1995

Applying Neural Networks to Find the Minimum-Cost Coverage of a Boolean Function

Pong P. Chu
Cleveland State University, p.chu@csuohio.edu

Original Citation

Chu, P.P. (1995). Applying Neural Networks to Find the Minimum-Cost Coverage of a Boolean Function. VLSI Design 3, 13-19.

VLSI Design
1995, Vol. 3, No. 1, pp. 13–19
Reprints available directly from the publisher
Photocopying permitted by license only

Applying Neural Networks to Find the Minimum Cost Coverage of a Boolean Function

PONG P. CHU
Department of Electrical Engineering, Cleveland State University, Cleveland, Ohio 44115

(Received May 14, 1993 Revised October 29, 1993

To find a minimal expression of a boolean function includes a step to select the minimum cost cover from a set of implicants. Since the selection process is an NP-complete problem, to find an optimal solution is impractical for large input data size. Neural network approach is used to solve this problem. We first formalize the problem, and then define an "energy function" and map it to a modified Hopfield network, which will automatically search for the minima. Simulation of simple examples shows the proposed neural network can obtain good solutions most of the time.

Key Words: *Logic minimization; Neural networks; Optimization; Quine-McCluskey method; Hopfield network.*

1. INTRODUCTION

Logic minimization is to simplify a boolean function so that the implementation cost can be reduced. Among the different techniques, two-level minimization is the most well-understood one and has a wide application in the design of Programmable Logic Array (PLA) [2]. This technique normally involves two basic procedures: the generation of *Prime Implicants* (PIs), and the selection of PIs. The generation of PIs can be exhaustive, as in traditional Quine-McCluskey method [15] and in McBoole [6], or heuristic, as in Espresso [2]. Once the PIs are found, we need to select a subset that can cover the function completely and has a minimal total cost. This selection process is an NP-complete problem [9] and may take exponential computation time to obtain an optimal answer. Although certain pre-processing techniques, such as column reduction, row reduction [14] and partition [17], can reduce the size of the input table, there is no effective way to obtain the optimal solution from the reduced table.

A neural network is an interconnected network of a large number of simple processors [18]. Although neural networks are primarily used for information processing and biological modeling, it has been shown that neural networks can collectively compute good solutions to a wide range of complex optimization problems, such as Traveling Salesman, 3-Satisfiability, etc. [11] [12] [20] [19]. Since it is possible to implement the massive neural network in a VLSI chip [16] in the future, neural networks can be an alternative approach to obtain answer for some "hard" computational problems. In this paper, we investigate the possibility of applying neural network approach to solve the coverage problem. We first formalize this problem and define the "energy function", and then derive network configuration accordingly.

The remaining of the paper is organized as follows: Section 2 gives an overview on neural network and its application to optimization; Section 3 shows the formulation of the network; Section 4 discusses the implementation issues and gives two examples; and the last section summarizes the study.

2. OVERVIEW ON OPTIMIZATION NEURAL NETWORKS

An artificial neural network is a network of a large number of simple computation elements, known as *neurons*, connected by links with variable weights.

