Complex Systems **5** (1991) 509–524

Design and Analysis of Competition-Based Neural Networks

Tao Li
Department of Computer Science, Concordia University,
1455 de Maisonneuve Blvd. W., Montreal, Quebec H3G 1M8, Canada

Yun Peng
Department of Computer Science, University of Maryland Baltimore County,
Baltimore, MD 21228, USA

Abstract. A simple and systematic approach to the design of neural networks for combinatorial optimization is presented in this paper. This approach is based on competition among the neurons of a network. Our approach relies on the use of simple heuristics in network design. The activation rule governing neuron behavior is derived by breaking down the global constraints for a problem into local constraints for individual neurons. The local constraints can be expressed as simple heuristic rules for network design. Using this approach, high performance neural networks for optimization can be readily designed and the global properties can be easily verified. Simulation results indicate that this approach can outperform some sequential heuristic algorithms and simulated annealing algorithms.

1. Introduction

Analog neural networks have been used to solve many combinatorial optimizations problems. Since the pioneering work of Hopfield and Tank [7, 8, 9, 10, 23], neural networks have been applied to many combinatorial problems. Some examples can be found in [1, 13, 16, 18, 19, 26]. Neural network models that adopt probabilistic choice or added noise are also used for optimization. The Boltzmann machine [2], the Gaussian machine [3], and the Cauchy machine [22] are some examples.

In the popular approach to optimization neural network design, whether deterministic or probabilistic, an energy function is first specified for a given problem. This energy function must combine the optimization objective and the problem constraints. The constraints usually appear as soft constraints in the energy function. Equations of motion for neurons are then derived from the energy function. Although elegant, this approach is not always practical.

It is difficult to find the energy functions of certain problems for which the constraints are not readily expressed in some form of soft constraints in the energy functions.

In our research, we have adopted a heuristic approach to the design of neural networks for optimization. This is the *competition-based* neural network approach [17, 20, 21, 24]. In this approach, neurons are allowed to compete to become active under certain constraints. The activation rules are derived from simple heuristics. This approach has produced very competitive results as compared to some sequential approximation algorithms.

We aim at developing neural networks in which neurons compete to become active. The formulation of equations of motion in our approach is often not the same as those in [17, 20, 21, 24]. Hence, we shall use the name *competition-based neural network* throughout the paper. Our design methodology is general enough to be applicable to a large class of problems and simple enough to enable many engineers to quickly design neural networks for optimization problems.

Our approach consists of five stages:

1. the selection of representation topology,
2. the design of equations of motion,
3. preliminary simulation,
4. proof of network properties, and
5. performance study with simulation.

Stages 2 and 3 may be repeated several times until a satisfactory system of equations of motion is found. Although the neural network is based on local heuristic representation, it also exhibits globally verifiable behaviors such as the convergence property and the guarantee of global constraint satisfaction. This approach has been applied to solve many optimization problems, some simple and some difficult.

This paper is organized into six sections. This section is an introduction. Section 2 discusses the principle of neural network construction for various problems. A method for verifying stability is given in section 3. To demonstrate the feasibility of our approach, a number of example systems will be discussed. Section 4 studies the application of competition-based neural networks for solving the set-covering problem, a problem in the NP class. Another example, the design of a neural network for generalized assignment, is discussed in section 5. These example systems demonstrate the high performance of neural networks designed with our approach. Section 6 contains a summary of our research.

2. Neural network construction

In designing a neural network for combinatorial optimization, one of the important issues is the choice of network topology for representing the solution of a problem. We normally want to adhere to the following principles.

1. **Completeness**. A representation must encode all possible solutions of a problem.
2. **Ease of Design**. This means that a representation should facilitate the design of a neural network for the optimization problem.
3. **Economy**. One should not use too many redundant neurons and connections.

Example 1. *0/1 Knapsack Optimization.* In this problem a knapsack of capacity M and a set of objects each having a size w_i and a profit P_i are given. The purpose is to put the objects into the knapsack so as to maximize the total profit. For this problem every object is represented as a variable and is associated with a neuron. The variables may assume either 0 or 1 and the activation values of the neurons may assume any value in the interval $[0, 1]$. If the object is placed in the knapsack, the corresponding neuron will have a value 1. Otherwise the neuron will have a value 0.

The second and perhaps the most important step in network design is the formulation of equations of motion. Some basic techniques for formulating equations of motion include direct mapping, simulation of sequential heuristic algorithms, and analysis of approximate energy functions. An example of the direct-mapping technique can be found in the design of the linear programming neural net of Tank and Hopfield [23]. This section concentrates on the second technique, the simulation of sequential heuristics. Since we are interested in the design of competition-based networks, we first formalize the concept of competitors in such networks.

Definition 1. *Two neurons in a network are competitors if and only if* $\partial \dot{a}_i / \partial a_k \leq 0$ *during the network evolution.*

This definition applies if the neural network is defined as a dynamical system. The definition is slightly more general than Grossberg's concept of competitive networks in which the neurons adopt the on-center–off-surround competitive geometry. It captures the idea that two nodes in a network are in competition if the gain of one occurs at the expense of the other. This definition does not require explicit inhibitory links for the network to be competitive.

