Proceedings of the 1998 IEEE
International Conference on Robotics & Automation
Leuven, Belgium • May 1998

A New Evolutionary Approach to Developing Neural Autonomous Agents

Jinn-Moon Yang, Jorng-Tzong Horng*, and Cheng-Yan Kao

Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
*National Central University, Chungli, Taiwan
E-mail: {moon,cykao}@solab.csie.ntu.edu.tw
*E-mail: horng@db.csie.ncu.edu.tw

Abstract

This paper explores the use of neural networks to control robots in tasks requiring sequential and learning behavior. We propose a Family Competition Evolutionary Algorithm (FCEA) to evolve networks that can integrate these different types of behavior in a smooth and continuous manner. The approach integrates self-adaptive Gaussian mutation, self-adaptive Cauchy mutation, decreasing-based Gaussian mutation, and family competition. In order to illustrate the power of the approach, we apply this approach to two different task domains: the "artificial ant" problem and a sequential behavior problem – an agent learns to play football. From the experimental results, we find our approach performs much better than other evolutionary algorithms in these two tasks. The main contribution of the paper, based on the results from our experiments, is that our approach can evolve neural networks to provide a means of integrating sequencing and learning within a single control system.

I. Introduction

Traditional knowledge-based AI approaches of constructing intelligent robotic systems do not provide satisfactory solutions in application to dynamic real-world problems. Such systems have often required enormous computational power to make control decisions and then to accomplish their tasks. More serious problem is that these systems must make many assumptions about the information that is supplied by the sensor devices. These systems become useless and unreliable when the information become invalid. Behavior-based control systems have been offered as alternative methods to traditional techniques of designing robotic control systems. Therefore, the researchers on autonomous robots have focused on behavior-based robotics [3, 4].

A number of research for behavior-based robotics have successfully employed evolutionary approaches to develop the control systems of simulated robots. These adaptive evolutionary methodologies, including classifier system [19, 5, 6], forward neural networks [15, 17], recurrent neural networks [8, 13, 16 20], and genetic programming [14], focused on genetic algorithms (GAs) to extract learning rules or to train the structures and weights of neural networks. However, GAs have several disadvantages to train neural networks [1, 18]. First, traditional bit-strings GAs applying to numerical optimization problems have certain limitations and more inefficient than Gaussian mutation [9]. Therefore, several research used real valued to replace the bit-string representation in order to avoid ill effect [7, 2]. Second, both traditional GAs and real valued GAs employed random mutation that causes a large jump, so GAs may be insufficient for local tune. Third, GAs depend on recombination operator heavily but it causes competing conventions problem [18]. The number of competing conventions grows exponentially with the number of hidden units.

A behavior-based robot must possess self-adaptive ability and tolerate noise in order to adapt its behavior to any changes of its environment. Connectionist methodology allows the task demands to avoid the bias of the designer to be the primary force in shaping the system development. Neural networks are flexible, robust, and tolerated noise. Combining the evolutionary algorithm with neural networks can develop a plausible and powerful system.

The main focus of this article is on developing a new evolutionary algorithm called Family Competition Evolutionary Algorithm (FCEA) to train recurrent neural networks for adaptive robotic control systems. The approach integrates self-adaptive Gaussian mutation employed in evolution strategies (ESs) [2], self-adaptive Cauchy mutation, decreasing-based Gaussian mutation, and family competition. In order to illustrate the power of our approach, we considered two different types of tasks: the "artificial ant" problem and the sequential behavior problem – an agent learns to play football. Experimental results indicate that our FCEA outperforms these evolutionary approaches. Our FCEA is fast at least ten times of genetic algorithm and genetic programming.

