Pergamon

Neural Networks, Vol. 7, No. 4, p. 629–641, 1994

0893-6080/94 $6.00 + .00

0893-6080(93)E0021-X

CONTRIBUTED ARTICLE

A Deterministic Annealing Neural Network for Convex Programming

JUN WANG

University of North Dakota

(*Received* 19 *October* 1992; *revised and accepted* 15 *November* 1993)

Abstract—*A recurrent neural network, called a deterministic annealing neural network, is proposed for solving convex programming problems. The proposed deterministic annealing neural network is shown to be capable of generating optimal solutions to convex programming problems. The conditions for asymptotic stability, solution feasibility, and solution optimality are derived. The design methodology for determining design parameters is discussed. Three detailed illustrative examples are also presented to demonstrate the functional and operational characteristics of the deterministic annealing neural network in solving linear and quadratic programs.*

Keywords—Recurrent neural network, Convex programming, Convergence analysis.

1. INTRODUCTION

Convex programming deals with the problem of minimizing a convex function over a convex set of decision variables. Convex programming problems represent a class of widespread optimization problems arising in diverse design and planning contexts. Many large-scale and real-time applications, such as traffic routing and image restoration, require solving large-scale convex programming problems in real time. In such applications, because of the large number of variables and stringent time requirements, existing algorithmic solution methods are usually not efficient.

In recent years, neural networks have been developed as viable alternatives for solving various optimization problems. Although the mainstream of the neural network approach to optimization has focused on non-convex combinatorial optimization problems, a number of neural network paradigms have been developed for solving convex programming problems. For example, Tank and Hopfield (1986) demonstrated the potential of the Hopfield network for solving linear programming problems. Kennedy and Chua (1988a,b) analyzed Tank–Hopfield linear programming circuit and Chua–Lin nonlinear programming circuit from a circuit-theoretic viewpoint and reconciled a modified canonical nonlinear programming circuit that can also be applied to linear programming. Wang (1991) and Wang and Chankong (1992) developed a class of recurrent neural networks for optimization. Maa and Shanblatt (1992) presented an analysis of the stability properties of neural networks for linear and quadratic programming. Although the results of these investigations have demonstrated great potential, most of the proposed neural network models are not guaranteed to yield optimal solutions. Although the capability of the recurrent neural networks for solving convex programming problems (Wang, 1991; Wang & Chankong, 1992) is substantiated theoretically, the neural network models are not very easy to implement in hardware. Further investigations are deemed necessary.

In this paper, a deterministic annealing neural network for solving convex programming problems is proposed. The proposed deterministic annealing neural network is convergent and capable of solving convex programming problems. Compared with the recurrent neural networks (Wang, 1991; Wang & Chankong, 1992), the proposed deterministic annealing neural network is less restrictive in conditions for solution optimality, much simpler in configuration, and hence much easier for hardware implementation. Starting with problem reformulation, the configuration of the proposed deterministic annealing neural network is

Acknowledgements: The author thanks the anonymous reviewers for the insightful comments and constructive suggestions on an earlier version of this paper. This work was partially supported by a summer research professorship award from the Graduate School of the University of North Dakota and by a research grant from the Advancing Science Excellence in North Dakota Program.

Requests for reprints should be sent to the author in the Department of Industrial Technology, University of North Dakota, Grand Forks, ND 58202-7118.

described. Then its convergence properties and the design principles are discussed. Finally, simulation results are given.

This paper is organized in eight sections. In Section 2, the reformulation of the convex program under study is delineated. In Section 3, the dynamics of the proposed deterministic annealing neural network are discussed. In Section 4, the architectures of the deterministic annealing neural network are described. In Section 5, the analytical results on the convergence properties of the proposed deterministic annealing neural network for convex programming are presented. In Section 6, design principles for selection of parameters in the deterministic annealing neural network are addressed. In Section 7, the performance of the proposed deterministic annealing neural network is demonstrated via illustrative examples. Finally, the paper is concluded in Section 8.

2. PROBLEM REFORMULATION

In this study, we consider a general constrained convex programming problem described as follows:

$$\text{minimize} \quad f(v), \tag{1}$$

$$\text{subject to} \quad g(v) \le 0, \tag{2}$$

$$h(v) = 0, \tag{3}$$

where $v \in \mathbf{R}^N$ is a column vector of decision variables, $f:\mathbf{R}^N \to \mathbf{R}$ is a convex objective function to be minimized, $g:\mathbf{R}^N \to \mathbf{R}^m$ is an m-tuple convex function used to define m inequality constraints, and $h:\mathbf{R}^N \to \mathbf{R}^n$ is an n-tuple linear function used to define n equality constraints. The inequality constraints (2) and equality constraints (3) characterize a convex feasible region in the N-dimensional Euclidean space. Let $\hat{\mathbf{V}}$ denote the convex feasible region and $v^* = [v_1^*, v_2^*, \ldots, v_N^*]^T$ denote an optimal solution that minimizes $f(v)$ over $\hat{\mathbf{V}}$ where the superscript T denotes the transpose operator; that is, $\hat{\mathbf{V}} = \{v \in \mathbf{R}^N \mid g(v) \le 0, h(v) = 0\}$ and $v^* = \arg\min_{v \in \hat{\mathbf{V}}} f(v)$. For an obvious reason, we assume that the feasible region $\hat{\mathbf{V}}$ is nonempty and that the objective function value $f(v)$ is bounded from below over $\hat{\mathbf{V}}$. Furthermore, for operational reasons, we assume that $f(v)$, $g(v)$, and $h(v)$ are differentiable with respect to v over $\mathbf{R}^N$.

Two classes of important convex programming problems are the linear program (LP) and the quadratic program (QP). In the standard form of a linear program, $f(v) = c^T v$, and $g(v) = -v$, $h(v) = Av - b$, where c is an N-dimensional column vector, A is an $n \times N$ matrix, and b is an n-dimensional column vector. In a quadratic program, $f(v) = v^T Q v/2 + c^T v$, and $g(v) = -v$, $h(v) = Av - b$, where Q is an $N \times N$ matrix, and A, b, and c are the same as those defined in an LP.

For a constrained optimization problem to be solved through neural computing, it is usually necessary to characterize the feasible set $\hat{\mathbf{V}}$ using a penalty function $p(v)$ to penalize a constraint violation. A penalty function characterizes the feasible region of an optimization problem with a functional equality. The basic requirement for the penalty function is nonnegativity; precisely

$$p(v)\begin{cases} = 0, & \text{if } v \in \hat{\mathbf{V}} \\ > 0, & \text{otherwise.} \end{cases} \tag{4}$$

Additional requirements for the penalty function $p(v)$ are convexity and differentiability. If $p(v)$ is convex and differentiable, then $\min_v p(v) = 0$ and $\forall v \in \hat{\mathbf{V}}$, $\nabla p(v) = 0$.

There are basically two ways to construct the penalty function based on the functional constraints: functional transformation and state augmentation.

(i) *Functional transformation:* Functional transformation involves a mapping of an inequality constraint or an equality constraint to an equality. Specifically,

$$p_g(v)\begin{cases} = 0, & g(v) \le 0; \\ > 0, & g(v) > 0 \end{cases} \tag{5}$$

$$p_h(v)\begin{cases} = 0, & h(v) = 0; \\ > 0, & h(v) \ne 0 \end{cases} \tag{6}$$

There are several logical choices of $p_g(v)$ and $p_h(v)$. For example, for $g(v) \le 0$, the integral of the Heaviside function can be used; for $h(v) = 0$, $\|h(v)\|_2^2$ can be used where $\|\cdot\|_2$ is the Euclidean norm. Namely,

$$p_g(v) = \sum_{i=1}^{m} \int_{-\infty}^{g_i(v)} H(x)\, dx, \tag{7}$$

$$p_h(v) = \frac{1}{2} \sum_{j=1}^{n} h_j(v)^2, \tag{8}$$

where $H(\cdot)$ is the heaviside function defined as follows:

$$H(x) = \begin{cases} 0, & x < 0; \\ x, & x > 0. \end{cases} \tag{9}$$

Because both $g(v)$ and $h(v)$ are differentiable, $p_g(v)$ and $p_h(v)$ are differentiable with respect to v. Based on the convexity of $g(v)$ and $h(v)$, the convexity of $p_g(v)$ and $p_h(v)$ can be proven (cf. the proof in the Appendix).

(ii) *State augmentation:* State augmentation is an approach through converting the inequality constraints to equality ones by adding slack or surplus variables. First, the inequality constraint, $g(v) \le 0$, can be converted to equality by adding m slack variables; for example,

$$g(v) + v^+ = 0, \tag{10}$$

where $v^+ \in \mathbf{R}^m$ and $v^+ \ge 0$. For example, in an LP problem, letting $g(v) = Av - b$, then $g(v) + v^+ = Av + v^+ - b = 0$. If $g(v) = r(v) - s$ where $r(v) \ge 0$, and

$s \in \mathbf{R}^m$, then a surplus variable vector $v^- \in \mathbf{R}^m$ can also be applied as follows:

$$r(v) - s \circ v^- = 0, \tag{11}$$

where $0 \le v_i^- \le 1$ $(i = 1, 2, \ldots, m)$ and $\circ$ denotes the operator for array multiplication (element-by-element product). Once all the inequality constraints are converted to equality constraints, a penalty function can be constructed in a way similar to that of eqn (8); that is,

$$p_g^{(1)}(v) = \tfrac{1}{2}[g(v) + v^+]^T[g(v) + v^+], \tag{12}$$

$$p_g^{(2)}(v) = \tfrac{1}{2}[r(v) - s \circ v^-]^T[r(v) - s \circ v^-], \tag{13}$$

$$p_h(v) = \tfrac{1}{2}h(v)^T h(v). \tag{14}$$

Obviously, $p_g(v)$ and $p_h(v)$ are also differentiable and convex with respect to v.