A simple form of constraint, 1-out-of-n, is frequently encountered in many discrete optimization problems. We use the following equation of motion for *1-out-of-n constraint* with cost coefficients:

$$\frac{dx_i}{dt} = \left[(1 - \epsilon) - \sum_{j \neq i}^{n} x_j \right] c_i, \tag{2.1}$$

where ϵ is a very small constant near 0 and the c_is are cost coefficients.

It is important to observe that all neurons start from the value 0. Their activation may change later. The larger c_i is for the neuron, the faster increase of activation it will have. If the c_i are all distinct, then their activations

will increase at different speeds. As soon as the sum of the activations approaches $1 - \epsilon$, dx_i/dt for some neurons will become negative; hence their activations will decrease. Since the neuron k with the largest c_k will increase faster than others, it will force other neurons to decrease while it maintains its increase of activation. It would not be difficult to show this by analyzing equation (2.1). Notice that $\sum_{j\neq k}^{n} x_j < \sum_{j\neq i}^{n} x_j$ for all $i \neq k$ since $x_k > x_i$. At the point when all other dx_i/dt turn negative, x_k will still be positive. Hence its activation will increase, and this will further decrease the activation of other neurons. Once the system passed the turning point, all its neurons will keep moving in the same direction until it approaches equilibrium. This is true because all the neurons start from value 0. Hence we have the following theorem.

Theorem 1. *The equation of motion defined above guarantees to enter a global stable state in each run, and the global stable state entered must have one variable assuming the value 1 and the rest assuming 0 if the c_is are all different and if the variables are initialized to 0.*

Another useful form of constraints is the *weighted constraint*: $\sum_{i=1}^{n} x_i w_i \leq M$, where w_i $(1 \leq i \leq n)$ are the weights and M is the capacity. The variables must take on digital values (either 0 or 1) in a valid solution. If, in addition, each variable x_i is associated with a profit P_i, this becomes the well-known *knapsack* problem. The equations of motion are described below:

$$\frac{dx_k}{dt} = \left(M + \epsilon - \sum_{i\neq k} w_i f(x_i) - w_k \right) p_k, \qquad k = 1, 2, \ldots, n \tag{2.2}$$

where $p_k = P_k/w_k$ is the unit profit of x_i, and

$$f(x) = \begin{cases} 0, & \text{if } x < \theta \\ x, & \text{if } x \geq \theta \end{cases}$$

where θ is a threshold value near 1 (typically 0.98), $0 < \epsilon < 1$, and ϵ is less than the minimum difference among all pairs of weights. If the weights are integers, it is convenient to choose $\epsilon = 0.5$. The competition is manifested in the factor p_k for the above equation.

Theorem 2. *Assuming that θ can be adjusted, the above equations of motion always produce stable digital solutions satisfying the constraint $\sum_{i=1}^{n} x_i w_i \leq M$ if each p_i is different and if the variables (neuron activations) are all initialized to 0.*

The proof of the above theorem is similar to Theorem 1. It can be done with an analysis of equation (2.2), but in order not to confuse it with the main subject, details are left out here. The following examples illustrate the formulation of equations using the technique of simulating sequential heuristics.

Example 2. *0/1 Knapsack.* If we modify equation (2.2) into the following form,

$$\frac{dx_k}{dt} = \Phi\left(\left(M - \sum_{i \neq k} w_i f(x_i) - w_k\right), p_k, dec\right), \quad k = 1, 2, \ldots, n \tag{2.3}$$

where Φ is defined as

$$\Phi(x, p, d) = \begin{cases} p, & \text{if } x \geq \theta \\ -d, & \text{if } x < \theta \end{cases}$$

we have a network that always produce the same result as the sequential algorithm for the 0/1 knapsack in [11]. We now give an intuitive explanation. A sequential algorithm would typically sort the objects by p_k into nonincreasing order and then place the objects in the knapsack following that order until the knapsack can no longer hold any other object. The equation of motion given previously simulates that effect. If neuron k has a higher p_k value, its activation will increase more quickly because of the factor p_k in its equation of motion. Therefore, a neuron with a higher p_k factor is more likely to be placed in the knapsack. On the other hand, if the object is too large to fit in the knapsack, the factor $M - \sum_{i \neq k} w_i f(x_i) - w_k$ will become negative and will thus reduce the activation of the neuron (not to place it in the knapsack).

Example 3. *Quadratic Assignment.* This problem is to optimally locate m *processing plants* at n possible sites. Each plant is to be located at one site and only one plant can be located at each *site.* The cost of transporting 1 unit of goods from site i to j is c_{ij}. The amount of goods to be transported from plant k to plant l is d_{kl}. The objective is to minimize the cost $f(x)$ of transporting goods among the plants by assigning the plants to the proper sites, where

$$f(x) = \sum_{i=1}^{n} \sum_{j \neq i} \sum_{k=1}^{m} \sum_{l \neq k} c_{i,j} d_{k,l} x_{i,k} x_{j,l}.$$

Since there are m plants and n sites, a matrix $\mathbf{X}$ of dimension $m \times n$ can be formed to represent a solution. If plant k is to be located at site i, $x_{i,k} = 1$. Otherwise, $x_{i,k} = 0$.

