NEURAL NETWORK MODELS FOR THE MAXIMUM CLIQUE PROBLEM

Roberto Cruz Rodés[1] and Nancy López Reyes[2]
[1]Instituto de Ciencias y Tecnología Nucleares
[2]CEMAFIT-ICIMAF
Universidad de Antioquia, Departamento de Matemáticas, Medellín, Colombia

ABSTRACT
In this paper we describe two neural network based algorithms for the Maximum Clique Problem. The developed algorithms provide discrete and continuos descent dynamics respectively to approximate the solution of the quadratic 0-1 formulation of the Maximum Clique Problem. The discrete approach performed better, maintaining computational competitiveness to greedy randomized search procedures. Experimental results on test graphs of size up to 3361 vertices and 5506380 edges are presented.

Key words: Maximum clique problem, heuristics, neural networks, quadratric 0-1 problem, combinatorial optimization.

RESUMEN

Se describen dos algoritmos basados en redes neuronales para el Problema del Clique Máximo de un grafo. Los algoritmos desarrollados implementan dinámicas descendentes, en un caso continua y en el otro discreta, para aproximar la solución del problema planteado a partir de la formulación cuadrática del mismo. El algoritmo discreto presenta un mejor desempeño, alcanzando resultados similares a los obtenidos con otras heurísticas. Se discuten los resultados de la aplicación de los algoritmos en un conjunto de grafos de hasta 3361 vértices y 5506380 aristas.

Palabras clave: Problema del clique máximo, heurística, redes neuronales, problema cuadrático 0-1, optimización combinatoria.

1. INTRODUCTION

This paper describes two algorithms based on a neural network approach for solving the Maximum Clique Problem (MCP). Let G = (V,E) be an undirected graph where V = {1,2,...,n} is the set of vertices in G,n = |V| is the size of the set V, and $E \subseteq V \times V$ is the set of edges in V. The adjacency matrix of G is denoted by $A_G = (a_{ij})_{n\times n}$, where $a_{ij} = 1$ if $(i,j) \in E$, and $a_{ij} = 0$ if $(i,j) \notin E$. The vertex degree of vertex k is denoted by $d_G(k)$. The complement graph of G = (V,E) is denoted by $\overline{G} = (V,\overline{E})$, where $\overline{E} = \{(i,j)/ij \in V, i \neq j, (i,j) \notin E\}$. The adjacency matrix of $\overline{G}$ is denote by $A_{\overline{G}} = (\overline{a}_{ij})_{n\times n}$. For a subset $S \subseteq V$ we call G(S) = (S,E $\cap$ S×S) the subgraph induced by S. A graph G = (V,E) is complete if and only if for all $i,j \in V$, $(i,j) \in E$. A clique C is a subset of V such that the induced subgraph G(C) is complete. A clique is maximal if no strict superset of it is a clique too. The MCP is the problem of finding a clique C of maximum cardinality in graph G. Denote the size of a maximum clique of graph G = (V,E) by $\omega(G)$.

All known exact algorithms for finding the maximum clique in a graph take exponential time in the worst case. This is not unexpected since the MCP is well known to be NP-hard (see Garey and Johnson [1979]). Hence, for practical purposes, these exact algorithms cannot solve very large problems. Since the problem is

[1]E-mail:rcruz@rsrch.isctn.edu.cu
[2]E-mail: nancylr@cidet.icmf.inf.cu
nlopez@matematicas.udea.edu.co

NP-hard even to approximate, in order to obtain polynomial time complexity one must use heuristics, that is, algoritms without performance guarantee.

Beginning with the initial paper of Hopfield [1982] the resolution of several Combinatorial Optimization Problems (COP) has been approached by the neural network dynamics (see Takefuji [1990]). The MCP is one of these COP which has been studied using this heuristic, due to its computational complexity. The Jagota´s work (Jagota [1995]) is a representative example of this. In this paper we describe two algorithms for solving a MCP, based on a Hopfield neural network whose stable states are identified with maximal cliques. One algorithm provides a discrete descent dynamics and the second one follows a continuos descent dynamics to approach the solution of the quadratic 0-1 formulation of the MCP, described in section 2.Since our objective is to find the energy global minimum in the configuration space, our strategy attempts to avoid local minima, which correspond to maximal but not necesarily maximum cliques.

