is satisfied, then there is not state change for the module m. Consequently, $\Sigma_i (dU_{m,i}/dt)^2 \, dV_{m,i}/dU_{m,i}$ must be zero.

If *2)* is satisfied, then

$$\sum_i \left(\frac{dU_{m,i}}{dt}\right)^2 \frac{dV_{m,i}}{dU_{m,i}}$$

$$= \left(\frac{U_{m,a}(t + dt) - U_{m,a}(t)}{dt}\right)^2 \frac{V_{m,a}(t + dt) - V_{m,a}(t)}{U_{m,a}(t + dt) - U_{m,a}(t)}$$

$$+ \left(\frac{U_{m,b}(t + dt) - U_{m,b}(t)}{dt}\right)^2 \frac{V_{m,b}(t + dt) - V_{m,b}(t)}{U_{m,b}(t + dt) - U_{m,b}(t)}$$

$$= \left(\frac{U_{m,a}(t + dt) - U_{m,a}(t)}{dt}\right)^2 \frac{1}{U_{m,a}(t + dt) - U_{m,a}(t)}$$

$$+ \left(\frac{U_{m,b}(t + dt) - U_{m,b}(t)}{dt}\right)^2 \frac{-1}{U_{m,b}(t + dt) - U_{m,b}(t)}$$

$$= \frac{U_{m,a}(t + dt) - U_{m,a}(t)}{(dt)^2} - \frac{U_{m,b}(t + dt) - U_{m,b}(t)}{(dt)^2}$$

$$= \frac{1}{(dt)^2}\{U_{m,a}(t + dt) - U_{m,a}(t) - U_{m,b}(t + dt) + U_{m,b}(t)\}$$

$$= \frac{1}{(dt)^2}\{U_{m,a}(t + dt) - U_{m,b}(t + dt) + U_{m,b}(t) - U_{m,a}(t)\} > 0$$

because $U_{m,a}(t + dt)$ is the maximum at time $t + dt$ and $U_{m,b}(t)$ is the maximum at time t for the module m. The contribution from each term is either 0 or positive, therefore

$$\sum_i \left(\frac{dU_{m,i}}{dt}\right)^2 \frac{dV_{m,i}}{dU_{m,i}} \geqslant 0 \text{ and}$$

$$-\sum_m \sum_i \left(\frac{dU_{m,i}}{dt}\right)^2 \frac{dV_{m,i}}{dU_{m,i}} \leqslant 0 \Rightarrow \frac{dE}{dt} \leqslant 0 \text{ Q.E.D.}$$

References

Cook SA (1971) Proc. of 3rd Ann. ACM Symp. on Theory of Computing. ACM, New York, pp 151–158

Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco

Geman S, Geman D (1984) Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. IEEE Trans Patt Anan Mach. Int PAMI-6:721

Hanan M, Wolff Sr PK, Agule BJ (1976). J Design Automat Fault Tolerant Comput 1:28–61

Hopfield JJ, Tank DW (1985) Neural computation of decisions in optimization problems. Biol Cybern 52:141–152

Kirkpatrick S, Gelatt Jr CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220:4598:671–680

Lee KC (1991) Ph.D. thesis. Case Western Reserve University, Cleveland

Lee KC, Takefuji Y (1991) CAISR Tech. Rep. TR91-104 (Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland)

Libeskind-Hadas R, Liu CL (1989) Proc. 26th Design Automation Conference. ACM, New York, pp 400–405

McCulloch WS, Pitts WH (1943) A logical calculus of the ideas imminent in nervous activity. Bull Math biophys 5:115

Preas BT, Karger PG (1988) Physical design automation of VLSI systems. In: Preas B, Lorenzetti M (eds) chap 4. Benjamin-Cummings, Menlo Park, Calif, pp 87–155

Takefuji Y, Lee K C (1990) A parallel algorithm for tiling problems. IEEE Trans Neural Networks 1:143–145