The penalty function can be constructed based on $p_g(v)$ and $p_h(v)$; for example,

$$p(v) = p_g(v) + p_h(v). \tag{15}$$

Because $p_g(v)$ and $p_h(v)$ are convex and differentiable, $p(v)$ is also convex and differentiable. The convex programming problem described in eqns (1), (2), and (3) thus can be reformulated as the following equivalent convex programming problem:

$$\text{minimize} \quad f(v), \tag{16}$$

$$\text{subject to} \quad p(v) = 0. \tag{17}$$

3. NETWORK DYNAMICS

The dynamical equations of the deterministic annealing neural network can be described as follows:

$$\frac{du(t)}{dt} = -T(t)\nabla f[v(t)] - \nabla p[v(t)], \tag{18}$$

$$v(t) = F[u(t)], \tag{19}$$

where $u(t) \in \mathbf{R}^N$ is a column vector of instantaneous net inputs to neurons, $v(t) \in \mathbf{V} \subset \mathbf{R}^N$ is a column vector of activation states corresponding to the decision variables, $\mathbf{V}$ is the activation state space, $u(0)$ and $v(0)$ are unspecified, $T(t)$ is a time-varying temperature parameter, $F: u \mapsto v$ is an N-tuple activation function, and ∇ is the gradient operator.

Because the temperature parameter is a function of time, the proposed deterministic annealing neural network is virtually a time-varying dynamic system. Equation (18) defines the neural network dynamics, in which the first term of the right-hand side enforces the minimization of the objective function and the second term enforces penalization of violation of constraint (17). Equation (19) is the activation function that defines the sensitivity and range of the activation state $v(t)$.

The major difference between the existing recurrent neural networks for optimization and the proposed deterministic annealing neural network lies in essentially the treatment of their penalization procedures for violation of constraints. In the Hopfield network for optimization, the penalization is realized by constant penalty parameters (Hopfield & Tank, 1985, Tagliarini, Christ, & Page, 1992). As evidenced by many empirical studies, the steady state of the Hopfield network may not represent a feasible solution or an optimal solution to the optimization problem. In the recurrent neural networks proposed earlier (Wang, 1991; Wang & Chankong, 1992), the penalization is realized by a monotonically increasing penalty parameter. Because the penalty parameter is supposed to be unbounded above, it is not easy to implement physically. In the proposed deterministic annealing neural network, the penalization is realized by using a time-varying temperature parameter. The role of the time-varying temperature parameter will be more apparent in Section 5.

4. NETWORK ARCHITECTURE

To solve an optimization problem via neural computation, the key is to cast the problem into the architecture of a recurrent neural network for which the stable state represents the solution to the optimization problem. As shown in eqn (18), the architecture of the deterministic annealing neural network depends on specific forms of $f(v)$ and $p(v)$ [i.e., $f(v)$, $g(v)$, $h(v)$, and the method for constructing $p(v)$].

If the functional transformation method [e.g., eqns (7) and (8)] is used to construct a penalty function, then

$$\nabla p_g(v) = \sum_{i=1}^{m} H[g_i(v)]\nabla g_i(v), \tag{20}$$

$$\nabla p_h(v) = \sum_{j=1}^{n} h_j(v)\nabla h_j(v). \tag{21}$$

Obviously, $\forall v \in \hat{\mathbf{V}}$, $\nabla p_g(v) = \nabla p_h(v) = 0$. Furthermore, because $H(x)$ is continuous, $\nabla p_g(v)$ is also continuous.

If the state augmentation method is used to construct a penalty function based on $g(v)$, then the dimension of the activation-state vector increases from N to $N + m$; so does the size of the neural network architecture. According to eqns (12) and (13),

$$\nabla p_g^{(1)}(v^+) = g(v) + v^+, \tag{22}$$

$$\nabla p_g^{(2)}(v^-) = -s \circ r(v) + s \circ s \circ v^-. \tag{23}$$

In view of the fact that $f(v)$ has nothing to do with the augmented-state variables, eqns (22) and (23) imply that there is no connection between the neurons cor-

responding to the augmented-state variables $v^+(t)$ or $v^-(t)$ in a neural network architecture.

In the standard form of LP, for example, $f(v) = c^T v$, $g(v) = -v$, and $h(v) = Av - b$. Because the nonnegativity constraint, $v \geq 0$, can be realized by using a nonnegative activation function $F[u(t)] \geq 0$, only $h(v)$ needs to be used in the penalty function. Let $p(v) = p_h(v) = h(v)^T h(v)/2 = (Av - b)^T(Av - b)/2$, thus $\nabla p(v) = \nabla p_h(v) = A^T Av - A^T b$. Equation (18) becomes

$$\frac{du(t)}{dt} = -A^T Av(t) + A^T b - T(t)c. \tag{24}$$

In the standard form of QP, $f(v) = v^T Qv/2 + c^T v$, $g(v) = -v$, and $h(v) = Av - b$. Letting Q be symmetric and positive semidefinite, similar to the case of LP, eqn (18) becomes

$$\frac{du(t)}{dt} = -[A^T A + T(t)Q]v(t) + A^T b - T(t)c. \tag{25}$$

There are two ways to map the dynamical equations, eqn (24) or (25), to a neural network architecture. The first method results in a single-layer recurrent neural network. The single-layer architecture of the deterministic annealing neural network for convex programming consists of N laterally connected neurons, as shown in Figure 1. Each neuron represents one decision variable. The lateral connection weight matrix is defined as $-A^T A$ for an LP and $-A^T A - T(t)Q$ for a QP. The biasing threshold vector of the neurons is defined as $A^T b - T(t)c$ for both LP and QP. In other words, the connection weight w_{ij} from neuron j to neuron i is defined as $-\sum_{k=1}^{n} a_{ki}a_{kj}$ for an LP and $-\sum_{k=1}^{n} a_{ki}a_{kj} - T(t)q_{ij}$ for a QP, the biasing threshold θ_i of neuron i is defined as $\sum_{k=1}^{n} a_{ki}b_k - T(t)c_i$, where a_{ij} and q_{ij} are the element in the ith row and the jth column of A and Q respectively, b_i and c_i are the ith elements of b and c, respectively. Because Q is symmetric, the connection weight matrices are symmetric for both LP and QP (i.e., $\forall i, j$, $w_{ij} = w_{ji}$). If Q is positive semidefinite and $T(t) \geq 0$, then the eigenvalues of the weight matrices are always real and nonpositive for both LP and QP.

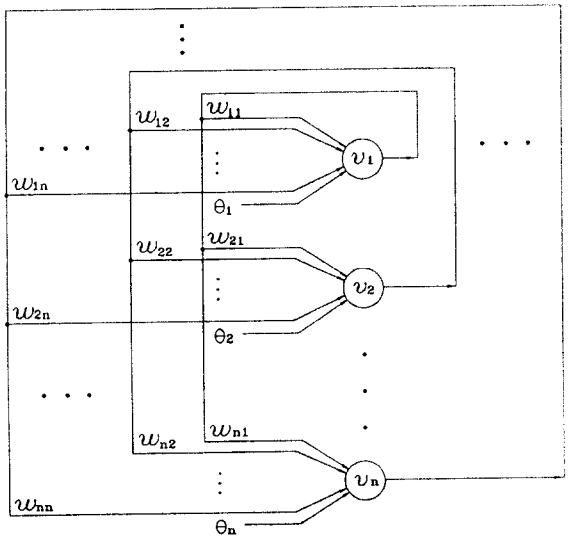

FIGURE 1. Single-layer architecture of the deterministic annealing neural network.

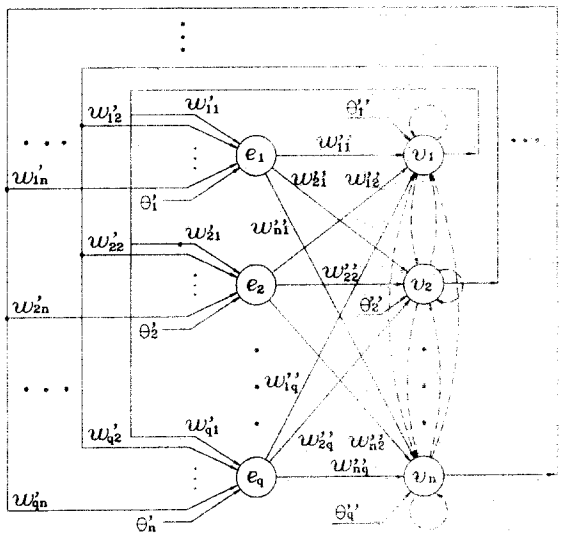

FIGURE 2. Two-layer architecture of the deterministic annealing neural network.