2. QUADRATIC 0-1 FORMULATION

An unconstrained quadratic zero-one programming problem has the following form:

$\min f(x) = x^T Ax, \qquad x \in \{0,1\}^n;$

where A is an $n \times n$ matrix. The MCP can be formulated as a global quadratic zero-one problem. To facilitate our discussion, define for a graph G = (V,E) a transformation T from $\{0,1\}^n$ to 2^V,

$T(x) = \{i | x_i = 1, i \in V\}, \forall x \in \{0,1\}^n.$

Denote the inverse of T by T^{-1}. If $x = T^{-1}(S)$ for some $S \subset V$ then $x_i = 1$ if $i \in S$ and $x_i = 0$ if $i \notin S$, i = l,...,n. Pardalos and Xue [1994] formulated the MCP as a minimization problem, as stated in the following theorem:

Theorem 2.1: The MCP is equivalent to the following global quadratic zero-one problem:

$$\text{global min } f(x) = \frac{1}{2} x^T (A_{\bar{G}} - I)x, \qquad x \in \{0,1\}^n. \tag{1}$$

If x^* solves (1) then the set $C = T(x^*)$ is a maximum clique of G with $\omega(G) = |C| = -2f(x)$.

Next, we obtain some characterizations of the solution to the quadratic 0-1 problem and prove some relations among the maximal subgraphs of G and the discrete local minima of the corresponding function f(x) from (1). Pardalos and Desai [1991] obtained similar results for the maximum weighted independent set problem. A vector $x \in \{0,1\}^n$ is said to be a discrete local minimum (d.l.m.) of f(x) if and only if $f(x) \leq f(y)$, for any $y \in \{0,1\}^n$ adjacent to x.

LEMMA 2.1: If x´ is a d.l.m. of function f(x) then $x'_i x'_j = 0$ for all edges $(i,j) \notin E$.

Proof **.** Let x´ be a d.l.m. of function f(x) and suppose there is some pair $(i,j) \notin E$, such that $x'_i x'_j = 1$.

Let $x'' = x' - e_i$, where $e_i \in \{0,1\}^n$ is the ith unit vector. It is easily seen that:

$$f(x'') = \frac{1}{2}(x' - e_i)^T (A_{\bar{G}} - I)(x' - e_i) = f(x') + \frac{1}{2} - (A_{\bar{G}} x')_i \tag{2}$$

since $x'_j = 1$ and $\bar{a}_{ij} = 1$ then $(A_{\bar{G}} x')_i \geq 1$ and $f(x'') \leq f(x') - \frac{1}{2} < f(x')$. Hence, x´ is not a d.l.m., which is a contradiction.

While Theorem 2.1 relates the global minimum of the function f(x) to the solution of the MCP, the following Theorem proves a relationship among all discrete minima of f(x) and the maximal cliques of the graph G.

Theorem 2.2. x' is a d.l.m. of function f(x) if and only if the set $C' = T(x')$ is a maximal clique of graph G.

Proof. Let C' be a maximal clique of graph G and $x' = T^1(C')$. Let $y \in \{0,1\}^n$ be an adjacent vector to x', that is, they differ in only one coordinate, for example in the ith coordinate.

Case 1. $x'_i = 1$ and $y_i = 0$. That is, $y = x' - e_i$. Since $i \in C'$, then $\bar{a}_{ik} = 0$ for all $k \in C'$. Using (2) we obtain $f(y) = f(x') + ½ > f(x')$. Hence, x' is a d.l.m.

Case 2. $x'_i = 0$ and $y_i = 1$. That is, $y = x' + e_i$ and

$$f(y) = \frac{1}{2}(x' + e_i)^T (A_{\bar{G}} - I)(x' + e_i) = f(x') - \frac{1}{2}(A_{\bar{G}} x')_i \qquad (3)$$

Since $i \notin C'$, there is some $j \in C'$ such that $(i,j) \notin E$ and $\bar{a}_{ij} x'_j = 1$. Hence, $(A_{\bar{G}} x')_i \geq 1$ and $f(y) \geq f(x') + ½$, so that x' is a d.l.m.