Since the task is to assign m plants to n sites, we choose to use a two-dimensional array of neurons. The array consists of $m \times n$ neurons. In a solution, if a neuron $x_{i,j}$ assumes the value 1, the plant j is assigned to site i. Two constraints which must be satisfied in a feasible solution are

$$\sum_{i=1}^{n} x_{i,j} = 1, \quad \text{for } j = 1, 2, \ldots, m \tag{2.4}$$

and

$$\sum_{j=1}^{m} x_{i,j} \leq 1, \quad \text{for } i = 1, 2, \ldots, n \tag{2.5}$$

Since the task is to minimize the cost of assignment $\sum_{i=1}^{n}\sum_{j\neq i}\sum_{k=1}^{m}\sum_{l\neq k} c_{i,j}d_{k,l}x_{i,k}x_{j,l}$, it is natural to consider $1/(c_{i,j}d_{k,l})$, the reciprocal of the link cost, as a coefficient for competition. This means, if a node has many links with high costs, it will compete weakly; and if a node has many links with low costs, it will compete strongly. Hence, a node with many low-cost links is likely to be selected in an assignment. Let $c_{\min} = \min\{c_{i,j}\}$ and $d_{\min} = \min\{d_{k,l}\}$. From this consideration, we naturally arrive at the following equation of motion for a neuron $x_{a,b}$:

$$\frac{dx_{a,b}}{dt} = \Phi\left(U\left(1-\sum_{j=1}^{m}x_{a,j}^{p}\right), U\left(1-\sum_{i=1}^{n}x_{i,b}^{p}\right), \alpha\sum_{i\neq a}\sum_{j\neq b}x_{i,j}\left(\frac{1}{c_{a,i}d_{b,j}}\right)\right) \tag{2.6}$$

where $\alpha \geq c_{\min}d_{\min}$ is a constant, U is the step function defined by

$$U(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{if } x < 0 \end{cases}$$

and Φ is defined as

$$\Phi(A,B,C) = C \times \operatorname{sgn}(AB + (A-1)B + A(B-1)) \tag{2.7}$$

The choice $\alpha \geq c_{\min}d_{\min}$ is such that $\alpha\sum_{i\neq a}\sum_{j\neq b}x_{i,j}(1/c_{a,i}d_{b,j})$ is not too small. As long as $\alpha \geq c_{\min}d_{\min}$, the algorithm will work well. The system is not sensitive to the choice of α.

3. Proving the convergence property

To prove the convergence property, we use the Lyapunov function approach as in [5, 8]. There are numerous texts on ordinary differential equations and on the design of automatic control systems that provide good discussion of Lyapunov stability theory. However, we cannot find one that really suits our purpose here. Our goal is to find a heuristic approach to derive Lyapunov functions for a dynamical system, so we have developed our own heuristics for solving this problem. An energy function is *good* if it is monotonically increasing (or decreasing) in time before an equilibrium state is reached.

Although we start from a *local* representation in the design of an optimization neural network, we expect the network to converge to a globally stable state that represents a feasible solution. A few heuristics for constructing good energy functions from equations of motion are developed here.

Several forms of terms commonly encountered in equations of motion and their corresponding forms in the energy function are:

1. Constant term C. An energy function term is $\sum_{i=1}^{n} x_i C$ for a one-dimensional representation and $\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}C$ for a two-dimensional representation.

2. Indexed constant β_k. The corresponding energy function term should be $\sum_{i=1}^{n} x_i\beta_i$ for a one-dimensional representation and $\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}\beta_{ij}$ for a two-dimensional representation.

3. Single variable term $x_k w_k$. This typically has the energy function term $\sum_{i=1}^{n} w_i \int_0^{x_i} \varphi_i d\varphi_i$.

4. Single sum $\sum_{i=1}^{n} x_i w_{ik}$. If $w_{ik} = w_{ki}$, then the corresponding energy function term is $\sum_{k=1}^{m}\sum_{i=1}^{n} x_i x_k w_{ik}$. If the term is $\sum_{i=1}^{n} x_i$, the energy function term is $(\sum_{i=1}^{n} x_i)^2$.

5. Single sum with x_i to the pth power $\sum_{i=1}^{n} x_i^p w_i$. A precise energy function term is difficult to obtain. Instead the term $(1/2p)(\sum_{i=1}^{n} x_i^p w_i)^2$ is used in energy functions.

6. Sum within function $\sum_{j=1}^{m} D_{jk} f(\sum_{i=1}^{n} D_{ji}x_i - B_j)$. Assume the existence of $F(z)$ such that $f(z) = dF(z)/dz$. $\sum_{j=1}^{m} F(\sum_{i=1}^{n} D_{ji}x_i - B_j)$ is the energy function term since $\sum_{k=1}^{m} \partial/\partial x_k \sum_{j=1}^{m} F(\sum_{i=1}^{n} D_{ji}x_i - B_j) = \sum_{j=1}^{m} D_{jk} f(\sum_{i=1}^{n} D_{ji}x_i - B_j)$.