II. Recurrent Neural Networks

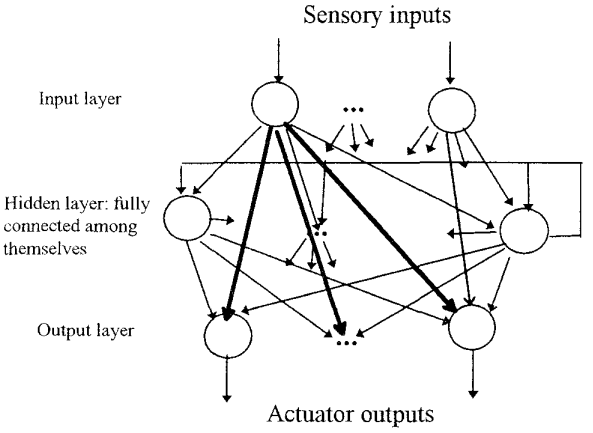
In this paper, the recurrent network is a mutilayer fully connected with short cut recurrent networks of continuous sigmoid nodes (Fig. 1). Each hidden node and output node of the network is govern by the following form:

$$y_i(t) = \sum_{k=1}^{S} w_{ki} I_k + \sum_{j=1}^{H} w_{ji} f(y_j(t-1) - \theta_j) \qquad (1)$$

where y_i is the state of the neuron; S is number of sensory inputs; I_k is the output of the k^{th} sensor; H is the number of hidden nodes; θ is a bias threshold input; w_{ki} and w_{ji} are the strength of the connection; $f(.)$ is standard sigmoidal activation which is given below.

$$f(\eta) = (1 + e^{-\eta})^{-1} \quad (2)$$

From Equation (1), we know the network maps the current sensory inputs into all hidden nodes and output nodes. The hidden nodes are fully connected themselves. This control network gathers input data from the sensor and determines how to set actuator outputs for the next step. The initial state of all neurons is set to zero. The initial weights of all connection and bias are set to the range [-0.1,+0.1] in this paper.



⟶: short cut : connected from input nodes to output nodes directly

Fig. 1. The control network is fully connected with short cut recurrent networks.

III. The Learning Method

First, FCEA generates a population of N networks, each network is represented as a quadratic real vector, $(\vec{x}_i, \vec{\sigma}_i, \vec{\nu}_i, \vec{\psi}_i), \forall i \in \{1,..., N\}$. The dimensions corresponding to the connection weights in the neural network are assume to m. $\vec{x}$ is the desired optimizing variable vector, i.e., the vector of connection weights of a network. $\vec{\sigma}$, $\vec{\nu}$, and $\vec{\psi}$ are n-dimensional real vectors that correspond to the strategy variable parameters of decreasing-based Gussian mutation, self-adaptive Gaussian mutation, and self-adaptive Cauchy mutation, respectively. The initial values of each component of $\vec{x}_i$, $\forall i \in \{1,..., N\}$, are ranged over [-0.1, 0.1]. The initial values of $\vec{\sigma}_i$, $\vec{\nu}_i$, and $\vec{\psi}_i$, $\forall i \in \{1,..., N\}$, are set to 4.0, 1.0, and 1.0 respectively. To evaluate the fitness score for each network $\vec{x}_i, \forall i \in \{1,..., N\}$, of the population is based on objective function $f(\vec{x}_i)$. FCEA then enters main evolutionary procedures: decreasing-based Gaussian mutation stage, self-adaptive Cauchy mutation stage, and self-adaptive Gaussian mutation stage. Each stage has four steps: recombination, mutation, family competition and population selection (used in decreasing-based Gaussian mutation stage) or replacement (used in self-adaptive mutation stage). The FCEA algorithm is given in Fig. 2.

In order to illustrate the genetic operators, let us denote two parents as $\vec{a} = (\vec{x}_a, \vec{\sigma}_a, \vec{\nu}_a, \vec{\psi}_a)$ and $\vec{b} = (\vec{x}_b, \vec{\sigma}_b, \vec{\nu}_b, \vec{\psi}_b)$ respectively, and the generated offspring as $\vec{c} = (\vec{x}_c, \vec{\sigma}_c, \vec{\nu}_c, \vec{\psi}_c)$. In the introduction of recombination operators, we identify two parents as a family parent ($\vec{a}$) and a parent ($\vec{b}$). In the following subsection, the symbol, x_j^d, in this paper denotes jth connection link of the individual $\vec{d}$ and $\forall j \in \{1,..., m\}$, where m is the number of links of a network.