Now let x' be a d.l.m. of f(x). From Lemma 2.1, $x'_i x'_j = 0$ for all edges $(i,j) \notin E$. Hence, $C' = T(x')$ represents a complete subgraph of G. If C' is not maximal then there is some y adjacent to x' such that for some i, $x'_i = 0$ and $y_i = 1$. Then $f(y) = f(x') - ½ < f(x')$. Hence, x' is not a d.l.m. of f(x), which is a contradiction. Thus, $C' = T(x')$ is a maximal clique of G.

Denote by $g \in Z^n$ the gradient of the function f(x), $x \in \{0.1\}^n$.

$$g_i = \nabla_i f(x) = \left[\left(A_{\bar{G}} - I\right) x\right]_i = \sum_{k \neq i} \bar{a}_{ik} x_k - x_i \,. \qquad (4)$$

Lemma 2.2*:* Let x be a vector such that $C = T(x)$ is a clique. Then the following conditions hold for C:

a) If $i \in C$ then $g_i = -1$

b) If $i \notin C$ and $g_i = 0$ then the vertex i is adjacent to every vertex of C and the superset $C \cup \{i\}$ is therefore a clique.

c) If $g_i \neq 0$ for all i = l,…,n then the clique C is maximal.

Proof

a) $x_i = 1$. Since C is a clique $\bar{a}_{ik} = 0$ for all $k \in C$. Hence, from (4) we obtain $g_i = - x_i = - 1$.

b) $x_i = 0$ and $g_i = 0$. From (4) we obtain $\bar{a}_{ik} = 0$ for all $k \in C$. Hence, the vertex i is adjacent to every vertex $k \in C$.

c) $x_i = 0$ and $g_i \neq 0$. From (4) we obtain $g_i > 0$ and at least one vertex of C is not adjacent to vertex i. Therefore, if for all i such that $x_i = 0$ we have $g_i \neq 0$ then there is not a vertex $i \notin C$ adjacent to every vertex of C. Hence, C is maximal.

3. NEURAL NETWORK APPROACH

The goal of the artificial neural network for solving COP is to minimize the fabricated computational energy function E. The energy function is constructed by considering all the constraints and/or the cost function from the given problem. The change of the network state at the time t is expressed by a differential equation system and the output of each neuron is given by the transfer function (Takefuji [1992]).

It was considered a neural network model with n neurons, where n is the number of vertices in graph G. Each neuron is enlaced with all resting. The differential equation system that expresses state of the network at the time is

$$\begin{cases} \dfrac{du_k}{dt} = -g_k = -\left[\left(A_{\overline{G}} - I\right)x\right]_k & k = 1,\cdots,n \\ x_k = F_T(u_k) \end{cases} \tag{5}$$

In this system $x \in \{0,1\}^n$ represents the vector state of neurons at determined time t. The vector $u \in R^n$ in the input vector to neuron units. The function vector g is the gradient vector of the function f(x) from (1) and $F_T(u_k)$ is the transfer function. Using a proper transfer function this system provides a parallel gradient descent method to minimize the energy function f(x) from (1). Applying first order Euler method to system (5) with $\Delta t = 1$, we obtain:

$$u_k^{t+1} = u_k^t - g_k^t \qquad k = 1,\ldots n \tag{6}$$

3.1. Discrete dinamics

Assume that the initial state of the system is $x^0 = e_i$, where e_i is ith unit vector. Since the initial state vector can be interpreted as a clique which contains only one vertex i, we can use as initial input vector the vector $u^0 = g^0$; which has zero components for all vertices adjacent to vertex i, and introduce a transfer function which adds only one vertex to a clique in each step until a clique becomes maximal . Using the results of Lemma 2.2, the transfer function can be defined as follows: one and only one neuron among all neurons with $g_k^t = 0$ and $u_k^{t+1} = 0$ is encouraged to fire in each step t+1. The neurons with input $u_k^{t+1} \neq 0$ are not changed. The input to the kth neuron is calculated using (6).

Lemma 3.1*:* From an initial state $x^0 = e_i$, the neural network converges to a maximal clique of G containing the vertex i in a number of iterations equal to or less than $\omega_i(G) \leq \omega(G) \leq n$, where $\omega_i(G)$ is the size of the maximum clique containing the vertex i.