7. Multiple sum $A\sum_{a=1}^{n}\sum_{b=1}^{m}[\sum_{j\neq a}\sum_{l\neq b} C_{a,j}D_{b,l}x_{j,l}]$. The corresponding energy function term is of the form $\sum_{i=1}^{n}\sum_{j\neq i}\sum_{k=1}^{m}\sum_{l\neq k} C_{i,j}D_{k,l}x_{i,k}x_{j,l}$ if the coefficients $c_{i,j}$ and $d_{k,l}$ are symmetric. In particular, $A\sum_{i=1}^{n}\sum_{j\neq i}\sum_{k=1}^{m}\sum_{l\neq k} x_{i,k}x_{j,l}$ is the energy function term for $A\sum_{a=1}^{n}\sum_{b=1}^{m}[\sum_{j\neq a}\sum_{l\neq b} x_{j,l}]$.

To find a good energy function for a set of equations of motion, one would follow these steps. First, use the distributive law to completely separate an equation of motion into atomic terms (constants, single variables, and simple sums). Next, for each term, derive the corresponding energy function term according to the previous list or by some symbolic manipulation mathematical knowledge. Finally, if all the energy function terms match, then adding these terms would result in a good energy function. If some do not match the rest, then adjust these terms by adding appropriate variables and constants so they can match the others.

For example, consider the equation of motion $dx_k/dt = (p - \sum_{i=1}^{n} x_i^3 + (x_k - 1))w_k$. The indexed constant w_k can be ignored initially. The energy function term for p is $p\sum_{i=1}^{n} x_i$. The energy function term to $x_k - 1$ is $\sum_{i=1}^{n} x_i^2/2 - \sum_{i=1}^{n} x_i$ (this is obtained from the list of common terms). The energy function term for $\sum_{i=1}^{n} x_i^3$ is $\frac{1}{6}(\sum_{i=1}^{n} x_i^3)^2$. Differentiating this last term with respect to x_k yields $x_k^2\sum_{i=1}^{n} x_i^3$. This term does not match the others because of the factor x_k^2, so the other terms must be adjusted to match the third term. The adjustments require that x_i^2 and a constant be multiplied with each of the other terms. The resulting energy function is $E = p\sum_{i=1}^{n} x_i x_i^2/3 + \sum_{i=1}^{n}(x_i^2 x_i^2/4 - x_i x_i^2/3) - \frac{1}{6}(\sum_{i=1}^{n} x_i^3)^2$. Differentiating this energy function yields

$$\frac{dE}{dt} = \sum_{k=1}^{n} x_k^2 \frac{1}{w_k}\left[w_k\left(p - \sum_{i=1}^{n} x_i^3 + (x_i - 1)\right)\right]\frac{dx_k}{dt} = \sum_{k=1}^{n}\frac{1}{w_k}\left(x_k\frac{dx_k}{dt}\right)^2.$$

This energy function is good if all w_k $(1 \leq k \leq n)$ are positive.

4. Neural network for vertex covering

The vertex-covering problem (VCP) is a problem of finding a set of minimum number of vertices that covers all edges in a given graph. A number of approximation algorithms for the VCP were proposed to reduce the complexity of the problem [12]. Let $G = \langle V, E \rangle$ be an undirected graph, where $V = \{v_1, v_2, ..., v_n\}$ is the set of vertices and $E \subseteq \{(v_i, v_j) | v_i, v_j \in V\}$ is the set of undirected edges. A minimum vertex cover of graph $G = \langle V, E \rangle$ is a subset of vertices $V' \subseteq V$ such that (1) for each edge $e_k = (v_i, v_j)$, either v_i or v_j is in V', and (2) among all covers of E, V' has the smallest cardinality (number nodes).

4.1 VCP network design

It is natural to adopt the network topology in which each neuron represents a node of G and each connection represents an edge of G. This representation is non-redundant. To form the equations of motion, the following constraints must be taken into account.

1. Two adjacent nodes compete with each other to cover an edge, so the higher the activation of one node, the less support the other should receive from that node.
2. At least one of the two adjacent nodes to an edge should be fully activated at equilibrium to be part of the solution.
3. If all neighbors of a node are fully active, all edges to this node are covered by its neighbors, and this node should completely lose support and eventually becomes fully inactive.
4. If a node is fully inactive, then, to cover each of its adjacent edges, each of its neighbors should get full support from the node.
5. For two adjacent nodes to an edge, the winner for this edge should not prevent the loser from being activated by support from other neighbors.
6. A node adjacent to more edges having less activated neighbors should be more likely to be in the solution.
7. The nodes with higher activations should compete more strongly.