A. Recombination Operators

Modified Discrete Recombination: The child $\vec{c}$ is generated by using the following modified discrete recombination operator.

$$x_j^c = \begin{cases} x_j^a & \text{with probabilty } 0.8 \\ x_j^b & \text{with probabilty } 0.2 \end{cases} \quad \forall j \in \{1,...,\text{m}\} \quad (3)$$

Generally, the original discrete recombination generates a child that inherits genes from two parents with equal probability. We modify discrete recombination such that the child inherits genes from the family parent $\vec{a}$ with probabilities 0.8 and another parent $\vec{b}$ with probability 0.2. That is, a child inherits genes from the family parent with high higher probability.

BLX-0.5 and Intermediate Recombination: The BLX-0.5 is used successfully in GAs and ESs. It generates a child $\vec{c}$ based on Equation (4).

$$\varpi_j^c = \varpi_j^a + \beta(\varpi_j^b - \varpi_j^a), \forall \text{j} \in \{1,...,\text{m}\} \quad (4)$$

Where β is chosen from uniform distribution in [-0.5, 1.5]. BLX-0.5 is called as intermediate recombination when β is equal to 0.5. FCEA employed the intermediate recombination in strategy variables, i.e. $\vec{\sigma}$, $\vec{\nu}$, and $\vec{\psi}$. In contrast to strategy variables, FCEA applied discrete recombination, BLX-0.5, and intermediate recombination to connection weights, $\vec{x}$, with different probabilities. They are p_{dc}, p_{bc}, and p_{ic}, respectively. In this paper, p_{dc}, p_{bc}, and p_{ic} are set to 0.7, 0.1 and 0.2, respectively.

---

t←0 /* t is the generation */ /* P(t), P'(t) is the t-th population and t-th quasi-population N Solutions */

Initialize population P(t) ($(\vec{x}_i, \vec{\sigma}_i, \vec{\nu}_i, \vec{\psi}_i), i \in \{1,..., N\}$,

Evaluate fitness score of each individual $f(\vec{x}_i, \vec{\sigma}_i, \vec{\nu}_i, \vec{\psi}_i)$

while termination criteria is not satisfied **do**

/* Decreasing-based Gaussian Mutation Stage */

$C \leftarrow \varnothing$ /* set children set C to empty */

for each individual $\vec{a} = (\vec{x}_i, \vec{\sigma}_i)$ of P(t)

$\vec{c}_{best} \leftarrow +\infty$

Repeat L_d times: ***/* family competition */***

a. Randomly select another individual $\vec{b}$ from P(t)

b. $\vec{a}' = (\vec{x}_i', \vec{\sigma}_i')$ ←**recombination(**$\vec{a}, \vec{b}$**)**

c. $\vec{a}'' = (\vec{x}_i'', \vec{\sigma}_i'')$ ←**GaussianMutation(**$(\vec{x}_i', \vec{\sigma}_i')$

d. if $f(\vec{a}'') \le \vec{c}_{best}$ then $\vec{c}_{best} \leftarrow \vec{a}''$ **endif**

Add $\vec{c}_{best}$ to the children set C.

endfor

P'(t) ←select the best N candidates from P(t) and children set C

/* Self-adaptive Gaussian Mutation Stage */

for each individual $\vec{a} = (\vec{x}_i, \vec{\nu}_i)$ in P'(t)

$\vec{c}_{best} \leftarrow +\infty$

Repeat L_a times ***/* family competition */***

a. Randomly select another individual $\vec{b}$ from P'(t)

b. $\vec{a}' = (\vec{x}_i', \vec{v}_i') \leftarrow$**recombination**$(\vec{a}, \vec{b})$

c. $\vec{a}'' = (\vec{x}_i'', \vec{v}_i'') \leftarrow$**GaussianMutation**$(\vec{x}_i', \vec{v}_i')$

d. **if** $f(\vec{a}'') \le \vec{c}_{best}$ then $\vec{c}_{best} \leftarrow \vec{a}''$ **endif**

if $f(\vec{c}_{best}) \le f(\vec{a})$ **then** /* ***replacement selection*** */

replace $\vec{a}$ with $\vec{c}_{best}$ in P'(t)

else $\vec{v}_i' = \gamma * \vec{v}_i'$

endif

endfor

/* **Self-adaptive Cauchy Mutation Stage** */

for each individual $\vec{a} = (\vec{x}_i, \vec{\psi}_i)$ in P'(t)