Proof. Let $x^0 = e_i$ be an initial state and $C^0 = T(x^0)$ is a clique containing only the vertex i. Let's calculate g^t for each iteration t = 1,2,... For each t we select only one neuron to fire among all neurons with $g_k^t = 0$ and input $u_k^{t+1} = 0$. Once for $t = t_1$ we have $g_k^t = -1$, then from Lemma 2.2 we have that the vertex k was incorporated to a clique in the previous iteration $C^t = C^{t-1} \cup \{k\}$. Thus we obtain $u_k^{t+1} > 0$, for all $t > t_1$. On the other hand, once for $t = t_2$ we have $g_k^{t+1} > 0$ (the vertex k cannot be incorporated to the current clique), we obtain $u_k^{t+1} < 0$, for all $t < t_2$. Hence, if for $t = t_3$ we have $g_k^{t+1} \neq 0$ for all k = 1,...,n, then the network has reached the stable state x^t which is a maximal clique containing a vertex i. Since in each iteration we incorporate only one vertex to a clique, the number of iteration is equal to the size of the maximal clique found, which is equal to or less than $\omega_i(G)$.

As may be noticed, in each iteration we need to update only those inputs u_k^{t+1} which were equal to zero in the previous iteration and represent the candidate neurons to be fired in the current iteration. Suppose the mth neuron was fired in the previous calculation, then for all neurons such that $x_k^t = 0$ and $u_k^t = 0$ ($k \neq m$), the input value can be calculated using (6) as follows:

$$u_k^{t+1} = u_k^t - g_k^t = -g_k^t = -\sum_{j \neq k} \overline{a}_{kj} x_j^t = -\sum_{\substack{j \neq k \\ j \neq m}} \overline{a}_{kj} x_j^{t-1} - \overline{a}_{km} x_m^t = g_k^{t-1} - \overline{a}_{km} = -\overline{a}_{km} \tag{7}$$

Since finding large clique is our objective, we shall improve the process of selection of the neuron to be fired in order to find a stable state as large as possible. We shall select among candidate neurons that one

which guarantees the largest number of candidates in the next iteration. Let N^t be the number of candidates in the previous iteration and N_m^{i+1} the number of candidates if we chose x_m to be fired. Using (7) we have:

$$N_m^{t+1} = N^t + \sum_{k \neq m} u_k^{t+1} = N^t - \sum_{k \neq m} \bar{a}_{km} \tag{8}$$

where the index k runs only for those neurons such that $u_k^t = 0$. Hence, we must select the neuron, which has the maximum value of N_m^{t+1}. Using this modified transfer function and starting from initial state $x^0 = e_i$ we are trying to find the largest clique which contains the vertex i. It is described as follows:

---

1) $t \leftarrow 0$, $x^0 \leftarrow e_i$, $u^0 \leftarrow g^0$, $C^0 \leftarrow \{i\}$, $N^0 \leftarrow d_G(i)$.,
2) *while* $(N^t \neq 0)$ *do*
3) $N^{t+1} = 0$
4) *For* all k such that $u_k^t = 0$ *do* update u_k^{t+1} using (7)
5) *For* all m such that $u_m^{t+1} = 0$ *do* update N_m^{t+1} using (8)
6) select neuron l such that $N_l^{t+1} = \max_m \left(N_m^{t+1} \right)$
7) $x^{t+1} \leftarrow x^t + e_l$
8) $C^{t+1} \leftarrow C^t \cup \{l\}$
9) $N^{t+1} \leftarrow N_l^{t+1}$
10) $t \leftarrow t+1$
11) end

This algorithm, referred as discrete neural network algorithm (DNNA), provides a discrete descent dynamics to approximate the MCP in the subgraph $G_i \subseteq G$ that contains vertices, that are adjacent to the vertex i. In order to find the largest clique of graph G, the algorithm is used for each vertex, which does not belong to a clique found before and has vertex degree d_G greater than the size of the largest clique found.

