VSP International
Science Publishers
P.O. Box 346, 3700 AH Zeist
The Netherlands

Lecture Series on Computer and Computational Sciences
Volume 1, 2004, pp. 375-378

Multivalued Neural Network for Graph MaxCut Problem

E. Mérida-Casermeiro[1] and D. López-Rodríguez[2]

Department of Applied Mathematics,
E. T. S. I. Informática,
University of Málaga
29071 Málaga, Spain

Received 30 July, 2004; accepted in revised form 28 August, 2004

Abstract: In this paper we have used a multivalued neural model (MREM) in order to solve the maximum cut problem. A new technique, based in the problem, that allows to escape of certain bad local minima has been incorporated in order to improve the goodness of the obtained solutions. Finally, we have made some comparisons with other neural formulations for that problem obtaining better solutions in a reasonable time.

Keywords: Maximum cut problem. Multivalued neural network. Combinatorial Optimization.

Mathematics Subject Classification: 82C32, 68W15

PACS: 120304

1 Introduction

One of the most important combinatorial optimization problems (COP) in graph theory is the maximum cut problem (MAXCUT). The goal of this problem is to find a partition of vertices of a weighted undirected graph $G = (V, E)$ into two disjoint sets, such that the sum of edges with endpoints in different set is as large as possible.

This problem has many applications to pattern recognition, clustering, statistical physics and the design of communication networks, VLSI circuits and circuit layout among them [2].

Although for planar graphs there exists a solution for this problem in polynomial time [5], the problem is known to be NP-complete [4] for a generic graph, even for unweighted graph. So, to obtain the optimum is computationally intractable and it is important to develop an algorithm that achieves near optimum solutions in reasonable time [9, 6, 3].

Neural networks have demonstrated to be a good alternative method to classical algorithms [10]. So, almost every type of COP has been tackled by neural networks, and many of the approaches are very competitive and efficient in terms of quality, time-consuming and resources.

In 1997, Alberti et al. presented a type-Hopfield neural model for MAXCUT [1], but its performance is worse than the presented by Bertoni et al [3]. Takefuyi and his colleagues [11] have developed a powerful neural model labelled 'maximum' and they have shown that their neural model performs better than the rest of algorithms in solving a wide range of COP.

[1]Corresponding author. E-mail: merida@ctima.uma.es
[2]E-mail: dlopez@ctima.uma.es

Recently, Galán-Marín et al. proposed a new neural model named OCHOM which obtains much more efficient solutions than 'maximum'. Moreover, it can be used for many COP and it also has the advantage of fast convergence to a valid solution without tuning any parameter.

In order to make OCHOM escape from local minima, Wang et al.[12] have recently proposed a stochastic dynamics for OCHOM, permitting temporary decreases of the objective function.

The model proposed in this paper obtains better solutions than those of Wang and OCHOM for MAXCUT in very short time-consumption. Moreover, the model can be applied to a generalization of MAXCUT and a new technique to scape from local minima is provided for this problem. This technique improves previous solutions and can be easily used with other COP.

2 The Maximum Cut Problem

Let $G = (V, E)$ be an undirected graph without self-connections. $V = \{v_i\}$ is the set of vertices and $E = (e_{i,j})$ is a symmetric binary matrix where $e_{i,j} = 1$, if and only if, an arc with endpoints v_i and v_j exists. For each edge in E there is a weight $c_{i,j} \in \mathbb{R}$. All weights can be expressed by a symmetric real matrix C, with $c_{i,j} = 0$ when it does not exist an arc with endpoints v_i and v_j.

The Maximum Cut Problem (MAXCUT) consists in finding a partition of V into two subsets A_1 and A_2, such that $\sum_{v_i \in A_1, v_j \in A_2, i>j} c_{i,j}$ is maximum.

Generalization of the MAXCUT Problem (K-MAXCUT): It looks for a partition of V into k disjoint sets A_i such that the sum of the weights of the edges from E that have their endpoints in different elements of the partition is maximum. So, the function to be maximized is

$$\sum_{v_i \in A_m, v_j \in A_n, i>j} c_{i,j} \tag{1}$$

3 Neural implementation

In order to solve the K-MAXCUT problem, we have used the MREM neural model since this model has been successfully used for other COP [7, 8].

The MREM neural model: It consists in a series of multivalued neurons, where the state of i-th neuron is characterized by its output (s_i) that can take any value in any finite set M. This set can be a non numerical one, but, in this paper, the neuron outputs only take value in $M \subset \mathbb{Z}^+$.

