A Hybrid Neural Network Model for Solving Optimization Problems

K. T. Sun and H. C. Fu, *Member, IEEE*

Abstract—**In this paper, we propose a hybrid neural network model for solving optimization problems. We first derive an energy function, which contains the *constraints* and *cost criteria* of an optimization problem, and we then use the proposed neural network to find the global minimum (or maximum) of the energy function, which corresponds to a solution of the optimization problem. The proposed neural network contains two subnets: a *Constraint network* and a *Goal network*. The *Constraint network* models the constraints of an optimization problem and computes the gradient (updating) value of each neuron such that the energy function monotonically converges to satisfy all constraints of the problem. The *Goal network* points out the direction of convergence for finding an optimal value for the cost criteria. These two subnets ensure that our neural network finds feasible as well as optimal (or near-optimal) solutions. We use two well-known optimization problems—the *Traveling Salesman Problem* and the *Hamiltonian Cycle Problem*—to demonstrate our method. Our hybrid neural network successfully finds 100% of the feasible and near-optimal solutions for the Traveling Salesman Problem and also successfully discovers solutions to the Hamiltonian Cycle Problem with connection rates of 40% and 50%.**

Index Terms—**Energy functions, feasible solutions, neural network, optimization problems.**

I. Introduction

NEURAL networks have been used to solve a wide variety of optimization problems [2], [10], [14], [20], [22], [23], [28], [31]. Hopfield and Tank [8] suggested that the Traveling Salesman Problem [6] can be represented by an energy function that can be iteratively solved on a neural network. The difficulty in using a neural network to optimize an energy function is that the iteration procedure may often be trapped into a local minimum, which usually corresponds to an invalid solution. Moreover, the values assigned to the parameters of an energy function can greatly affect the convergence rate of iterations. Aiyer *et al.* [1] proposed a mathematical method for predicting a set of parameters on the Hopfield model that allow the Hopfield net to reach a feasible solution. However, Aiyer's method is still restricted to quadratic functions (the Hopfield net) only. Many optimization problems, such as satisfiability problems, cannot be solved by the Hopfield net. In this paper, we propose a hybrid neural network for solving energy functions of different orders. The solutions found by our network are all feasible.

Manuscript received February 15, 1991; revised September 10. 1991. This work was supported in part by the National Science Council under Grant NSC 79-0408-E009-23.

The authors are with the Department of Computer Science and Information Engineering, National Chiao-Tung University Hsinchu, Taiwan 30050, R.O.C.

IEEE Log Number 9205428.

We first express an optimization problem by a set of logical expressions and then map the logical expressions into algebraic equations. Based on these algebraic equations, an energy function is formulated. The energy function has two parts: constraints and the cost criteria. The proposed neural network also has two parts: a *Constraint net* (for the constraints) and a *Goal net* (for the cost criteria). Benchmark tests on the traveling salesman problem show that our neural network can provide 100% of the feasible and near-optimal (or optimal) solutions to the test problems.

The contents of this paper are as follows. In Section II, we describe the proposed method for constructing an energy function from an optimization problem. In Section III, we present the operations and functions of the proposed neural network model. A neural state updating method, the *coordinate Newton method*, and some convergence theorems are also introduced in Section III. In Section IV, the results of experimental tests of our network are presented and discussed. Concluding remarks are given in Section V.

II. Problem Representation and the Transformation Method

Since Hopfield and Tank [8] constructed an energy function to represent the Traveling Salesman Problem (TSP) so that it could be iteratively solved on their neural network, many energy functions have been proposed to represent different optimization problems. To date, there is no systematic transformation method for constructing an energy function to represent an optimization problem. In this section, we shall propose such a systematic transformation method. Our transformation method contains three steps: 1) representing an optimization problem as a set of logical expressions; 2) mapping these logical expressions into a set of algebraic equations; and 3) formulating an energy function from these algebraic equations. Each of these steps will be discussed as follows.

A. Representing an Optimization Problem by a Set of Logical Expressions

Since most optimization problems contain two parts: constraints and cost criteria, to achieve an optimal solution, the constraints must be satisfied and the cost criteria must be minimized (or maximized). For example, in the Traveling Salesman Problem [9], the constraints are that each city must be visited exactly once and that a salesman can arrive at only one city at a time during the tour, and the minimization of the cost criteria is to find the shortest tour length. For this problem,

we use the logical symbol C_{xj} or its complement to represent whether or not a city x is being visited at time j during a tour, and we use the algebraic symbol d_{xy} to represent the distance between cities x and y. Thus, the constraints and the minimization of cost criteria can be represented by the logical expressions at the bottom of this page and in the following.

In both of these constraints, N represents the number of cities in the problem, and $1 \leq i, j \leq N$.

Cost criteria: To find the shortest tour length.

$$Min \sum_{x,y}^{N} \sum_{i}^{N} [d_{xy} C_{xi} \wedge (C_{yi+1} \oplus C_{yi-1})] \tag{3}$$

where $\oplus$ is an exclusive-OR operator (XOR).

Under constraints (1) and (2), the space of feasible solutions can be represented by a binary value matrix $[\boldsymbol{C}]_{N \times N}$. In matrix $[\boldsymbol{C}]$, there can be only one "1" (True) in each row and each column; all other variables must be "0" (False), so that constraints (1) and (2) are satisfied. Mapping logical expressions into algebraic equations is the second step of the transformation method. If we try to map expressions (1) and (2) directly into algebraic equations, however, the transformed algebraic equations will contain N^Nth power terms, making it difficult and complicated to update the new value on a neural net and slowing down the convergence speed. Therefore, we propose to rewrite the constraints of the problem using shorter logical expressions with fewer state variables. The rewritten constraints and the logical expressions are as follows:

Constraint 1: Each city i must be visited at least once and no more than once. This constraint can be expressed by the following set of logical equations:

$\backslash *$ At least one $C_{ik}, 1 \leq k \leq N$, is True. $* \backslash$

$$C_{i1} \vee C_{i2} \vee C_{i3} \vee \cdots \vee C_{iN} = True,$$

$\backslash *$ No more than one $C_{ik}, 1 \leq k \leq N$, is True. $* \backslash$

$$\begin{aligned} C_{i1} \wedge C_{i2} = False, C_{i1} \wedge C_{i3} = False, \cdots, \\ C_{i1} \wedge C_{iN} = False, \\ C_{i2} \wedge C_{i3} = False, \cdots, C_{i2} \wedge C_{iN} = False, \cdots, \\ C_{iN-1} \wedge C_{iN} = False. \end{aligned} \tag{4}$$

Constraint 2: A salesman arrives in at least one and no more than one city at any time j during the tour. Similarly, this constraint can be expressed by the following set of logical equations:

$\backslash *$ At least one $C_{kj}, 1 \leq k \leq N$, is True. $* \backslash$

$$C_{1j} \vee C_{2j} \vee C_{3j} \vee \cdots \vee C_{Nj} = True,$$

$\backslash *$ No more than one $C_{kj}, 1 \leq k \leq N$, is True. $* \backslash$

$$\begin{aligned} C_{1j} \wedge C_{2j} = False, C_{1j} \wedge C_{3j} = False, \cdots, \\ C_{1j} \wedge C_{Nj} = False, \\ C_{2j} \wedge C_{3j} = False, \cdots, C_{2j} \wedge C_{Nj} = False, \cdots, \\ C_{N-1j} \wedge C_{Nj} = False. \end{aligned} \tag{5}$$

The logical expression "$C_{yi+1} \oplus C_{yi-1} = True$" in the cost expression can also be formulated as

$$(C_{yi+1} \wedge C_{yi-1}) \vee (\overline{C}_{yi+1} \wedge \overline{C}_{yi-1}) = False. \tag{6}$$

Now, we shall simplify the cost expression in (6) by eliminating the term "$(C_{yi+1} \wedge C_{yi-1}) = False, 1 \leq y, i \leq N$" that also appears in the constraints [i.e., in (4)]. Thus, we can delete this term from (6) so that expression (6) can be simplified to $\overline{C}_{yi+1} \wedge \overline{C}_{yi-1} = False$. Since "$\overline{C}_{yi+1} \wedge \overline{C}_{yi-1} = False$" is logically equivalent to "$C_{yi+1} \vee C_{yi-1} = True$," the cost expression then becomes

$$Min \sum_{x,y}^{N} \sum_{i}^{N} [d_{xy} (C_{xi} \wedge (C_{yi+1} \vee C_{yi-1}))]. \tag{7}$$

Now the constraints and the cost criteria of a TSP can be represented by expressions (4), (5), and (7).