3.2. Continous dynamics

In this case we used as initial input to the neural network the vector u^0, which components were randomly generated in the interval (-1, 1). The McCulloch Pitts function:

$$x_k = F_T(u_k) = \begin{cases} 1 & \text{if} \quad u_k \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

was used as a transfer function, which guarantees the convergence of the networks. The obtained iterative scheme was used until the state of neurons corresponds to a clique, not necessarily maximal. It can be notices that this network works over all the vertices of the graph, and once a neuronal state represents a clique it is only necessarily to update the states of neurons corresponding to the vertices that are adjacent to the current clique found. From condition a) of the Lemma 2.2, it can be seen that the state of neurons x^t correspond to a clique if and only if $A_{\bar{G}} x^t$ is a vector with zero components. As soon as this condition is met, we only update by the equation (6) the inputs to candidate neurons which correspond to adjacent vertices of the current clique, i.e inputs to neurons which hold the conditions $x_k^t = 0$ and $g_k^t = 0$. In the following iterations we use a maximum transfer function, it means we select among candidate neurons the one which has the greatest input. In this case the selection is different in comparison with the algorithm described above,

because we are not firing the neuron which guarantees the largest number of candidates in the next iteration. The neuron selected with this dynamics in each iteration depends of the initial state of the network, which is randomly generated, hence the obtained algorithm is not deterministic. It may be noticed that from the moment when the network reach a state corresponding to a clique, the Lemma 3.1 is hold.

In order to find the largest clique of graph G, we use this algorithm n times, one time for each vertex, trying to find in each trial a maximal clique which contains a current vertex. To achieve this we generated randomly the initial input vector and gave to corresponding component of the vector a great value. This algorithm is referred as continous neural network algorithm (CNNA).

In order to reduce computer time, when each one of the algorithms is attempting to find the maximum clique containing a particular vertex, the sum of the number of vertices already incorporated to a clique and the number of candidates is checked out in each iteration. If this value is less than the size of the largest clique previously found, the algorithm is interrupted without finding the maximal clique containing this vertex. In order to speed up the performance of the heuristics in large graphs, the neural network is used for a maximum of 10 vertices in graphs with density > 0.9, 100 vertices in graphs with density > 0.8 and 500 vertices in graphs with lower density.

4. EXPERIMENTS AND RESULTS

The codes were written in FORTRAN and compiled using the FL32 compiler. All experiments were conducted using a Pentium Power P5 133 MHz PC. We tested our algorithm on a total of 66 graphs ranging in size from 28 to 3361 vertices and up to 5506380 edges.

We use 9 different classes of DIMACS benchmark graphs for the MCP. These problems are from the Second DIMACS Implementation Challenge at the NSF Science and Technology Center in Discrete Mathematics and Theoretical Computer Science in 1993. In the following tables the column headed with t_{DNNA} and t_{CNNA} contains the CPU time in seconds for the DNNA and CNNA algorithms respectively. We compare our results with the sizes found by DMCLIQUE heuristic, that are the best-reported results on these benchmark graphs. We obtained the code of DMCLIQUE program from the DIMACS public access directories and compiled it using the VisualC 4.0 software. We ran a few experiments using DMCLIQUE and determined a time conversion factor of approximately 0.8, that is, the times are slightly faster on our machine. In the following discussion t_{DM} refers to the times reported in DIMACS file "results.dmclique" multiplied by 0.8. The terms speed-up refers to $\left\lfloor \frac{t_{DM}}{t_{DNNA}} \right\rfloor$. The size of retrieved maximum clique is also compared with the results of CBH (Gibbons *et al*. [1996]) on these benchmark graphs.

The CFAT graphs arise in the Fault Diagnosis Problem (Hasselberg *et al*. [1993]). DNNA and CNNA found the maximum clique in all seven instances. For `c-fat500-10` both algorithm found a clique with size of 126 which matches the largest clique found by DMCLIQUE. For all seven instances $t_{DM} > t_{DNNA}$, the speed up ranged from a factor of 2 to a factor of 298 in the case of `c-fat500-1`. DNNA and CNNA found the maximum clique in all four instances of Johnson graphs (Hasselberg **et al** [1993]), and the algorithm DNNA performed with speed-up of 12, 3, 22 and 58 respectively. The performance of the DNNA and CNNA algorithms were similar to CBH on these benchmark graphs.