The state vector $\vec{S} = (s_1, s_2, \ldots, s_N) \in M^N$ describes the network state at any time, where N is the number of neurons in the net. Associated with any state vector, there is an energy function $E : M^N \to \mathbb{R}$, defined by the expression:

$$E(\vec{S}) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{i,j} f(s_i, s_j) \tag{2}$$

where $W = (w_{i,j})$ is a matrix and $f : M \times M \to \mathbb{R}$ is usually a similarity function since it measures the similarity between the outputs of neurons i and j. At each step, the state vector will be evolving to decrease the energy function.

To solve the K-MAXCUT problem with this neural net, we need as many neurons as number of nodes N in the graph. Each neuron taking value $s_i \in M = \{1, 2, \ldots, k\}$ points to the subset of the partition where the i-th node is assigned to.

The cost function of the K-MAXCUT, given by equation-1, must be identified with the energy function of equation-2. So, $w_{i,j} = -c_{i,j}$ and $f(x, y) = \delta_{x,y}$ (Krönecker delta function).

Initially the state vector is randomly selected. At any time, the net is looking for a better solution than the current one, in terms of the energy function.

Some dynamics can be used for MREM, but we have used one that we have named best-2. It consists in selecting the greatest decrease of the energy function by changing the state of only two neurons at the same time.

Denoting by $s_i = s_i(t)$ the state of the i-th neuron at time t and $s'_i = s_i(t+1)$ the next state of the neuron, an expression for the energy increment when two neurons p and q change their states ($s_p(t) = a$, $s_p(t+1) = a'$, $s_q(t) = b$, $s_q(t+1) = b'$), is given by:

$$\Delta(E) = E(\vec{S}(t+1)) - E(\vec{S}(t)) = \sum_{j=1}^{N} w_{p,j} \left[\delta_{a',s'_j} - \delta_{a,s_j}\right] + \sum_{i=1, i\neq p}^{N} w_{i,q} \left[\delta_{s'_i,b'} - \delta_{s_i,b}\right] \quad (3)$$

So, the dynamics of the neural net can be summarized as follows:

- A state vector is randomly assigned.
- Repeat until there is not any change in the state vector
  - At each step, only two neuron outputs can be updated.
  - The scheduling sequentially selects one value $d \in \{1, 2, \ldots, N-1\}$.
  - The next process can be made in parallel. The p-th neuron builds a $k \times k$ matrix B^p whose element $b^p_{i,j}$ equals the decrease of energy resulting from making $s_p(t+1) = i$, $s_q(t+1) = j$, where $q = (p+d) \mod (N)$, $0 < q \leq N$, by applying equation-3.
  - Neuron p then calculates the minimum $\beta_p = b^p_{i,j}$ of the elements in matrix B^p.
  - After this parallel process, the scheduling calculates the minimum of the vector $\vec{\beta}$: $\beta_0 = b^{p_0}_{i_0,j_0}$ and the state vector is updated by changing outputs of neurons p_0 and $q_0 = p_0 + d$ to $s_{p_0}(t+1) = i_0$, $s_{q_0}(t+1) = j_0$.

4 Improving solutions

A new technique to escape from local minima of the energy function has been developed. It improves greatly the quality of the solutions with a little increment of time-consuming.

Given an estimated solution (e.g. the stable state of the previous neural algorithm), a good solution usually has the following property:

"High-weighted arcs must have their endpoints in different subsets".

So, we can study the high-weighted arcs. Let A be the set of arcs with weights greater than a threshold and endpoints in the same subset. Let $V^* \subset V$ be the set of endpoints of arcs in A. Then the current solution is saved and the shake phase begins. It consists in:

- Selecting the nodes that are endpoints of arcs in A and their neighbors: $H = \{v_i / \exists v_j \in V^*, e_{i,j} = 1\}$.
- Nodes in $V - H$ are clamped to their current values, while nodes in H are randomly assigned.
- With this new initial state vector, the network evolves with the usual dynamics, but only nodes in H will be selected in order to be modified, until a new stable state is reached.
- This new solution is compared with the previous saved one and the best one is selected.

5 Simulation Results

We have compared our proposed algorithms to OCHOM and Wang's. All of them have been implemented and tested in MATLAB, on a Pentium IV (3.06 Ghz). More specifically, Wang's network has been tested with its default parameter $\lambda = 30$. In the proposed model, the set A was built by including every edge $e_{i,j}$ whose cost $c_{i,j} > \overline{c} + 3\sigma$, where $\overline{c}$, σ are respectively the mean and the standard deviation of $c_{i,j}$. So, A is forced to include exclusively high-weighted edges.