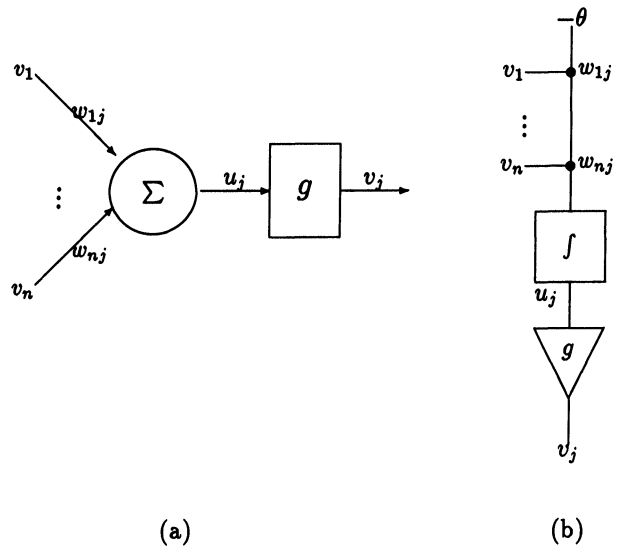

FIGURE 1 Diagram of a Single Neuron

Instead of performing a program of instructions sequentially, neural network model explores many competing hypothesis simultaneously using its massively parallel net. A typical neuron is shown in Figure 1(a). The neuron's operation contains two basic steps. The first step calculates the *activation* (u) of the neuron by summing n weighted incoming signals (i.e., for neuron j, performs $\sum_{i=1}^{n} w_{ij}v_i(t) - \theta_j$ and assign the value to $u_j(t+1)$). The second step passes the results through an output function $g(\cdot)$(i.e., for neuron j, performs $g(u_j(t+1))$ and assign the value to $v_j(t+1)$). Normally, $g(\cdot)$ is a nonlinear *sigmoid* function, which is monotone increasing and bounded. A neuron can be extended to a "nonlinear" neuron if the summation in the first part is replaced by a more general nonlinear function $f(v_1(t), \ldots, v_n(t))$. A neuron can also be implemented as an analog device with a nonlinear amplifier of gain $g(\cdot)$ [16]. The analog version of the neuron is shown in Figure 1(b). The two steps now can be described as $du_j(t)/dt = \sum_{i=1}^{n} w_{ij}v_i(t) - \theta_j$ and $v_j(t) = g_j(u_j(t))$. The time unit here is normalized to the RC time constant of the integrator. The analog neuron can also be extended to a nonlinear one by generalizing the fist part: $du_j(t)/dt = f_j(v_1(t), \ldots, v_n(t))$. In the remaining discussion, we will use the analog version. While the primary research interests in neural network are concentrated in the information processing and biological modeling, it has been shown that the neural network can collectively compute good solutions to complex optimization problems [19]. This approach comes from the observation that in certain properly designed networks, the dynamic of the networks will force the network to converge to a *minimal energy state* [10]. If we can map the problem solution to this minimal energy state, the network can "automatically" solve the optimization problem.

The procedure listed below (extended over [7]) outlines the basic steps of this approach:

1. introduce a set of variable $\{v_1(t), v_2(t), \ldots, v_n(t)\}$ to represent the quantities to be optimized.
2. determine an *energy function*, $E(v_1(t), v_2(t), \ldots, v_n(t))$, as a measurement of "optimality". E is a scalar function. It should be bounded below and its minimum should correspond to the desired solutions of the optimization problem.
3. introduce $u_i(t)$ as the activation for neuron i; and define $v_i(t)$ as $v_i(t) = g(u_i(t))$, where $g(\cdot)$ is a high gain sigmoid function:

$$g(u_i) = \frac{1}{2}(1 + \tanh(\lambda u_i)), \qquad \lambda \gg$$

4. define the rate of change of $u_j(t)$ as

$$\frac{du_i(t)}{dt} = -\frac{dE(v_1(t), v_2(t), \ldots, v_n(t))}{dv_i(t)}$$

5. construct the network accordingly.
6. add *relaxation* mechanism to the network if necessary.

In this procedure, step (1) determines the number of the required neurons; step (4) and step (3) specify the activation function and output function respectively. Once these entities are determined, the actual network can be constructed accordingly. By choosing $du_i(t)/dt = -dE/dv_i(t)$, we can show that $dE/dt \leq 0$ (the proof is in the Appendix). The inequality implies that as time progresses, the energy of the network will decrease or remain the same. However, since E is bounded below, the energy will evnetually reach the minimal point (in a local sense). In other words, becuase of our choice of E, output function and activation function, the dynamics of the network will force itself to converge to a minimal energy state, which corresponds the desired solution of the optimization problem. The purpose of the high-gain sigmoid function is to reduce the

width of the "linear" region (the region with a value close to 0.5) and ensures that the output moves away from the "undetermined" area. Since steps (1) to (5) only guarantee that the network converges to a local minima, sometimes we may need to apply relaxation techniques, such as simulated annealing, to obtain a global minima [1, 13]. Relaxation can be thought as "coarse searching", which will help the network to escape from local minima.

3. FORMULATION

In logic minimization, the set of PIs and the vertices to be covered are normally represented as a table, with rows as PIs and columns as vertex. For each column, a check mark is placed in a row if that vertex is covered by the corresponding PI. For each PI, there is a positive number associated with it to represent the *cost* of implementing this PI (such as the number of literals). The goal of the coverage process is to select a set of PIs so that there is at least one check mark in each column and the total cost of the set is minimal.

