SOLVING MAX-CLIQUE USING CELLULAR NEURAL NETWORK

Y. SANTOSH REDDY

NIT
ROURKELA, ORISSA, INDIA

ABSTRACT

In this paper, we present an artificial life method of the Cellular neural network for Max-Clique problem. The method is intended to provide an optimum parallel algorithm for solving the Max-Clique problem. To do this we use the Cellular neural network to get a maximum Clique. Some of the instances are simulated to verify the proposed method with the simulation results showing that the solution quality is superior to that of best existing parallel algorithm. We also test the learning method on total coloring problem.

INTRODUCTION

The two-dimensional binary cellular automata (CA) have a greed structure. Each node contains the cell that changes its state in discrete manner due to the transition function (rule). There are thousands different CA in the automata space. Wolfram puts them to the four different classes, according to their dynamical behavior. Here we used 5-elements neighborhood cellular automata. In this automaton a cell's next state is dependent on four of the neighborhood cells, which are located above, below, left, right of the central cell (CC). The 5-elements neighborhood is shown in fig.1.

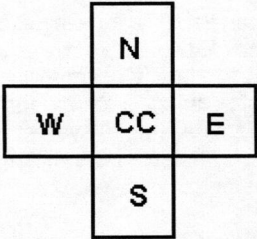|	N	
W	CC	E
	S	

Fig.1. 5-elements neighborhood

Each cell has eight adjacent elements among those four are used to change the state of central cell (CC).

Max-Clique problem is NP-Complete problem. NP-Complete problems are hardest problems in the set NP. Clique is the complete subgraph of an existing graph. If there is a graph G=(V, E) where V is the set of vertices and E is the set of edges. Complete graph has edge between

i and j where $i, j \in V$. Complete graph with N vertices having N*(N-1) /2 edges.

DECOMPOSITION OF RULE

Here we used the rule, which has properties given below.
(1) Among 5-elements shown in fig.1 if 1's are more next CC is 1.
(2) Else next CC is 0.
The set of their states is called the configuration v **(t).**

$$V (t)=(CC (t), E (t), N (t), W (t), S (t))$$

Apart from central cell (CC) we divided the four neighbors into six types of categories

Σ_i = i number of 1's exist where $i \in \{0,1,\dots,5\}$
We can decompose this set into two based on CC. Those are shown in fig.2 and fig.3. When CC is 0 then the state table is fig.2. When CC is 1 then the state table is fig.3. In fig.2 there is no such state of all having 1. In fig.3 there is no such state of all having 0.

E W N S	00	01 10	11
00	Σ_0	Σ_1	Σ_2
01 10	Σ_1	Σ_2	Σ_3
11	Σ_2	Σ_3	Σ_4

Fig.2. State table of Cellular automata rule when CC=0

The combined state table of the cellular automata rule is shown in fig.4. The difference between fig.2 and fig.3 is the diagonal from right top to left bottom. That is called the critical diagonal, which shows difference between TRUE and FALSE states.

Fig.3. State table of Cellular automata rule when CC=1

Fig.4.Generelized State table of Cellular automata rule

This study we can use in the finding of maximum Clique in a given graph. How our problem is depending on these cellular automata rule is described in the next section. After that we can discuss the algorithm, which is more important part in the paper where we discuss cellular automata updating and neural network usage in the problem solving.

INTERCONNECTION NETWORK

I selected rule FEE87880 because this rule has the property of self-extendibility and there must be a state called constant state. When there are more 0's than 1's that will make FALSE to that cell. Otherwise make TRUE that cell. Because of this the cellular automata state space will increase or decrease 1's or 0's based on the initial states. Here only two states 0 or 1.

In finding of Max-Clique of a given graph the main thing is discovering of the nodes, which are not connected directly. Using this rule first non-neighbor nodes are discovered and they are combined according to that of algorithm. So at every stage there will be new state space produced. In this problem we must avoid loosely connected nodes, but finding loosely connected nodes is very much difficult because all loosely connected are not victims. To overcome such difficulties we used neural networks because it is the universal learner.

Here we introduced a cellular neural network structure, which gives input as cellular automata state space to the cellular automata next state generator, it will give next state of the state space. Next there is a comparator is there which compares input state space and next state space, after comparison it gives differences in the state space to the neural network. Neural network updates the state space of the cellular automata and produces the modified state space. This process continues up to a constant state that is all having TRUE state or all having FALSE state. But according to our algorithm all having FALSE state is not reachable because neural network is takes care about it and it make state space not all FALSE. If it reaches all cells FALSE after comparison neural network will make cells TRUE. The cellular automata rule given has the property of making cells all, which are dominating. Interconnection network among all modules is shown in fig.5.
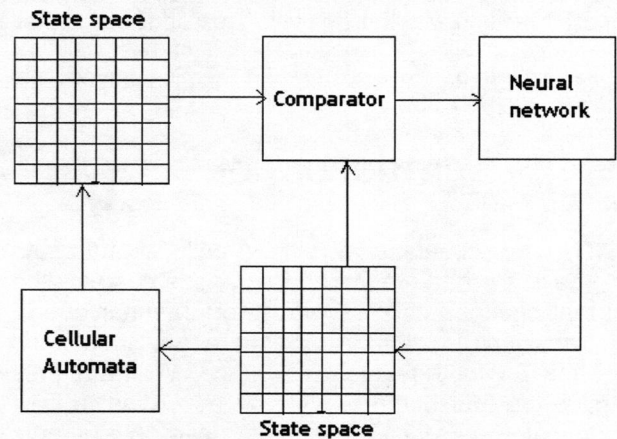

