An Efficient Learning Algorithm of the Hopfield Neural Network for the Minimum Set Cover Problem

Rong-Long Wang[1], Pei Zhang[1,2] and Kozo Okazaki[1]

[1]*Faculty of Engineering, Fukui University, Fukui-shi, Japan 910-8507*
[2]*Faculty of Information Science and Technology, Beijing University of Chemical Technology, Beijing , China*

Summary
The minimum set cover problem (MSCP) is an important NP-hard problem. In this paper, we propose a learning algorithm of the Hopfield neural network which can escape from local minima, for efficiently solving the problem. Extensive simulations are performed, and the simulation results show that the proposed learning algorithm works much better than the other existing algorithms on random instances of hypergraphs.

Key words:
Hopfield neural network, Learning algorithm, Minimum set cover problem, NP-complete problem

Introduction

The minimum set cover problem (MSCP) is the problem of finding the smallest set cover in a given hypergraph [1]. It has many practical applications in important fields such as production, capital investment, project selection; it includes airline crew scheduling [2], location of emergency services [3], and assembly line balancing [4]. Since MSCP is an NP-hard problem [1], it is useful to study approximation algorithms for it. Some greedy algorithms are proposed. One of the best greedy heuristic was discovered by Johnson [5], which is referred to as *G2*. However, the classical algorithms cannot be applied under parallel computation platforms. For solving such problems, the Hopfield neural network [6] constitutes an important avenue. Using the Hopfield neural network technique, D. Kaznachey et al. [7] proposed a parallel algorithm for the problem. Unfortunately, due to its inherent difficulties at dealing with local minima, the probability of obtaining the minimum set cover using the Hopfield neural network is very low.

In this paper, a learning algorithm of the Hopfield neural network is presented for solving the problem. The learning algorithm adjusts a parameter in the energy function so that the local minimum that the network once falls into vanishes and the network can continue updating in a gradient descent direction of energy. We evaluate the proposed learning algorithm by simulating a large number of random instances of hypergraphs. The simulation results are compared with those of Simulated Annealing (SA) [8], a greedy algorithm called G2[5] and the Hopfield neural network method presented by Kaznachey et al. [7]. The simulation results show that the proposed learning algorithm works much better than the other algorithms on random instances of hypergraphs.

2. Problem Formulation

Given a hypergraph $H=(V,E)$ with a vertex set $V=\{v_1,v_2,\cdots,v_n\}$ and an edge set $E=\{e_1,e_2,\cdots,e_m\}$, a set cover is a subset of vertices ($C\subseteq V$) that covers all the edges. The MSCP is the problem of finding the smallest set cover in a given hypergraph [1]. In this paper, we deal with *k*-cardinality hypergraphs, i.e., hypergraphs whose edges have maximum cardinality *k*. A hypergraph *H* can be represented by an incidence matrix $A = (a_{ij})$ in which a_{ij} is 1 if vertex *j* is in the edge *i* and 0 otherwise.

In general, an *n*-vertex MSCP can be mapped onto the Hopfield neural network with *n* neurons. Neuron y_i represents #*i* vertex. The output of neuron y_j is 1 if the vertex *i* is included into the set cover and 0 otherwise. Then the goal of the problem is to minimize $\sum_{i=1}^{n} y_i$ subject to constraints $\sum_{i=1}^{n} a_{ei} \geq 1$ for each edge $e=1,2,\cdots,m$. When we follow the mapping procedure by Hopfield [6], the energy function for the MSCP is given by:

$$E=\alpha\cdot\sum_{i=1}^{n} y_i+\beta\cdot\sum_{e=1}^{m}\left(d-\sum_{i=1}^{n} a_{ei}y_i\right)^2 \quad (1)$$

Where α,β are parameters. The first term in Eq.(1) is cost term and the second one represents the quadratic constraints, and is minimized when the number of vertices covering each edge $\sum_{i=1}^{n} a_{ei}y_i$ equals parameter *d*. If *d* is chosen properly, this term will be maximized if the edge is not covered. Evidently, *d* must be between 0 and *k*. Parameter *d* is very important, but difficult to be selected. If *d* is too small, the second term in Eq.(1) does not

Manuscript received September 5, 2006.
Manuscript revised September 19 , 2006.

penalize uncovered edges. On the other hand, if *d* is too large, then even covered edges are penalized a lot.