Letting $e(t) = Av(t) - b$, eqns (24) and (25) can be decomposed, respectively, as

$$\frac{du(t)}{dt} = -A^T e(t) - T(t)c, \tag{26}$$

$$\frac{du(t)}{dt} = -A^T e(t) - T(t)Qv(t) - T(t)c. \tag{27}$$

The second approach for mapping the dynamical equations to a neural network architecture results in a two-layer architecture composed of an output layer and an input layer, as shown in Figure 2. The output layer consists of N neurons and the input layer consists of n neurons. The state vectors of the input and output layers are denoted by $e(t)$ and $v(t)$, respectively. The output state vector $v(t)$ represents the decision variable vector v and the input state vector $e(t)$ represents the error vector that measures the solution feasibility of the output-state vector. The connection weight matrix from the input layer to output layer is defined as $-A^T$ and the weight matrix from output layer to input layer is A. In the LP case, there is no lateral connection among input and output neurons. In the QP case, there is no lateral connection among input neurons, and the connection weight matrix among output neurons is $-T(t)Q$. The bias vectors are $-b$ in the input layer and $-T(t)c$ in the output layer, respectively, for both the LP and QP neural networks. The advantage of the second configuration is that the connection weight matrices and bias vectors can be determined directly from

the coefficient matrices and vectors without involving matrix multiplication.

5. CONVERGENCE ANALYSIS

In this section, analytical results on the stability of the proposed deterministic annealing neural network and feasibility and optimality of the steady-state solutions to convex programming problems are presented.

5.1. Asymptotic Stability

THEOREM 1. *Assume that the Jacobian matrix of* $F[u(t)]$ *exists and is positive semidefinite. If the temperature parameter* $T(t)$ *is nonnegative, strictly monotone decreasing for* $t \geq 0$, *and approaches zero as time approaches infinity, then the deterministic annealing neural network is asymptotically stable in the large; that is,* $\forall t \geq 0$, $J\{F[u(t)]\}$ *is p.s.d.,* $T(t) \geq 0$, $dT(t)/dt < 0$, *and* $\lim_{t\to\infty} T(t) = 0$ *imply* $\forall v(0) \in V$, $\exists \bar{v} \in V$ *such that* $\lim_{t\to\infty} v(t) = \bar{v}$, *where* $J\{F[u(t)]\}$ *is the Jacobian of* $F[u(t)]$ *and* $\bar{v}$ *is a steady-state of* $v(t)$.

Proof. Because $f(v)$ is bounded from below, there exists M such that $-\infty < M < \inf_{v\in V} f(v)$. Let

$$E[v(t), T(t)] = T(t)\{f[v(t)] - M\} + p[v(t)]. \quad (28)$$

Because $\forall t \geq 0$, $T(t) \geq 0$, $f[v(t)] - M > 0$, and $p[v(t)] \geq 0$, $\forall v \in V$, $E[v(t), T(t)] \geq 0$. Furthermore, because $\lim_{t\to\infty} T(t) = 0$ and $\min_v p(v) = 0$, $\min_{v,T} E[v, T] = 0$. To facilitate the stability analysis, let us consider $T(t)$ as a dummy-state variable for the time being; that is, let $\tilde{v}(t)^T = [v(t)^T, T(t)]$ be the augmented-state vector. In view of the fact that $f(v)$ and $p(v)$ are convex and $T(t) \in [0, \infty)$, $E \to \infty$ as $\|\tilde{v}\| \to \infty$; that is, E is radially unbounded. Because $-du(t)/dt = T(t)\nabla f[v(t)] + \nabla p[v(t)]$ and $v(t) = F[u(t)]$ according to eqns (18) and (19), $dv(t)/dt = J\{F[u(t)]\}du(t)/dt$ and

$$\begin{aligned}
\frac{dE[v(t), T(t)]}{dt} &= \{f[v(t)] - M\}\frac{dT(t)}{dt} \\
&\quad + T(t)\frac{df[v(t)]}{dt} + \frac{dp[v(t)]}{dt} \\
&= \{f[v(t)] - M\}\frac{dT(t)}{dt} \\
&\quad + \{T(t)\nabla f[v(t)] + \nabla p[v(t)]\}^T \frac{dv(t)}{dt} \\
&= \{f[v(t)] - M\}\frac{dT(t)}{dt} - \frac{du(t)^T}{dt}\frac{dv(t)}{dt} \\
&= \{f[v(t)] - M\}\frac{dT(t)}{dt} \\
&\quad - \frac{du(t)^T}{dt} J\{F[u(t)]\}\frac{du(t)}{dt}.
\end{aligned}$$

Because $J\{F[u(t)]\}$ is positive semidefinite, $du(t)^T/dtJ\{F[u(t)]\}du(t)/dt \geq 0$. Furthermore, because $f[v(t)] - M > 0$ and $dT(t)/dt < 0$, $dE[v(t), T(t)]/dt < 0$. Therefore, $E[v(t), T(t)]$ is positive definite and radially unbounded, $dE[v(t), T(t)]/dt$ is negative definite. According to the Lyapunov's theorem, the deterministic annealing neural network is asymptotically stable in the large. ■

5.2. Solution Feasibility

For the deterministic annealing neural network to realize a solution procedure for a convex programming problem, its steady state must represent, at least, a feasible solution to the problem. Toward this end, the following theorem is derived.

THEOREM 2. *Assume that* $J\{F[u(t)]\}$ *exists and is positive semidefinite. If* $T(t) \geq 0$, $dT(t)/dt < 0$ *and* $\lim_{t\to\infty} T(t) = 0$, *then the steady state of the deterministic annealing neural network represents a feasible solution to the convex programming problem described in eqns* (16) *and* (17); *that is,* $\bar{v} \in \hat{V}$.

Proof. The proof of Theorem 1 shows that the energy function $E[v(t), T(t)]$ is positive definite and strictly monotone decreasing with respect to time t, which implies $\lim_{t\to\infty} E[v(t), T(t)] = 0$. Because $\lim_{t\to\infty} T(t) = 0$, $\lim_{t\to\infty} E[v(t), T(t)] = \lim_{t\to\infty} T(t)\{f[v(t)] - M\} + p[v(t)] = \lim_{t\to\infty} p[v(t)]$. Therefore, $\lim_{t\to\infty} E[v(t), T(t)] = \lim_{t\to\infty} p[v(t)] = 0$. Because $p(v)$ is continuous, $\lim_{t\to\infty} p[v(t)] = p[\lim_{t\to\infty} v(t)] = p(\bar{v}) = 0$.

REMARK. Theorems 1 and 2 show that the asymptotic stability of the deterministic annealing neural network implies the solution feasibility of the steady state.

5.3. Solution Optimality

Theorems 1 and 2 ensure the asymptotic stability and solution feasibility. The optimality of the steady-state solutions, however, is not guaranteed and deserves further exploration.

There are two possibilities for the locality of the optimal solution v^*: an interior point or a boundary point of the feasible region $\hat{V}$. If v^* is an interior point of $\hat{V}$, then v^* is the same as the global minimum of the objective function $f(v)$, hence $\nabla f(v^*) = \nabla p(v^*) = 0$, because both $f(v)$ and $p(v)$ are convex and differentiable. In this case, according to eqn (18), a steady state of the deterministic annealing neural network is an optimal solution if the temperature parameter $T(t)$ sustains sufficiently long. The solution process for this case is trivial. In most convex programming problems, such as linear programs, v^* is a boundary point of $\hat{V}$. In this case, $\forall v \in \hat{V}$, $\nabla f(v) \neq 0$. Theorem 3 provides a sufficient condition for the deterministic annealing

neural network to solve convex programming problems in which the optimal solutions are boundary points.

THEOREM 3. *Assume that* $\forall t \geq 0$, $J\{F[u(t)]\} \neq 0$ *and is positive semidefinite, and* $\nabla f[v(t)] \neq 0$. *If* $dT(t)/dt < 0$, $\lim_{t\to\infty} T(t) = 0$, *and*

$$T(t) \geq \max\left\{0, \frac{\begin{array}{r}\nabla p[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)] \\ -\nabla f[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)]\end{array}}{\begin{array}{r}\nabla f[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)] \\ -\nabla p[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)]\end{array}}\right\}, \quad (29)$$

then the steady state of the deterministic annealing neural network represents an optimal solution to the convex programming problem described in eqns (16) *and* (17), *that is,* $\bar{v} = \arg\min_{v\in\hat{\mathbf{V}}} f(v)$.