The test set was built by using the same parameters that Wang et al in [12]. It is formed by random graphs, dependent on two parameters, N and $\rho \in (0, 1)$ (density) meaning that the number of edges in E is the closest integer to $\rho \frac{N(N-1)}{2}$. Ten items for each $N \in \{20, 50, 80, 100\}$ and each $\rho \in \{0.05, 0.15, 0.25\}$ have been used. Weights were integers randomly chosen in $[-1, 5]$.

On each graph in the test set, 10 simulations of each algorithm were performed. Both best and average solutions obtained are shown in Table 1. So, we can verify that the proposed algorithm outperforms others, not only giving the best results, but even on average.

Table 1: Best and average performance on test set.

			Wang			OCHOM			MREM			MREM-shake		
N	ρ	N_a	Best	Av.	t	Best	Av.	t	Best	Av.	t	Best	Av.	t
20	0.05	10	26	21.8	0.003	26	24.4	0.001	26	25.4	0.024	26	25.4	0.023
20	0.15	27	67	29.0	0.002	69	66.5	0.001	69	68.0	0.027	69	68.0	0.027
20	0.25	43	80	63.6	0.002	86	78.5	0.001	86	84.4	0.024	86	84.4	0.025
50	0.05	58	144	113.4	0.016	142	137.4	0.005	149	143.5	0.243	149	143.5	0.265
50	0.15	161	278	248.8	0.015	273	264.6	0.005	284	276.7	0.234	284	277.0	0.369
50	0.25	256	460	397.9	0.012	476	448.8	0.006	469	460.6	0.244	472	463.9	0.482
80	0.05	143	270	238.0	0.031	266	258.6	0.011	279	271.5	0.713	279	271.5	1.025
80	0.15	403	715	702.5	0.034	739	712.4	0.014	754	735.3	0.943	754	742.2	1.954
80	0.25	680	1100	878.2	0.034	1106	1080.5	0.016	1117	1091.0	0.857	1117	1095.1	1.717
100	0.05	204	400	323.7	0.048	407	390.2	0.017	418	406.0	1.539	418	406.6	2.374
100	0.15	631	1071	843.6	0.068	1060	1029.1	0.023	1081	1062.3	1.629	1084	1068.5	3.257
100	0.25	1058	1697	834.4	0.043	1728	1682.3	0.025	1741	1702.7	1.323	1741	1714.8	2.407

References

[1] A. Alberti, A. Bertoni, P Campadelli, G. Grossi and R. Posenato. *A neural algorithm for MAX-2SAT: performance analysis and circuit implementation*, Neural Networks **10-3**, 555-560(1997).

[2] F. Barahona, M. Grotschel, M. Junger and G. Reinelt, *An Application of combinatorial optimization to statistical physics and circuit layout design.* Operat. Research **36** 493-513(1988).

[3] A. Bertoni, P. Campadelli and G. Grossi, *An approximation algorithm for the maximum cut problem and its experimental analysis.* Proceedings: Algorithms and experiments. Trento, **9-11**, 137-143(1998).

[4] M.R. Garey and D.S. Johnson, *Computers and Intractability. A guide to the theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[5] F. Hadlock, *Finding a maximum cut of a planar graph in polinomial time.* SIAM J. Computation **4-(3)**, 221-225(1975).

[6] C.P. Hsu, *Minimum-via topological routing.* IEEE Transaction Computer-Aided Design. Integrated Circuits Systems, **2** 235-246(1983).

[7] E. Mérida-Casermeiro, G. Galán-Marín and J. Muñoz-Pérez. *An Efficient Multivalued Hopfield Network for the Traveling Salesman Problem.* Neural Processing Letters **14** 203-216(2001).

[8] E. Mérida-Casermeiro, J. Muñoz-Pérez and R. Benítez-Rochel *Neural Implementation of Dijkstra's Algorithm.* Lecture Notes in Computer Science **2686** 342-349(2003).

[9] S. Sahni and T. Gonzalez, *NP-complete approximation problems.* J.ACM **23** 555-565(1976).

[10] K. A. Smith, *Neural Networks for Combinatorial Optimization: A review of more than a decade of research.* INFORMS Journal on Computing **11-(1)**, 15-34(1999).

[11] Y. Takefuyi and J. Wang, *Neural computing for optimization and combinatorics.* Singapore, World Scientific, Chapter **3**, (1996).

[12] Jiahai Wang and Zheng Tang. *An improved optimal competitive Hopfield network for bipartite subgraph problems.* Neurocomputing (In press).