$\vec{c}_{best} \leftarrow +\infty$

Repeat L_a times /* ***family competition*** */

a. Randomly select another individual $\vec{b}$ from P'(t)

b. $\vec{a}' = (\vec{x}_i', \vec{\psi}_i') \leftarrow$**recombination**$(\vec{a}, \vec{b})$ $\vec{a}'' = (\vec{x}_i'', \vec{\psi}_i'')$

c. $\vec{a}'' = (\vec{x}_i'', \vec{\psi}_i'') \leftarrow$**CauchyMutation**$(\vec{x}_i', \vec{\psi}_i')$

d. **if** $f(\vec{a}'') \le \vec{c}_{best}$ then $\vec{c}_{best} \leftarrow \vec{a}''$ **endif**

if $f(\vec{c}_{best}) \le f(\vec{a})$ then /* ***replacement selection*** */

replace $\vec{a}$ with $\vec{c}_{best}$ in P'(t)

endif

endfor

P(t+1)←P'(t)

t←t+1

endwhile

Fig. 2. The FCEA Algorithm

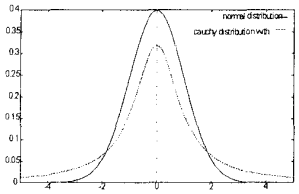


Fig. 3. The probability density distribution N(0,1) and Cauchy distribution with t=1.0.

B. Mutation Operators

Self-adaptive Gaussian Mutation: Schwefel [2] proposed a self-adaptative technique that has been widely applied to numeric optimization problems successfully. Gaussian mutation generates a child by first mutating step size, $\vec{v}$, and then by mutating the object variables, $\vec{x}$, according to the normal probability density function. Self-adaptive Gaussian works as follows:

$$v_j^c = v_j^a * \exp(\tau' * N(0,1) + \tau * N_j(0,1)) \quad (5)$$

$$x_j^c = x_j^a + v_j^c * N(0,1), \ \forall j \in \{1,...,m\} \quad (6)$$

where *N(0,1)* shown in Fig. 3 is a normal distribution with mean 0 and standard derivation 1. $N_j(0.1)$ is a new value with distribution *N(0,1)* that must be generated for each connection weight. τ and τ' are respectively set to $(\sqrt{2m})^{-1}$ and $(\sqrt{2\sqrt{m}})^{-1}$ which are proposed by Schwefel [2].

Self-adaptive Cauchy Mutation: The behavior of self-adaptive Cauchy mutation is exactly the same as self-adaptive Gaussian mutation except the Equation (6) is replaced with Equation (8).

$$\psi_j^c = \psi_j^a * \exp(\tau' * N(0,1) + \tau * N_j(0,1)) \quad (7)$$

$$x_j^c = x_j^a + \psi_j^c * C(t), \ \forall j \in \{1,...,m\} \quad (8)$$

where C(t) is a Cauchy random number with parameter t = 1.0. The parameter values of Equation (8) are exactly the same as those in Equation (6). The Cauchy mutation has higher probability to escape from local optimum. This can be seen from Fig. 3 that the dot line extended to both ends infinitely.

Decreasing-based Gaussian Mutation: Decreasing-based Gaussian mutation uses an annealing-like concept to control the step size by using the same decreasing rate in each weight of n-dimension and works as follows:.

$$\sigma_j^c = \gamma * \sigma_j^a \quad (9)$$

$$x_j^c = x_j^a + \sigma_j^c * N(0,1), \forall j \in \{1,...,m\} \quad (10)$$

In our experiments, γ is set 0.95 and initial step-size is set to 4.0. From Equations (6), (8), and (10), we find Equation (10) can save computational time because it did not generate a random normal number and did not compute exponential function for each strategy variable. We will further describe the relationship between decreasing-based mutation and self-adaptive mutation later.

C. Selection

Recombination Selection: For each individual ($\vec{a}$) in the population, FCEA employed recombination selection to select two individuals, one is itself ($\vec{a}$) called family parent and another is randomly selected from the population. One important philosophy of FCEA is that each individual has equal probability to generate the same number of offspring. Recombination selection and modified recombination operator are designed in order to achieve this principle.

Family Competition: Each individual uses recombination and mutation to generate L offspring (L is the length of family competition) in order to explore fairly the search space. These L generated offspring from the same individual are called family. These similar offspring competes each other and only the best one survives in order to avoid the premature problem. Family competition can avoid the domination of early superstar because exactly one child in a family survives.

