Block-Based Neural Networks

Sang-Woo Moon and Seong-Gon Kong, *Member, IEEE*

***Abstract*—This paper presents a novel block-based neural network (BBNN) model and the optimization of its structure and weights based on a genetic algorithm. The architecture of the BBNN consists of a two-dimensional (2-D) array of fundamental blocks with four variable input/output nodes and connection weights. Each block can have one of four different internal configurations depending on the structure settings. The BBNN model includes some restrictions such as two-dimensional array and integer weights in order to allow easier implementation with reconfigurable hardware such as field programmable logic arrays (FPGAs). The structure and weights of the BBNN are encoded with bit strings which correspond to the configuration bits of FPGAs. The configuration bits are optimized globally using a genetic algorithm with 2-D encoding and modified genetic operators. Simulations show that the optimized BBNN can solve engineering problems such as pattern classification and mobile robot control.**

***Index Terms*—Evolvable hardware, genetic algorithm, neural network, structure optimization.**

I. Introduction

ARTIFICIAL neural networks have been successfully applied to diverse engineering problems due to their model-free approximation capability to complex decision making processes. Despite the success and the promise of artificial neural networks in solving practical problems, their design procedure still requires a trial-and-error. Obtaining the optimal structure of artificial neural networks for a given problem is one of typical problems in neural network design. Popular multilayer perceptron (MLP) neural networks trained using the backpropagation learning algorithm, for example, provide a practical approach to pattern classification tasks because of their postlearning processing speed and to their wide range of applications, but the problem of determining its structure has no general solution. Design of artificial neural networks requires identification of network configuration and associated parameters. Gradient-descent learning algorithms that modify connection weights based on a error function gradient have difficulties in determining optimal network topology such as the number of nodes and connection configurations. Training a neural network typically requires a human operator to make many training runs with different choices of structural and parameter settings. For practical purposes, neural-network structure and connection weights should be optimized at the same time.

Manuscript received January 4, 2000; revised July 31, 2000. This work was supported by Brain Science and Engineering Research Program sponsored by Korean Ministry of Science and Technology.

The authors are with Intelligent Signal Processing Lab, Department of Electrical Engineering, Soongsil University, Seoul 156-743, Korea (e-mail: skong@ee.ssu.ac.kr).

Publisher Item Identifier S 1045-9227(01)02038-0.

Another problem is hardware implementation of artificial neural networks. Due to nonlinear activation functions and real-valued connection weights of artificial neural networks, their implementation based on digital hardware is not an easy task. Hardware realization of artificial neural networks has attempted mainly with analog devices [1], [2]. For neural network hardware containing analog components, dynamic reconstruction and on-line parameter update is a difficult task.

Evolutionary algorithms [3], [4], based on the mechanics of natural selection and natural genetics, provide us with a solution to the structure and weight optimization problems. The evolutionary computation uses a problem-dependent fitness function to search for the global optimum. Previous works in the field of neural-network optimization with the evolutionary algorithms include a method of finding optimal network structure by using evolutionary computation dynamically while still using traditional gradient-based learning method [5]. Also, weight optimization in fixed structure has been proposed to avoid local minimum problems [6]. Both structure and weights are optimized by evolutionary computation simultaneously [7]–[9]. The competing conventions problem or the permutations problem can occur when genetic algorithm is used for weight optimization of feedforward multilayer neural networks [10]. To overcome such problems, Montana and Davis [11] use intelligent crossover and real encoding for connection weights. Korning [12] optimizes neural-network connection by using binary encoding scheme and fitness function with entropy concepts. Genetic algorithm has been successfully applied to neural network optimization. Whitley [13], [14] determines the neural-network structure by the GA with conventional gradient descent learning algorithm. Vittorio [15] represents structure and weights by binary encoding, and determines network size by granularity. Kumagai [16] optimizes neural network with recurrent neurons by the GA with internal copy operator.

In this paper, a novel block-based neural network (BBNN) model is proposed in order to achieve simultaneous optimization of network structure and connection weights. The BBNN consists of a two-dimensional (2-D) array of basic neural-network blocks with integer weights for easier implementation using reconfigurable hardware such as field programmable logic arrays (FPGAs). The structure and internal parameters of an individual BBNN are represented by fixed-length binary codes, which correspond to network configuration bit strings of FPGAs to determine internal structures. The structure and weights of the BBNN are encoded as a 2-D chromosome for easier partial on-line reconfiguration. The 2-D encoding method used in this paper differs from the multidimensional encoding [19] method. A genetic algorithm evolves configuration bit strings to search for an optimal structure and weights setting of the BBNN among many possible choices of structure and weight combinations. Simula-

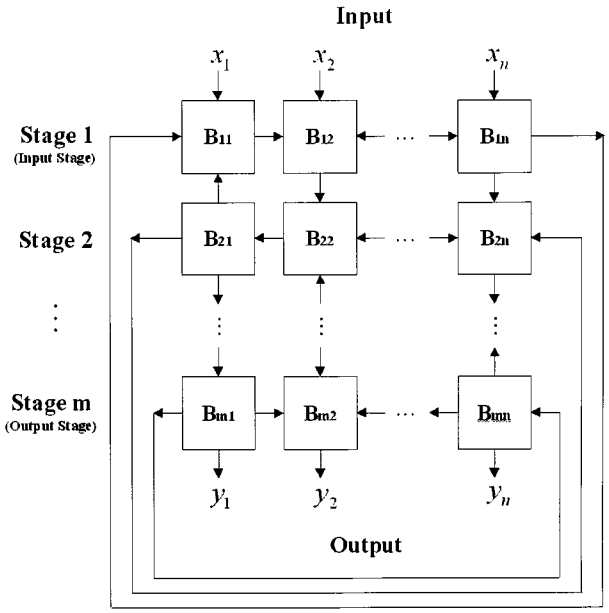


Fig. 1. Structure of the block-based neural network (BBNN).

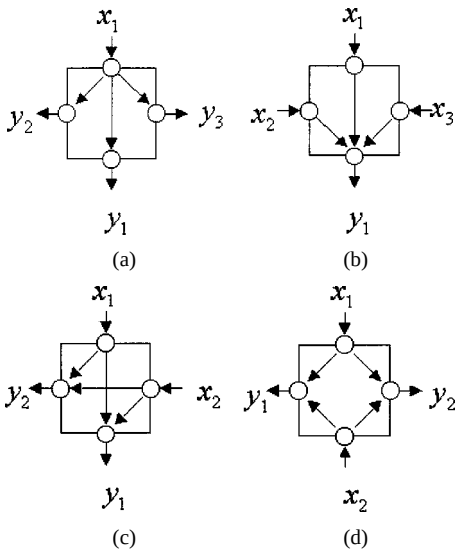


Fig. 2. Four different internal configuration types of a basic block. (a) 1/3. (b) 3/1. (c) 2/2. (d) 2/2.

tions show that the BBNN with structures and weights obtained by the genetic algorithm successfully solve pattern classification and autonomous mobile robot control problems.