Mathematically, we can use a matrix and a cost vector to describe the PI-vertex table. Let m and n be the number of vertex and PIs. The cost vector can be defined as $\langle c_j \rangle$, $1 \le j \le n$, where c_j is the cost associated with jth PI and $c_j \le 0$. The relation of vertex and PIs can be defined by a matrix $[a_{ji}]$, where $a_{ji} \in \{0, 1\}, 1 \le j \le n, 1 \le i \le m$; $a_{ji} = 1$ if jth PI covers ith vertex and $a_{ji} = 0$ otherwise.

After introducing the two entitites, we can formalize the coverage problem as an optimization problem [3]:

let $\{x_1, x_2, \ldots, x_n\}$ be a set of variables, where $x_j \in \{0, 1\}$ and $x_j = 1$ if P_j is selected; assign proper values to x_j so that they can

$$\text{minimize} \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \sum_{j=1}^{n} a_{ji} x_j \ge 1, \qquad i = 1, \ldots, m$$

The first line represents the total cost that needs to be minimized, and the second line represents m constraints which state that for each column there must be at least one check mark.

3.1 Formal Description of the Network

With the formal description, a neural network can be constructed as follows:

1. use n neurons to represent set $\{x_1, x_2, \ldots, x_n\}$ and use the output of ith neuron (v_i), to represent the value of x_i.
2. define the energy function E as

$$E = \sum_{j=1}^{n} c_j v_j + D \sum_{i=1}^{m} F\left(\sum_{j=1}^{n} a_{ji} v_j - 1 \right)$$

where $F(y) = \begin{cases} y^2 & \text{if } y < 0 \\ 0 & \text{if } y \ge 0 \end{cases}$

3. define the output function as a high-gain sigmoid function:

$$v_i = g(u_i) = \frac{1}{2}(1 + \tanh(\lambda u_i)), \qquad \lambda \gg 1$$

4. define the activation of jth neuron (u_j) as

$$\frac{du_j}{dt} = -c_j + D \sum_{i=1}^{m} (-a_{ji}) f\left(\sum_{k=1}^{n} a_{kj} v_k - 1 \right)$$

where $f(y) = \begin{cases} 2y & \text{if } y < 0 \\ 0 & \text{if } y \ge 0 \end{cases}$
and $D > \Sigma_{j=1}^{n} c_j$

3.2 The Energy Functionxq

There are two major terms in the energy function, the first term $\Sigma c_j v_j$ represents the total cost of the selected PIs, and the second term $D\Sigma F(\Sigma a_{ji} v_j - 1)$ represents the penalty for the violation of constraints. The first term is always greater than 0 because c_j is always positive. The second term is greater than or equal to 0, depending whether the constraints are satisfied. It returns a positive value if any constraint is violated (i.e., for some $i, \Sigma a_{ji} v_j < 1$) and returns 0 if all constraints are satisfied (i.e., for all $i, \Sigma a_{ji} v_j \ge 1$).

There is a constant scaling factor D in penalty term. The value of D can be interpreted as the penalty for the violation of a single constraint and can be used to specify the relative weights between the cost term and the penalty term. Since it is more desirable to obtain a valid sub-optimal solution than an invalid solution, any constraint violation should

be given a larger penalty. The D is used to assure this.

Our choice of D comes from the following observation. $\Sigma_{j=1}^{n} c_j$ can be interpreted as the cost incurred to the case that all PIs are selected. This is a valid solution since all the vertices are covered. Also, this is the worst solution since its cost is larger for any other valid solutions. In other words, $\Sigma_{j=1}^{n} c_j$ represents the cost (and energy value as well) of the worst valid solution. Since any invalid solution is worse than the worst valid solution, its energy value should be larger than $\Sigma_{j=1}^{n} c_j$. Thus, we choose $D > \Sigma_{j=1}^{n} c_j$ and guarantee that constraint violation will always be penalized more.