B. Mapping Logical Expressions into Algebraic Equations

By applying a mapping method similar to that used in [29], logical expressions can be formulated as algebraic equations without changing their semantic meaning. The mapping method is listed as follows: Replace each instance of

1) *True* by 1,
2) *False* by 0,
3) Logical variable C_{ij} by c_{ij},
4) A NOT operator by subtraction from one, and
5) An AND operator by multiplication.

Constraint 1: Each city i must be visited exactly once.

$$\begin{array}{ccccc} & (C_{i1} \wedge \overline{C}_{i2} \wedge \overline{C}_{i3} \wedge \cdots \wedge \overline{C}_{iN}) & \vee & (\overline{C}_{i1} \wedge C_{i2} \wedge \overline{C}_{i3} \wedge \cdots \wedge \overline{C}_{iN}) & \\ \vee & \cdots & \vee & \cdots & \\ \vee & (\overline{C}_{i1} \wedge \cdots \wedge \overline{C}_{iN-2} \wedge C_{iN-1} \wedge \overline{C}_{iN}) & \vee & (\overline{C}_{i1} \wedge \overline{C}_{i2} \wedge \cdots \wedge \overline{C}_{iN-1} \wedge C_{iN}) = True. & \end{array} \tag{1}$$

Constraint 2: A salesman arrives at only one city at any time j during the tour.

$$\begin{array}{ccccc} & (C_{1j} \wedge \overline{C}_{2j} \wedge \overline{C}_{3j} \wedge \cdots \wedge \overline{C}_{Nj}) & \vee & (\overline{C}_{1j} \wedge C_{2j} \wedge \overline{C}_{3j} \wedge \cdots \wedge \overline{C}_{Nj}) & \\ \vee & \cdots & \vee & \cdots & \\ \vee & (\overline{C}_{1j} \wedge \cdots \wedge \overline{C}_{N-2j} \wedge C_{N-1j} \wedge \overline{C}_{Nj}) & \vee & (\overline{C}_{1j} \wedge \overline{C}_{2j} \wedge \cdots \wedge \overline{C}_{N-1j} \wedge C_{Nj}) = True. & \end{array} \tag{2}$$

When an OR operator is needed, it can be derived by combining the NOT and AND operators. For example, $X_i \vee Y_i = \overline{\overline{X}_i \wedge \overline{Y}_i}\cdot(X_i \vee Y_i)$ can then be transformed into the algebraic equation $1 - (1 - x_i)(1 - y_i)$.

Based on this mapping method, the logical expressions of a TSP can be transformed into the following algebraic equations:

Constraint 1: Each city i must be visited at least once and no more than once.

$$\begin{aligned}(1 - c_{i1})(1 - c_{i2})(1 - c_{i3})\cdots(1 - c_{iN}) = 0,\\ c_{i1}c_{i2} = 0, c_{i1}c_{i3} = 0, \cdots, c_{i1}c_{iN} = 0,\\ c_{i2}c_{i3} = 0, \cdots, c_{i2}c_{iN} = 0, \cdots, c_{iN-1}c_{iN} = 0.\end{aligned} \quad (8)$$

Constraint 2: A salesman arrives in at least one and at most one city at any time j during the tour.

$$\begin{aligned}(1 - c_{1j})(1 - c_{2j})(1 - c_{3j})\cdots(1 - c_{Nj}) = 0,\\ c_{1j}c_{2j} = 0, c_{1j}c_{3j} = 0, \cdots, c_{1j}c_{Nj} = 0,\\ c_{2j}c_{3j} = 0, \cdots, c_{2j}c_{Nj} = 0, \cdots, c_{N-1j}c_{Nj} = 0.\end{aligned} \quad (9)$$

Cost criteria: To find the shortest tour length.

$$Cost = \sum_{x,y}^{N}\sum_{i}^{N} d_{xy}c_{xi}(c_{yi+1} + c_{yi-1} - c_{yi+1}c_{yi-1}). \quad (10)$$

C. Constructing an Energy Function from the Algebraic Equations

For the constraints of the TSP [(8) and (9)], a squared error function E_s can be formulated as follows:

$$E_s = \sum_{i=1}^{N'}(t_i - a_i)^2, \quad (11)$$

where t_i represents the target value (at the right-hand side) and a_i represents the iteration value (at the left-hand side) of each algebraic expression of (8) and (9). N' is the total number of algebraic equations in (8) and (9). An energy function E can be obtained by combining (10) and (11) as follows:

$$E = E_s + Cost = A\sum_{i=1}^{N'}(t_i - a_i)^2 + BCost \quad (12)$$

where A and B are the parameters. When the constraints are satisfied (i.e., $E_s = 0$) and the cost criteria is minimized, the shortest tour length of the TSP is obtained.

Parameter Setting: There are two adjustable parameters in (12). Previous studies of neural networks for optimization problems have had to assign appropriate values to the parameters in an energy function to find a feasible solution [1], [22]. Consequently, choosing improper values for the parameters in the energy function slows down the convergence speed and results in invalid solutions [1], [22]. In the next section, we propose a neural network and a neural state updating method in which the parameters of the energy function can be fixed values and no initial setting of the proper parameters is required. In addition, our method accelerates the convergence speed and generates feasible solutions.

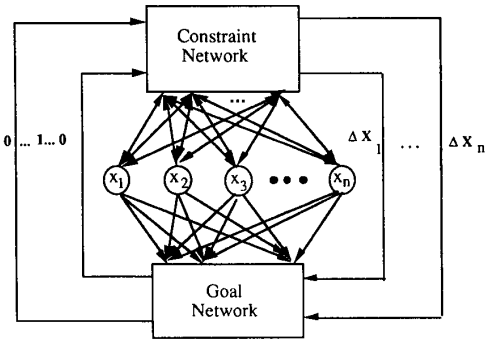

Fig. 1. The structure of the hybrid neural network.

III. Minimization of the Energy Function

Many neural network models and algorithms [4], [11], [12], [18]–[20], [23], [30], [32] have been proposed for solving optimization problems. Most of the algorithms often converge to an invalid solution or require a large amount of computation time to reach an optimal (or near-optimal) solution. In this section, we propose a neural network model that finds an optimal (or near-optimal) solution within a short computation time.

The proposed neural network, which we called the hybrid neural network, contains two subnets: a *Constraint network* and a *Goal network* (see Fig. 1).

The *Constraint network* models the constraints of the problem and computes the gradient (updating) value of each neuron (i.e., $\Delta x_1, \Delta x_2, \cdots, \Delta x_n$). By using these gradient value Δx's, the *Goal network* computes the direction of convergence during each iteration for minimizing (or maximizing) the cost criteria (functions). The underlying point of this method is to ensure the constraints are satisfied (or at least are closer to being satisfied) at each stage of the gradient descent search on the cost functions. Therefore, the hybrid neural network produces feasible and near-optimal solutions.