II. Block-Based Neural Network Model

A. Architecture of the Block-Based Neural Network

The BBNN model consists of a 2-D array of basic blocks. Fig. 1 represents the structure of the BBNN model of $m \times n$ size. Each block is labeled as B_{ij}. The first stage ($i = 1$) and the last stage ($i = m$) denote the input and the output stages, respectively. The BBNN can have more than two middle stages. The $m \times n$ BBNN can have up to n inputs and n outputs. Redundant inputs take a constant value and redundant outputs are "don't care." Any block in the middle stages is connected directly with its neighboring blocks. The leftmost and rightmost blocks are connected with each other. The incoming arrow to a block denotes an input to the block, the outgoing arrow indicates output. Output of nodes become inputs for their neighboring blocks. For example, the output of block B_{12} becomes an input for block B_{22}. The output of blocks at the same level are calculated concurrently in order for system output to reflect current input. The BBNN can be implemented using reconfigurable hardware by arranging the schematic design of each basic block.

The structure of the BBNN defines signal flow represented by the arrows between the blocks. This type of structure allows signal flow in both forward and backward directions. Signal flow determines automatically internal configuration or input-output connections of basic blocks. Each block corresponds to a simple feedforward neural network having four nodes which are reconfigurable depending on network structure. The block has four different types of internal configurations depending on the input-output connections of the network structure. The different internal connections of the block allow far powerful generalization capabilities.

Fig. 2 shows four different internal structures. The signal flow between the blocks determines the I/O mode and thus internal structure of the basic block. Fig. 2(a) and Fig. 2(b) denote one input/three outputs and three inputs/one output cases, respectively. Fig. 2(c) and Fig. 2(d) correspond to two inputs/two outputs but with different internal connections. x_i and y_i indicate the input and output of the block. All the weights and biases have integer values for the purposes of easy hardware implementation. BBNNs have problem-dependent structures with both feedforward and feedback characteristics. Basic blocks have three possible I/O modes or three different internal structures: 1/3 (one input and three outputs), 2/2 (two inputs and two outputs), or 3/1 (three inputs and one output). The extreme case of all nodes becoming inputs (4/0) or outputs (0/4) is not allowed.

Each node is characterized by an activation function $g()$. Input nodes use a linear activation function, while the output nodes use symmetric saturated linear activation functions for hardware implementation. The output node can have bias terms with a constant (=1) input. Fig. 3 shows the shape of the symmetric saturating linear activation function with a gradient of $1/d$. For a more continuous output space with restricted training patterns, the slope of the activation function will be changed.

The BBNNs are designed for implementation using digital hardware such as FPGA and other reconfigurable chips that allow on-line partial reorganization. BBNNs have modular characteristics. The size can be expanded by adding more basic blocks. Network structure simply corresponds to determining signal flows. Connection weights are learned by the GA-based optimization procedure.

B. Characteristics of the BBNN

Let $MLP(n, m-1)$ denote an MLP network with a maximum number of neurons n in each layer and a maximum number of middle layers $m-1$. And let $w_{ij}^{(k,r)}$ denote the connection weight between the ith input neuron and the jth

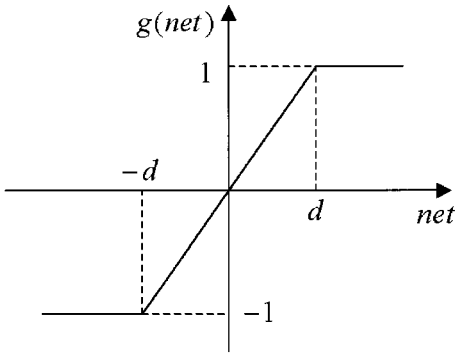


Fig. 3. Symmetric saturated linear activation function of the BBNN.

output neuron in the rth basic block B_{kr} in the kth stage. ($1 \le k \le m, 1 \le r \le n$).

Theorem 1: Let N_b be the maximum number of weights of the BBNN of size $m \times n$. Let N_m be the maximum number of weights of the MLP network $MLP(n, m-1)$. Then $N_b \ge N_m$ for $1 \le n \le 5$.

Proof: A basic block of the BBNN can have a maximum number of weights, which is six including the bias, when all the blocks are configured with the mode 2/2. The maximum number of weights of the $m \times n$ BBNN is $N_b = 6mn$. The maximum number of weights of a fully connected MLP network with a maximum number of neurons n in each layer and the maximum number of middle layers $m-1$ equals $N_m = mn(n+1)$.

$$\begin{aligned} N_m - N_b &= mn(n+1) - 6mn \\ &= m(n^2 - 5n). \end{aligned}$$

For $m > 0$ and $1 \le n \le 5$, $n^2 - 5n \le 0$. Therefore $N_m \le N_b$. □

Theorem 2: Any connection weight of an MLP network is represented by a combination of more than one weight of the BBNN, if all the basic blocks are of mode 2/2.

Proof: Any connection weight between pth neuron at kth layer and qth neuron at $(k+1)$th layer in the MLP MLP$(n, m-1)$ is represented by a set C_{pq} of weights of kth stage blocks $w_{ij}^{(k,r)}$. ($1 \le p \le n, 1 \le q \le n, 1 \le k \le m, 1 \le r \le n$)

$$C_{pq} = \begin{cases} \left\{ w_{21}^{(k,q)}, w_{22}^{(k,q-1)}, \cdots, w_{22}^{(k,p-1)}, w_{12}^{(k,p)} \right\} & \text{if } p > q \\ \left\{ w_{22}^{(k,1)}, \cdots, w_{22}^{(k,p-1)}, w_{12}^{(k,p)}, w_{21}^{(k,q)}, w_{22}^{(k,q+1)}, \cdots, w_{22}^{(k,n)} \right\} & \text{if } p < q \\ \left\{ w_{22}^{(k,p)} \right\} & \text{if } p = q \end{cases}.$$

□

Theorem 3: A BBNN of $m \times n$ represents the equivalent structure of the MLP network $MLP(n, m-1)$.

Proof: According to Theorem 2, an $m \times n$ BBNN can represent all the weights of the MLP network MLP$(n, m-1)$. Therefore the $m \times n$ BBNN can represent the equivalent structure of the MLP$(n, m-1)$. □

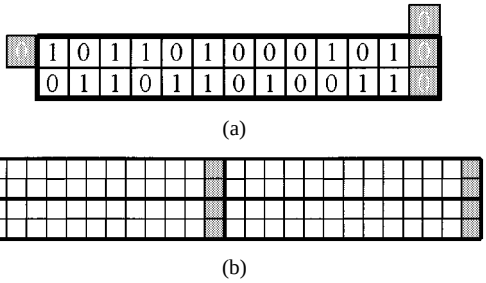


Fig. 4. Two-dimensional encoding of the BBNN. (a) Chromosome representation of a basic block. (b) Two-dimensional encoding.

The BBNN of size $m \times n$ can represent successfully the input–output characteristics of any MLP network for $n \le 5$. For example, both a 3-3-1 MLP and a 2×3 BBNN can classify three-bit odd parity patterns successfully. Since the BBNN can represent both feedforward and feedback connection characteristics, the BBNN can have an optimal structure for a given problem by learning through evolutionary algorithms.

III. Genetic Algorithm for Evolving BBNN

Optimization of the BBNN is composed of two tasks: structure and parameter optimizations. Structure optimization corresponds to the determination of the internal configuration of each block. Parameter optimization or learning is the process that determines the connection weight values for given input-output data. The search space is not continuous since BBNNs have integer value of weights. In a discrete search space and for a small amounts of training data, learning through evolutionary algorithms is better than through gradient-based search algorithms. Genetic algorithm finds the optimal solution in search space consisted of chromosomes according to fitness function, which represent a set of possible settings of structure and weights of the BBNN.