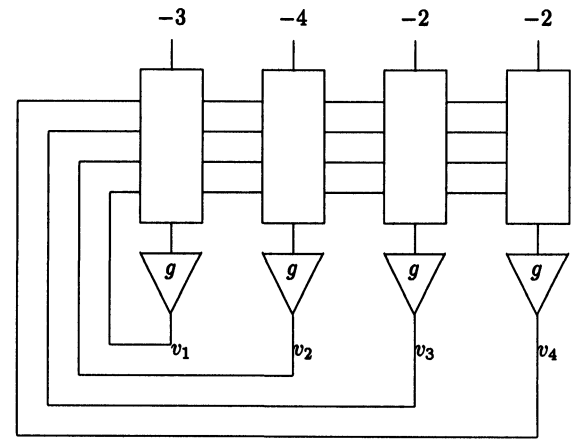

FIGURE 2 First Implementation of the Proposed Network

4. IMPLEMENTATION AND EXAMPLES

In our proposed network, both activation function and output function are non-linear. While the output function is a regular signmoid function, the activation function (which is $-c_j + D\Sigma_{i=1}^{m} - a_{ji} f(\Sigma_{j=1}^{n} a_{ji} v_j - 1)$) is fairly complicated for a single neuron to compute. However, close observation shows that the calculation of $f(\Sigma_{j=1}^{n} a_{ji} v_j - 1)$ resembles the operation of a regular neuron except the output function is $f(\cdot)$ and there is no integrator. Thus, we can introduce a set of slightly modified neurons to perform this task. The detailed implementation is explained by the following example.

The coverage table of the first example is shown in Table 1 which has 4 PIs and 5 vertices. In this example, there are total 2^4 possible selections, and $\{P_2, P_4\}$ is the optimal solution.

The implementation derived from section 3.1 is shown in Figure 2 The D is chosen to be 11.1. In this configuration, the activation function (denoted by a square box) is complicated and hard to implement. Further, computation to obtain $D\Sigma_{i=1}^{m}(-a_{ji}) f(\Sigma_{j=1}^{n} a_{ji} v_j - 1)$ is duplicated in every neuron.

A better alternative implementation is shown in Figure 3. In this configuration, we simplify the implementation of original neurons by distributing the computation of activation functions to a new group of neurons. There are two groups of neurons in this implementation. The first group contains n neurons. The input part now is very simple, which is in a standard linear summation form, similar to the convetional neuron. The second group contains m neurons and each of them performs the computation of $f(\Sigma_{j=1}^{n} a_{ji} v_j - 1)$. They are regular neurons with f as their output functions and without the integrators. This network is similar to the one used in [20].

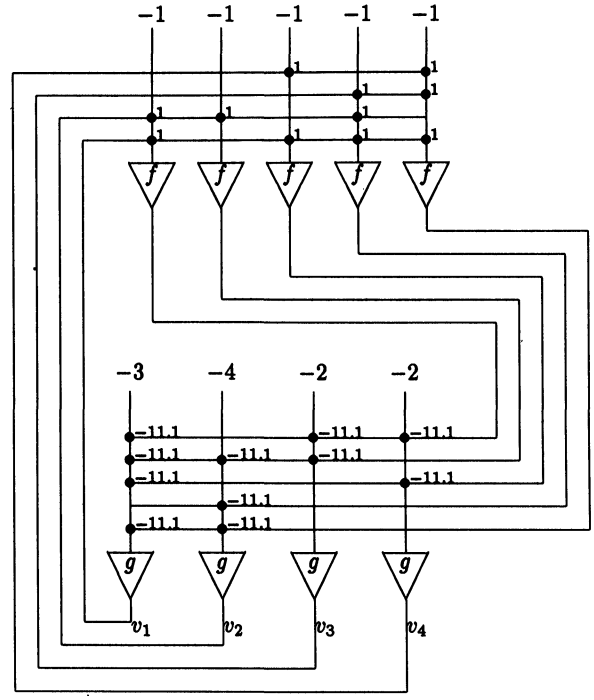

FIGURE 3 The Alternative Implementation of the Proposed Network

TABLE 1
The Coverage Table for the First Example

PI	Vertex					Cost
P_j	y_1	y_2	y_3	y_4	y_5	c_j
P_1	✓		✓	✓	✓	3
P_2	✓	✓		✓		4
P_3				✓	✓	2
P_4			✓		✓	2