Population Selection: In decreasing-based Gaussian mutation stage, FCEA employs higher recombination rate and unbiased Gaussian mutation to make a large jump. The difference between the parent and its children may be larger than self-adaptive mutation. FCEA applied population selection, based on elitist and deterministic principle, to select the best N individuals from the union of parent set and offspring set. In this stage, population selection has two advantages. First, FCEA discards bad individuals in order to speed up convergence. Second, FCEA avoids premature problem in the early search time because of large diversity between parent and offspring.

IV The Tracker Task: The Ant Problem

A. The Environment and The Task

The first experiment was aimed at an artificial ant problem, "John Muir Trail" defined by Jefferson et al.[13], in order to compare the performance of our approach with other evolutionary algorithms, including genetic algorithm [13] and evolutionary programming [1]. Fig. 4 shows the trail studied by Jefferson et al. [13]. Each black box in the trail represented as food. The tracker task requires a simulated ant to follow a

crooked and broken trail of food on a two-dimensional toroidal grid, so that the cells on the left edge are considered to be adjacent to those on the right edge. The ant traverses the grid to collect any contacted food along the road. The ant will eat the food of a cell as soon as it stands on. The goal of the problem is to evolve a neural network, i.e., a simulated ant, that collects the maximum number of pieces of food in a given time steps. The maximum score is 89 because it is the maximum number of food. According to the environment of Jefferson's ant, the ant stands on one cell, facing one of the cardinal directions, and it can sense only the cell ahead of it. After sensing the cell ahead of it, the ant must take one of four actions: move forward one step, turn right 90° (without moving), turn left 90° (without moving), and no-op (do nothing). There are 89 food cells, 38 no food cells, and 20 turns in the optimizing shortest path of "John Muir Trail", so that the minimum steps of eating all food is 147 steps.

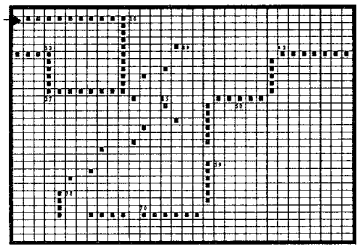


Fig. 4. The ant problem defined by Jefferson et al. [13]. The trail is 32 X 32 toroidal grid. The symbol '■' indicates a food on the trail. The symbol '→' denotes the start position and starting facing direction of an ant. The number in side of food cell gives the order of shortest path and is mere explanation.

B. The Control Network and Implementation Detail

In order to compare the results with GAs, we follow the work of Jefferson et al. [13]. They used finite state machines and recurrent neural networks to represent the problem and used the traditional bit-string genetic algorithm to train the structures. Each ant is controlled by a network with two input nodes, food and no-food, and four output nodes: move forward one step, turn right 90°, turn left 90°, and no-op. The food input is 1 when the presence of food in the cell ahead of the ant; the no-food input is 1 as the absence of food in the cell ahead of the ant. Each input node is connected to each of five hidden nodes and to each of four output nodes. There are 5 hidden nodes and these nodes are fully connected themselves in the hidden layer. This structure is a full connection with short cut recurrent neural network. So, each node of hidden node and output node has 7 links, and total number of links with bias input is 72. We implement the task on Intel Pentium Processor 200 MHz.

The strategy parameters, such as initial population, family competition length and recombination rate are shown in Table 1. FCEA evolved a population of 50 networks. Adaptation begins by initializing all the weights (x) of each network to random values between -0.1 and 0.1. The initial values of step size for self-adaptive Gaussian mutation (ν), Cauchy mutation (ψ) , and decreasing-based Gaussian mutation (σ) are set to 1.0, 1.0, and 4.0, respectively. The family competition length of self-adaptive and decreasing-based stage is set to 9 and 3, respectively. Thus, FCEA generates (3+9+9)*50=1050 networks in one generation. The recombination rate is set to 0.5 (p_d) and 0.2 (p_a) for decreasing-based mutation operator and self-adaptive mutation operator (see Fig. 2). The fitness is defined as the number of eaten food within 200 steps for "John Muir Trail".