We have designed an array processor system [5], [13] (see Fig. 2) for our hybrid neural network. In Fig. 2, each processor element (neuron) x_i in the *Constraint Net* computes the updating value Δx_i, and all updating values $(\Delta x_1, \Delta x_2, \cdots, \Delta x_N)$ are sent to the *Max-Min Net* to determine which neuron is to be updated. Each processor in the *Goal net* computes $\frac{\partial Cost}{\partial x_i}$. Both the Δx_i and $\frac{\partial Cost}{\partial x_i}$ are sent to the Max-Min net to determine which variable x_i is to be updated. The output of the Max-Min Net enables the *Selector module* to output the corresponding updating value Δx_i, which will be passed back to the input neurons to update their states. The iteration procedure continues until a stable state (i.e., $\Delta x_i = 0$) is reached.

The Coordinate Newton Method: To minimize the squared error function E_s, we propose a neural state updating method called the *coordinate Newton method* to compute the updating value of each neuron (i.e., each variable) at each iteration. This method is based on the concept of the coordinate descent method [17], in which a function $f(\boldsymbol{x})$ is minimized with respect to one of the coordinate variables x_i of $\boldsymbol{x}$ at each

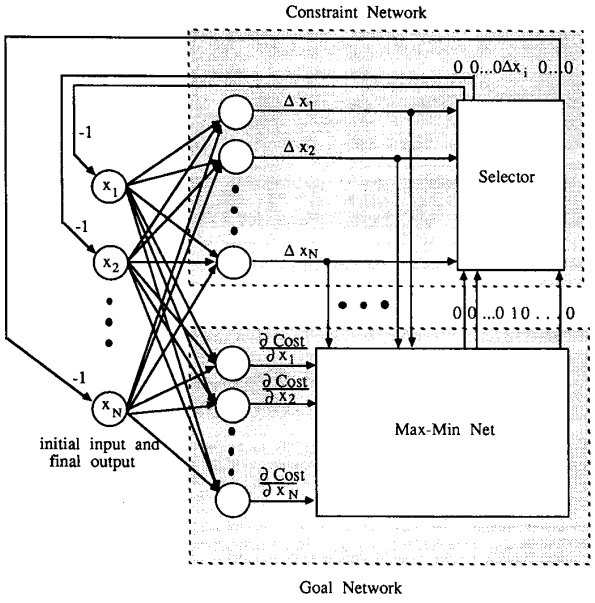

Fig. 2. The architecture of our hybrid neural network system.

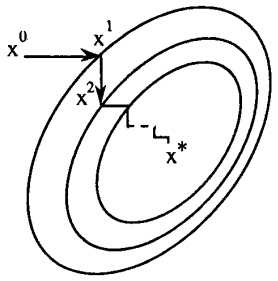

Fig. 3. For a function f, an initial point $\boldsymbol{x}^0$ is monotonically reduced to a minimum point $\boldsymbol{x}^*$.

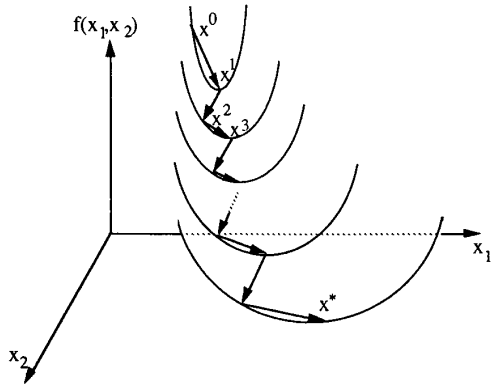

Fig. 4. A convex function $f(x_1, x_2)$ converges to the minimum point $\boldsymbol{x}^i$ on the updating coordinate in each iteration.

iteration until the gradient of function f reaches zero (i.e., $\nabla f(\boldsymbol{x}) = 0$). In order to achieve faster convergence speed, we apply the *Newton method* [17] instead of the gradient method to search for the minimum. By using the Newton method to find the minimum of a function f, the updating value Δx for a vector $\boldsymbol{X}$ at an iteration t is defined by (13).

$$\Delta \boldsymbol{X}^t = [\boldsymbol{F}(\boldsymbol{X}^t)]^{-1} \nabla \boldsymbol{f}(\boldsymbol{u}X^t)^T, \text{ and}$$
$$[\boldsymbol{F}(X^t)] = \left[\frac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{x}_i \partial \boldsymbol{x}_j}\right], \forall i, j. \tag{13}$$

For updating a function $\boldsymbol{f}$ with only one variable x, (13) can be simplified as

$$\Delta x = -\frac{\dfrac{d\boldsymbol{f}}{dx}}{\dfrac{d^2\boldsymbol{f}}{dx^2}}. \tag{14}$$

Thus, the updating value along a coordinate variable x_i will be

$$x_i^{t+1} = x_i^t + \Delta x_i = x_i^t - \frac{\dfrac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}_i}}{\dfrac{\partial^2 \boldsymbol{f}}{\partial \boldsymbol{x}_i^2}}, \tag{15}$$

where the value x_i^{t+1} is the new value on the coordinate x_i such that the new point $\boldsymbol{x}^{t+1}$ is closer to the minimum point. Some interesting properties of the coordinate Newton method are presented in the following theorems.

Theorem 1: The coordinate Newton method monotonically reduces a function f to a stable state, where f has nonzero second-order partial derivatives over a convex set Ω in which each variable is of order two.

Fig. 3 illustrates that an initial point $\boldsymbol{x}^0$monotonically reduces to a minimum point $\boldsymbol{x}^*$ by the coordinate Newton method. Each ellipse curve in Fig. 3 represents an equal value (potential) curve of a function f. The proof of this theorem is given in Appendix A.

Theorem 2: The coordinate Newton method always finds the minimum point of a function f along the updating coordinate at each iteration, where f has nonzero second-partial derivatives over a convex set Ω in which each variable is of order two.

Theorem 2 means that the new point $\boldsymbol{x}^{t+1}$ is the minimum point along the updating coordinate x_i of the iteration t. The proof of this theorem is shown in Appendix B. Fig. 4 illustrates that an initial point $\boldsymbol{x}^0$ of a convex function $f(x_1, x_2)$ converges to the minimum point $\boldsymbol{x}^1$ along the updating coordinate x_1. At the next iteration, the point $\boldsymbol{x}^1$ converges to the minimum point $\boldsymbol{x}^2$ along the updating coordinate x_2, and the convergence procedure proceeds continuously until a stable point $\boldsymbol{x}^*$ is reached (i.e., $\nabla f = \boldsymbol{0}$).

Another property of the coordinate Newton method is that *it converges to a minimum with order two of convergence.*

Order of Convergence: When a sequence $\{\boldsymbol{x}_k\}$ converges to $\boldsymbol{x}^*$, the order of convergence [16] of $\{\boldsymbol{x}_k\}$ is defined as the supremum of the nonnegative number p satisfying

$$0 \le \overline{\lim_{k \to \infty}} \frac{| x_{k+1} - x^* |}{| x_k - x^* |^p} < \infty. \tag{16}$$

A larger order of p implies a faster convergence speed.

Theorem 3: Let f be a function on R^n. Assume that there exists a minimum point $\boldsymbol{x}^*$, and the Hessian $\boldsymbol{F}(\boldsymbol{x}^*)$ is positive

definite. If the initial point $\boldsymbol{x}^0$ is not a minimum and it is sufficiently close to $\boldsymbol{x}^*$, the sequence of points generated by the coordinate Newton method converge to $\boldsymbol{x}^*$ with order two convergence.