The Keller graphs in Table 2 arise in connection with Keller´s conjecture on tilling using hypercubes (Hasselberg **et al** [1993]). DNNA found the maximum cliques in two of the three instances with speed-up to 18, 2 and 12. In 41766.40 seconds DMCLIQUE was able to find cliques with sizes larger than 51 in 78 out of 1000 runs that were performed on `keller6`, including a clique of size 55 which was found exactly once. The algorithm CNNA was only applied to the `keller4` instance and succeeded in finding the maximum clique. The performance of DNNA and CNNA algorithms on Hamming graphs, arising in Coding Theory Problems (Hasselberg **et al** [1993]), is also presented in Table 2. DNNA found the maximum cliques in five of the six instances and in all cases $t_{DM} > t_{DNNA}$ with speed-up ranging from 2 to 136. However, DNNA could only find a clique of size 32 in `hamming10-4`.DMCLIQUE found larger cliques, including a clique with a size of 40 that

was found in 1 out of the 1000 runs performed on this graph. CBH also found a larger clique of size 35 for this instance. The algorithm CNNA performed worst, finding the maximum clique only in two instances.

Table 1. The CFAT and Johnson benchmark graphs.

Graph	Nodes	Edges	DNNA	t_{DNNA}	CNNA	t_{CNNA}	MaxClique
c-fat200-1	200	1534	12	0.1	12	22.7	12
c-fat200-2	200	3235	24	0.3	24	27.1	24
c-fat200-5	200	8473	58	3.2	58	47.3	58
c-fat500-1	500	4459	14	0.2	14	385.1	14
c-fat500-10	500	46627	126	140.5	126	997.8	≥ 126
c-fat500-2	500	9139	26	0.7	26	420.0	26
c-fat500-5	500	23191	64	20.9	64	517.9	64
johnson16-2-4	120	5460	8	0.6	8	15.4	8
johnson32-2-4	496	107880	16	24.3	16	2604.9	16
johnson8-2-4	28	210	4	0.1	4	0.1	4
johnson8-4-4	70	1855	14	0.1	14	2.8	14

Table 2. The Keller and Hamming benchmark graphs.

Graph	Nodes	Edges	DNNA	t_{DNNA}	CNNA	t_{CNNA}	MaxClique
Keller4	171	9435	11	0.7	11	49.5	11
Keller5	776	225990	27	245.8	-	-	27
Keller6	3361	4619898	51	3322.0	-	-	≥ 59
hamming10-2	1024	518656	512	99.1	-	-	512
hamming10-4	1024	434176	32	129.7	-	-	≥40
hamming6-2	64	1824	32	0.01	32	2.2	32
hamming6-4	64	704	4	0.1	4	1.0	4
hamming8-2	256	31616	128	1.2	97	299.8	128
hamming8-4	256	20864	16	1.2	13	166.5	16

The graphs in Table 3 were contributed by L. Sanchis [1992]. DNNA found maximum cliques in seven of the eleven instances of Sanchis graphs while CNNA failed in all instances, For `san200_0.7_2`, `san200_0.9_2` and `san400.0.7_3` DMCLIQUE found larger cliques in 11, 37 and 6 out of 1000 runs respectively. In the case of `san400_0.9_3` DMCLIQUE did not find a maximum clique, finding a clique of 42 only in 2 out of 1000 runs. For all the eleven instances DNNA found cliques larger than CBH. In the case of random Sanchis benchmark graphs, DNNA found a maximum clique only in one of the four instances and CNNA failed in all the four instances. DMCLIQUE found cliques of size 18, 41, 12 and 21 and CBH found cliques of size 18, 41, 12 and 20 respectively. In all eleven cases of Sanchis graphs and 4 instances of random Sanchis graphs, $t_{DM} > t_{DNNA}$. With speed-ups ranging from 2 to 38. The performance of DNNA on these instances was better than CBH, but the algorithm CNNA performed very poorly, finding stable states very far from the global optimum in several instances.

The DNNA algorithm performed quite similar to CBH on Brockington graphs (Brockington and Culberson [1993]) in Table 4. Except for `brock200_1`, DNNA was unable to find any maximum clique. For `brock800_1` DNNA found a clique of size 21, which matches the largest clique found by DMCLIQUE in 2 out

of 1000 runs performed on this graps. For all other 10 instances DMCLIQUE found cliques larger in one vertex, than those found by DNNA. In all twuelve cases $t_{DM} > t_{DNNA}$ with speed-ups ranging from 2 to 78.

Table 3. The Sanchis and random Sanchis benchmark graphs.