From constraints 1 and 4, the support for node i from a neighbor j should have the form $(1 - a_j(t))$. Constraints 2, 3, and 5 can be captured by the total support for node i, which is $\sum_{v_j \in N(i)}(1 - a_j(t))$. In addition, this also captures constraint 6. From constraint 7, we multiply an addition factor $(1 + Aa_i(t))$ to arrive at $in_i(t) = (1 + Aa_i(t)) \sum_{v_j \in N(i)}(1 - a_j(t))$. So $in_i(t)$ is the total support received by node i at time t. Finally, we add a decay term to $in_i(t)$ and then multiply it by a shunting factor $(1 - a_i(t))$, which ensures that the activation of the neuron will be kept in the interval $[0, 1]$.

The resulting equation of motion is

$$\dot{a}_i(t) = \frac{da_i}{dt} = [in_i(t) - Aa_i(t)][1 - a_j(t)] \tag{4.1}$$

4.2 Convergence of the VCP network

If the neurons start from a small value in the range $(0, 1)$, then it is easy to see that they will remain in the range $[0, 1]$. This point will become more clear if we expand equation (4.1) into the following form:

$$\dot{a}_i(t) = \left[(1 + Aa_i(t)) \sum_{v_j \in N(i)} (1 - a_j(t)) - Aa_i(t)\right][1 - a_i(t)] \tag{4.2}$$

When a_i approaches 1, $\dot{a}_i(t)$ approaches 0, so its activation will stop increasing at 1. When its activation approaches 0, $Aa_i(t)$ approaches 0, $(1 + Aa_i(t))$ approaches 1, and $(1 - a_i(t))$ also approaches 1. Hence $\sum_{v_j \in N(i)}(1 - a_j(t)) > Aa_i(t)$, which implies $\dot{a}_i(t)$ becomes positive and its activation cannot decrease any more. Now we take the partial differential of a_i with respect to a_k, which is

$$\frac{\partial \dot{a}_i}{\partial a_k} = -(1 - a_i)(1 + Aa_i) \sum_{v_j \in N(i)} \frac{\partial a_j}{\partial a_k}$$

Therefore,

$$\begin{aligned} \frac{\partial \dot{a}_i}{\partial a_k} &< 0, \quad \text{if } v_k \in N(i) \\ &= 0, \quad \text{otherwise.} \end{aligned}$$

So we arrive at the following theorem.

Theorem 3. *Two distinct nodes v_i and v_j are competitors in the VCP neural network only if they are immediate neighbors.*

By following the steps of section 3 and using the energy function terms, it is not difficult to construct a good energy function of the form

$$H(t) = \sum_i \left[\int_0^{a_i} g(\phi)d\phi + \frac{1}{2}(1 - a_i) \sum_{v_j \in N(i)} (1 - a_j)\right] \tag{4.3}$$

where $g(a_i) = Aa_i/(1 + Aa_i)$. By taking the total differential of H, we obtain

$$\frac{dH(t)}{dt} = -\sum_k f(a_k) \left[g(a_k) - \sum_{v_j \in N(k)} (1 - a_j)\right]^2 \leq 0 \tag{4.4}$$

where $f(a_k) = (1 + Aa_k)(1 - a_k)$. Since $f(a_k) \geq 0$, $dH(t)/dt \leq 0$ always. Hence $H(t)$ is a Lyapunov function of the neural network, and the network is stable.

	$N_c = N_m$	$N_c = N_m + 1$	$N_c = N_m + 2$	$N_c = N_m + 3$
$\|V\|$=20	35 (87.5%)	5 (12.5%)	0 (0%)	0 (0%)
$\|V\|$=50	18 (45%)	15 (37.5%)	5 (12.5%)	2 (5%)
Total	53 (66.25%)	20 (25%)	5 (6.25%)	2 (2.5%)

Table 1: Covers obtained by a neural network.

If the neurons take the value of either 0 or 1 in a state that does not represent a set cover, then there exists some edge (v_i, v_k) with $a_i = a_k = 0$. Hence $\dot{a}_i = \sum_{v_j \in N(i)}(1 - a_j) \geq (1 - a_k) > 0$, so the activation a_i will increase. The state is thus not stable.

On the other hand, if such a state represents a redundant cover, it is also not stable. This can be shown by considering the node a_i and all its neighbors. If a_i is redundant, its neighbors must all approach 1. Thus $(1 + Aa_i)\sum_{v_j \in N(i)}(1 - a_j) - Aa_i < 0$. The activation a_i must decrease, and the state is not stable. So we arrive at the following theorem.

Theorem 4. *When the network enters a stable equilibrium state, this state must represent an irredundant cover. In addition, every irredundant cover is a stable state of the network. A smaller size irredundant cover corresponds to a lower energy state.*

4.3 Experiments and results

To study the performance of the VCP neural network, a set of 80 randomly generated graphs was used to test it. A graph $G = \langle V, p \rangle$ is generated in a way such that the probability of generating an edge between each pair of nodes is p. Among the 80 graphs tested, half are "small" with $|V| = 20$ and half are "large" with $|V| = 50$. For each size, there are four groups of 10 and the graphs in each groups have the same p. The four groups have $p = 0.5$, 0.25, 0.125, and 0.0625, respectively. The sizes of these examples are practical. The solutions found by the neural networks are compared with the solutions produced by the greedy approximation algorithm [12] and with the optimal solutions obtained by the heuristic A^* algorithm. The greedy algorithm in [12] has been widely used in comparison with other techniques.