Fig.5.Interconnection network among all modules

In fig.5 cellular automata module has some architecture, which is shown in fig.6. This is the flow diagram of the cellular automata. Here next state is defined from the present state variables called neighbors.

Above flow chart is only for single cell. For all cells there may be large network. So we can construct that using neural network. Here we approach distributed and parallel method in simulation, because for large networks it is difficult to simulate in single system. This method of solving Max-Clique problem is efficient than all other previous parallel methods.

This method is very simple to analyze and implement. The proposed method was implemented in C++ on PENTIUM 4 (RAM 1024) for large number of graphs. It is peculiar method because any other method which using neural networks may calculate energy function. But here we can achieve final state by making all cells TRUE. Because of this there is no optimal solutions. Here we have to make more than one trails with same graph and input is in different permutation.
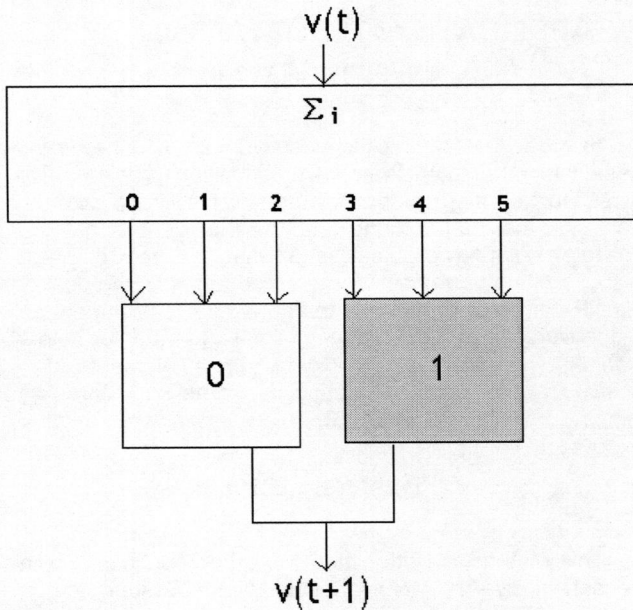
$v(t)$

Σ_i

| 0 | 1 | 2 | 3 | 4 | 5 |

| 0 | 1 |

$v(t+1)$

Fig.6.Flow chart of the cellular automata

ALGORITHM

Input: Adjacency matrix A [][]
Output: Maximum complete subgraph

Adjacency graph is the graph, which represents the adjacent elements of the graph with corresponding node.

Here we use + and ▬ instead of 1 and 0 respectively. We used adjacency matrix as described above as the state space of the cellular automata. The algorithm described below.

1.Pass adjacency matrix as CA state space
2.Change state space by CA rule, which we are using
3.Compare state spaces by using comparator
4.Send differences to the neural network
5.Update neural network
   (Here we using competitive learning)
6.The neural network gives information about new updated graph by which we can construct state space.
7.Repeat from step 2 until one of the termination conditions satisfies.

Here we described some rules, which are used to update state space.

Case (1):If changes are from only negative to positive
   A. Delete nodes which are more participated in the change for example if changes are occurred at (1,2), (3,4), (3,5)
      Then we can delete node 3 and share those edges with nodes 3 and 5.
   B. If there are single occurrences of pairs (x, y). Then combined them.

Case (2):If changes are from only positive to negative
   A. If ordered pair (x, y) occurred then take nodes x-1, x+1,y-1, y+1 which are negative and apply same rules as Case (1).
   B. In Case (1) first prior to the neighbors of the positive to negative changed nodes.

We used some heuristics to simplify the problem.
Heuristics:
   1.If diagonal elements are the only positive elements then we   leave it and solution is 0.
   2.We can remove which have no connection or least connections.

Here we used an example to describe the algorithm which shown in fig.7. The corresponding state space and steps to solve is given.
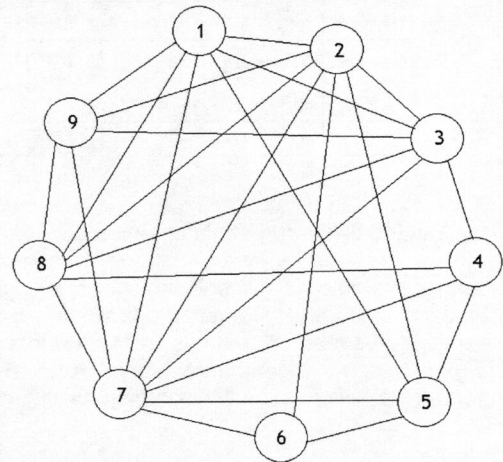