A. Encoding

The structure of the BBNN determines the signal flow between the blocks, and the signal flow between the blocks determines in turn the internal configuration or the input/output mode of each block. In order to optimize structure and parameters simultaneously, network structure and connection weights should be encoded in one chromosome at the same time. In conventional multidimensional encoding, each gene is assigned to one dimension. In this paper, the network structure and weights of each block are assigned to a two-dimensional chromosome in order to improve the performance of the crossover operation.

Fig. 4(a) represents a basic block chromosome a of the BBNN. Fig. 4(b) shows the result of two-dimensional encoding of the BBNN, whose component is a block encoded as in Fig. 4(a). All the weights are encoded by four-bit binary numbers. □ denotes connection weights, and ■ shows structure. Structure of BBNN is defined by signal flows. The value "0" indicates downward(↓) and leftward(←) signal flows , while "1" denotes upward(↑) and rightward(→) directions, respectively. Signal flows are common for neighboring blocks, and therefore adjacent blocks share the same signal flow arrows. The first stage of the BBNN has input nodes which corresponds

to all "0" signal flows, while the last stage also contains all "0" for output signal flow. The connection weight has value w from $-(2^{l-1}-1)$ to $(2^{l-1}-1)$ with two zeros which means the weight is disconnected. Equation (1) transforms positive integer w_b expressed by l-bit binary number into integer value w from $-(2^{l-1}-1)$ to $(2^{l-1}-1)$

$$w = w_b - 2^{l-1} + s(w_b) \tag{1}$$

where

$$s(w_b) = \begin{cases} 1 & \text{if } w_b < 2^{l-1} \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

Block-based neural networks have restrictions comparing to other generally known neural-network models. The BBNNs are designed for digital hardware implementation as FPGA that allows on-line partial reorganization. This means that reconfiguration bit strings must be generated without the help of microprocessors. Binary encoding of the BBNN is advantageous for digital hardware implementation and dynamic reconfiguration. In order to convert the structure and weights to reconfiguration bit strings by means of digital logic circuits, the BBNN parameters should be represented by bit strings. A genetic algorithm based on binary encoding has inherent shortcomings[17]. To overcome these problems, a modified encoding schemes and genetic operations are used.

B. Genetic Operators

A stochastic selection by the roulette wheel method is a basic reproduction mechanism used frequently in the genetic algorithm. The roulette wheel selection method is based on the fitness ratio, which has some weaknesses. In early stage of evolution, a chromosome with a larger fitness value than other chromosomes has a high survival probability in the reproduction process, which might cause premature convergence. Also, when individuals converge to near solution, an average fitness might be close to the population's best fitness. If this is the situation, the solution candidates with average and best fitness will have nearly the same number of copies in future generations. Then competition between individuals by genetic operators becomes low, and so individuals wander around the solution.

To overcome this problem, one can reduce the relatively high fitness values of the individual chromosomes, and the fitness difference between the chromosomes can be scaled by the distribution of the individual state of all fitness. The fitness scaling and ranking methods is one of the solutions for this problem [18]. The fitness ranking method ranks the chromosomes by fitness values and then redistributes fitness exponentially according to rank. The fitness ranking method does not consider the relation between object function and fitness. The linear fitness scaling method scales all fitness using maximum, minimum, and average fitness by a linear function.

The linear fitness scaling considers the state of all fitness, but if the average fitness is close to the maximum fitness, a fitness less than the average fitness can be evaluated as a negative value. To prevent evaluation as a negative fitness in the linear fitness scaling procedure, a modified scaling procedure transforms the maximum and average fitness to average and minimum fitness when negative fitness values occur. The modified linear fitness scaling method uses two different linear functions based on maximum, minimum, and average fitness values to avoid negative fitness values and genetic drift.

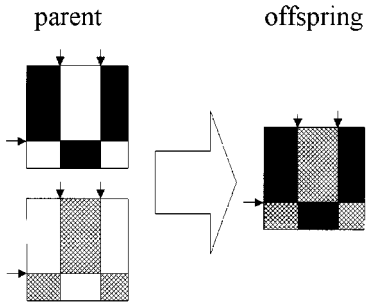


Fig. 5. Generalized 2-D crossover operation.

$$h(f_i) = \begin{cases} a_1 f_i + b_1 & \text{if } f_{\min} < f_s \text{ and } f_i < f_{avg} \\ a_2 f_i + b_2 & \text{otherwise} \end{cases}. \tag{3}$$

Equation (3) shows the linear scaling functions for each case. f_i is the fitness of ith chromosome and $h(f_i)$ is the scaled fitness of ith chromosome. $f_{\min}$, f_{avg}, $f_{\max}$ are the minimum, average, and maximum fitness by object function, respectively. The parameters in scaling functions are $a_1 = f_{avg}/(f_{avg} - f_{\min})$, $a_2 = (\alpha-1)f_{avg}/(f_{\max} - f_{avg})$, $b_1 = -f_{\min} f_{avg}/(f_{avg} - f_{\min})$, and $b_2 = f_{avg}(f_{\max} - \alpha f_{avg})/(f_{\max} - f_{avg})$. $f_s = (\alpha f_{avg} - f_{\max})/(\alpha - 1)$ is a criterion for $h(f_{\min}) < 0$ in the linear fitness scaling procedure. α is a linear scaling constant and effects the slope of linear scaling function. In general, for a small number of chromosomes, the slope takes the range $1.2 \leq \alpha \leq 2$. This fitness scaling helps the evolution scheme maintain evolution trend and diversity of individuals as f_{avg} gets close to $f_{\max}$. Together with the modified linear fitness scaling, elitist selection is used.

Genetic operators are modified in accordance with 2-D encoding. A chromosome composed of multidimensional encoding considers three types of crossover operations, extended from one-dimensional (1-D) encoding. First, one can select a crossover point for each dimension. Next, selection of the multicrossover point for each dimension is done, which is an extension of the first method. Finally, one can fix the total number of crossover points $\sum_{i=1}^{n} k_i = k(k_i \geq 0)$. The final method is a generalized form, which allows a more diverse schema than the crossover operations with fixed crossover points.

Fig. 5 shows an example of a generalized multidimensional crossover operation with $k_1 = 1$ and $k_2 = 2$ [19]. In the case of a 2-D generalized crossover for 2-D encoding, having one crossover point results in maintaining a variety of chromosomes and having two crossover points improves convergence. Different mutation probabilities are applied to the structure and weight optimizations bit in binary representation. p_{ms} denotes the mutation probability of the structure, and p_{mw} also represents the mutation probability for the weights.

The transmission of data between the blocks has feedforward and feedback characteristics. The internal copy operator

which played an important role in improving the evolution performance of neural network included a feedback characteristics [16]. In this paper, the internal copy operator enables a sufficient variety of all chromosomes after crossover. In 2-D encoding, a chromosome selected by internal copy probability p_{ic} copies a part of the chromosome to other parts. The size and location of the copy are randomly determined. For better performance, the inversion operator searches for a solution by the information of a chromosome selected by inversion probability p_i. Structural genes do not use inversion operators.

IV. Optimization of BBNN

A. Problem Definition

In order to verify the capability of the BBNN for solving practical engineering problems, pattern classification tasks and a mobile robot control problem were considered. First, the exclusive-OR (XOR) and the four-bit symmetry problems are used for showing the performance of pattern classification. The XOR problem contains two inputs and one output. If both inputs have the same values, the output equals -1. The output is one, if the inputs differ from each other. For the four-bit symmetry problem, the output is set to one if two most significant bits(x_1, x_2) and two least significant bits(x_3, x_4) are symmetric; and -1, otherwise. Inputs of the BBNN are normalized to have values between -1 and 1, and the output has values from -1 to 1 since the symmetric saturating linear activation function is used. For mobile robot control problem, the BBNN takes five inputs from sensors for successful navigation in a path.