Graph	Nodes	Edges	DNNA	t_{DNNA}	CNNA	t_{CNNA}	MaxClique
san1000	1000	25050	15	124.2	-	-	15
san200_0.7_1	200	13930	30	3.6	17	111.7	30
san200_0.7_2	200	13930	15	2.3	14	109.9	18
san200_0.9_1	200	17910	70	12.4	53	185.8	70
san200_0.9_2	200	17910	53	6.7	42	128.4	60
san200_0.9_3	200	17910	40	7.6	33	135.6	44
san400_0.5_1	400	39900	13	11.1	8	889.7	13
san400_0.7_1	400	55860	40	43.2	22	1448.9	40
san400_0.7_2	400	55860	30	33.2	18	1248.3	30
san400_0.7_3	400	55860	17	21.6	15	1147.2	22
san400_0.9_1	400	71820	100	48.7	55	2101.7	100
sanr200_0.7	200	13868	17	2.0	16	92.5	18
sanr200_0.9	200	17863	41	8.4	37	135.7	≥ 42
sanr400_0.5	400	39984	13	7.3	11	811.1	13
sanr400_0.7	400	55869	20	13.3	18	1112.6	≥ 21

Table 4. The Brockington benchmark graphs.

Graph	Nodes	Edges	DNNA	t_{DNAA}	CNNA	t_{CNAA}	MaxClique
brock200_1	200	14834	21	3.1	19	100.1	21
brock200_2	200	9876	11	0.6	10	70.1	12
brock200_3	200	12048	13	0.9	12	79.4	15
brock200_4	200	13089	16	1.5	15	85.6	17
brock400_1	400	59723	23	21.5	21	1216.1	27
brock400_2	400	59786	24	30.0	21	1202.7	29
brock400_3	400	59681	24	22.6	21	1202.3	31
brock400_4	400	59765	24	28.4	23	1158.5	33
brock800_1	800	207505	21	108.4	-	-	23
brock800_2	800	208166	20	111.1	-	-	24
brock800_3	800	207333	20	109.6	-	-	25
brock800_4	800	207643	20	103.7	-	-	26

The Mannino graphs in Table 5 are obtained from Set Covering Problem. DNNA and CNNA found maximum clique only for `mann_a9` graph. For `mann_a27` DNNA found a clique with size 125, that is, a clique smaller in one vertex than the maximum one. In 16362.04 seconds DMCLIQUE found cliques of size larger than 342 in 70 out of 1000 runs performed on the graph `mann_a45`, including a largest clique of size 344 which was found exactly once. On these two graphs CBH found cliques of size 121 and 336 respectively. In

the case of mann_a81, DMCLIQUE found in 26865.24 seconds a clique of size 1097 in 10 out of 1000 runs performed on this graphs. In all cases $t_{DM} > t_{DNNA}$ with speed.ups in favor of DNNA of 27, 36, 2 and 150.

Tabla 5. The Mannino benchmark graphs.

Graph	Nodes	Edges	DNNA	t_{DNAA}	CNNA	t_{CNAA}	MaxClique
MANN_a27	378	70551	125	19.2	120	2488.5	126
MANN_a45	1035	533115	342	444.4	-	-	345
MANN_a81	3321	5506380	1096	18303.9	-	-	≥ 1100
MANN_a9	45	918	16	0.1	16	0.8	16

The DNNA and CNNA results on PHAT graph (Soriano and Gendreau [1993]) are presented in Table 6. DNNA found maximum cliques in six of the fifteen instances and CNNA found the maximum clique only in one out of the six instances with 500 or less vertices. For `p_hat500-3` DNNA found a clique of size 49 which matches the largest size found by CBH and also by DMCLIQUE in 4 out of 1000 runs. DMCLIQUE found cliques larger in one vertex than those found by DNNA for graphs `p_hat300-3`, `p_hat700-1`, `p_hat700-3` and `p_hat100-2`. In the case of `p_hat1500-2` and `p_hat1500-3`, DMCLIQUE found cliques of size 64 and 94 respectively. In all cases $t_{DM} > t_{DnNA}$, with speed-ups ranging from 2 to 46. On this category of graphs CBH found larger cliques than those found by DNNA in six instances and cliques with less size in 2 instances.

Tabla 6. The PHAT benchmark graphs.