Fig.7.Example graph

+	+	+	—	+	—	+	+	+
+	+	+	—	+	+	+	+	+
+	+	+	+	—	—	+	+	+
—	—	+	+	+	—	+	+	—
+	+	—	+	+	+	+	—	—
—	+	—	—	+	+	+	—	—
+	+	+	+	+	+	+	+	+
+	+	+	+	—	—	+	+	+
+	+	+	—	—	—	+	+	+

Fig.8.State space of graph in Fig.7

After modified by cellular automata we got the state space given in fig.9. Then we observe the differences between the state spaces shown in fig.8 and fig.9. Differences are at (1,6), (2,4), (3,5), (4,6), which are

91

changed from negative to positive. Positive to negative changes are at (1,4), (1,6), (4,9).

The most occurrences according to Case (1) are node 6. So we share node 6 with nodes 1 and 4. In Case (2) changes (1,4) are there, we already combined 1 and 6, 4 and 6. So we combine nodes 1 and 4. Then there is unique node pair (3, 5) exists. So share 5 with 3,which adds nodes 3 and 5. Finally we got 6 nodes the updated graph is shown in fig.10.

+	+	+	−	⊝	⊕	+	+	+
+	+	+	⊕	+	+	+	+	+
+	+	+	+	⊕	−	+	+	+
−	⊕	+	+	+	⊕	+	+	−
⊝	+	⊕	+	+	+	+	−	−
⊕	+	−	⊕	+	+	+	−	−
+	+	+	+	+	+	+	+	+
+	+	+	+	−	−	+	+	+
+	+	+	−	−	−	+	+	+

Fig.9.Modified state space
(Circled cells represents changes occurred)

In above example we got solution with few updating. I worked graphs with up to 500 vertices in single processor. I observed that this method reduces more amounts of time and complexity. This algorithm is very useful to graph coloring problem, which is the NP-complete problem.
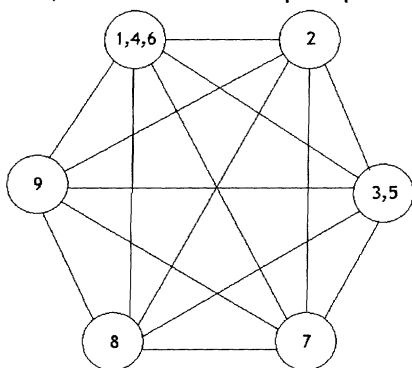

Fig.10.Updated graph

+	+	+	+	+	+
+	+	+	+	+	+
+	+	+	+	+	+
+	+	+	+	+	+
+	+	+	+	+	+
+	+	+	+	+	+

Fig.11. State space of the graph in Fig.10

APPLICATION TO TOTAL COLORING PROBLEM

In order to examine the effectiveness of our learning method to other combinatorial optimization problems, we tested our learning method on one of the graph coloring problems: the total coloring problem. For general graphs, the total coloring problem is NP-hard. Given a graph $G=(V,E)$ with a vertex set V and an edge set E, the goal of this NP-complete problem is to find a color assignment on all the vertices in V with the minimum number of colors such that no adjacent or incident pair of elements in V receives same color. Updating is same but here we combine and allocate a color.

CONCLUSION

This method is better than the other methods, which are solved by Ant Colony Optimization, Scatter search algorithm, k-optimal local search algorithm. In order to examine the effectiveness of our learning method to other combinatorial optimization problems, we have also tested our learning method on the total coloring problem. The simulation results showed that our learning method found optimal solution in every test graph. As far as the computation time required by the proposed learning method, although our simulation results verified that our learning method finds good solutions in short computation time, the analysis of the time complexity of the proposed algorithm should be an issue in the future works.

REFERENCES

[1] R. Garey and S. Johnson, Computers and Intractability, A Guide to the Theory of NP-Completeness. San Francisco, CA: Freeman, 1991.

[2] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer, "Some simplified NP-complete graph problem," Theor. Comput. Sci., vol. 1, pp. 237–267, 1976.

[3] R. M. Karp, "Reducibility among combinatorial problems," in Complexity of Computer Computations. New York: Plenum, 1972, pp. 85–104.

[4] S. Even and Y. Shiloach, "NP-completeness of several arrangement problems," Dept. Computer Science, Technion, Haifa, Israel, Tech. Rep. 43, 1975.

[5] J. J. Hopfield and D. W. Tank, "Neural computation of decisions in optimization problems," Biol. Cybern., no. 52, pp. 141–152, 1985.

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

[7] E.A. Akkoyunlu. The enumeration of maximal cliques of large graphs. SIAM Journal on Computing, 2(1):1-6, 1973.

[8] B. Bollobás and P. Erdös. Cliques in random graphs. Mathematical Proceedings of the Cambride Philosophical Society, 80:419-427, 1976.

[9] Coen Bron and Joep Kerbosch. Finding all cliques of an undirected graph. Communications of the ACM, 16(9):575-577, 1973.