Proof. According to Theorems 1 and 2, the steady state of the deterministic annealing neural network represents a feasible solution to the convex programming problem: that is, $\lim_{t\to\infty} v(t) = \bar{v} \in \hat{\mathbf{V}}$ and $p(\bar{v}) = 0$. Because $E[v(t), T(t)] = T(t)\{f[v(t)] - M\} + p[v(t)]$ and $\nabla E[v(t), T(t)] = T(t)\nabla f[v(t)] + \nabla p[v(t)]$, $du(t)/dt = -\nabla E[v(t), T(t)]$, where $\nabla E[v(t), T(t)] = \{\partial E[v(t), T(t)]/\partial v_1, \partial E[v(t), T(t)]/\partial v_2, \ldots, \partial E[v(t), T(t)]/\partial v_N\}^T$ is the gradient of $E[v(t), T(t)]$ with respect to v. Moreover, because $v(t) = F[u(t)]$ and $du(t)/dt = -\nabla E[v(t), T(t)]$, $dv(t)/dt = J\{F[u(t)]\}du(t)/dt = -J\{F[u(t)]\}\nabla E[v(t), T(t)]$. In view of the fact that $J\{F[u(t)]\} \neq 0$ and is positive semidefinite, the state $v(t)$ of the deterministic annealing neural network approaches a local minimum of the energy function in a gradient descent manner. Furthermore, because $T(t) \geq 0$ and both $f(v)$ and $p(v)$ are convex, $E[v(t), T(t)]$ is also convex. The convexity property of $E[v(t), T(t)]$ guarantees that a local minimum of E is always a global one. Because of the presence of $T(t)$ in $E[v(t), T(t)]$, the global minimum is time-varying. Equation (29) implies

$$\begin{aligned} &T(t)\{\nabla f[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)] \\ &\quad - \nabla p[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)]\} \\ &\geq \nabla p[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)] \\ &\qquad - \nabla f[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)]. \end{aligned}$$

Rearranging the above inequality, we have

$$\begin{aligned} &T(t)\nabla f[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)] \\ &\quad + \nabla f[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)] \\ &\quad - \{T(t)\nabla p[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)] \\ &\qquad + \nabla p[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)]\} \geq 0; \end{aligned}$$

that is,

$$\begin{aligned} &\nabla f[v(t)]^T J\{F[u(t)]\}\{T(t)\nabla f[v(t)] + \nabla p[v(t)]\} \\ &\quad - \nabla p[v(t)]^T J\{F[u(t)]\}\{T(t)\nabla f[v(t)]\} \\ &\qquad + \nabla p[v(t)]\} \geq 0. \end{aligned}$$

Substituting eqn (18) and $dv(t)/dt = J\{F[u(t)]\} \times du(t)/dt$ into the above inequality, we have

$$\begin{aligned} &\nabla f[v(t)]^T J\{F[u(t)]\}\frac{du(t)}{dt} \\ &\quad - \nabla p[v(t)]^T J\{F[u(t)]\}\frac{du(t)}{dt} \\ &\qquad = \nabla f[v(t)]^T \frac{dv(t)}{dt} - \nabla p[v(t)]^T \frac{dv(t)}{dt} \leq 0. \end{aligned}$$

Note that $\nabla f[v(t)]^T dv(t)/dt = df[v(t)]/dt$, and $\nabla p[v(t)]^T dv(t)/dt = dp[v(t)]/dt$, eqn (29) implies $df[v(t)]/dt \leq dp[v(t)]/dt$. Furthermore, $df[v(t)]/dt \leq dp[v(t)]/dt$ implies $f[v(t'')] - f[v(t')] \leq p[v(t'')] - p[v(t')]$ for any $t' \leq t''$. Let t^* be the time associated with an optimal solution v^*. We have $f[v(\infty)] - f[v(t^*)] \leq p[v(\infty)] - p[v(t^*)]$; that is, $f(\bar{v}) - f(v^*) \leq p(\bar{v}) - p(v^*)$. Because $p(\bar{v}) = p(v^*) = 0$, $f(\bar{v}) \leq f(v^*)$. Also, because $\bar{v} \in \hat{\mathbf{V}}$ and $v^* = \arg\min_{v\in\hat{\mathbf{V}}} f(v)$, $f(\bar{v}) \geq f(v^*)$ by definition of v^*. Consequently, $f(\bar{v}) = f(v^*) = \min_{v\in\hat{\mathbf{V}}} f(v)$. ■

REMARK. Theorem 3 gives a lower bound on the temperature parameter $T(t)$. Note that $T(t) \geq 0$, $dT(t)/dt < 0$, and $\lim_{t\to\infty} T(t) = 0$ imply $T(t) \not\equiv 0$. Because $\lim_{t\to\infty} p[v(t)] = p(\bar{v}) = \min_v p(v) = 0$, $p(v)$ is convex, and $\nabla p[v(t)]$ is continuous, $\nabla p(\bar{v}) = \lim_{t\to\infty} \nabla p[v(t)] = 0$. Hence

$$\lim_{t\to\infty} \frac{\begin{array}{r}\nabla p[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)] \\ -\nabla f[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)]\end{array}}{\begin{array}{r}\nabla f[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)] \\ -\nabla p[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)]\end{array}} = 0. \quad (30)$$

In addition, Theorem 3 relaxes the infeasibility requirement for initial states of the recurrent neural networks proposed earlier (Wang, 1991; Wang & Chankong, 1992).

6. DESIGN METHODOLOGY

The convergence analysis in the preceding section reveals the desirable properties of the deterministic annealing neural network for convex programming. In this section, design principles are discussed, based on the results of convergence analysis discussed above, for determining the activation function and temperature parameter of the deterministic annealing neural network.

6.1. Activation Function

In light of Theorems 1, 2, and 3, the activation function F should be differentiable and its Jacobian should be nonzero and positive semidefinite at any time. The existing activation functions are all uncoupled; that is, $v_i(t) = F_i[u_i(t)]$ for all i. In this case, $J\{F[u(t)]\} = \text{diag}\{dF_i[u_i(t)]/du_i | i = 1, 2, \ldots, N\}$ and the nonzero

and positive semidefinite requirements on $J\{F[u(t)]\}$ can be replaced by strictly monotone increasing F_i for all i. The activation function F determines the domain of the state variables $v(t)$ (i.e., the state space $\mathbf{V}$) and depends on the feasible region $\hat{\mathbf{V}}$. Specifically, it is necessary for $\hat{\mathbf{V}} \subseteq \mathbf{V}$ unless prior knowledge on the locality of v^* is available. Any explicit bound on decision variable vector v can be realized by properly selecting the range of F. Figure 1 of Wang (1991) provides a set of potential choices of the activation functions for different domain of state variables.

In the typical sigmoid activation function, $F_i[u_i(t)] = v_{\max}/[1 + \exp(-\xi u_i(t))]$ where $v_{\max} > 0$. To ensure increasing monotonicity, parameter ξ must be positive. Moreover, parameter ξ can be used to control the slope of the sigmoid activation functions, as shown in Figure 3a. Because the activation sensitivity increases as ξ increases, a large value of parameter ξ is usually selected to expedite the convergence of the deterministic annealing neural network. The range of the sigmoid activation function is defined as $[0, v_{\max}]$, because the infimum and supremum of the activation state are 0 and $v_{\max}$, respectively. The state space of the deterministic annealing neural network is then defined as a hypercube $\mathbf{V} = [0, v_{\max}]^N$, where $v_{\max}$ depends on the feasible region of decision variables. Furthermore, $v_{\max}$ is also a factor for sensitivity of the sigmoid activation function, as illustrated in Figure 3b. Because the slope of the sigmoid activation function $F_i[u_i(t)]$ increases as $v_{\max}$ increases and the upper bound $v_{\max}$ needs not to be tight unless an explicit upper bound on v exists. A large upper bound could increase the sensitivity of activation and hence increase the convergence rate as long as the hypercube-state space contains the feasible region. Therefore, the rule for determining $v_{\max}$ is to set it to maximum such that $\hat{\mathbf{V}} \subseteq \mathbf{V}$. If there is no explicit bound on decision variable vector v, then a linear activation function, $F[u(t)] = \xi u(t)$, can be used where $\xi > 0$ is also a scaling constant. In this case, $\mathbf{V} = \mathbf{R}^N$.

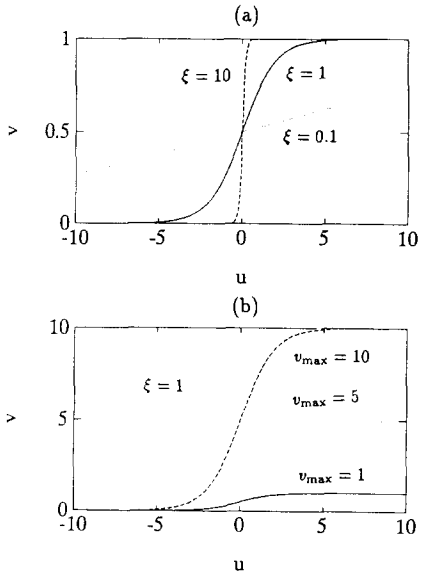

FIGURE 3. Effects of ξ and $v_{\max}$ on sigmoid activation function.

6.2. Temperature Parameter

Because $T(t)$ represents the *annealing* effect, the determination of the cooling temperature parameter $T(t)$ is very important for solution optimality. Theorems 1 and 2 provide the necessary conditions for selecting the temperature parameter by requiring the nonnegativity and decreasing monotonicity of $T(t)$ with respect to time. The following polynomial, exponential, and logarithmic functions of t are three simple examples of open-loop $T(t)$.

$$T(t) = \beta(1 + t)^{-\eta}, \tag{31}$$

$$T(t) = \beta\alpha^{-\eta t}, \tag{32}$$

$$T(t) = \beta[\log_\alpha(t + \alpha)]^{-\eta}, \tag{33}$$

where $\alpha > 1$, $\beta > 0$, and $\eta > 0$ are constant parameters. Parameters β and η can be used to scale the temperature parameter.