Table 1. The Values for Different Parameters Used in FCEA

parameter name	the value of parameters
family competition length	L_d = 3 (decreasing-based mutation) and L_a = 9 (self-adaptive mutation)
recombination rate	p_{cD}=0.8 (crossover rate of decreasing mutation), p_{cA}=0.2 (crossover rate of adaptive mutation). p_{cd}=0.7 (discrete recombination), p_{cb}=0.1 (BLX-0.5), and p_{ci}=0.1 (intermediate recombination).
decreasing rate	γ=0.95

C. The Experimental Results and Comparison

Fig. 5 shows the typical convergent curve of the ant problem. Fig. 5 indicates that the best network in the first generation can find 24 food, 78 food at 10-th generation, and 81 food at 20-th generation. FCEA only requires about 20,000 function evaluations to a train neural controller for an ant to find 82 food within 200 time steps. FCEA requires 45000 networks and 65000 networks in order to find 86 and 88 food respectively. Each problem is tested over 20 runs and the successful rate to find 89 food is 65%. The rest of 35% tests can forage at least 86 food. The successful rate can be improved to 85% when the population is 100 and the maximum function evaluation is 500,000. The ant problem has many deep local optima and it prevents many algorithms from finding optimal solutions. Fig. 6 shows a typical behavior and its traveled path of a simulated ant that is controlled by a evolved neural network. The number in the food cell is the time step order of the ant ate the food. The symbol, '*', denotes a traveled cell by the ant and the cell is empty. Fig. 6 indicates an obtained solution that the ant requires 195 time steps to seek all 89 food. These results demonstrated that one of Jefferson's conclusions, i.e. ,the ant problem is difficult because it must be achieved in given time steps. They demonstrated by randomly generating about $1.3*10^9$ networks and the best solution is only 82.

Table 2 summarized the performance of various evolutionary algorithms including the genetic algorithm [13], evolutionary programming [1], and our method (FCEA). FCEA simulated each problem over 20 runs. To facilitate comparison with other algorithm more fairly, we list these results based on population size, function evaluations (evolved networks), best performance, average performance, and average time. Jefferson et al. [13] used traditional bit-string GA to evolve the same neural structure. They encoded the problem with 448 bits and used a population of 65,536 to solve the task in 100 generations on a 16K-processor Connection Machine (CM2). In their research, finding the solution required 6,553,600 networks and spent about one hour. The solution exactly needed 200 time steps to forage 89 food. In contrast to Jefferson's solution, FCEA uses small population size 50 and 100, and only needs about 86,000 and 344,00 function evaluations to solve the task based on the same structure. The evolved ant exactly needs 195 time steps to seek 89 food. GA needs about 70 times number of networks evaluated by FCEA. Angeline [1] proposed a system, called GNARL, that use evolutionary programming to train the weight and topology of recurrent network simultaneously. Evolutionary programming [9] only employs mutation operator and never uses recombination operator. The second row of Table 2 shows the results. Obviously, FCEA performs more stable and better than GNARL. FCEA will be more robust if we enlarge the population size. The third row in Table 2 shows the results.

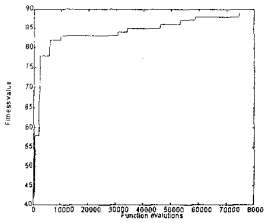


Fig. 5

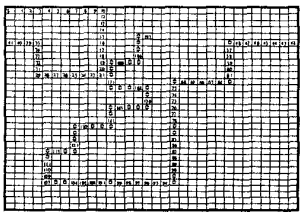


Fig. 6

Fig. 5. The typical convergent of best individual fitness value at each generation for ant problem. Fig. 6. The typical behavior of a simulated ant that iscontrol by evolved neural controller within 200 time steps. The number in the food cell is the order of the ant ate the food. '❀' represents a traveled cell by the ant.

Table 2. FCEA compares with genetic algorithm and evolutionary programming based on some criteria, including population size, function evaluations, performance, successful rate and time, in the ant problem. The performance of FCEA averaged over 20 runs. The first number in parentheses is the number of runs that FCEA finds all 89 food, and the second number denotes total runs.

applied method	populatio n size	function evaluation	best Performance	average Performance	average Time(s)
Genetic Algorithms [13]	65536	6,553,600	89	*	3600
Evolutionary programming [1]	100	184,250	83	82	*
FCEA with 5 hidden nodes	50	86,000	89(13/20)	88.54	430
	100	344,000	89(17/20)	88.85	1935

The symbol, '*', stands for the author did not report the results.