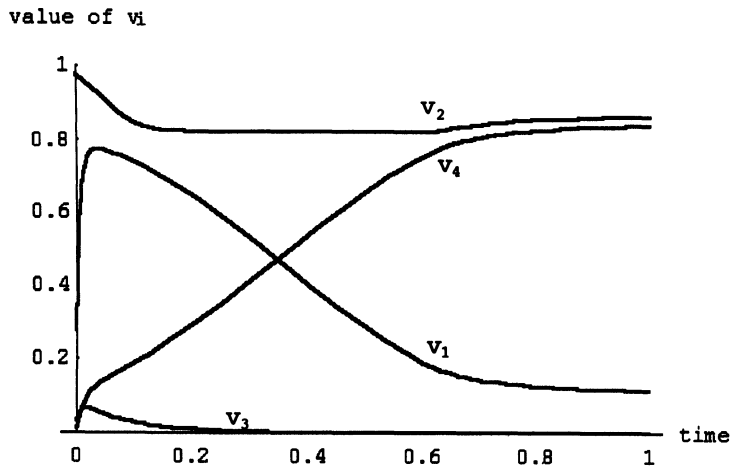

FIGURE 4 A Representative Convergence Trace for Example 1

Although the two types of neurons look similar, their operation speeds need to be different. Since the neurons in the second group are not involved the network energy, they are essentially just "computational elements" that compute the activation functions. Their computation should be completed before any neuron in the first group changes to a new output value. Therefore, the neurons in the second group should be operated much faster than the neurons in the first group. We may need to add additional capacitance in the inputs of the neuron of the first group to ensure that it has a proper time constant.

The operation of the network is simulated by software. 50 randomly generated initial conditions, in which the u_i is uniformly distributed between $[-0.5, 0.5]$, are used. Simulation results show that all of them converge to the correct value and the convergence normally takes less than one time constant. A typical convergence trace is shown in Fig ure 4.

The convergence table of the second example is shown in Table 2, which is adopted from [15]. $\{P_2, P_3, P_6\}$ and $\{P_1, P_2, P_5\}$ are the two optimal selections. This is a difficult case since the table is cyclic and no heuristic rules, such as essential PI, dominance etc., can be used to reduce the complexity. The D is chosen to be 12.1. Again, 50 randomly generated initial conditions are used. Simulation results show that 46% converge to $\{P_2, P_3, P_6\}$, and 40% converge to $\{P_1, P_2, P_5\}$, and 14% converge to undeterministic states (with some output values near 0.5). The convergence is normally within one time constant. A typical convergence trace is shown in Figure 5.

TABLE 2
The Coverage Table for the Second Example

PI	Vertex						Cost
P_j	y_1	y_2	y_3	y_4	y_5	y_6	c_j
P_1	✓	✓					2
P_2	✓		✓				2
P_3		✓		✓			2
P_4			✓		✓		2
P_5				✓		✓	2
P_6					✓	✓	2

5. SUMMARY

In this paper, we apply the neural network approach to obtain the minimal cost coverage of a given PI-vertex table. We first formalize the problem and derive the energy function, and then map it to a neural network. The actual implementation includes two groups of neurons. One of them is used as regular neurons that works towards minimal energy state and another group is used as computation elements that assist the calculation of activation functions. Simulation on simple examples show that the neural network can obtain good solutions in a short amount of time.

In the current form, the major problem of this approach is the required computation time. Simulating the neural network is essentially using numerical analysis to solve a set of non-linear partial differential equations, which is computation intensive. This limits the total number of PIs to a small number and makes it not feasible for large practical problems. However, this scheme may be an attractive alternative when the analog neural network VLSI is available [16]. The VLSI device eliminates the need of

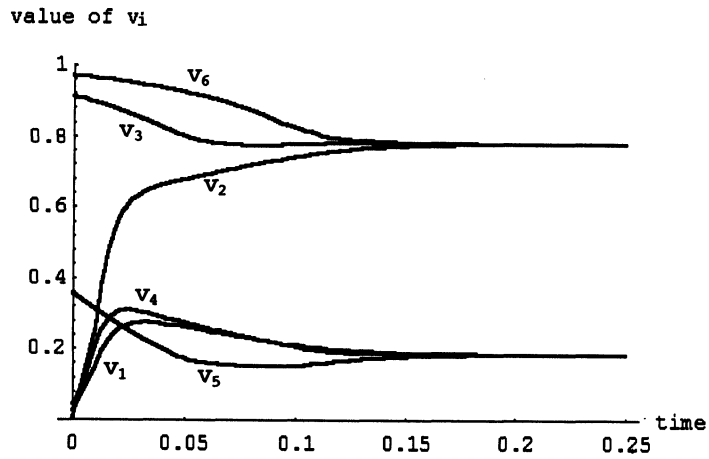

FIGURE 5 A Representative Convergence Trace for Example 2

simulation and can obtain the solution in a fraction of the circuit's time constant. It can house massive neural networks, which, in turn, can support a large number of PIs. Several other studies also suggest to use Hopfield-like network to solve CAD related problems including partition, placement, routing and test generation [4, 5, 8, 21]. In the future, it may be possible to include a generic programmable neural network as an "accelerator" chip in a regular CAD system, and to use the chip to assist to solve the "core computation" of certain optimization problems.