The proof of this theorem is presented in Appendix C. From Theorem 3, the coordinate Newton method has the same convergence—of order two—as the Newton method. In order to apply the coordinate Newton method in the proposed neural network for solving optimization problems, we propose the following algorithm:

Hybrid Network Updating Algorithm (HNUA)

1. Apply the coordinate Newton method to minimize the squared error function E_s by computing the updating value Δx_i $(= \frac{\frac{\partial E_s}{\partial x_i}}{\frac{\partial^2 E_s}{\partial x_i^2}})$ for all variables (neurons) x_i's.
2. Calculate the partial derivative of the $Cost$ function over each variable x_i (i.e., $\frac{\partial Cost}{\partial x_i}$).
3. Determine the maximum value Δx^* of $N \mid \Delta x_i \mid$'s
$$\Delta x^* = \max(\{\mid \Delta x_i \mid, 1 \leq i \leq N\}), \tag{17}$$
then form a set Γ of variables x_j that corresponds to the maximum updating value Δx^*, i.e.,
$$\Gamma = \{x_j \mid\mid \Delta x_j \mid = \Delta x^*, \forall j\}. \tag{18}$$
4. Among the x_j variables in set Γ, select a variable x_k which corresponds to the minimum value of the partial derivative of the $Cost$ function,
$$x_k = \min_{arg}(\frac{\partial Cost}{\partial x_j}, \forall x_j \in \Gamma). \tag{19}$$
Update x_k^t by adding the Δx_k^t obtained in Step 1,
$$x_k^{t+1} = x_k^t + \Delta x_k^t. \tag{21}$$
5. Check the gradient of function E_s (i.e., $\nabla E_s = \frac{\partial E_s}{\partial x_i}, \forall i$). If ∇E_s is equal to $\mathbf{0}_{1\times N}$, i.e., the energy function converges to a stable state ($\Delta x_i = 0, \forall i$), then stop. Otherwise, return to step 1 for the next iteration.

The HNUA is implemented by the *Constraint network* and the *Goal network*. Steps 1 and 2 can be executed in parallel on these two subnetworks. The max and min operations in steps 3 and 4 are performed by the Max-Min net. Based on the output from the Max-Min net, the *Selector* module selects the neuron x_k for updating. Finally, the Constraint net checks the gradient value ∇E_s to see whether the iteration procedure should be stopped or not. If the gradient ∇E_s is equal to $\mathbf{0}_{1\times N}$, then the energy function has converged to a stable state, which represents a solution to the optimization problem.

We will apply the HNUA to solve two well-known optimization problems: the *Traveling Salesman Problem* and the *Hamiltonian Cycle Problem.*

Example 1: Traveling Salesman Problem (TSP): By applying the transformation method, the TSP can be represented by an energy function E [(12)]. The steps involved in using the HNUA to solve the TSP are described as follows.

1. Apply the *coordinate Newton method* on the squared error function E_s [see (11)] to compute the Δc_{xi} for each variable c_{xi}, $1 \leq x, i \leq N$:
$$\Delta c_{xi} = -\frac{\frac{\partial E_s}{\partial c_{xi}}}{\frac{\partial^2 E_s}{\partial c_{xi}^2}}, 1 \leq x, i \leq N. \tag{22}$$
2. Calculate $\frac{\partial Cost}{\partial c_{xi}}$ of (10), $1 \leq x, i \leq N$:
$$\frac{\partial Cost}{\partial c_{xi}} = \sum_{y=1, y\neq x}^{N} d_{xy}(c_{yi+1} + c_{yi-1} - c_{yi+1}c_{yi-1}). \tag{23}$$
3. Determine the maximum value Δc^* of $N^2 \mid \Delta c_{xi} \mid$'s,
$$\Delta c^* = \max(\{\mid \Delta c_{xi} \mid, 1 \leq x, i \leq N\}), \tag{24}$$
then form a set Γ of variables c_{yj} that corresponds to the maximum updating value Δc^*, i.e.,
$$\Gamma = \{c_{yj} \mid\mid \Delta c_{yj} \mid = \Delta c^*, \forall y, j\}. \tag{25}$$
4. Among the c_{yj}'s in set Γ, select a variable c_{wk} which corresponds to the minimum value of the partial derivative of the $Cost$ function.
$$c_{wk} = \min_{arg}(\frac{\partial Cost}{\partial c_{yj}}, \forall c_{yj} \in \Gamma). \tag{26}$$
Update c_{wk} by adding the updating value obtained in Step 1.
$$c_{wk}^{t+1} = c_{wk}^t + \Delta c_{wk}. \tag{27}$$
5. Test the gradient of function E_s (i.e., ∇E_s). If ∇E_s is equal to $\mathbf{0}_{1\times N^2}$, i.e., the energy function converges to a stable state (i.e., $\Delta c_{xi} = 0, \forall x, i$), then stop. Otherwise, return to step 1 for the next iteration.

Based on the HNUA, the squared error function E_s of the TSP monotonically converges to a stable state, because the squared error function E_s has nonzero second partial derivatives over a convex set in which each variable is of order two (see Theorems 1 and 2). Many interesting properties of the squared error function E_s are presented with proofs in Appendix D.

Example 2: Hamiltonian Cycle Problem (HCP): A Hamiltonian cycle in a graph $G = (V, E)$ is a cycle in graph G containing all vertices in V. If G is directed, then the Hamiltonian cycle is directed; if G is undirected, then the Hamiltonian cycle is undirected. Note that not all graphs have a Hamiltonian cycle, and the problem of determining whether a graph G has a Hamiltonian cycle is NP-complete. To solve an HCP, we need to derive an energy function for the HCP. The energy function for an HCP can be derived in a way similar to the derivation of (12) for TSP. Since the nodes in an HCP are not fully connected, the cost (distance) d_{ij} between nodes is defined as follows:

$d_{ij} = 1$, if node i is connected to node j.
$d_{ij} = N$, if there is no connection between node i and node j. (28)

We represent the distance between two unconnected nodes i and j by a number N for ease of formulation and computation of the energy function.

The procedure of using the Hybrid Network Updating Algorithm to solve an HCP is similar to the procedure for solving the TSP. To avoid repetition, in the following we will discuss only the major procedure differences between these two problems. For an HCP, a valid tour length is N for an N nodes problem. The total distance connected to a node x at time i is equal to the value of $\frac{\partial Cost}{\partial c_{xi}}$. Using the distance defined in (28), a value of $\frac{\partial Cost}{\partial c_{xi}}$ greater than or equal to N indicates a disconnected path to node x at time i. In order to continue searching for a connected cycle, we randomly select a node y to visit at time i and remove the selected nodes at time $i+1$ and time $i-1$. Thus, another path is tried in order to establish a complete cycle. This process can be seen as a hill-climbing technique used to escape from a spurious minimum $\boldsymbol{x}_i^*$ and to search for the global minimum $\boldsymbol{x}^*$ (an optimal solution).

From the TSP and HCP, we see that the coordinate Newton method is suitable for rapidly solving different optimization problems with different orders of the squared error function E_s. Therefore, a wide variety of optimization problems, such as the satisfiability problem [23], the traffic control problem in interconnection networks [25], [26], the restrictive channel routing problem [27], the independent set problem, the multiprocessor scheduling problem, and the partition problem can be also solved by this method.

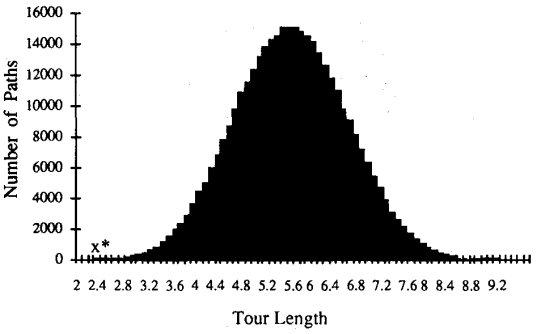

Fig. 5. The distribution of tour lengths of a 10-city TSP. Our solution falls at the position $\boldsymbol{x}^*$ (= 2.28), which is an optimal solution.