In our simulation experiments, the constant $A = 3.0$. For all 80 graphs, the neural network was able to reach stable states (in which every node has activation either greater than 0.99 or less than 0.01) after 100 iterations. All solutions represented irredundant covers of their respective graphs. The results for the neural network in comparison with the optimal results are summarized in table 1. The results for the greedy algorithm are given in table 2. Obviously, the neural network significantly outperforms the greedy sequential algorithm.

In table 1, N_c denotes the size of a cover produced by the neural net, and N_m denotes the size of a minimum cover. In table 2, N_g denotes the size of a cover by the sequential greedy algorithm.

	$N_g = N_m$	$N_g = N_m + 1$	$N_g = N_m + 2$	$N_g = N_m + 3$
$\|V\|$=20	24 (60%)	13 (32.5%)	3 (7.5%)	0 (0%)
$\|V\|$=50	8 (20%)	18 (45%)	9 (22.5%)	5 (12.5%)
Total	32 (40%)	31 (38.75%)	12 (15%)	5 (62.5%)

Table 2: Covers obtained by a greedy algorithm.

5. Neural network for generalized assignment

The generalized assignment problem concerns the assignment of tasks to agents. The generalized assignment has applications in assigning software development tasks to programmers, assigning jobs to computer networks, scheduling variable length television commercials, and so forth.

5.1 Problem description

In a generalized assignment problem, a set of agents $A = (1, 2, 3, \ldots, m)$ and a set of tasks $T = (1, 2, 3, \ldots, n)$ are given. In addition, a cost matrix C of dimension $m \times n$, a resource matrix R of dimension $m \times n$, and a resource limit vector B of size m are also given. An element $c_{i,j}$ in the C matrix is the cost when agent i is assigned task j and an element $r_{i,j}$ in the R matrix is the resource required by agent i to perform task j. An element β_i in B is the amount of resource available to agent i. The problem is to assign tasks to agents such that the total cost $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j}$ can be minimized, where $x_{i,j} = 1$ if agent i is assigned task j and $x_{i,j} = 0$ otherwise.

This is one of the very difficult problems that are complete under ϵ approximation. That is, no ϵ approximation algorithm is known. The cost function for the problem is $\sum_{i=1}^{m} \sum_{j=1}^{n} c_{i,j} x_{i,j}$, the resource constraints are $\sum_{j=1}^{n} r_{i,j} x_{i,j} \leq \beta_i$ for $i = 1, 2, 3, ..., m$, and the assignment constraints are $\sum_{i=1}^{m} x_{i,j} = 1$ for $j = 1, 2, 3, ..., n$. The assignment constraints mean that each task must be assigned to exactly one agent.

The neural network consists of an array of $m \times n$ elements. The neuron at position (i, j) represents the assignment of task j to agent i. This representation is intuitive and complete. If task j is assigned to agent i, the activation (state) of neuron $x_{i,j}$ is 1; otherwise it is 0.

5.2 Formulating equations of motion

The set of resource constraints is a set of weighted constraints. The set of assignment constraints is a set of 1-out-of-n constraints. Hence the equations of motion for generalized assignment can be derived (with slight modifications) from the equations of these constraints.

For the resource constraints, the equation of motion for $x_{a,b}$ should have a factor $(\beta_a - \sum_{j \neq b}^{n} r_{a,j} f(x_{a,j}) - r_{a,b})$. This is adopted from the 0/1 knapsack equation. For the assignment constraints, the equation of motion for $x_{a,b}$ should have a factor $(1 - \epsilon - \sum_{i \neq a}^{m} f(x_{i,b}))$ (adopted from the 1-out-of-n

equation). These two types of equations for $x_{a,b}$ are referred to as the *resource condition* and the *assignment condition* for $x_{a,b}$, respectively. When none of the conditions are violated (both produce positive values), a neuron increases its activation according to the reciprocal of its cost; that is, $dx_{a,b} = A(1/c_{a,b})dt$, which favors those neurons with lower costs. When either one or both conditions for $x_{a,b}$ are violated, the neuron activation must decrease; that is, $dx_{a,b}/dt = -B$, where B is a constant.