B. Genetic Algorithm Parameters

The genetic algorithm for the optimization of the BBNN requires seven parameters: fitness scaling factor α, crossover probability p_c, mutation probability for weight p_{mw}, mutation probability for structure p_{ms}, internal copy probability p_{ic}, inversion probability p_{iv}, and population. All the parameters in the pattern classification and robot control problems are selected as the one not having premature convergence and reaches the fastest in 20 repeated trials.

Table I shows all the parameters used in genetic evolution for optimizing the BBNN. Parameters p_{ic} and p_{iv} are chosen to be small in order to improve the search performance and to avoid destruction of the schema. Connection weights of the BBNN for pattern classification and robot control simulations are represented by six bits.

C. Optimization of the BBNN for Pattern Classification

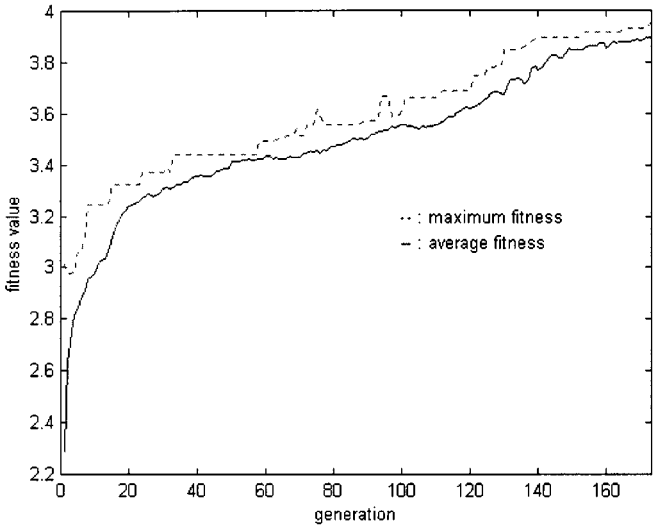
Genetic algorithms perform optimization with the fitness function. Maximizing fitness functions corresponds to minimizing error between the desired output and the actual output

$$\text{Fitness} = p_1 p_2 \left(N n_o - \sum_{j=1}^{N} \sum_{k=1}^{n_o} e_{jk}^2 \right) \tag{4}$$

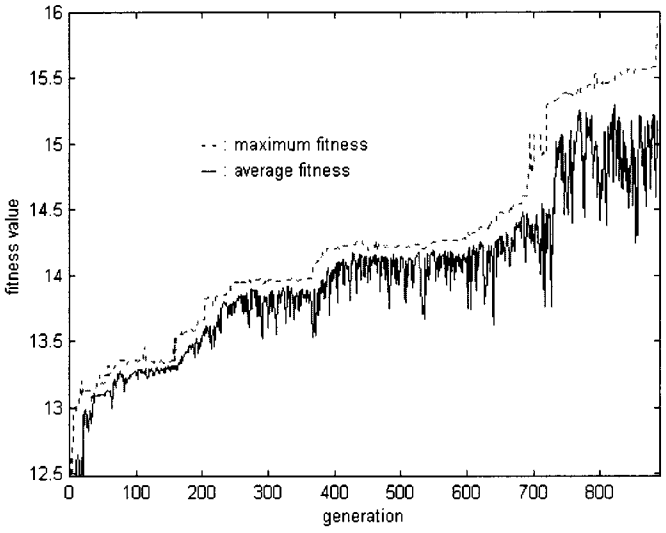
$$e_{jk} = d_{jk} - y_{jk}(x). \tag{5}$$

TABLE I
Parameter Settings for Optimizing BBNN

Parameter	Value
Fitness scaling factor (α)	1.6
Crossover probability (p_c)	0.35
Mutation for weight (p_{mw})	0.001
Mutation for structure (p_{ms})	0.005
Internal copy (p_{ic})	0.002
Inversion (p_{iv})	0.002
population	100



(a)



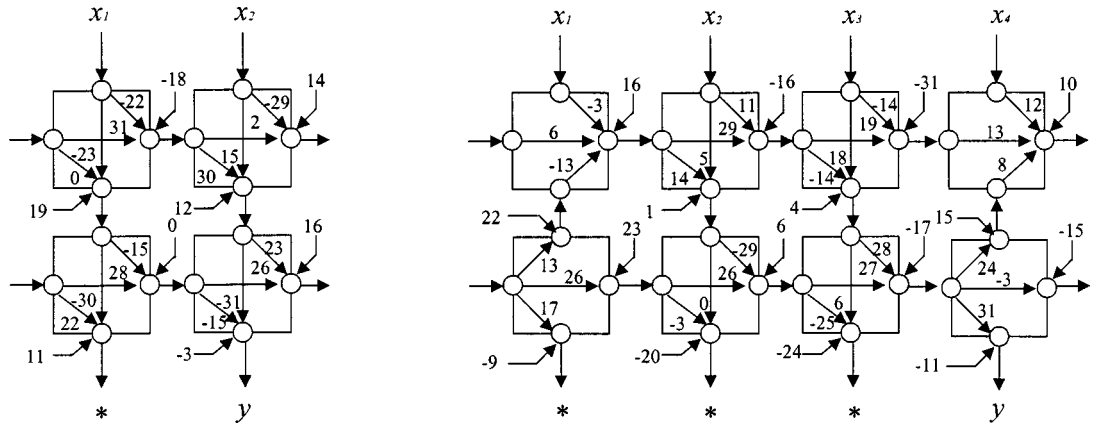
(b)

Fig. 6. Trend of fitness values during evolution. (a) XOR (b) Four-bit symmetry.

Equation (4) shows the fitness function used to optimize the structure and weights of the BBNN.

N number of training data;

n_o number of actual output nodes;



(a) (b)

0	0	1	0	0	0	0	1	1	1	1	1	1	1	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	0	1	0	1	1	0	0	1
0	0	1	0	0	1	1	1	1	1	1	1	0	0	1	1	0	1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	1	1	1	0	0
0	0	0	0	0	1	1	1	0	1	1	0	1	0	1	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	0	0	1
0	1	0	0	0	0	1	1	1	1	0	0	1	0	0	0	0	0	1	1	0	1	1	1	1	1	1	0	1	0	1	1	0	0	0	0	0

(c)

Fig. 7. Structure and weights of the BBNN optimized using the genetic algorithm. (a) XOR. (b) Four-bit symmetry. (c) 2-D encoding for the XOR problem.

d_{jk} and y_{jk} desired and actual outputs of the kth output block referred to jth pattern;

p_1 and p_2 penalty terms for assuring evolution of valid BBNN structures only.

Penalty term p_1 excludes invalid internal configuration of a block having all input nodes or all output nodes only, which means all the signal sof a block only coming in or going out. The term p_2 prevents invalid structures in which all the outputs of middle stage blocks are composed of only upward signal flow $\uparrow$. Normally, the penalty terms are set to $p_1 = 1$, $p_2 = 1$. When a structure chromosome contains at least one invalid structure, the penalty terms are assigned to a low fitness of 0.9.