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

$$e = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} y_i y_j - \sum_{i=1}^{n} I_i y_i \tag{2}$$

where w_{ij} (i,j=1,…,n) is weight of a synaptic connection from the j-th neuron to the i-th one, I_i is external input of neuron #i and is also called threshold.

For the MSCP problem, the resulting weight and threshold can now be obtained by equating the energy specified by Eq.(2) with the energy as in Eq.(1). The weight of the Hopfield network is

$$w_{ij} = -2\beta\sum_{e=1}^{m} a_{ei} a_{ej} \tag{3}$$

And the threshold is

$$I_i = 2d\beta\sum_{e=1}^{m} a_{ei} - \alpha \tag{4}$$

It is proved that the state of the Hopfield network converges to a stable state with the energy taking on lower and lower values [6]. It can be viewed as seeking a minimum in a mountainous terrain. Thus, we can find the solution to the minimum set cover problem simply by observing the stable state that the Hopfield network reaches. However once the network fall into a local minimum, the updating procedure will stop. There is no way for the network to reach the global minimum from a local minimum. Because this local minimum problem and the difficulty of selecting the value of parameter *d*, the solution found by the existing parallel algorithm based on neural network are not good. In the next section, focusing on the problems, we propose a learning algorithm that can help network efficiently searching the optimal solution.

3. Learning Algorithm for the MSCP

In order to realize the global minimum convergence of the Hopfield neural network, we now propose a learning algorithm that adjusts the balance of two terms in Eq.(1) by modifying the parameter *d*, thus the local minimum vanishes. Because the energy terrain is determined partially by the parameter *d*, the learning can be performed by changing parameter *d* once the network fall into a local minimum so that the local minimum vanishes. The variation of energy of network with the state change of *i*# (i=1,…,n) neuron can be written as:

$$\Delta E_i = \frac{\partial E(y_1, y_2, \cdots, y_n)}{\partial y_i} \cdot \Delta y_i \qquad for\ i = 1,2,\cdots,n \tag{5}$$

We analyze the characteristics of binary Hopfield neural network. It is well known that a local minimum satisfies[9]

$$\Delta E_i \geq 0 \qquad for\ i=1,2,\ldots,n. \tag{6}$$

We now propose a learning algorithm to make $\Delta E_i < 0$ (it means that the local minimum vanishes) with the state change of neuron in local minimum. Using Eq.(1) we have:

$$\Delta E_i = \left[\left(\alpha + 2\beta\sum_{e=1}^{m}\sum_{k=1}^{n} a_{ek} y_k a_{ei}\right) - d \cdot 2\beta\sum_{e=1}^{m} a_{ei}\right]\Delta y_i \tag{7}$$

We note that:

$$\left(\alpha + 2\beta\sum_{e=1}^{m}\sum_{k=1}^{n} a_{ek} y_k a_{ei}\right) > 0 \tag{8}$$

$$\left(-2\beta\sum_{e=1}^{m} a_{ei}\right) < 0 \tag{9}$$

Thus, we can modify the parameter *d* using the following learning rule:

$$d = \frac{\alpha + 2\beta\sum_{e=1}^{m}\sum_{k=1}^{n} a_{ek} y_k a_{ei}}{2\beta\sum_{e=1}^{m} a_{ei}} + \delta \cdot \Delta y_i \tag{10}$$

Where δ is a small positive constant that controls the learning speed. We know that with the state change of the *i*# neuron, the variation of the energy of the network can be described by the following formula by substituting Eq.(10) into Eq.(7).

$$\Delta E_i = \left[-2\beta \cdot \delta \cdot \sum_{e=1}^{m} a_{ei}\right]\Delta y_i^2 \tag{11}$$

It is evident that:

$$\Delta E_i = \left[-2\beta \cdot \delta \cdot \sum_{e=1}^{m} a_{ei}\right]\Delta y_i^2 < 0 \tag{12}$$

The derivatives of Eqs. (11) and (12) show that the energy of the network decreases with the state change of the *i*# neuron by the above learning rule (Eq.(10)). Thus, the learning (Eq.(10)) eliminate the local minimum that the network falls into. Besides the learning (Eq.(10)) also provide a method to select a critical value of parameter *d*.