This set of two equations describes the dynamic behavior of a neuron in the optimization network for generalized assignment. This set of equations has been simulated and tested on many examples. The set of equations can be combined using a function ϕ as defined below:

$$\frac{dx_{a,b}}{dt} = \phi\left(A,\ B,\ c_{a,b},\ 1-\epsilon-\sum_{i\neq a}^{m} f(x_{i,b}),\ \beta_a - \sum_{j\neq b}^{n} r_{a,j} f(x_{a,j}) - r_{a,b}\right) \tag{5.1}$$

where A and B are constants, f is the threshold function with threshold θ, and the function ϕ is defined by

$$\phi(A,B,C,\rho_1,\rho_2) = \begin{cases} A/C, & \text{if } \rho_1 \geq 0 \quad \text{and} \quad \rho_2 \geq 0 \\ -B, & \text{if } \rho_1 < 0 \quad \text{or} \quad \rho_2 < 0 \end{cases}$$

In this equation of motion, each neuron competes for an activation that is proportional to the reciprocal of the cost of assignment. The larger the cost, the slower the neuron activation increases (hence the neuron competes more weakly). Therefore, this scheme favors the assignments with smaller costs. The system is not very sensitive to the choice of the constants. The constants A and B control the speed of convergence but they do not significantly affect the final results.

5.3 Network convergence

The function ϕ used in the equations of motion is rather complicated. This function gives a negative value if either the resource condition or the assignment condition for a neuron is negative. This disjunction of the two conditions guarantees that, if the activation of a neuron $x_{a,b}$ exceeds the threshold θ, this neuron will remain active. That is, it will approach the value 1 and remain there afterwards. To show this, we need another condition that requires that the costs for neurons in the same row and in the same column are all distinct (this does not imply that all the costs are different). If they are not all distinct in the same row and in the same column, they must be initialized to different values.

The requirement of different costs in the same row and in the same column guarantees that the activations of the neurons in a given row and in a given column do not approach the threshold θ at the same time. If the activation of a neuron $x_{a,b}$ does exceed θ, it will force the neurons in the same column to

decrease their activations because of the logic disjunction in the function ϕ. Hence, these neurons will not be able to cause $x_{a,b}$ to decrease activation. On the other hand, if we choose θ large enough such that $n(1-\theta)r_{\max} < r_{\min}$, we can also guarantee that the activation of $x_{a,b}$ will remain high if it exceeds θ. The value $r_{\max}$ is defined as $\max\{r_{i,j}\}$ and $r_{\min}$ is defined as $\min\{r_{i,j}\}$, over $1 \leq i \leq n$ and $1 \leq j \leq m$. This can be readily observed from the equations for resource conditions. Therefore, we can choose appropriate θ such that the activation of a neuron will approach 1 and remain there if it has exceeded θ.

Now we want to show $dx_{a,b}/dt$ can change sign at most once in each run, so the system cannot oscillate. We consider three cases.

1. $dx_{a,b}/dt$ changes sign due to the resource condition. Then we know that the resource limit of agent a is not sufficient for processing task b. From the previous observation, we know that those neurons that have activations above θ forced $dx_{a,b}/dt$ to change sign. But the activations of these neurons will not decrease from that point in time. Therefore, the resource condition of task b will never be satisfied by agent a, and $dx_{a,b}/dt$ will remain negative (and cannot become positive again).

2. $dx_{a,b}/dt$ changes sign due to the assignment condition. In this case, the activation of another neuron in the same column (for the same task) exceeds θ, which forces $dx_{a,b}/dt$ to remain negative.

3. $dx_{a,b}/dt$ changes sign due to both conditions. From the above discussion, it will also not change sign again.

Therefore we can derive the following result.

Theorem 5. *The system defined in equation (5.1) is stable if the costs in the same column and in the same row are different and θ is properly chosen.*

One can see from the above discussion that, although the system is stable, there is no implication that it will always produce feasible solutions. By "feasible" solution we mean one in which all the tasks are assigned to some agents and the constraints are all satisfied.

Indeed, the network occasionally produces infeasible solutions. But such situations rarely arise. They arise only when the number of feasible solutions is very small, which happens when the average resource requirement for the tasks exceeds the average resource limit for the agents and the variance of the resource requirements for the tasks is large. But these are not reasonable requirements and they do not occur in most real applications. Less than 5% of all the simulation experiments we performed have produced infeasible solutions, and among the 5% of experiments some did not have feasible solutions at all.

5.4 Experimental results

Four groups, each with 10 sets of data, were randomly generated to test the generalized assignment network. The first group contained 10 sets of data for

	net > sim	sim > net	net = sim
10 jobs / 3 agents	4	0	6
10 jobs / 5 agents	6	1	3
10 jobs / 7 agents	5	2	3
10 jobs / 10 agents	3	2	5

Table 3: Results for generalized assignment network.

10 tasks and 3 agents; the second group contained 10 sets of data for 10 tasks and 5 agents; the third group contained 10 sets of data for 10 tasks and 7 agents; and the last group contained 10 sets of data for 10 tasks and 10 agents. The resource bounds were generated according to the average resource for each agent and the total number of agents. Since this problem is known to be ϵ-approximation complete [11], there exists no heuristic algorithm that can find optimal or good near-optimal solutions in reasonable time. So we choose to compare our neural network with the simulated annealing method, which is known to produce near-optimal solutions. The constant A was chosen such that $A/\min\{c_{a,b}\} = 0.02$, and B was set to 0.01. The results are summarized in table 3.

As can be seen from the table, the neural networks produce better results than slow cooling simulated annealing for every group of data.