Fig. 6 shows the trend of fitness values in evolution for the two pattern classification problems. The dotted line and the solid lines denote the maximum and average fitness values, respectively. In (4), the maximum fitness of a single output is the same as the number of data (N). Fitness trends show the evolution procedure takes longer in the latter part of the evolution in order to satisfy the end conditions of approaching 99.9% of the maximum fitness. After an evolution procedure achieving 99.9% of max fitness, the error between the actual output and the desired output is quite big, since the defined fitness function has a near uniform distribution in the interval. Increasing the interval for higher fitness values could solve this problem. The mean-square errors by the BBNN classifiers after evolution procedure were zero for the XOR and 0.062 for the four-bit symmetry problems.

The structure and weights of the BBNN optimized by the genetic algorithm for different pattern classification problems are shown below. The number of inputs, which is bigger than that of outputs, determines the size of the BBNN. Two stages were enough to classify the patterns in the two problems.

Fig. 7 shows the structure and weights of the optimized BBNN for the given patterns after the evolution procedure. The rightmost node was arbitrarily chosen as the output node. All the redundant output nodes are marked as *. All integer numbers denote optimized weights and biases. The optimized BBNN structure differs from that of the multilayer feedforward neural networks. The XOR pattern classifier based on the BBNN having different structure from the multilayer perceptron neural network with two input nodes, two middle layer nodes, and one output node(2-2-1). The BBNN optimized with the GA successfully classified the patterns with integer weights. The number of possible BBNN structures depends on the network size. BBNNs of size 2×2 can have up to 64 possible structures, 2×4 BBNN can have 4096 possible structure combinations.

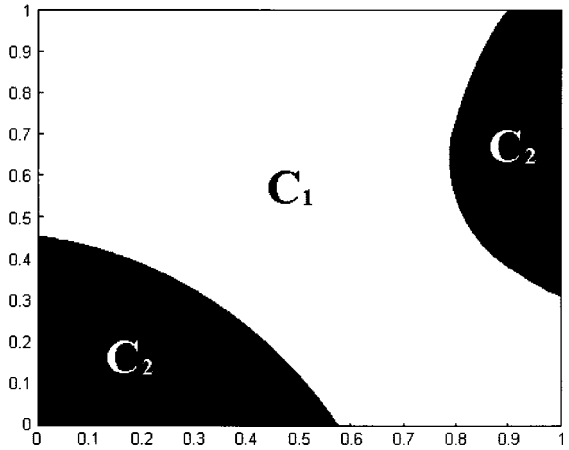
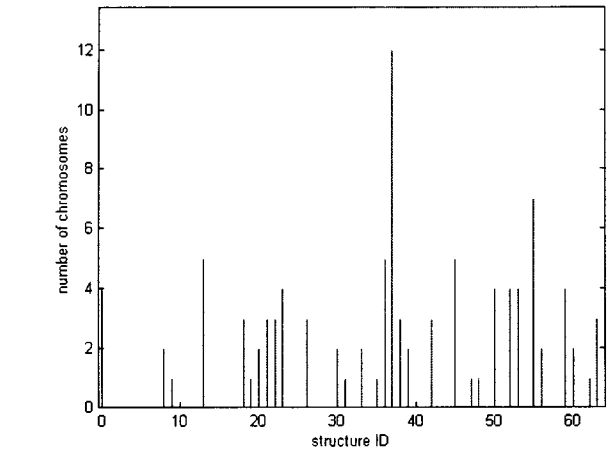


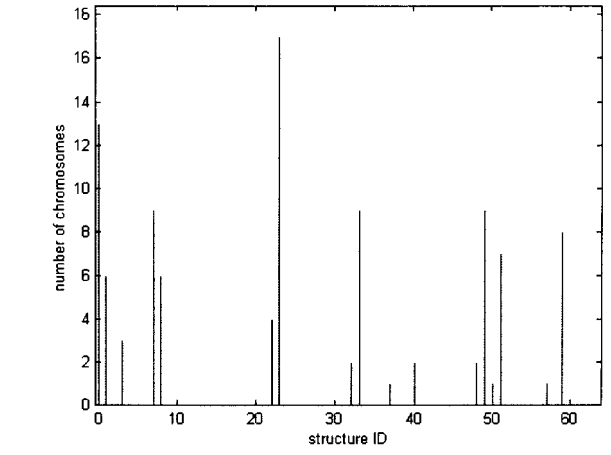
Fig. 8. Decision boundary for the XOR classification problem.

Fig. 8 shows the decision boundary for the XOR problem generated by the BBNN trained with the genetic algorithm. The decision boundary represents the nonlinear characteristics of the XOR pattern classification problem.

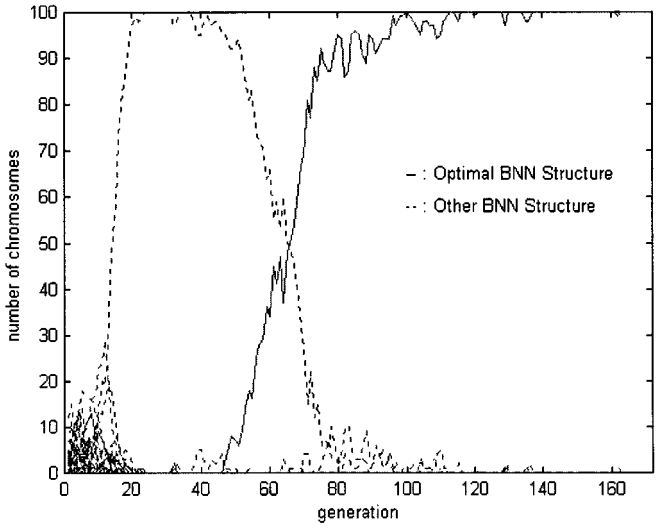
Fig. 9 shows the evolution procedures of the structure optimization for the XOR problem. Fig. 9(a) represents the ini-



(a)

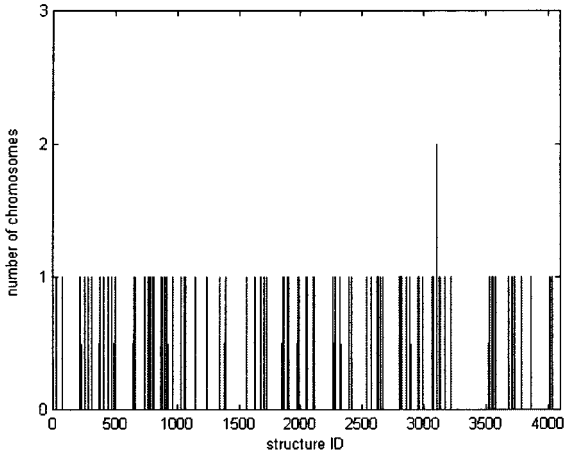


(b)

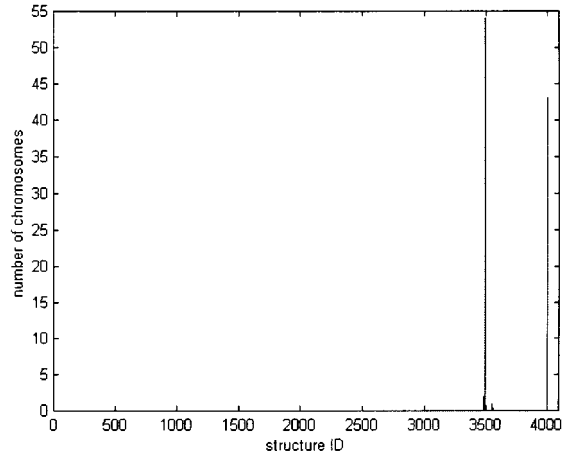


(c)