APPENDIX

Theorem

For the configuration in section 2, $\frac{dE}{dt} \leq 0$

Proof:

$$\frac{dE}{dt} = \sum_{i=1}^{n} \frac{dE}{dv_i}\frac{dv_i}{dt}$$

$$= \sum_{i=1}^{n} -\frac{du_i}{dt}\frac{dv_i}{dt}$$

$$= -\sum_{i=1}^{n} \frac{du_i}{dv_i}\frac{dv_i}{dt}\frac{dv_i}{dt}$$

$$= -\sum_{i=1}^{n} \left(g^{-1}(v_i)\right)'\left(\frac{dv_i}{dt}\right)^2$$

Since $g(\cdot)$ is monotone increasing, $g^{-1}(\cdot)$ is monotone increasing and $(g^{-1}(\cdot))' \geq 0$. Thus,

$$\frac{dE}{dt} \leq 0$$

□

References

[1] E. Aarts and J. Korst, *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons, 1989.
[2] Robert K. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, 1984.
[3] Melvin A. Breuer ed., *Design Automation of Digital Systems, Volume 1*, Prentice-Hall, 1972.
[4] S.T. Chakradhar, M.L. Bushnell and V.D. Agrawal, "Automatic Test Generation Using Neural Networks," *Proceedings of ICCAD*, 1988.
[5] R.I. Chang and P.Y. Hsiao, "Forced Directed Self-Organizing Map and its Application to VLSI Cell Placement," *Proceedings of IEEE International Conference on Neural Networks*, 1993.
[6] Michel R. Dagenais et al., "McBOOLE: A New Procedure for Exact Logic Minimization," *IEEE Trans. on Computer-Aided Design*, May, 1986.
[7] G.C. Fox and J.G. Koller, "Code Generation by a Generalized Neural Network: General Principles and Elementary Examples," *J. of Parallel and Distributed Computing*, November, 1989.
[8] N. Funabiki and Y. Takafuji, "A Parallel Algorithm for Channel Routing Problems," *IEEE Trans. On Computer-Aided Design*, April, 1992.
[9] Michael R. Garey and David S. Johnson, *Computers and Intractability*, Freeman and Company, 1979.
[10] J.J. Hopfield, "Neurons with Graded Response Have Collective Computational Properties like those of Two-state Neurons", *Proc. of the National Academy of Science USA*, vol. 81, 1984.
[11] J.J. Hopfield and D. W. Tank, "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, vol. 52, 1985.
[12] J.L. Johnson, "A Neural Network Approach to the 3-Satisfiability Problem," *J. of Parallel and Distributed Computing*, November, 1989.

[13] S. Kirkpatrick et al., "Optimization by Simulated Annealing", *Science*, vol. 220, 1983.
[14] Z. Kohavi, *Switching and Finite Automata Theory* McGraw-Hill, 1978.
[15] E.J. McCluskey, *Logic Design Principles*, Prentice-Hall, 1986.
[16] C. Mead, *Analog VLSI and Neural Systems*, Addison Wesley, 1989.
[17] R.L. Rudell and A. Sangiovanni-Vincentelli, "Multiple-Valued Minimization for PLA Optimization," *IEEE Trans. on Computer-Aided Design*, May, 1987.
[18] D.E. Rumelhart and J.L. McCelland, *Parallel Distributed Processing, Vol. 1 and 2* The MIT Press, 1986.
[19] Y. Takefuji, *Neural Network Parallel Computing* Kluwer Academic Publishers, 1992.
[20] D.W. Tank and J.J. Hopfield, "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Trans. on Circuits and Systems*, May, 1986.
[21] J-S Yih and P. Mazumder, "A Neural Network Design for Circuit Partition," *IEEE Trans. on Computer-Aided Design*, December, 1990.

Biographies

PONG P. CHU is currently an assistant professor of Electrical Engineering Department at Cleveland State University. He obtained the Ph.D. Degree from Iowa State University, majoring in Electrical and Computer Engineering. His research interests include digital systems, neural networks applications and high-speed computer networks. He is a member of Sigma Xi, Phi Kappa Phi as well as the IEEE and ACM.