6. Summary

A systematic approach to the design of competition-based neural networks for combinatorial optimization has been presented. We also give some general techniques for finding good energy functions to prove the convergence property of such neural networks.

Several neural networks have been designed to show the feasibility of this approach. These neural networks produce highly competitive performance as compared with sequential heuristic algorithms. In addition, extensive simulation experiments have been performed to compare the quality of the neural networks. The results are satisfactory. The approach developed in this paper is general, simple, and competitive. The important characteristics of our approach are summarized as follows.

1. Our approach is generally applicable. In addition to the examples presented in this paper, we have also developed neural networks for other optimization problems.

2. Our method is very simple and is easy to learn. Our students have been taught to design neural networks with this approach in very short time and they have been able to learn and apply the technique very quickly.

3. The neural networks designed by this approach can produce impressive performance and high-quality solutions.

Acknowledgments

Part of this research was supported by an Australian Research Council grant when the first author was with the Department of Computer Science, Monash University, Clayton, VIC 3168, Australia. Part of the research was performed with the help of the Department of Computer Science, Concordia University, Montreal, Quebec, Canada. The research of the second author was supported in part by NSF award IRI-8451430.

References

[1] S. Abe, "Theories on the Hopfield Neural Networks," *International Joint Conference on Neural Networks*, **1** (1989) 557–564.

[2] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A Learning Algorithm for Boltzmann Machines," *Cognitive Science*, **9** (1985) 147–169.

[3] Y. Akiyama, A. Yamashita, M. Kajiura, and H. Aiso, "Combinatorial Optimization with Gaussian Machine," *International Joint Conference on Neural Networks*, **1** (1989) 533–540.

[4] P. Bourret, S. Goodall, and M. Samuelides, "Optimal Scheduling by Competitive Activation: Application to the Satellite Antennae Scheduling Problem," *International Joint Conference on Neural Networks*, **1** (1989) 565–572.

[5] M. Cohen and S. Grossberg, "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks," *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-13** (1983) 815–826.

[6] S. Grossberg, "How Does a Brain Build a Cognitive Code?" *Psychological Review*, **87** (1989) 1–51.

[7] J. J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences*, **79** (1982) 2553–2558.

[8] J. J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons," *Proceedings of the National Academy of Sciences*,**81** (1984) 3088–3092.

[9] J. J. Hopfield and D. W. Tank, "'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics*, **52** (1985) 141–152.

[10] J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model," *Science*, **233** (1986) 625–633.

[11] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms* (New York, Computer Science Press, 1978).

[12] D. Johnson, "Approximation Algorithms for Combinatorial Problems," *Journal of Computer and Systems Science*, **9** (1974).

[13] A. B. Kahng, "Traveling Salesman Heuristics and Embedding Dimension in the Hopfield Model," *International Joint Conference on Neural Networks*, **1** (1989) 513–520.

[14] S. Kirkpatrick, C. Gelatt, and M. Vecci, "Optimization by Simulated Annealing," *Science*, **220** (1983) 671–680.

[15] L.-Y. Fang and T. Li, "Design of Competition-Based Neural Networks for Combinatorial Optimization," *International Journal of Neural Systems*, **1**(3) (1990) 221–235.

[16] Y. Peng, "A Connectionist Approach for Vertex Cover Problems," Technical report, University of Maryland (1989).

[17] Y. Peng and J. Reggia, "A Connectionist Model for Diagnostic Problem Solving," *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-19** (1989).

[18] C. Peterson and J. R. Anderson, "Neural Networks and NP-Complete Optimization Problems: A Performance Study on the Graph Bisection Problem," *Complex Systems*, **2** (1988) 59–89.

[19] C. Peterson and B. Soderberg, "A New Method for Mapping Optimization Problems onto Neural Networks," *International Journal of Neural Systems*, **1** (1989) 3–22.

[20] J. Reggia, "Methods for Deriving Competitive Activation Mechanisms," *International Joint Conference on Neural Networks*, **1** (1989) 357–363.

[21] J. Reggia and G. Sutton. "Self-Processing Networks and Their Biomedical Implications," *Proceedings of the IEEE*, **76** (1988) 680–692.

[22] Y. Takefuji and H. Szu, "Design of Parallel Distributed Cauchy Machines," *International Joint Conference on Neural Networks*, **1** (1989) 529–532.

[23] D. W. Tank and J. J. Hopfield, "Simple 'Neural' optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Transactions on Circuit Systems*, **CAS-33** (1986) 533–541.

[24] P. Wang, S. Seidman, and J. Reggia, "Analysis of Competition-Based Spreading Activation in Connectionist Models," *International Journal of Man-Machine Studies*, **28** (1988) 77–97.

[25] G. V. Wilson and G. S. Pawley, "On the Stability of the Traveling Salesman Problem Algorithm of Hopfield and Tank," *Biological Cybernetics*, **58** (1988) 63–70.

[26] D. E. Van den Bout and T. K. Miller, "Graph Partitioning Using Annealed Neural Networks," *International Joint Conference on Neural Networks*, **1** (1989) 521–528.