TABLE I
COMPARISON OF AVERAGE TOUR LENGTHS DERIVED USING THE HYBRID NETWORK (HN), POTTS NEURAL NETWORK (NN_p), ELASTIC NET (EN), GENETIC ALGORITHM (GA), SIMULATED ANNEALING (SA), HYBRID APPROACH (HA), AND RANDOM DISTRIBUTION (RD)

N (number of cities)	Different Approaches						
	HN	NN_P	EN	GA	SA	HA	RD
50	6.7	6.61	5.62	5.58	6.8	—	26.95
100	9.28	8.58	7.69	7.43	8.68	7.48	52.57
200	12.77	12.66	11.14	10.49	12.79	10.53	106.42

IV. EXPERIMENTAL RESULTS

Simulation systems for the proposed neural network were constructed on an IBM RS/6000 workstation. The Traveling Salesman Problem (TSP) and the Hamiltonian Cycle Problem (HCP) were used as examples to test the performance of the proposed neural network. For the TSP, we tested 10, 50, 100 and 200-city problems with randomly chosen city coordinates within a unit square, similar to [21]. One hundred test cases with different city coordinates were simulated and tested for problems with different number of cities. The initial values of the neurons for different tests were all set to "0", except for the starting city c_{11} (=1) of the tour. Fig. 5 shows the tour length distribution for a 10-city TSP. For this case, our test result, 2.28, is an optimal solution. Table I shows our simulation results along with the results for the Potts Neural Network (NN_p) [21], Elastic Net (EN) [4], Genetic Algorithm (GA) [19], Simulated Annealing (SA) [11], Hybrid Approach (HA) [12], and Random Distribution (RD).

Using our neural network, the simulation time for solving a 200-city problem on an IBM RS/6000 workstation was about two minutes. As shown in Table I, our results are far better than those obtained by RD (random distribution), and our results are comparable to those for the Potts Neural Network (NN_p). Although our method, HN, generates the longest or the second longest tour lengths, our object is to compare the overall performance of the different methods, including such factors as tour length, convergence speed, and feasible solution rate. As listed in Table I, our method is the only method that does not apply hill-climbing technique to escape from a local minimum. Obviously, applying hill-climbing technique requires a lot of computation. Davis [3] and Wilson *et al.* [30] have reported that some neural networks that do not use hill-climbing technique can become trapped into invalid solutions. Although our method guarantees that the solution search for the travelling salesman problem stops at the global minimum of the constraints, it may stop at the local minimum of the cost function. Thus we claim that our method generates feasible solutions for the TSP, although these solutions may not be optimal solutions. In addition, our method does not require setting proper values for the parameters in the energy function or estimating the temperature T_c for the annealing schedule, both of which are critical for obtaining a good solution using the Potts Neural Network. The Elastic Net (EN) can only be applied to the TSP using geometrical distances between cities, which is too restrictive for the general TSP. The distances for a general TSP may contain costs, times, etc., and the Elastic Net cannot be used to solved this general problem. The genetic algorithm has the best performance, since it provides the shortest tour lengths. When a genetic algorithm is used, the solution is obtained after many evolutions of generations. In each generation, thousands of feasible solutions are generated, and the better solutions are selected to evolve the next generation. Therefore, using the genetic algorithm to compute a solution provides a higher probability of discovering a near-optimal solution than other methods, which generates only one result. However, the genetic algorithm requires a large amount of computation to find a solution. In addition, different operations of the genetic algorithm, such as *crossover*, *mutation*, and *inversion*, must be specially defined to solve different optimization problems. For example, the crossover operation in the genetic algorithm can be defined to exchange the positions of cities on the tour for the TSP, and it can also be defined to change the assignment of objects to different sets for partition problem.

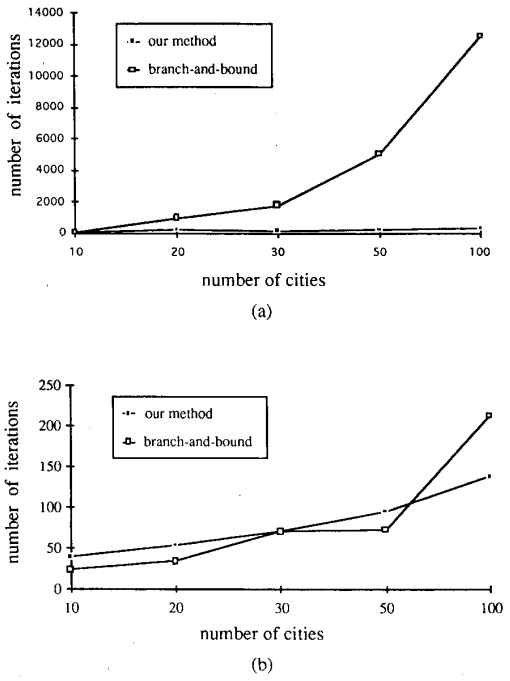

Fig. 6. The number of iterations needed by our method and another efficient algorithm to solve an HCP with connection density (a) 40% and (b) 50%.

So the genetic algorithm cannot provide a systematic method for solving different optimization problems. The simulated annealing (SA) method uses much time to reach a stable state, which makes it impractical for larger problems. The hybrid approach (HA), which combines three methods—the greedy method, simulated annealing, and exhaustive search—is a complex and computation-inefficient algorithm for solving the TSP. Compared with the other methods listed in Table I, our method provides greater computational efficiency.

For the HCP, we also tested 10, 20, 30, 50, and 100-node problems with connection densities of 40% and 50%. One hundred tests with different connection distributions between nodes were simulated for each different size and density problem. A comparison of our method with the branch-and-bound algorithm [9], is shown in Figs. 6(a) and (b).

The branch-and-bound algorithm is a tree search procedure in which the number of iteration steps grows exponentially as the size of the problem is increased. However, our simulation results show that the number of iteration steps in our neural network approach grows approximately in linear. When the connection density is 50%, the results of our method are comparable to those of the branch-and-bound algorithm. When the connection density is reduced to 40%, our method performs much better than the branch-and-bound algorithm. Our method seems to be suitable for solving HCP's with lower connection densities.

V. Conclusions

In this paper, a hybrid neural network and a neural state updating method for solving optimization problems have been proposed. A transformation method for representing an optimization problem by an energy function has also been presented. The transformation method can be applied to any type of optimization problem that can be represented by logical expressions such as those used in Section II of this paper. The energy function has two parts: the constraints and the cost criteria. It is solved using a hybrid neural network that comprises two subnets: the *Constraint net* and the *Goal net*, which correspond to the two parts of the energy function. The energy of the function is iteratively minimized (or maximized) by the neural network operations until a stable state is reached. The *coordinate Newton method*, a neural state updating method that combines the concepts of the coordinate descent method and the Newton method, is proposed for computing the updating value of each variable (neuron). Various merits of using the *coordinate Newton method* to determine the gradient value of each variable in order to reduce (or increase) the energy of the function are 1) the coordinate Newton method's convergence speed (order two convergence) is faster than that of the gradient descent method; 2) the coordinate Newton method is more suitable for parallel implementation than the Newton method; 3) if a function f has nonzero second-order partial derivatives over a convex set Ω in which each variable is of order two, then by applying the coordinate Newton method, a) the function f is monotonically reduced to a stable state, b) the function f always reaches the minimum point along the updating coordinate at each iteration; and 4) the stable state of the energy function represents a feasible solution.

In summary, the proposed neural network technique for solving optimization problems has the following advantages:

1) It provides a mapping technique for systematically transforming an optimization problem into an energy function.
2) It does not require selection of the proper values for parameters in an energy function in order to obtain an optimal (or near-optimal) solution.
3) It prevents the energy function from being trapped into an invalid solution.

Appendix A
Proof of Theorem 1

The *coordinate Newton method* monotonically reduces a function f to a stable state, where f has nonzero second-order partial derivatives over a convex set Ω, in which each variable is of order two.