Theorem 3 provides a sufficient condition for determining $T(t)$. If $J\{F[u(t)]\}$ is a symmetric matrix such as the aforementioned diagonal matrix, then $\nabla f[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)] = \nabla p[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)]$, hence eqn (29) becomes

$$T(t) \geq \max\left\{0, \frac{\nabla p[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)] - \nabla f[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)]}{\nabla f[v(t)]^T J\{F[u(t)]\}\nabla f[v(t)] - \nabla f[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)]}\right\}. \tag{34}$$

Because $J\{F[u(t)]\}$ is positive semidefinite, $\nabla p[v(t)]^T J\{F[u(t)]\}\nabla p[v(t)] \geq 0$ and $\nabla f[v(t)]^T \times J\{F[u(t)]\}\nabla f[v(t)] \geq 0$. In the cases of LP and QP, if $v(0)$ and $J\{F[u(t)]\}$ is symmetric, as in most of neural network models, then eqn (29) implies

$$\max_{t\geq 0} T(t) = T(0) \geq \max\left\{0, \frac{b^T A J\{F[u(t)]\}(c + A^T b)}{c^T J\{F[u(t)]\}(c + A^T b)}\right\}. \tag{35}$$

In addition, if $F(u) = [F_1(u_1), F_2(u_2), \ldots, F_N(u_N)]$ and $F_1 = F_2 = \ldots = F_N$, then eqn (35) becomes

$$\max_{t\geq 0} T(t) = T(0) \geq \max\left\{0, \frac{b^T A(c + A^T b)}{c^T(c + A^T b)}\right\}. \tag{36}$$

As discussed in Section 3, the terms $-T(t)\nabla f[v(t)]$ and $-\nabla p[v(t)]$ in the right-hand side of the dynamical eqn (18) represent, respectively, the objective minimization effect and the constraint satisfaction effect. Because a feasible solution is not necessarily an optimal one, it is essential for the first term not to vanish until $v(t)$ reaches an optimal solution. Theorem 3 suggests

essentially that $T(t)$ sustain sufficiently long to obtain an optimal solution. In other words, similar to the temperature parameter in the stochastic counterpart, the value of the temperature parameter in the deterministic annealing neural network should decrease slowly. A simple way to implement this requirement is to select a sufficiently small value of η in eqns (31), (32), or (33). Because eqn (29) is not a necessary condition, violation of this condition could still result in an optimal solution. As will be shown in the illustrative examples in the ensuing section, optimal solutions can still be obtained when $df[v(t)]/dt \not\leq dp[v(t)]/dt$.

An exponential function, $\beta \exp(-\eta t)$ is used implicitly as the temperature parameter in the recurrent neural networks for solving the assignment problem and linear programming problems (Wang, 1992, 1993); that is, $a = e$ in eqn (32) where $e = 2.71838\ldots$. In this case, if the first term of the right-hand side of dynamical eqn (18) is dominant as time elapses, then the convergence rate of the recurrent neural network is exclusively dictated by η and the deterministic annealing neural network needs approximately $5/\eta$ units of time to converge, because η is the time constant of the temperature parameter. This feature enables one to control or determine the convergence rate of the deterministic annealing neural network.

Because $\beta = T(0)$ in eqns (31)–(33), parameter β defines the maximal and initial value of temperature parameter $T(t)$. According to eqn (34),

$$\beta \geq \max\left\{0, \frac{\begin{array}{l}\nabla p[v(0)]^T J\{F[u(0)]\}\nabla p[v(0)] \\ -\nabla f[v(0)]^T J\{F[u(0)]\}\nabla p[v(0)]\end{array}}{\begin{array}{l}\nabla f[v(0)]^T J\{F[u(0)]\}\nabla f[v(0)] \\ -\nabla p[v(0)]^T J\{F[u(0)]\}\nabla f[v(0)]\end{array}}\right\}. \quad (37)$$

In addition, β is a design parameter related to convergence rate of the deterministic annealing neural network and can be used to balance the penalization effect on constraint violation and minimization effect on the objective function. In general, a small β results in an underpenalization and a large β results in an overpenalization of constraint violations. The convergence is slow in either case. An excessively small β could even result in convergence to a suboptimal solution, as will be shown in Example 2. In other words, a properly selected parameter β can make the deterministic annealing neural network converge rapidly. If $p_g(v)$, $p_h(v)$, and $f(v)$ are normalized in the same scale, then a design rule is to set $\beta \approx 1$.

7. ILLUSTRATIVE EXAMPLES

Simulations have been performed using a simulator for a number of sample problems. The following illustrative examples are to demonstrate the operating characteristics of the proposed deterministic annealing neural network in solving linear and quadratic programs. Specifically, Example 1 shows the performance of the deterministic annealing neural network with different initial states, Example 2 shows the effects of the design parameters β, η, and ξ on the performance of the deterministic annealing neural network, and Example 3 shows the performance of the neural network in solving a quadratic program.

EXAMPLE 1. Consider the following linear program with two decision variables and three functional constraints.

$$\begin{aligned} \min \quad & -2v_1 - 3v_2, \\ \text{s.t.} \quad & v_1 + 2v_2 \leq 3, \\ & -v_1 + 2v_2 \leq 2, \\ & 3v_1 + v_2 \leq 5, \\ & v_1 \geq 0,\ v_2 \geq 0. \end{aligned}$$

The optimal solution of this problem is $v^* = [1.4, 0.8]^T$ with the minimal objective value $f(v^*) = -5.2$. In this example, the inequality constraints are converted to equality constraints by adding three slack-state variables v_3, v_4, and v_5.

$$\begin{aligned} & v_1 + 2v_2 + v_3 = 3, \\ & -v_1 + 2v_2 + v_4 = 2, \\ & 3v_1 + v_2 + v_5 = 5, \end{aligned}$$

where $v_1, v_2, v_3, v_4, v_5 \geq 0$.

Numerical simulations have been performed using the developed simulator based on $v_i(t) = v_{\max}/[1 + \exp(-\xi u_i(t))]$, $T(t) = \beta(1 + t)^{-\eta}$, $v_{\max} = 2$, $\xi = 10^5$, $\beta = 1$, $\eta = 10^3$, and $\Delta t = 10^{-6}$. The results of the simulations show that the steady states of the deterministic annealing neural network can always accurately generate the optimal solution starting from four different corners of the 2×2 square on the $v_1 - v_2$ plane; that is, $\bar{v} = [1.400000, 0.800001]^T$ and $f(\bar{v}) = -5.200001$. Figures 4 and 5 illustrate, respectively, the

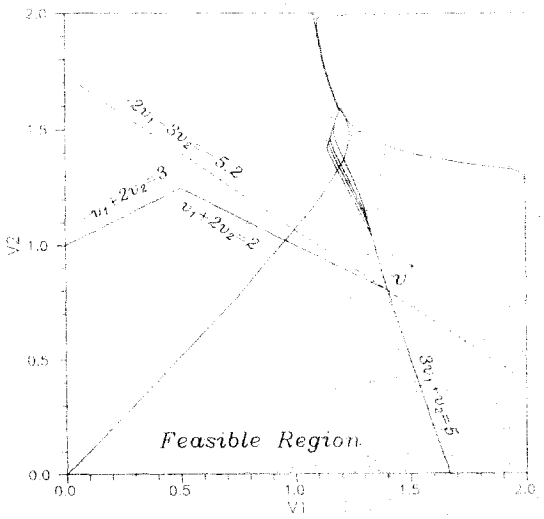

FIGURE 4. Feasible region and state trajectories in Example 1.

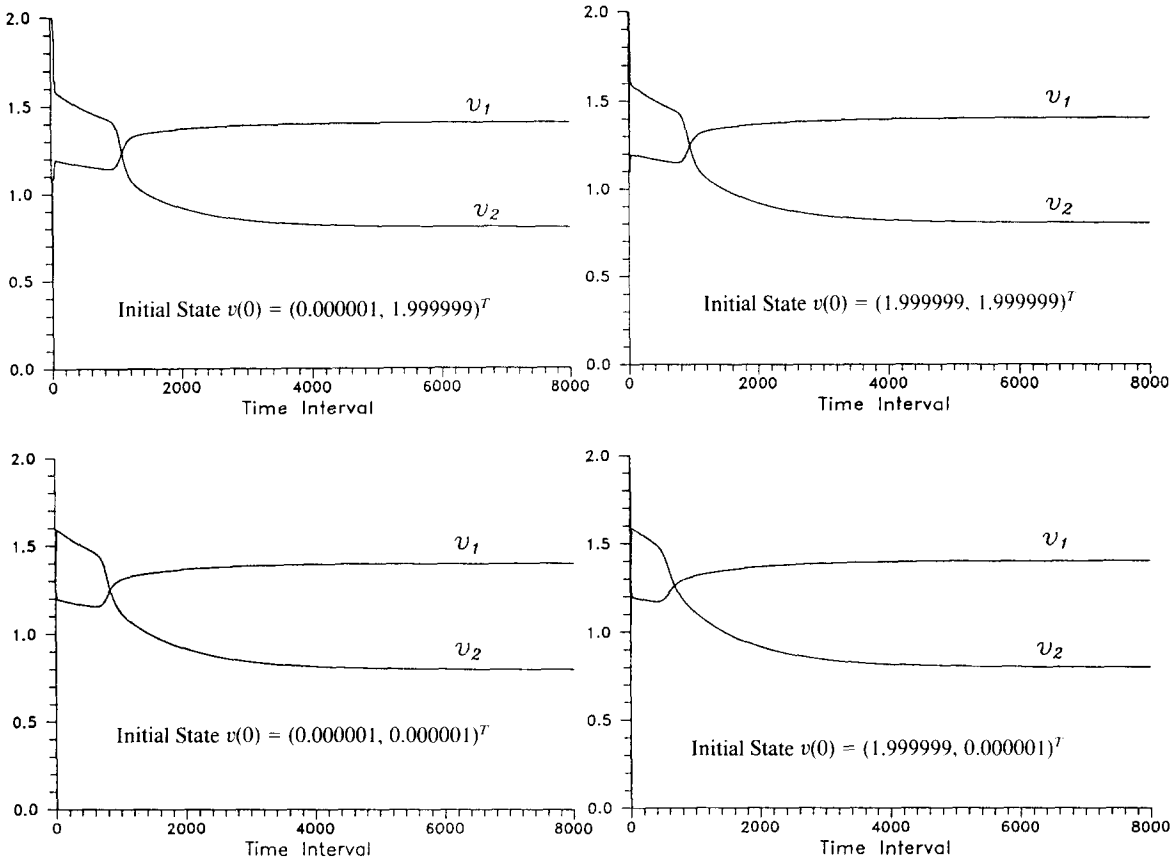

FIGURE 5. Transient states of the deterministic annealing neural network in Example 1.