Graph	Nodes	Edges	DNNA	t_{DNNA}	CNNA	t_{CNNA}	MaxClique
p_hat300-1	300	10933	8	0.6	7	253.3	8
p_hat300-2	300	21928	25	4.5	24	440.7	25
p_hat300-3	300	33390	35	18.3	32	563.3	36
p_hat500-1	500	31569	9	3.1	9	1474.7	9
p_hat500-2	500	62946	36	33.1	32	2971.0	36
p_hat500-3	500	93800	49	115.0	45	3536.9	≥ 49
p_hat700-1	700	60999	10	7.4	-	-	11
p_hat700-2	700	121728	44	85.5	-	-	44
p_hat700-3	700	183010	61	328.2	-	-	≥ 62
p_hat1000-1	1000	122253	10	15.6	-	-	10
p_hat1000-2	1000	244799	45	170.8	-	-	≥ 46
p_hat1000-3	1000	371746	64	661.1	-	-	≥ 65
p_hat1500-1	1500	284923	11	38.6	-	-	12
p_hat1500-2	1500	568960	62	489.9	-	-	≥ 64
p_hat1500-3	1500	847244	90	1693.6	-	-	≥ 94

The algorithm CNNA was only applied to instances with up to 500 vertices, because this algorithm performed very slowly for graphs with a great number of vertices.

5. CONCLUDING REMARKS

We presented two neural network algorithms for the MCP. The discrete neural network algorithm (DNNA) performed well on many instance categories of benchmark graphs. In 34 of the 66 DNNA found cliques with equal size to the largest cliques found by DMCLIQUE. In 54 instances DNNA found cliques of equal or larger size than those found by CBH. The performance of the algorithm could be improved changing the selection scheme of the vertices to be used as initial input of the neural network. The continuos neural network algorithm (CNNA) performed worst than DNNA in almost all cases, finding maximal clique with less size than the cliques found by DNNA. It is due to the use in the DNNA the transfer function which selects among candidate neurons, the neuron which guarantees the greater number of candidates in the next iteration. On

the other hand, the CNNA transfer function selects among candidate neurons, the neuron with the maximal input and this value depends on the initial state of the networks, which was set randomly. The continuos algorithm also performed slower than DNNA, because the discrete algorithm starts with the state which is a clique of size 1 and only update the inputs to units corresponding to candidate neurons, while the continuos algorithms update all the imputs to units, until a clique is found.

ACKNOWLEDGMENTS

This work has been supported by ICTP-TWAS under the Research Project Grant No. 95-344 RG/MATHS/LA and by Antioquia University under the Research Project "Neural Networks Development and Applications".

REFERENCES

BROCKINGTON, L. and J. CULBERSON (1993): "Camouflaging Independent Sets in Quasi-random Graphs", Working paper, **Second DIMACS Implementation Challenge**.

GAREY M.R. and D.S. JOHNSON (1979): "Computers and Intractability: A Guide to the Theory of NP-Completeness", Freeman, San Francisco.

GIBBONS, L.E.; D.W. HEARN and P. PARDALOS (1996): "A Continuos Based Heuristic for the Maximum Clique Problem", **DIMACS Series in Discrete Mathematics and Theoretical Computer Science** 26, 103-124.

HASSELBERG, J.; P.M. PARDALOS and G. VARIAKTARAKIS (1993): "Test Case Generator and Computational Results for the Maximum Clique Problem", **Journal of Global Optimizaction**, 3, 463-482.

HOPFIELD, J.J. (1982): "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", **Proc. Nat. Academy Sci.** 79.

JAGOTA, A: (1995): "Approximating Maximum Clique with a Hopfield Network", **IEEE Trans. Neural Networks** 6, 724-735.

PARDALOS, P.M. and N. DESAI (1991): "An Algorithm for Finding a Maximum Weighted Independent set in an Arbitrary Graph", **Intern. J. Computer Math.** 38, 163-175.

_____ and J. XUE (1994): "The Maximum Clique Problem", **Journal of Global Optimization**, 4, 301-328.

SANCHIS, L. (1992): "Test Case Construction for the Vertex Cover Problem", Working paper, **DIMACS**.

SORIANO, P. and M. GENDREAU (1993): "Solving the Maximum Clique Problem using a Tabu Search Approach", **Annals of Operations Research** 41, 385-403.

TAKEFUJI, Y. (1992): **Neural Network Parallel Computing**. KLUWER Acad. Pub. N. York.