Proof: According to Taylor's series expansion, we can expand a function $f(\boldsymbol{x})$ at point $\boldsymbol{x}^0$ as follows:

$$f(\boldsymbol{x}) = f(\boldsymbol{x}^0)+\boldsymbol{d}f(\boldsymbol{x}^0)+\frac{1}{2!}\boldsymbol{d}^2 f(\boldsymbol{x}^0)+\frac{1}{3!}\boldsymbol{d}^3 f(\boldsymbol{x}^0)+\cdots, \quad (29)$$

where $\boldsymbol{d}^k f(\boldsymbol{x}^0) = \sum_{i1}\sum_{i2}\cdots\sum_{ik}\frac{\partial^k f(\boldsymbol{x}^0)}{\partial x_1 \partial x_2 \cdots \partial x_k} q_1 q_2 \cdots q_k$, and $q_i = x_i - x_i^o$. By applying the *coordinate Newton method*, the updating value of x_i along the coordinate x_i at iteration t is

$$x_i^{t+1} = x_i^t - \frac{\frac{\partial f(x^t)}{\partial x_i}}{\frac{\partial^2 f(x^t)}{\partial x_i^2}}, \text{ and}$$

$$x_j^{t+1} = x_j^t, \forall j \neq i. \quad (30)$$

Let $\boldsymbol{x} = x^{t+1}$ and $\boldsymbol{x}^0 = x^t$. Then $q_i = x_i - x_i^0 = x_i^{t+1} - x_i^t = \Delta x_i$, such that the value of q_i is equal to Δx_i. Rewrite (29) in terms of $\boldsymbol{x}^t, x^{t+1}$ and $\Delta \boldsymbol{x}$. Then

$$f(\boldsymbol{x}^{t+1}) = f(\boldsymbol{x}^t) + \Delta x_i \frac{\partial f(\boldsymbol{x}^t)}{\partial x_i} + \frac{1}{2}\Delta x_i^2 \frac{\partial^2 f(\boldsymbol{x}^t)}{\partial x_i^2} + \frac{1}{3!}\Delta x_i^3 \frac{\partial^3 f(\boldsymbol{x}^t)}{\partial x_i^3} + \cdots. \tag{31}$$

Since each variable in function f is of order two, the higher order terms $(\frac{1}{3!}\Delta x_i^3 \frac{\partial^3 f(\boldsymbol{x}^t)}{\partial x_i^3} + \cdots)$ vanish and (31) becomes

$$\begin{aligned} f(\boldsymbol{x}^{t+1}) &= f(\boldsymbol{x}^t) - \frac{\frac{\partial f(x^t)}{\partial x_i}}{\frac{\partial^2 f(x^t)}{\partial x_i^2}} \frac{\partial f(\boldsymbol{x}^t)}{\partial x_i} + \frac{1}{2}\frac{(\frac{\partial f(x^t)}{\partial x_i})^2}{(\frac{\partial^2 f(x^t)}{\partial x_i^2})^2} \frac{\partial^2 f(\boldsymbol{x}^t)}{\partial x_i^2} \\ &= f(\boldsymbol{x}^t) - \frac{1}{2}\frac{(\frac{\partial f(x^t)}{\partial x_i})^2}{\frac{\partial^2 f(\boldsymbol{x}^t)}{\partial x_i^2}}. \end{aligned} \tag{32}$$

Then,

$$\Delta f(\boldsymbol{x}^t) = -\frac{1}{2}\frac{(\frac{\partial f(x^t)}{\partial x_i})^2}{\frac{\partial^2 f(\boldsymbol{x}^t)}{\partial x_i^2}}. \tag{33}$$

Since $f(\boldsymbol{x})$ is a convex function, the second partial derivatives of function $f(\boldsymbol{x})$ are greater than 0 (i.e., $\frac{\partial^2 f(\boldsymbol{x}^t)}{\partial x_i^2} > 0$). Therefore, we can prove that

$$\Delta f(\boldsymbol{x}^t) \leq 0. \tag{34}$$

This implies that the function f is monotonically reduced to a stable state. Q.E.D.

Appendix B
Proof of Theorem 2

The coordinate Newton method can always find the minimum point of a function f along the updating coordinate at each iteration, where f has nonzero second-partial derivatives over a convex set Ω in which each variable is of order two.

Proof: The function $f(\boldsymbol{x}), \boldsymbol{x} = (x_1, x_2, \cdots, x_n)$, can be expressed in terms of the variable x_i as follows:

$$f(\boldsymbol{x}) = Ex_i^2 + Fx_i + G, \text{ and } E > 0; F, G \in R, \tag{35}$$

where E, F and G are the terms that contain variables $x_j, \forall j \neq i$ in the function $f(\boldsymbol{x})$. By applying the coordinate Newton method to function f on the coordinate x_i, the variable x_i is updated by

$$x_i' = x_i - \frac{\frac{\partial f(\boldsymbol{x})}{\partial x_i}}{\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i^2}}. \tag{36}$$

If new point $\boldsymbol{x}'$ is the minimum point on the coordinate x_i, then the partial derivative of f over x_i at the next iteration should equal zero.

$$\begin{aligned} \frac{\partial f(\boldsymbol{x}')}{\partial x_i} &= \frac{\partial}{\partial x_i}(E(x_i')^2 + Fx_i' + G) \\ &= \frac{\partial}{\partial x_i}E(x_i')^2 + \frac{\partial}{\partial x_i}Fx_i' + 0 \\ &= E\frac{\partial}{\partial x_i}(x_i - \frac{\frac{\partial f(x)}{\partial x_i}}{\frac{\partial^2 f(x)}{\partial x_i^2}})^2 + F\frac{\partial}{\partial x_i}(x_i - \frac{\frac{\partial f(x)}{\partial x_i}}{\frac{\partial^2 f(x)}{\partial x_i^2}}) \\ &= 2E(\frac{\partial x_i}{\partial x_i} - \frac{\partial}{\partial x_i}\frac{\frac{\partial f(x)}{\partial x_i}}{\frac{\partial^2 f(x)}{\partial x_i^2}}) + F(\frac{\partial x_i}{\partial x_i} - \frac{\partial}{\partial x_i}\frac{\frac{\partial f(x)}{\partial x_i}}{\frac{\partial^2 f(x)}{\partial x_i^2}}) \\ &= 2E(1-1) + F(1-1) \\ &= 0 + 0 \\ &= 0. \end{aligned} \tag{37}$$

From the minimization of convex functions, (37) shows that the new point $\boldsymbol{x}'$ is the minimum point on the coordinate x_i at the next iteration. Thus, the *coordinate Newton method* always finds the minimum point along the updating coordinate at each iteration. Q.E.D.

Appendix C
Proof of Theorem 3

Let f be a function on R^n. Assume that there exists a minimum point $\boldsymbol{x}^*$ and that the Hessian $\boldsymbol{F}(x^*)$ is positive definite. If the initial point $\boldsymbol{x}^0$ is not a minimum and is sufficiently close to $\boldsymbol{x}^*$, the sequence of points generated by the coordinate Newton method converge to $\boldsymbol{x}^*$ with order two convergence.