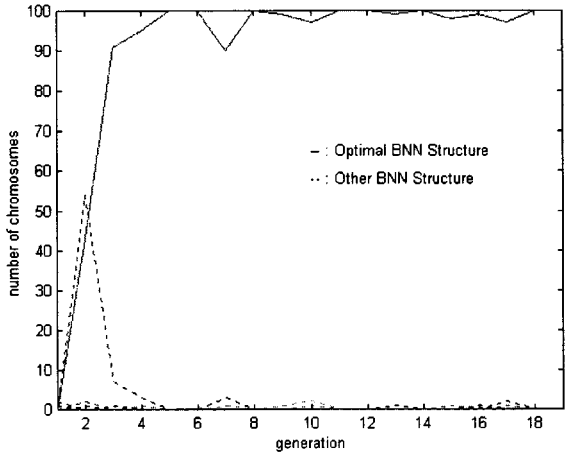
Fig. 9. Convergence of the optimal BBNN structure for the XOR problem. (a) Initial structure distribution. (b) Structure distribution after eight generations. (c) Convergence of the optimal BBNN structure.



(a)



(b)



(c)

Fig. 10. Convergence of the optimal BBNN structure for the four-bit symmetry problem. (a) Initial structure distribution. (b) Structure distribution after 50 generations. (c) Convergence of the optimal BBNN structure.

tial distributions of all the possible combinations of the BBNN structures. Initial BBNN structures are randomly selected, while the BBNN with structure ID 37 was dominant. The number in the x-axis denotes the decimal ID number decoded from a bit string that represents a specific BBNN structure. Fig. 9(b) indicates a structure distribution after eight generations. Many nonoptimal structures disappeared in natural selection. Fig. 9(c) represents the trends of an optimal structure that survive in an evolution procedure. The solid line denotes the number of optimized chromosomes in population, the dotted line shows the sum of the number of the other structures. After optimization procedure, GA finds the optimal structure (ID 51).

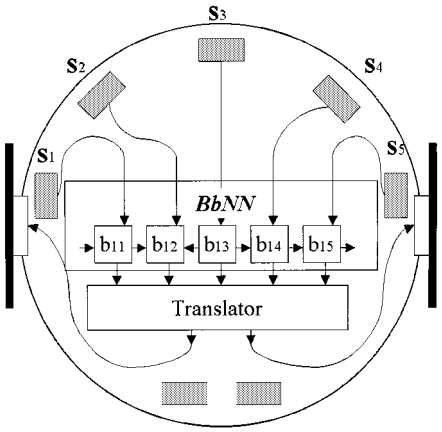


Fig. 11. Simulated Khepera robot with the BBNN controller.

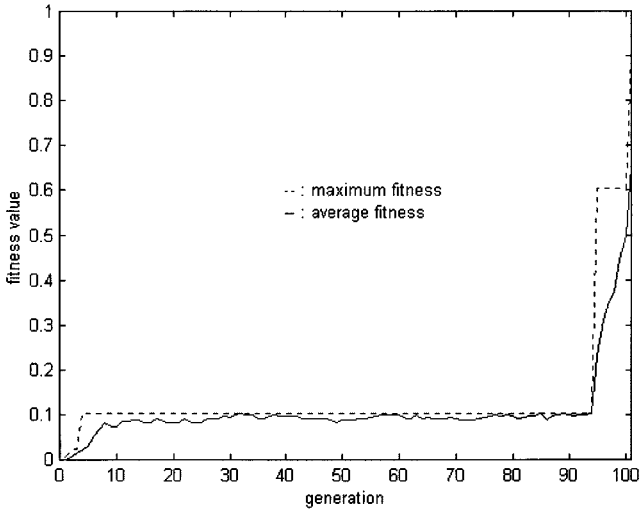


Fig. 12. Trend of fitness values for mobile robot control.

Fig. 10 shows the evolution procedure of the structure optimization for the four-bit symmetry problem. Fig. 10(a) represents the initial distributions of all the possible BBNN structures, and Fig. 10(b) represents BBNN structure distribution after 50 generations. In an initial state of evolution, the population involves diverse types of structures while the structure ID 3100 was in the majority in early evolution process. As the evolution procedure went on, most structures disappeared and only a few structures survived in the evolution process after 50 generations. Finally, the BBNN converged to an optimal structure with structure ID 3999 after 884 generations. This structure is expressed in Fig. 7(b). Fig. 10(c) represents the trends of an optimal structure at every 50 generations that survive in an evolution procedure as in Fig. 9(c).

D. Optimization of the BBNN for Mobile Robot Control

Autonomous mobile robots recognize objects and move freely by means of their wheels based on their own decisions from sensor input data taken from surroundings. A mobile robot model Khepera [20] is used to test the control capability of the BBNN. In the simulation, all the robot parameters such as sensor locations and sensing range are the same as those of real Khepera robots. Two backward sensors are not used since only forward movement is assumed. Two forward sensors are combined as a single forward sensor for simplicity. A maximum sensing range is 32 [mm], and diameter of a robot is assumed 48 [mm]. The path is assumed to be 128 [mm] wide in order to guarantee the robots can sense wall accurately.

A BBNN of 1×5 size was used to control mobile robots for navigation in a path. Fig. 11 shows the simulated Khepera robot with the BBNN controller. The BBNN takes five inputs from proximity sensors and produce coded output for motor control. Five output signals are converted to two motor control signal for the two wheel motors. The robots are assumed to move forward in a constant speed. The motor output determines rotation angles of the mobile robots. The coded output of the BBNN is translated to actual motor control signal. In the simulation, the robot can rotate by 0°, $\pm 45^\circ$, and $\pm 90^\circ$.

The objective of mobile robot control problem is to achieve navigation along a path with no collision. Genetic algorithm can achieve object dependent optimization based on the fitness function. The fitness function is defined as

$$f(c, s) = \left(1 - \frac{c}{c_{\max}}\right)\left(1 - \frac{r}{r_{\max}}\right) \tag{6}$$

where c denotes the number of collisions and r indicates the number of materials that robots must collect along the navigation path. $c_{\max}$ and $r_{\max}$ are the maximum values of c and r. In a training path, optimal BBNN is obtained by genetic procedure and tested in other navigation path.

Fig. 12 shows trend of fitness values for the mobile robot control problem. Theoretical maximum fitness value equals one according to (6). After 100 generations, no collisions occur and the number of materials found was 111. The maximum fitness value was $f(c, s) = 0.909\,836$.

Fig. 13 shows the initial distribution of structures for the mobile robot control problem. In all 32 possible choices of structure settings, structure of ID 14 survived and becomes a dominant structure after 87 generations.

Fig. 14 shows the final structure of ID 14 and weights of the BBNN for mobile robot control. The obtained BBNN corresponds to the one with the maximum fitness value by genetic procedure. Outputs of the BBNN consists of 4-bit control codes y_1, y_2, y_3, and y_4. The output expresses rotation angle of mobile robot at each step.

Fig. 15 shows a navigation by best individual during whole genetic procedure. For training of path navigation, 122 materials are put uniformly over the path. Robots are required to collect all the materials on the path in order to learn navigation path correctly. The trained robot finds the optimal navigation path with no collisions.