4. Algorithm

The following procedure describes the proposed algorithm for the MSCP problem of *k*-cardinality hypergraphs. 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 traveling salesman problem and the MSCP 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 MSCP problem,

Table 1: Simulation Results

Test Graphs			SetCover Size				Validity Ratio(%)			
No.	Notes	Edges	G2	SA	Kaznachey et al.	Our Method	G2	SA	Kaznachey et al.	Our Method
1	50	19	12	20	13	10	100	100	100	100
2	50	115	29	25	29	22	100	100	80	100
3	50	17628	48	50	48	47	100	100	70	100
4	100	154	44	66	83	40	100	100	82	100
5	100	796	76	85	85	60	100	100	60	100
6	100	1632	82	78	89	75	100	100	83	100
7	100	16053	95	90	96	90	100	100	84	100
8	200	162	65	108	129	56	100	100	96	100
9	200	785	123	180	157	120	100	100	40	100
10	200	1348	138	160	185	131	100	100	70	100
11	400	10570	280	280	361	240	100	100	63	100
12	400	6524	328	380	361	320	100	100	40	100
13	400	105765	395	400	395	390	100	100	58	100

we found that the network can always find good solutions within 10 learning times; therefore, we selected 20 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 the following algorithm for solving a *k*-cardinality hypergraphs:

1. Set *learn_time*=0, *learn_limit=20,* and $\alpha = 2.0$, $\beta = 1.0$, $d = (k+1)/2$
2. The initial value of y_i for *i=1,…,n* are randomized in 0 or 1.
3. The updating procedure is performed on the Hopfield network with original weights and thresholds until the network converges a stable state.
4. Record the stable state.
5. Use the learning rule (Eq.(10)) to modify the parameter *d*.
6. Compute the new weights and new thresholds (Eq.(3) and Eq.(4)) using the new *d*.
7. The updating procedure is taken on the Hopfield network with the new weights and thresholds until the network reaches a stable state.
8. If the new stable state is better than the recorded stable state, then the recorded stable state is replaced by the new stable state obtained from step 7.
9. Increment the *learn_time* by 1. If *learn_time=learn_limit* then terminate this procedure, otherwise go to the step 5.

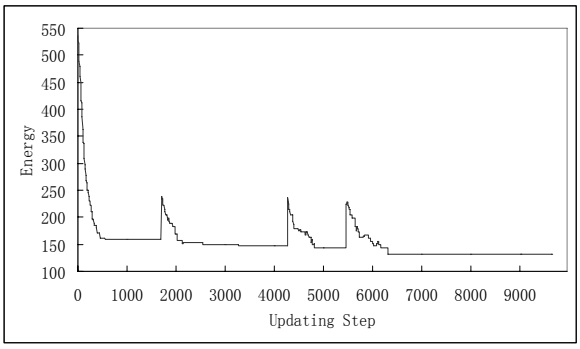

Fig. 1 The variation process of the energy during learning

5. Simulation Result

In order to assess the effectiveness of the proposed learning method, extensive simulations were carried out over randomly generated instances on PC Station. Simulations referred to parameter set at $\alpha = 2.0$ $\beta = 1.0$ and $\delta = 0.1$. The first instance that we tested was a randomly generated 100-vertex 154-edges hypergraph. Figure.1 showed the change of energy on the instance that illustrates a typical progressive intermediate solution during the learning. Initially the Hopfield network converged to a stable state. From this stable state, we found that there were two uncovered edges. It was obviously not a valid solution. After the first learning, the Hopfield network found a valid set cover with 44 vertices. In this problem, the network performed totally 3 learning and finally found the set cover with 40 vertices.