Proof: According to the coordinate Newton method, the updating value for each neuron is $x_i^{t+1} = x_i^t + \Delta x_i^t$, which can be represented in vector form for the purpose of proving the convergence of the function f. In vector form the updating value is

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - [\boldsymbol{e}_i \boldsymbol{F}(\boldsymbol{x}^t)\boldsymbol{e}_i^T]^{-1} \nabla f(\boldsymbol{x}^t)\boldsymbol{e}_i^T \boldsymbol{e}_i \tag{38}$$

where $\boldsymbol{e}_i$ is the ith coordinate unit vector (i.e., $\boldsymbol{e}_1 = (1,0,0,\ldots,0), \boldsymbol{e}_2 = (0,1,0,\ldots,0), \ldots, \boldsymbol{e}_n = (0,0,0,\ldots,1)$); $\boldsymbol{F}(x^t)$ is the Hessian matrix [17], $[\partial^2 f/\partial x_i \partial x_j], \forall i,j$, of the function f; and $\nabla f(\boldsymbol{x}^t)$ is the gradient of function f at time $\boldsymbol{t}$. There exists $\rho, \alpha > 0$, $\beta > 0$ such that $\mid \boldsymbol{x} - x^* \mid < \rho$, $\mid [\boldsymbol{e}_i \boldsymbol{F}(x)\boldsymbol{e}_i^T]^{-1} \mid \leq \alpha$ and $\mid \nabla f(\boldsymbol{x})\boldsymbol{e}_i^T \boldsymbol{e}_i + [\boldsymbol{e}_i \boldsymbol{F}(x)\boldsymbol{e}_i^T](\boldsymbol{x}^* - \boldsymbol{x}) \mid \leq \beta \mid \boldsymbol{x} - \boldsymbol{x}^* \mid^2$, for all $\boldsymbol{x}$. Now suppose $\boldsymbol{x}^t$ is selected with $\alpha\beta \mid \boldsymbol{x}^t - \boldsymbol{x}^* \mid < 1$. Then

$$\begin{aligned} \mid \boldsymbol{x}^{t+1} - \boldsymbol{x}^* \mid &= \mid \boldsymbol{x}^t - \boldsymbol{x}^* - [\boldsymbol{e}_i \boldsymbol{F}(x^t)\boldsymbol{e}_i^T]^{-1} \nabla f(\boldsymbol{x}^t) e_i^T \boldsymbol{e}_i \mid \\ &= \mid [\boldsymbol{e}_i \boldsymbol{F}(\boldsymbol{x}^t)\boldsymbol{e}_i^T]^{-1} (\nabla f(\boldsymbol{x}^t)\boldsymbol{e}_i^T \boldsymbol{e}_i \\ &\quad + [\boldsymbol{e}_i \boldsymbol{F}(\boldsymbol{x}^t)\boldsymbol{e}_i^T](\boldsymbol{x}^* - \boldsymbol{x}^t)) \mid \\ &\leq \mid [\boldsymbol{e}_i \boldsymbol{F}(\boldsymbol{x}^t)\boldsymbol{e}_i^T]^{-1} \mid \beta \mid \boldsymbol{x}^t - \boldsymbol{x}^* \mid^2 \\ &\leq \alpha\beta \mid \boldsymbol{x}^t - \boldsymbol{x}^* \mid^2 \qquad (39) \\ &< \mid \boldsymbol{x}^t - \boldsymbol{x}^* \mid . \qquad (40) \end{aligned}$$

Equation (40) shows that the new point $\boldsymbol{x}^{t+1}$ is closer to $\boldsymbol{x}^*$ than the point $\boldsymbol{x}^t$, and (39) shows that the order of convergence of the coordinate Newton method is two. Q.E.D.

Appendix D
Properties of the Squared Error Function E_s of the TSP

Property 1: Based on the coordinate Newton method, the value of each variable c_{xi}, $1 \leq x, i \leq N$, in the squared error function E_s is bound within $[0, 1]$.

Proof: The squared error function E_s [(11)] can be rewritten as

$$E_s = \sum_x \sum_i \sum_{j \neq i} c_{xi}^2 c_{xj}^2 + \sum_x \sum_{y \neq x} \sum_i c_{xi}^2 c_{yi}^2 + \sum_x \prod_i (1 - c_{xi})^2 + \sum_i \prod_x (1 - c_{xi})^2. \quad (41)$$

Equation (41) can be expressed in terms of the variable c_{xi}:

$$E_s = A c_{xi}^2 + B c_{xi} + C, \quad (42)$$

where $A \geq 0$, $C \geq 0$, and $B \leq 0$. Therefore, (42) is a convex function of c_{xi} and can be expressed as

$$E_s = A(c_{xi}^2 + \frac{B}{A} c_{xi} + \frac{B^2}{4A^2}) + C - \frac{B^2}{4A} \quad (43)$$

$$= A(c_{xi} + \frac{B}{2A})^2 + C - \frac{B^2}{4A}. \quad (44)$$

From Theorem 2, c_{xi}^{t+1} is equal to the minimum point $(= \frac{-B}{2A})$ at iteration $t+1$, so the value of the variable c_{xi} is bound by

$$0 \leq c_{xi}^{t+1} = \frac{-B}{2A}. \quad (45)$$

From (41), the value of A (summation of the terms with order two) is greater than or equal to $\frac{1}{2}B$ (summation of the terms with order one), thus c_{xi}^{t+1} is further bound by

$$c_{xi}^{t+1} = \frac{-B}{2A} \leq 1. \quad (46)$$

This proves that the value of the variable c_{xi} is bound by [0,1]. Q.E.D.

Property 2: For the neuron state matrix $[C]_{N \times N}$ of a TSP, when the value of a variable c_{xi} equals one and the values of other elements in row x and column i are not all zeros, then the gradient value of $\frac{\partial E_s}{\partial c_{xi}}$ is not equal to zero.

Proof: When more than one variable in a row x and column i is one and the values of the other elements in row x and column i are not all equal to zero, (45) shows that the gradient value of $\frac{\partial E_s}{\partial c_{xi}}$ is not equal to zero (not in a stable state). Q.E.D.

Property 3: By applying the HNUA, when the squared error function E_s reaches a stable state and a variable c_{xi} equals one, then each row and each column of matrix $[C]_{N \times N}$ must contain only one "1" and all other entries of this row and column must be "0", which corresponds to a feasible solution.

Proof: From (41), the first-order partial derivative of E_s with respective to the variable c_{xi} is

$$\begin{aligned}\frac{\partial E_s}{\partial c_{xi}} =& 2[-(1-c_{x1})^2(1-c_{x2})^2 \cdots (1-c_{xi})(1-c_{xi+1})^2 \\ & \cdots (1-c_{xN})^2 \\ & - (1-c_{1i})^2(1-c_{2i})^2 \cdots (1-c_{xi})(1-c_{x+1i})^2 \\ & \cdots (1-c_{Ni})^2 \\ & + c_{xi}c_{1i}^2 + c_{xi}c_{2i}^2 + \cdots + c_{xi}c_{x-1i}^2 + c_{xi}c_{x+1i}^2 + \\ & \cdots + c_{xi}c_{Ni}^2 \\ & + c_{xi}c_{x1}^2 + c_{xi}c_{x2}^2 + \cdots + c_{xi}c_{xi-1}^2 + c_{xi}c_{xi+1}^2 + \\ & \cdots + c_{xi}c_{xN}^2]. \end{aligned} \quad (47)$$

When the function E_s reaches a stable state and the value of c_{xi} is equal to one, (47) becomes

$$\begin{aligned}\frac{\partial E_s}{\partial c_{xi}} &= 2[c_{x1}^2 + c_{x2}^2 + \cdots + c_{xi-1}^2 + c_{xi+1}^2 + \cdots + c_{xN}^2 \\ &\quad + c_{1i}^2 + c_{2i}^2 + \cdots + c_{x-1i}^2 + c_{x+1i}^2 + \cdots + c_{Ni}^2] \\ &= 0, \ 1 \leq x, i \leq N. \end{aligned} \quad (48)$$

Since (48) contains only squared terms, all variables in row x other than c_{xi} must be zero. Similarly, on the stable state, the partial derivative $\frac{\partial E_s}{\partial c_{xl}}, \forall l \neq i$, must be zero. Then,

$$\begin{aligned}\frac{\partial E_s}{\partial c_{xl}} &= -2(1-c_{1l})^2(1-c_{2l})^2 \cdots (1-c_{x-1l})(1-c_{x+1l})^2 \\ &\quad \cdots (1-c_{Nl})^2 \\ &= 0, \qquad \forall l \neq i. \end{aligned} \quad (49)$$

From property 2, there exists only one element c_{pl} that is "1" in column l of matrix $[C]_{N \times N}$, and the other elements are all zeros. Similarly, we can prove that any row or any column contains only one "1." Q.E.D.