Fig. 16 shows trajectory of a robot with the BBNN controller in a path not used in training procedure. A single stage BBNN could control a mobile robot for path navigation without collision. The robot with the optimized BBNN controller having the maximum fitness value shows no collisions to find the optimal navigation path. Mondada [21] demonstrated a navigation and

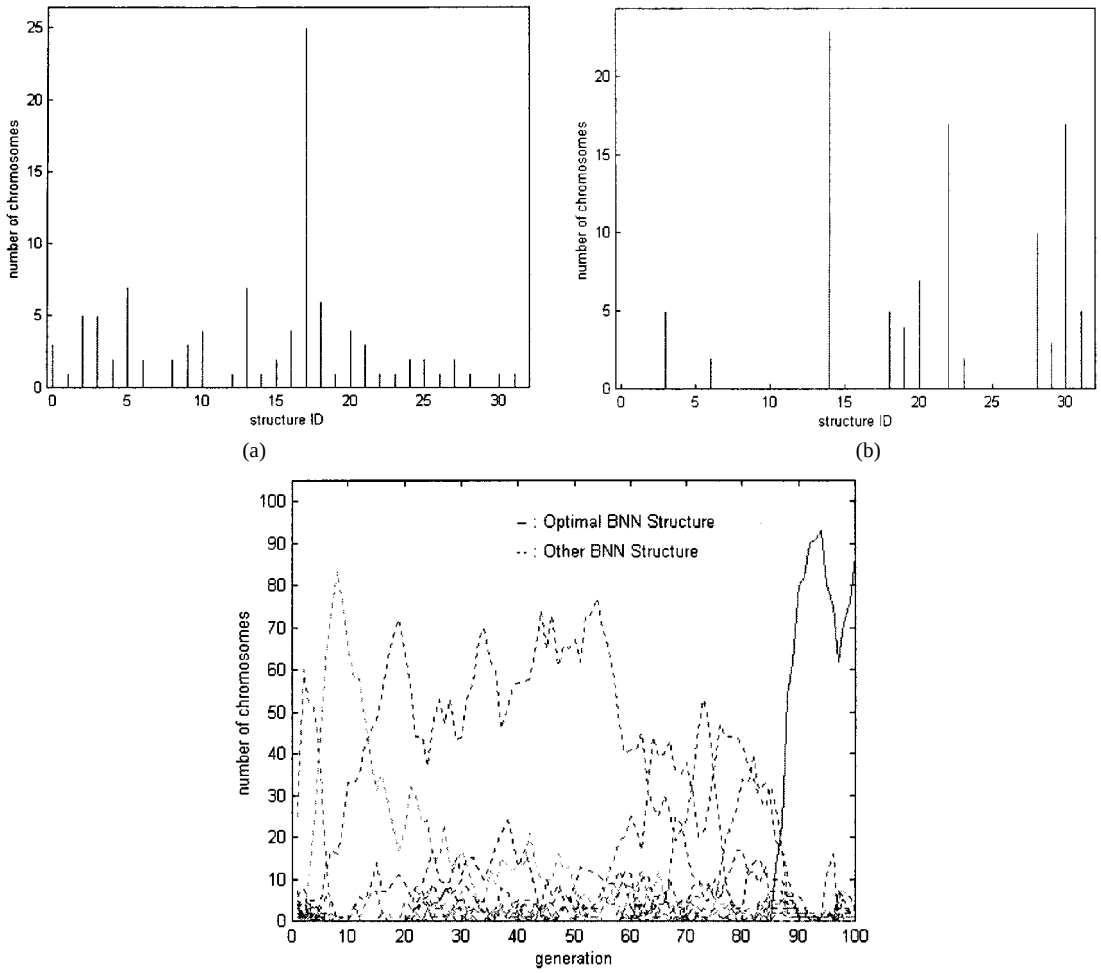


Fig. 13. Convergence of the optimal BBNN structure for the mobile robot control problem. (a) Initial structure distribution. (b) Structure distribution after 87 generations. (c) Convergence of the optimal BBNN structure.

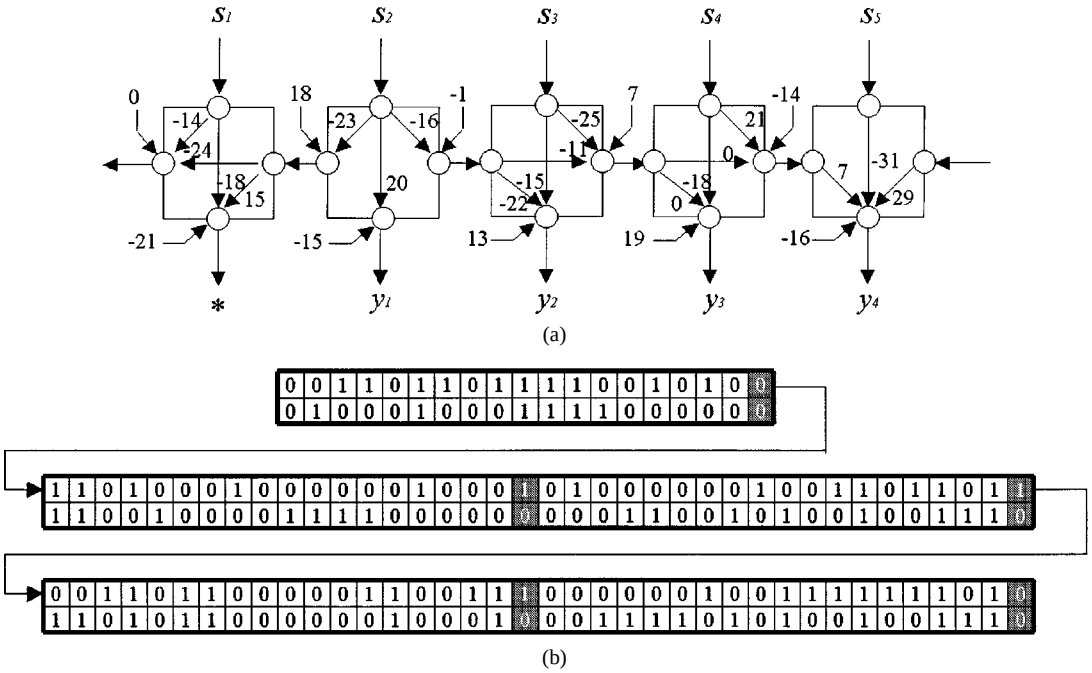


Fig. 14. Final structure and weights of the BBNN for the mobile robot control problem. (a) Final structure and weight of BBNN. (b) 2-D encoding of the optimized BBNN.

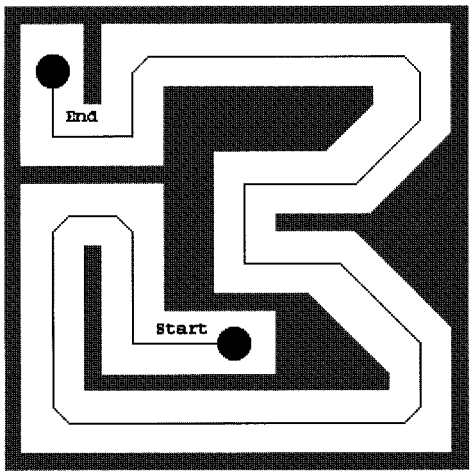


Fig. 15. Trajectory of a mobile robot with maximum fitness in training path.