To widely verify the proposed algorithm, we tested the algorithm with a large number of randomly generated *p*-random *n*-vertex *k*-cardinality hypergraphs in which each of the possible edges is independently included in the

graph with probability *p*. In the experiment, up to 400-vertex graphs with different probability are used. To evaluate our results, we compared our results with those of Simulated Annealing (SA) [8], a greedy algorithm called G2[5] and the Hopfield neural network method presented by Kaznachey et al.[7]. For each of instances, 100 simulation runs with different initial input values were performed. Information on the test graphs as well as all results are shown in Table.1. The results that we recorded for each graph are the solutions in number of vertex produced by each algorithm and the ratio of finding valid solution for each methods. From table.1, we can know that the proposed parallel algorithm can find better solution than other algorithms, and the solution found by the proposed learning algorithm is always a valid solution

Table 2: Computation Time

No	CPU Time(s)			
	G2	SA	Kaznachey et al.	Our Method
1	0.09	0.22	1.67	0.02
2	1.78	1.8	1.2	0.12
3	223	2204	94.2	165.14
4	11.3	3.7	2.72	0.17
5	10.51	46.1	31.3	2.05
6	31.34	76.7	3.25	5.12
7	218.2	4135	262.09	499.24
8	10.6	9.4	313.33	2.87
9	101.1	89.1	28.03	10.92
10	109.4	28.33	685.52	22.63
11	172.8	380.1	1352.7	71.86
12	823.4	9698	2270.9	3426.6
13	1517	97984	1883.2	67155

The average computation time within 100 runs are shown in the Table 2. From the table we can know that although the computation time of the proposed learning algorithm is larger than that of algorithms G2 and the algorithm of Kaznachey et al., it is realistic considering the size of the problem. Besides, it is worth to note that our simulations were performed on a series computer for generating optimal or near-optimal solution to the MSCP problem, and naturally result in large CPU times that were uncompetitive with alternative techniques. We adhered to the philosophy that the model being tested should be always having a possible "silicon implementation" [10]. Thus on a parallel computation device the computation time of the proposed method will become very short. Furthermore, because of the simplicity of the proposed learning method, it is easy to implement the proposed method on electronic circuits.

6. Conclusion

We have proposed a Hopfield network learning algorithm for minimum set cover problem and showed its effectiveness by simulation experiments. The simulation results showed that the proposed algorithm works much better than the other existing algorithms on random instances of hypergraphs, and has higher convergence rate to valid solution.

References

[1] M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Comleteness, Freeman, New York, 1979.

[2] J. Bartholdi, "A guaranteed-accuracy round-off algorithm for cyclic scheduling and set covering," Operations Research, Vol. 29, No.3, May/Jun, pp.501-510, 1981.

[3] C. Toregas, R. Swain, C. Revelle, L. Bergman, "the location of emergency service facilities," Operations Research, Vol.19, No.6, pp.1363-1373, 1971.

[4] S. T. Hackman, M. J. Magazine, T. S. Wee, "Fast effective algorithms for simple assembly line balancing problems," Operations Research, Vol.37, No.6, pp.916-924, 1989.

[5] D.S. Johnson, "Approximation algorithms for combinatorial problems", J. *Computer System Sci.* Vol. 9, pp.256-278, 1974.

[6] J. J. Hopfield, "Neurons with graded response have collective computational properties like those of two-state neurons," Proc. Natl. Acad. Sci. USA, Vol.81, pp.3088-3092, 1984.

[7] D. Kaznachey and A. Jagota, "Approximating Minimum set Cover in a Hopfield-Style Network," Information Sciences, Vol.98, No.1-4, pp.203-216, 1997.

[8] D. S. Johnson, C. R. Aragon, L. A. McGeoch and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; Part 1, graph partitioning," Operations Research, Vol.37, No.6, pp.865-892, 1989.

[9] M.Ohta,A.Ogihara,K.Fukunaga, "Binary neural networks with negative self-feedback and its application to N-Queens problem" IEICE Trans. Inf. Systems, Vol.E77-D, No.4, pp.459-465, 1997

[10] K. A. Smith, "Neural Networks for combinatorial optimization: A Review of more than a decade of research," INFORMS Journal on Computing, Vol.11, No.1, pp.15-34, 1999.