ACKNOWLEDGMENT

We are grateful to Profs. S. Y. Kung and J. N. Hwang and C. C. Chiang for their helpful discussions and suggestions. Also, we wish to thank anonymous reviewers for their insightful comments.

REFERENCES

[1] S. V. B. Aiyer *et al.*, "A theoretical investigation into the performance of the Hopfield model," *IEEE Trans. Neural Networks,* vol. 1, pp. 204–215, June 1990.
[2] B. Angeniol, G. De La Croix Vaubois, and J.-Y. Le Texier, "Self-organizing feature maps and the travelling salesman problem," *Neural Networks,* vol. I, pp. 289–293, 1988.
[3] G. W. Davis, "Sensitivity analysis in neural net solutions," *IEEE Trans. Syst., Man, Cybern.,* vol. 19, no. 5, pp. 1078–1082, 1989.
[4] R. Durbin, R. Szeliski, and A. Yuille, "An analysis of the elastic net approach to the traveling salesman problem," *Neural Computat.,* vol. 1, p. 384, 1989.
[5] H. C. Fu, J. N. Hwang, S. Y. Kung, W. D. Mao, and J. A. Vlontzos, "A universal digital VLSI design for neural networks," in *Proc. IJCNN'89,* Washington DC, June, 1989.
[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-compleness.* San Francisco, CA: Freeman, 1979.
[7] K. M. Gutzmann, "Combinatorial optimization using a continuous state Boltzmann machine," *in Proc. IEEE First Int. Conf. Neural Networks,* vol. III, San Diego, CA, June 1987, pp. 721–728.
[8] J. J. Hopfield and D. W. Tank, "Neural composition of decisions optimization problems," vol. 55, pp. 141–152, 1985.
[9] E. Horowitz and S. Sahni, "Dynamic programming," in *Fundamentals of Computer Algorithms.* Rockville, MD: Computer Science Press, 1978.
[10] J. J. Johnson, "A neural network approach to the 3-satisfiability problem," *J. Parallel Distributed Comput.,* pp. 435–449, 1989.
[11] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science,* vol. 220, p. 671, 1983.
[12] S. Kirkpatrick and G. Toulouse, "Configuration space analysis of the traveling salesman problem," *J. Phys.,* vol. 42, p. 1277, 1985.
[13] S. Y. Kung, *VLSI Array Processors.* Englewood Cliffs, NJ: Prentice-Hall, 1988.
[14] B. W. Lee and B. J. Sheu, "Combinatorial optimization using competitive-Hopfield neural network," *in Proc. Int. Joint Conf. Neural Networks,* vol. II, Washington DC, Jan. 1990. pp. 627–630,
[15] R. P. Lippmann, "Introduction to computing with neural nets," *IEEE ASSP Mag.,* pp. 4–22, Apr, 1987.
[16] D. G. Luenberger, "Speed of convergence," in *Linear and Nonlinear Programming.* Reading, MA: Addison-Wesley., 1984, ch. 6, pp. 189–192.

[17] ——, "Basic descent methods," in *Linear and Nonlinear Programming.* Reading, MA: Addison-Wesley, 1984, ch. 7, pp. 197–237.
[18] S. Mehta and L. Fulop, "A neural algorithm to solve the Hamiltonian cycle problem," in *Proc. Int. Joint Conf. Neural Networks,* vol. III, San Diego, CA, June 18–22, 1990 pp. 843–849.
[19] H. Muhlenbein, M. Gorges-Schleuter, and O. Kramer, "Evolution algorithms in combinatorial optimization, *Parallel Computat.,* vol. 7, p. 65, 1988.
[20] C. Peterson and B. Soderberg, "A new method for mapping optimization problems onto neural networks," *Int. J. Neural Syst.,* vol. 1, pp. 3–22, 1989.
[21] C. Peterson, "Parallel distributed approaches to combinatorial optimization: Benchmark studies on the traveling salesman problem," *Neural Computat.,* M.I.T. Press Journals, vol. 2, pp. 261–269, 1990.
[22] J. Ramanujam and P. Sadydppan, "Parameter identification for constrained optimization using neural networks," in *Proc. 1988 Connectionist Models Summer School,* Morgan Kaufmann, 1988 pp. 154–161.
[23] K. T. Sun and H. C. Fu, "Solving satisfiability problems with neural networks," in *Proc. IEEE Region 10 Conf. Comput. and Commun. Syst.,* vol. 1, Hong Kong, Sept. 24–27, 1990, pp. 17–22.
[24] ——, "A neural network for solving the satisfiability problems," in *Proc. Int. Comput. Symp. 1990,* National Tsing Hua University, Taiwan, R.O.C., Dec. 17–19, 1990 pp. 757–762.
[25] ——, "An O(n) parallel algorithm for solving the traffic control problem on crossbar switch networks," *Parallel Processing Lett. (PPL),* vol. 1, no. 1, pp. 51–58, 1991.
[26] ——, "A neural network algorithm for solving the traffic control problem in multistage interconnection networks," in *Proc. Int. Joint Conf. Neural Networks (IJCNN-91),* Singapore, Nov. 24–28, 1991 pp. 1136–1141.
[27] ——, "A neural network approach to restrictive channel routing problems," in *Proc. Int. Conf. Artificial Networks ICANN'92.*
[28] G. A. Taglianeri and E. W. Page, "Solving constraint satisfaction problems with neural networks," in *Proc. Int. Conf. Neural Networks,* San Diego, CA, June 1987, pp. 741–747.
[29] R. J. Williams, "Learning the logic of activation functions," in *Parallel Distributed Processing, Vol. 1.* Cambrideg, MA: M.I.T. Press, 1986, ch. 10, pp. 423–443.
[30] G. V. Wilson and G. S. Pawley, "On the stability of the travelling salesman algorithm of Hopfield and Tank," *Biol. Cybern.,* vol. 58, pp. 63–70, 1988.
[31] X. Xu and W. T. Tsai, "An adaptive neural algorithm for traveling salesman problem," in *Proc. Int. Joint Conf. Neural Networks,* vol. I, Washington, DC, Jan. 1990, pp. 716–719.
[32] A. Yuille, "Generalized deformable models, statistical physics, and matching problems," *Neural Computat.,* vol. 2, pp. 1–24, 1990.

K. T. Sun received the B.S. degree in information science from Tunghai University in 1985 and the M.S. and Ph.D. degrees in computer science and information engineering from National Chiao-Tung University in 1987 and 1992, respectively.

He is a Research Associate in Chung Shan Institute of Science and Technology, and an Adjunt Associate Professor at the Information Science Department in Tunghai University, Taiwan, R.O.C. His research interests include logical programming, parallel processing, array processors, fuzzy theorems, and neural networks.

H. C. Fu (S'79–M'80) received the B.S. degree in electrical and communication engineering from National Chiao-Tung University in 1972, and the M.S. and Ph.D. degrees in electrical and computer engineering from New Mexico State University in 1975 and 1981, respectively.

From 1981 to 1983 he was a member of Technical Staff in Bell Laboratories. Since 1983 he has been on the faculty of the Department of Computer Science and Information Engineering, National Chiao-Tung University, Hsinchu, Taiwan, R.O.C. From 1987 to 1988, he was a Director of the Department of Information Management at the Research, Development and Evaluation Commission in Executive Yuan, R.O.C. In the academic year 1988–1989, he was a Visiting Scholar at Princeton University. From 1989 to 1991, he was the Chairman of the Department of Computer Science and Information Engineering at N.C.T.U., where he is currently a Professor there. He is the author of over 40 technical papers in computer engineering. His current research interests are in computer architecture, array processing, parallel processing, and neural computing.

Dr. Fu is a member of the IEEE Computer Society, Phi Tau Phi, and the Eta Kappa Nu Electrical Engineering Honor Society.