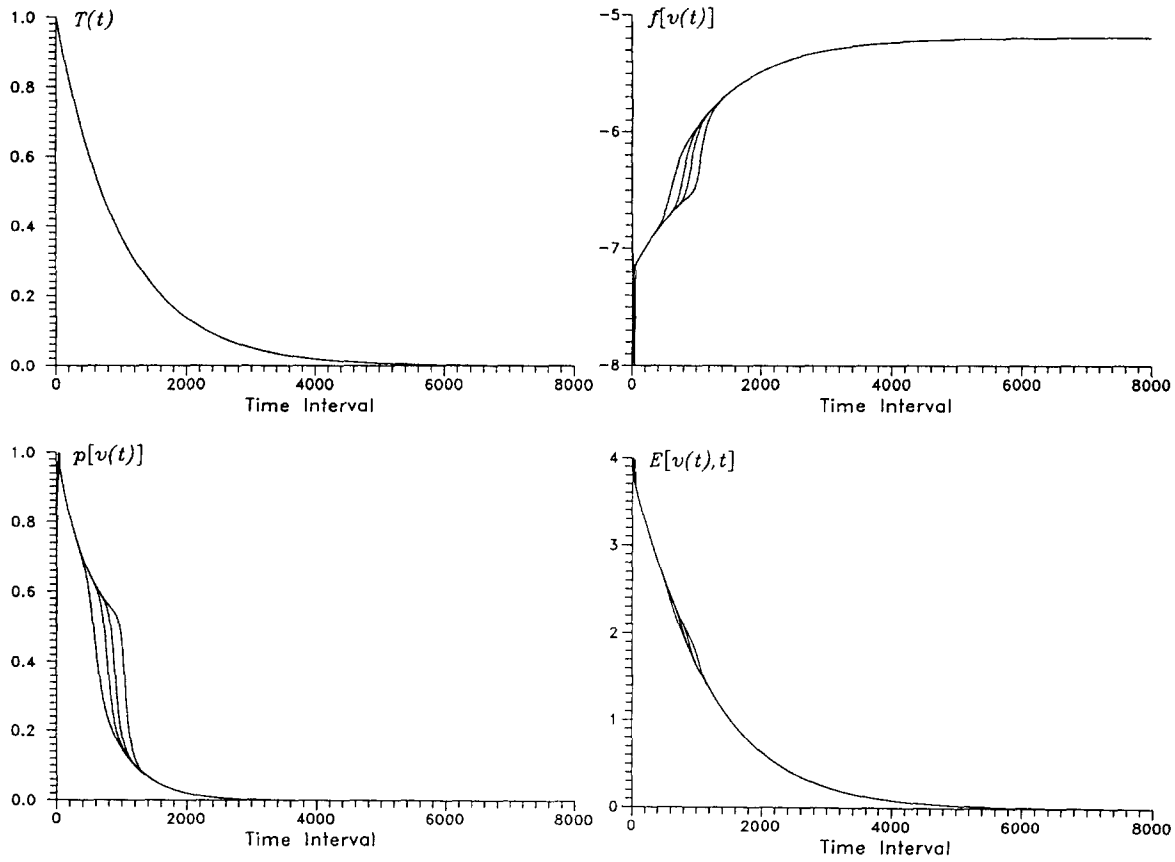

FIGURE 6. Transient behaviors of the temperature parameter, objective function, penalty function, and energy function in Example 1.

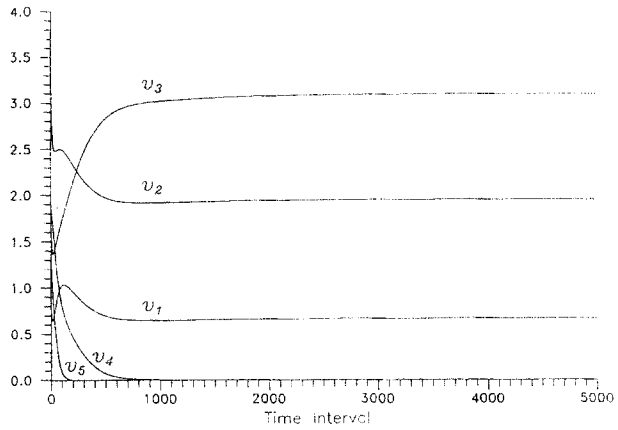

FIGURE 7. Transient states of the deterministic annealing neural network in Example 2.

convergent-state trajectories on the $v_1 - v_2$ plane and the convergent states versus the number of time intervals (iterations). Figures 4 and 5 indicate that the four trajectories merge in their final stages. This phenomenon indicates a deep valley of the energy function in the vicinities of the trajectories. Figure 5 also shows that the deterministic annealing neural network can converge in approximately 8000 time intervals or 0.008 time units. Figure 6 illustrates the convergence patterns of the values of temperature parameter, objective function, penalty function, and energy function versus the number of time intervals with four different initial states, which shows that the values of objective, penalty, and energy functions with different starting points make little difference after a few iterations.

EXAMPLE 2. Consider the following linear program with five variables and three constraints.

$$\begin{aligned}
\min \quad & 2.4v_1 + 1.6v_2 + 4.2v_3 + 5.2v_4 + 2.4v_5, \\
\text{s.t.} \quad & -4.3v_1 + 5.3v_2 + 1.6v_3 + 0.5v_4 - 2.1v_5 = 12.5, \\
& 7.2v_1 - 2.6v_2 + 2.4v_3 + 1.6v_4 + 2.9v_5 = 7.2, \\
& 1.3v_1 - 1.2v_2 + 2.5v_3 + 4.1v_4 - 2.7v_5 = 6.3, \\
& v_1 \geq 0,\ v_2 \geq 0,\ v_3 \geq 0,\ v_4 \geq 0,\ v_5 \geq 0.
\end{aligned}$$

The optimal solution of this problem $v^* = [0.671138, 1.963131, 3.113311, 0, 0]^T$ and the corresponding value of objective function $f(v^*) = 17.827650$. Simulation using the simulator shows that the steady state of the deterministic annealing neural network $\bar{v} = [0.671138, 1.963132, 3.113310, 0.000001, 0.000000]^T$ and $f(\bar{v}) = 17.827648$, where $v(0) = [2.5, 2.5, 2.5, 2.5, 2.5]^T$, $v_i(t) = v_{\max}/[1 + \exp(-\xi u_i(t))]$, $T(t) = \beta \exp(-\eta t)$, $v_{\max} = 5$, $\xi = 10^4$, $\beta = 1$, $\eta = 10^3$, $\Delta t = 10^{-6}$ unless otherwise indicated. Figure 7 illus-

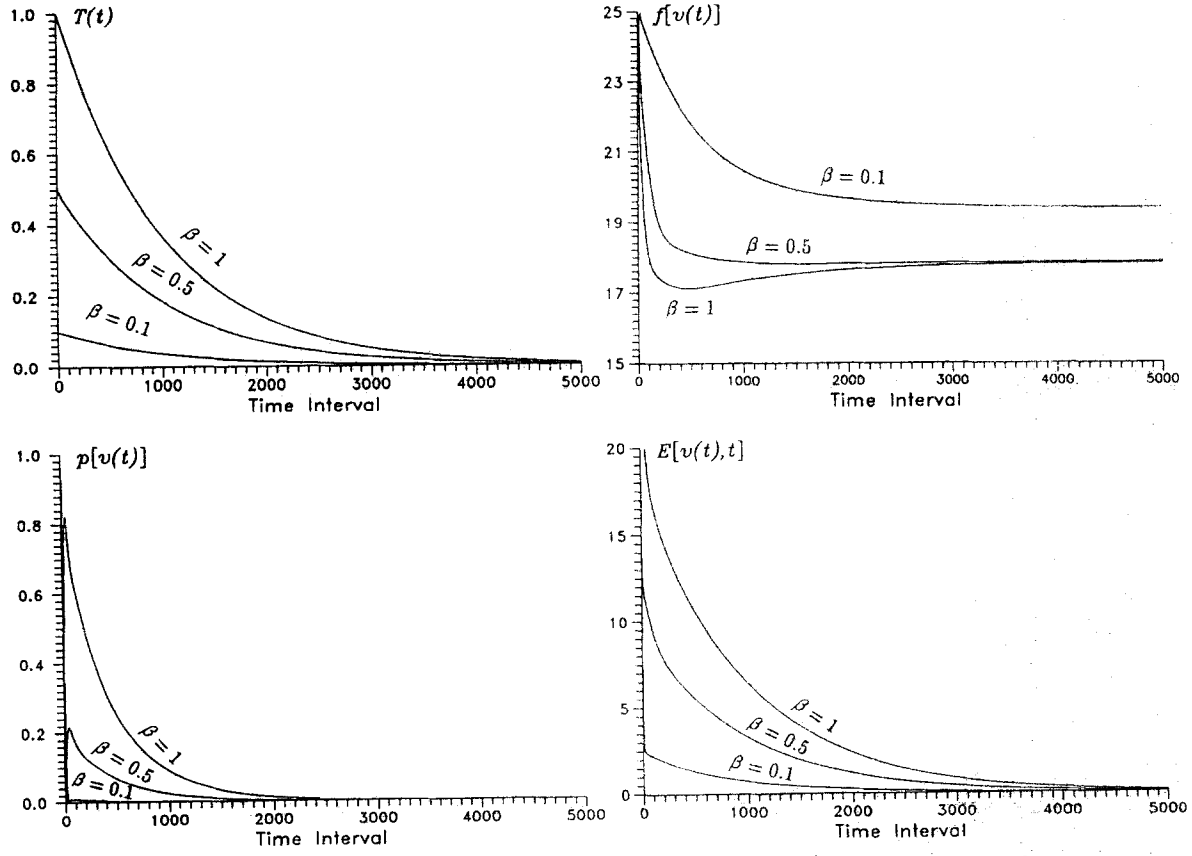

FIGURE 8. Transient behaviors of the temperature parameter, objective function, penalty function, and energy function based on different β in Example 2.