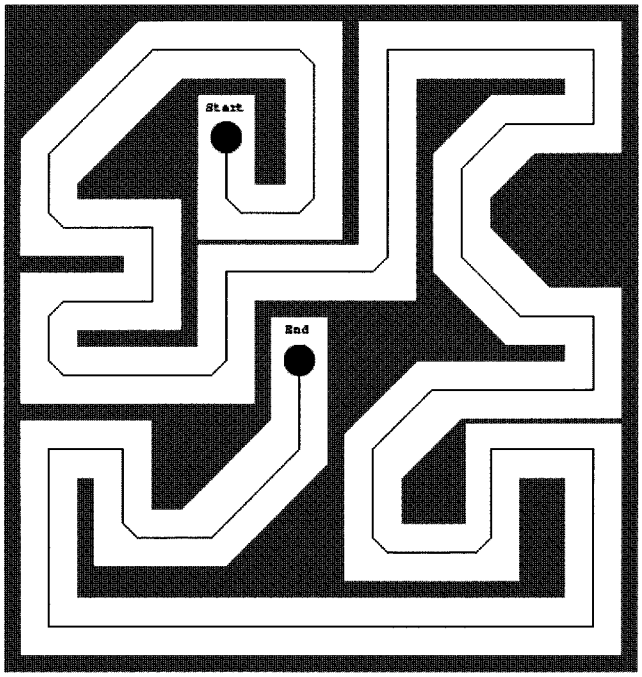


Fig. 16. Trajectory of a mobile robot in test path.

obstacle avoidance experiment of the autonomous mobile robot Khepera in an open space with an obstacle in the middle using a single-layer feedforward neural network. The neural network has self-feedback and allows lateral connection among the output neurons. The single-stage BBNN in Fig. 14(a) was enough to control the mobile robot navigation problems.

V. Conclusion

This paper presents a novel block-based neural network model and its application to pattern classification and mobile robot control problems. The BBNN consists of 2-D array of basic blocks which are suitable for implementation using reconfigurable hardware such as FPGAs. The structure and weights of BBNN are optimized simultaneously by a genetic algorithm. The genetic algorithm searches for global optimum of structure and weights among a possible combinations of structure and weights. The optimized BBNN can solve practical problems as pattern classification and mobile robot control. The genetic algorithm uses 2-D encoding, modified scaling, and the elitist method. A normalized 2-D crossover operator and different mutation probabilities in structure and weight bit strings are applied in the optimization of the BBNN. In order to enhance its performance, internal copy and inversion operators are used. Input–output patterns with nonlinear decision boundaries were used to evaluate pattern classification performance of the BBNN. The proposed BBNN models, evolved with the genetic algorithm, could solve pattern classification and mobile robot control problems.

References

[1] A. J. Montalvo, R. S. Gyurcsik, and J. J. Paulos, "Toward a general-purpose analog VLSI neural network with on-chip learning," *IEEE Trans. Neural Networks*, vol. 8, pp. 413–423, Mar. 1997.

[2] L. Raffo, S. P. Sabatini, and G. M. Bisio, "Analog VLSI circuits as physical structures for perception in early visual tasks," *IEEE Trans. Neural Networks*, vol. 9, pp. 1483–1494, Nov. 1998.

[3] D. B. Fogel, *Evolutionary Computation: The Fossil Record*: IEEE Press, 1998.

[4] T. Bäck, U. Hammel, and H.-P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 3–17, 1997.

[5] X. Yao and Y. Shi, "A preliminary study on designing artificial neural networks using co-evolution," in *Proc. IEEE Int. Conf. Intell. Contr. Instrum.*, Singapore, 1995, pp. 149–154.

[6] M. Scholz, "A learning strategy for neural networks based on a modified evolutionary strategy," in *Proc. Parallel Problem Solving from Nature*, H.-P. Schwefel and R. Männer, Eds. Heidelberg, Germany: Springer-Verlag, 1991, pp. 314–318.

[7] X. Yao, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Networks*, vol. 8, pp. 694–713, May 1997.

[8] J. R. McDonnell and D. Waagen, "Evolving recurrent perceptrons for time-series modeling," *IEEE Trans. Neural Networks*, vol. 5, pp. 24–38, 1994.

[9] P. J. Angeline, G. M. Sauders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 54–65, Jan. 1994.

[10] N. J. Radcliffe, "Genetic set recombination and its application to neural network topology optimization," Univ. Edinburgh, Edinburgh, U.K., Tech. Rep. EPCC-TR-91-21, 1991.

[11] D. J. Montana and L. Davis, "Training feedforward neural networks using genetic algorithms," in *Proc. 11th Joint Conf. Artificial Intell. (IJCAI)*, 1989, pp. 762–767.

[12] P. G. Korning, "Training neural networks by means of genetic algorithms working on very long chromosomes," *Int. J. Neural Syst.*, vol. 5, no. 3, pp. 299–316, 1995.

[13] D. Whitley and T. Starkweather, "Optimizing small neural networks using a distributed genetic algorithm," in *Proc. Int. Joint Conf. Neural Networks*. Hillsdale, NJ: Lawrence Erlbaum, 1990, vol. 1, pp. 206–209.

[14] D. Whitley, T. Starkweather, and C. Bogart, "Genetic algorithms and neural networks: Optimizing connections and connectivity," *Parallel Comput.*, vol. 14, pp. 137–170, 1995.

[15] M. Vittorio, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, pp. 39–53, Jan. 1994.

[16] T. Kumagai, M. Wada, S. Mikami, and R. Hashimoto, "Structured learning in recurrent neural network using genetic algorithm with internal copy operator," in *Proc. IEEE Int. Magn.Conf.*, 1997, pp. 651–656.

[17] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs 3rd Rev. and Extended Ed.*. New York: Springer-Verlag, 1998.

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[19] T. N. Bui and B. R. Moon, "On multi-dimensional encoding/crossover," in *Proc. 6th Int. Conf. Genetic Algorithms (ICGA)*, 1995, pp. 49–56.

[20] F. Mondada, E. Franzi, and P. Ienne, "Mobile robot miniaturization: A tool for investigation in control algorithms," in *Proc. 3rd Int. Symp. Experimental Robot.*, 1993, pp. 501–513.
[21] F. Mondada and D. Floreano, "Evolution and mobile autonomous robotics," in *Toward Evolvable Hardware*, E. Sanchez and M. Tommasini, Eds. New York: Springer-Verlag, 1996, pp. 221–249.

Sang-Woo Moon received the B.S. and the M.S. degrees in electrical engineering from the Soongsil University, Seoul, Korea, in 1998 and 2000, respectively.

His research interests include intelligent robot control, pattern recognition, real-time system, and evolutionary computation.

Seong-Gon Kong (S'89–M'92) received the B.S. and the M.S. degrees in electrical engineering from Seoul National University, Seoul, Korea, in 1982 and 1987. In 1991, he received the Ph.D. degree in electrical engineering from the University of Southern California (USC), Los Angeles.

Dr. Kong is an Associate Professor of electrical engineering at Soongsil University, Seoul, Korea, and was Department Chair from August 1999 to July 2000. Since August 2000 he has been with the School of Electrical and Computer Engineering at Purdue University, West Lafayette, IN, as a Visiting Scholar. His research interests include intelligent signal processing, artificial neural networks, fuzzy systems, pattern recognition, and evolutionary robotics.