trates the transient behavior of the activation states of the deterministic annealing neural network. Because $\eta = 10^3$ and $\Delta t = 10^{-6}$, $5/\eta = 0.005$ and $5/(\eta\Delta t) = 5000$. Figure 7 clearly shows that the deterministic annealing neural network can converge in approximately 5000 time intervals or 0.005 time units. Figures 8, 9, and 10 illustrate the transient behaviors of the values of the temperature parameter, objective function, penalty function, and energy function based on different values of design parameters β, η, and ξ, respectively.

EXAMPLE 3. Consider the following convex quadratic program with four variables and three constraints.

$$\begin{aligned}
\min \quad & 1.5v_1^2 - v_1v_2 + v_1v_3 - 2v_1v_4 + 2v_2^2 \\
& + 2v_2v_3 + 2.5v_3^2 + v_3v_4 + 3v_4^2 \\
& -6v_1 + 15v_2 + 9v_3 + 4v_4, \\
\text{s.t.} \quad & v_1 + 2v_2 + 4v_3 + 5v_4 = 12, \\
& 3v_1 - 2v_2 - v_3 + 2v_4 = 8, \\
& 2v_1 - 3v_2 + v_3 - 4v_4 = 6, \\
& v_1 \geq 0,\ v_2 \geq 0,\ v_3 \geq 0,\ v_4 \geq 0.
\end{aligned}$$

The optimal solution to this problem $v^* = [2.967033, 0.000000, 1.736264, 0.417582]^T$ and the corresponding value of objective function $f(v^*) = 24.157710$. Simulation using the simulator shows that the steady state of the deterministic annealing neural network $\bar{v} = [2.967033, 0.000000, 1.736263, 0.417583]^T$ and $f(\bar{v}) = 24.157701$, where the initial state is $v(0) = [2.5, 2.5, 2.5, 2.5]^T$, $v_i(t) = v_{\max}/[1 + \exp(-\xi u_i(t))]$, $T(t) = \beta[\ln(t + e)]^{-\eta}$, $v_{\max} = 5$, $\xi = 10^4$, $\beta = 1$, $\eta = 10^4$, $\Delta t = 10^{-6}$. Figure 11 illustrates the transient behavior of the activation states, temperature parameter, objective function, penalty function, and energy function in Example 3.

8. CONCLUSIONS

In this paper, a deterministic annealing neural network for solving convex programming problems has been proposed. The convergence properties of the deterministic annealing neural network have been analyzed, the design principles have been discussed, and three illustrative examples have also been detailed. It has been substantiated that the proposed deterministic annealing neural network is capable of generating optimal solutions to convex programming problems. Having the same convergence properties of its predecessor (Wang, 1991; Wang & Chankong, 1992), the proposed deterministic annealing neural network possesses more desirable features. From a theoretical point of view, the

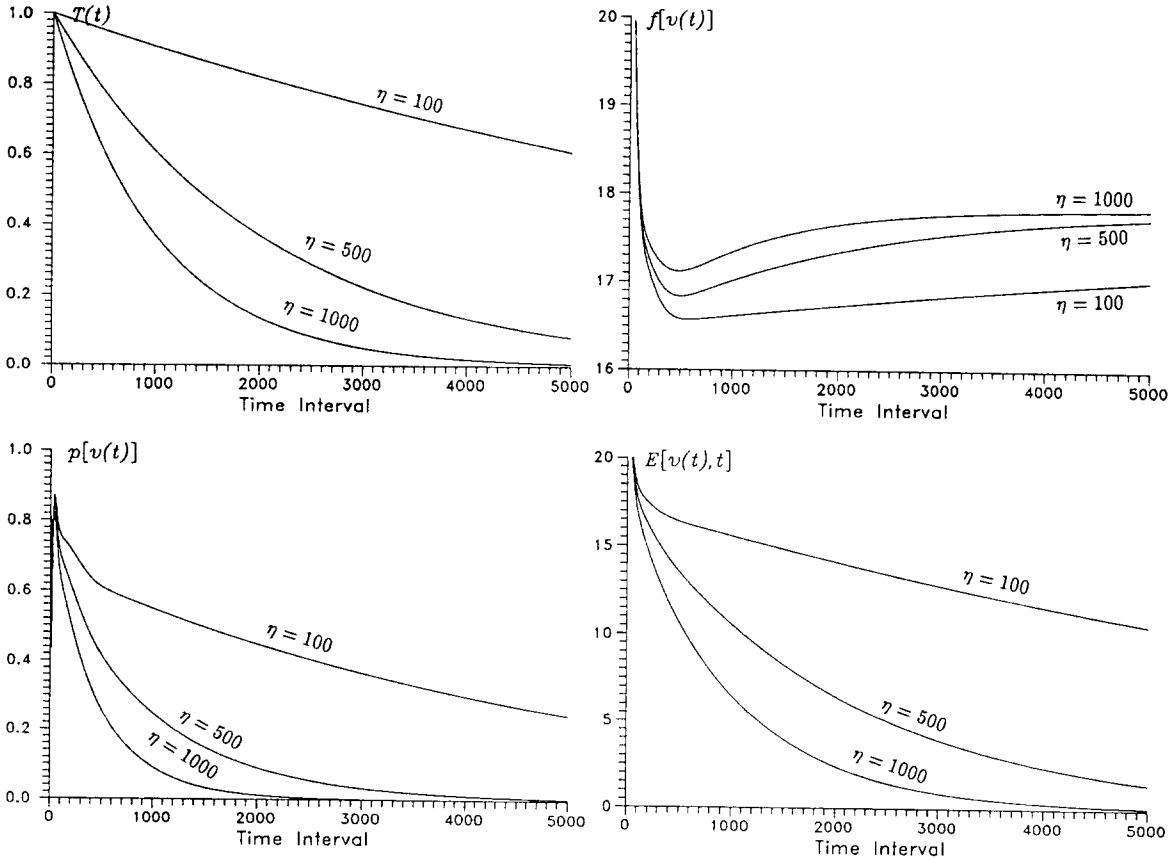

FIGURE 9. Transient behaviors of the temperature parameter, objective function, penalty function, and energy function based on different η in Example 2.

infeasibility requirement for initial solutions is relaxed. From a practical point of view, the proposed deterministic annealing neural network is more suitable for hardware implementation, as shown in Wang (1992, 1993). The proposed deterministic annealing neural network thus can serve as an effective computational model for solving real-time and large-scale convex programming problems.

Many avenues are open for future work. The proposed deterministic annealing neural network for convex programming could be extended for solving nonconvex optimization problems. Further investigations have been aimed at extension of the proposed deterministic annealing neural network to solve nonconvex quadratic programs and integer programs, applications of the deterministic annealing neural network to specific problems of interest, and implementation of the deterministic annealing neural network in analog VLSI circuits.

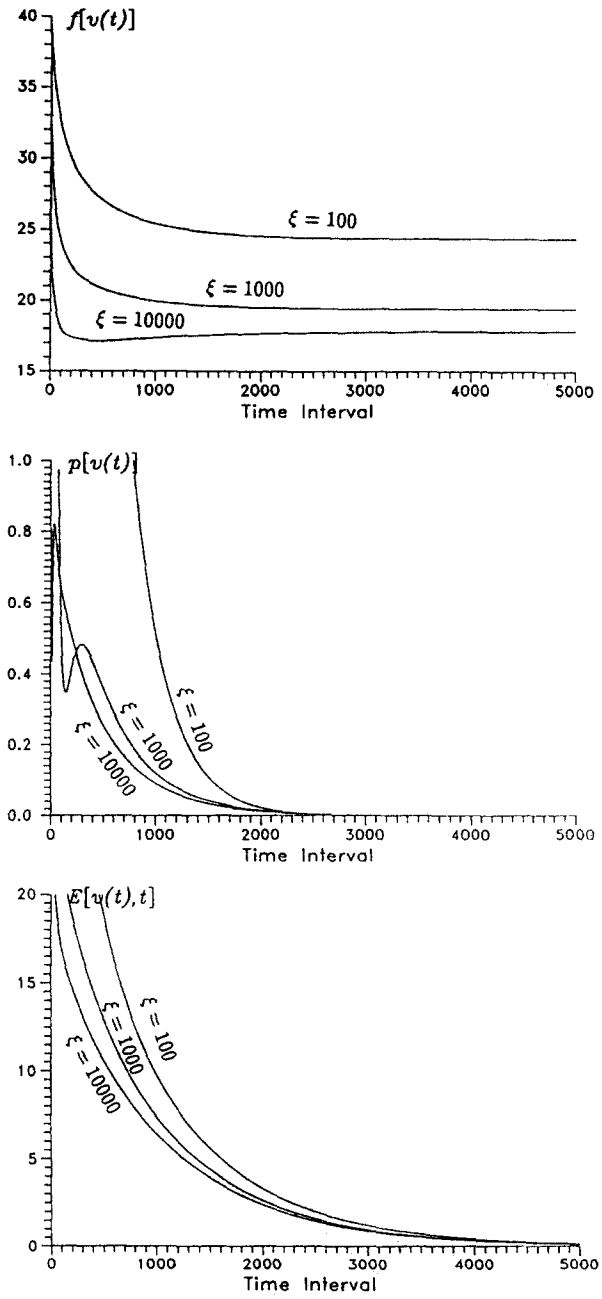

FIGURE 10. Transient behaviors of the temperature parameter, objective function, penalty function, and energy function based on different ξ in Example 2.

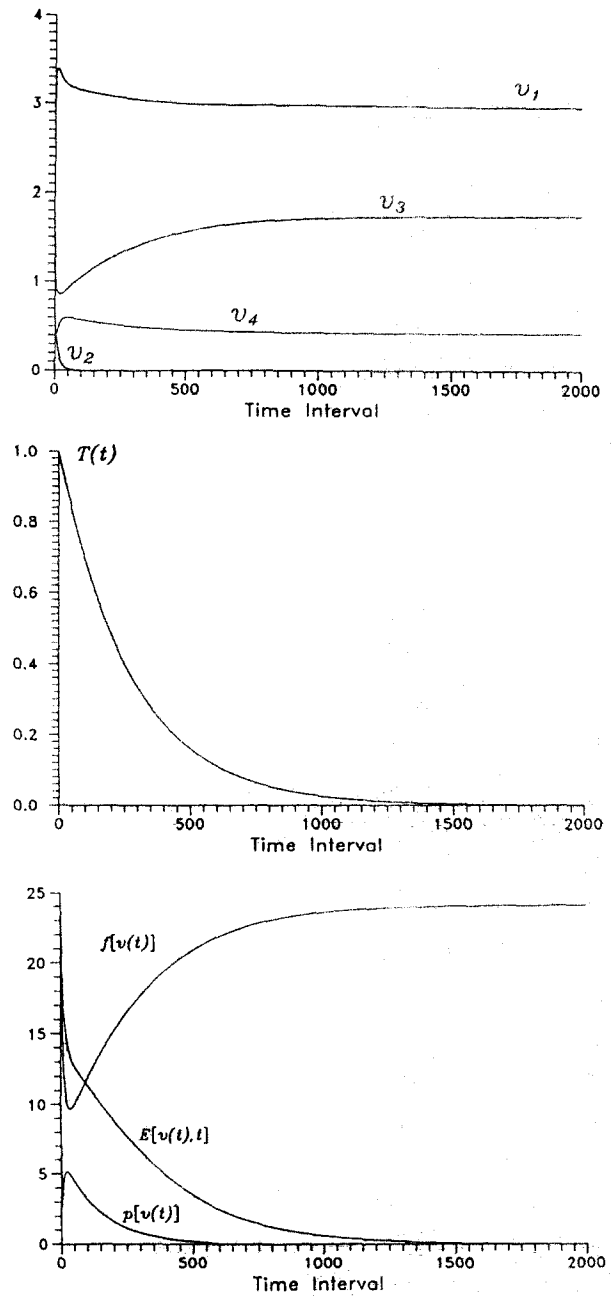

FIGURE 11. Transient states, temperature parameter, objective function, penalty function, and energy function in Example 3.

REFERENCES

Hopfield, J. J., & Tank, D. W. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, **52**, 141–152.

Kennedy, M. P., & Chua, L. O. (1988a). Unifying the Tank and Hopfield linear programming circuit and the canonical nonlinear programming circuit of Chua and Lin. *IEEE Transactions on Circuits and Systems,* **34**(2), 210–214.

Kennedy, M. P., & Chua, L. O. (1988b). Neural networks for nonlinear programming. *IEEE Transactions on Circuits and Systems,* **35**(5), 554–562.

Maa, C. Y., & Shanblatt, M. A. (1992). Linear and quadratic programming neural network analysis. *IEEE Transactions on Neural Networks,* **3,** 580–594.

Tagliarini, G. A., Christ, J. F., & Page, E. W. (1991). Optimization using neural networks. *IEEE Transactions on Computers,* **40**(12), 1347–1358.

Tank, D. W., & Hopfield, J. J. (1986). Simple neural optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Transactions on Circuits and Systems,* **33**(5), 533–541.

Wang, J. (1991). On the asymptotic properties of recurrent neural networks for optimization. *International Journal of Pattern Recognition and Artificial Intelligence,* **5**(4), 581–601.

Wang, J. (1992). Analog neural network for solving the assignment problem. *Electronics Letters,* **28**(11), 1047–1050.

Wang, J. (1993). Analysis and design of a recurrent neural network for linear programming. *IEEE Transactions on Circuits and Systems: Fundamental Theory and Applications,* **40**(9), 613–618.

Wang, J., & Chankong, V. (1992). Recurrent neural networks for linear programming: Analysis and design principles. *Computers & Operations Research,* **19**(3/4), 297–311.

APPENDIX

Proof of the convexity of $p_g(v)$ in eqn (7):

$p_g(v)$ is convex if $\lambda p_g(v') + (1 - \lambda)p_g(v'') \geq p_g[\lambda v' + (1 - \lambda)v'']$, $\forall v', v'' \in \mathbf{R}^N$, $\lambda \in [0, 1]$. According to the definition of the Heaviside function eqn (9), for $i = 1, 2, \ldots, m$

$$\int_{-\infty}^{g_i(v)} H(x)\,dx = \begin{cases} g_i(v)^2/2 & \text{if } g_i(v) > 0, \\ 0 & \text{if } g_i(v) \leq 0. \end{cases}$$

Because $0 \leq \lambda \leq 1$, if $g_i(v') \leq 0$ and/or $g_i(v'') \leq 0$, then the convexity is straightforward. Let us consider $g_i(v') > 0$ and $g_i(v'') > 0$.

$$\lambda p_g(v') + (1 - \lambda)p_g(v'') = \frac{1}{2}\sum_{i=1}^{m} [\lambda g_i(v')^2 + (1 - \lambda)g_i(v'')^2].$$

Because $[g_i(v') - g_i(v'')]^2 \geq 0$, $\lambda(1 - \lambda)[g_i(v')^2 + g_i(v'')^2] \geq 2\lambda(1 - \lambda)g_i(v')g_i(v'')$; that is, $\lambda g_i(v')^2 + (1 - \lambda)g_i(v'')^2 \geq \lambda^2 g_i(v')^2 + (1 - \lambda)^2 g_i(v'')^2 + 2\lambda(1 - \lambda)g_i(v')g_i(v'')$. Summing the above inequality over i and multiplying both sides with $\frac{1}{2}$, we have

$$\frac{1}{2}\sum_{i=1}^{m} [\lambda g_i(v')^2 + (1 - \lambda)g_i(v'')^2]$$

$$\geq \frac{1}{2}\sum_{i=1}^{m} [\lambda^2 g_i(v')^2 + (1 - \lambda)^2 g_i(v'')^2 + 2\lambda(1 - \lambda)g_i(v')g_i(v'')].$$

Because

$$\frac{1}{2}\sum_{i=1}^{m} [\lambda^2 g_i(v')^2 + 2\lambda(1 - \lambda)g_i(v')g_i(v'') + (1 - \lambda)^2 g_i(v'')^2]$$

$$= \frac{1}{2}\sum_{i=1}^{m} [\lambda g_i(v') + (1 - \lambda)g_i(v'')]^2$$

and $\lambda g_i(v') + (1 - \lambda)g(v'') \geq g_i(\lambda v' + (1 - \lambda)v'')$ due to the convexity of $g_i(v)$ for $i = 1, 2, \ldots, m$,

$$\frac{1}{2}\sum_{i=1}^{m} [\lambda g_i(v') + (1 - \lambda)g_i(v'')]^2 \geq \frac{1}{2}\sum_{i=1}^{m} g_i(\lambda v' + (1 - \lambda)v'')^2$$

$$= p_g[\lambda v' + (1 - \lambda)v''].$$

Thus, the convexity property of $p_g(v)$ is proven. Note that the convexity of $p_g(v)$ does require the convexity of $g(v)$. The proof of the convexity of $p_h(v)$ is similar.

NOMENCLATURE

N	the number of elements in a decision variable
m	the number of inequality constraints
n	the number of equality constraints
v	the N-dimensional activation state vector and vector of decision variables
u	the N-dimensional net input vector to neurons
$f(v)$	the real-valued objective function
$g(v)$	the m-tuple function for inequality constraints
$h(v)$	the n-tuple function for equality contraints
$p(v)$	the real-valued penalty function
$T(t)$	the temperature parameter