V. The Sequential Behavior Problem: The Simulation of Playing Football

In this subsection, FCEA aimed at developing a simulated robot capable of performing a sequence of behavior: an agent learns to play football [15]. The simulation of Maniezzo's football environment is that both robot and ball are randomly set to any positions on a field. To shoot ball into opponent's goal, an agent must learn four sequential tasks: reaching the ball, getting the ball, dragging it and reaching opponent's goal, and last kicking it into the goal. In sequential behavior environment, the controller will be triggered when environment is changed, and then the controller will refer previous actions to decide next action. The intelligent agent needed seven sensory inputs: two inputs for the distance of ball, two inputs for the distance of opponent's goal, two inputs for own goal (an agent must intercept ball in two-player environment when opponent got the ball earlier) , and one input for ball status. The ball status (free or controlled by a player) can be considered as a trigger input in this problem.

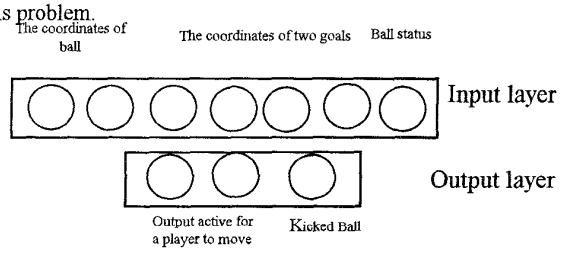


Fig. 7. The semantic of the input/output for playing football problems and the detailed structure is similar to Fig. 1.

A. The Control Network and Implementation Detail

Fig. 7 presents semantic of input/output of network for playing football problem. The five hidden nodes are fully connected to three motor output nodes that control the moving direction, speed and kicking ball, respectively. To facilitate the output action, the sigmoidal activation function defined in Equation (2) is modified as follows:

$$f(\eta) = \frac{(1-e^{-r\eta})^{-1}}{(1+e^{-r\eta})^{-1}}, \qquad (11)$$

where γ is set to 0.1 .

The fitness function in sequential behavior task is difficult to define, because the relationship between sequential steps is important. We divide the task into a sequence of subtasks, and then fitness function based on these subtasks. The overall fitness of a network was computed as follow:

$$fitness(x) = -R_b(x) + GotBall(x) - R_g(x) + \text{Kickedball(x)} \qquad (12)$$

where $R_b(.)$ and $R_g(.)$ are reward functions that an agent closes to the ball and closes to goal, respectively. The purpose of $R_b(.)$ is to prevent an agent from wandering aimlessly or standing still in the field when the robot did not get the ball. $R_g(.)$ is the reward to guide the robot to move to the goal while the agent has got the ball and was not close to the goal. These two functions are important because they will guide the robot to the goal. $R_b(.)$ and $R_g(.)$ are defined as follows:

$$R_b(x) = \begin{cases} 0 & \text{if robot controled the ball} \\ (p_s - P_b)^2 & \text{, others} \end{cases} \qquad (13)$$

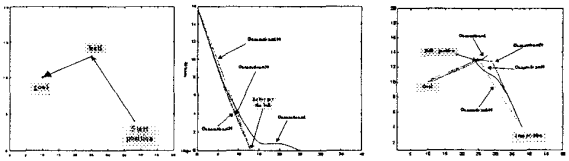
$$R_g(x) = \begin{cases} 0 & \text{if (robot kicked the ball)} \\ & \text{or (robot did not controled the ball)} \\ (p_s - P_g)^2 & \text{, others} \end{cases} \qquad (14)$$

where P_b and P_g, denote the positions of ball and goal respectively, and p_s is the final position of an agent in a trial. GotBall(.) defines the reward when an agent got the ball. Similarly, KickedBall(.) is defined the reward that an agent kicked a ball into the goal. GotBall(.) is set to 1000 and the value must be larger than $R_g(.)$. KickedBall(.) is set to 10000 and the value must be larger than the values of the four items in Equation (14). In this experiment, the value of parameters are the same as those used in the ant problem except the population size is changed to 30. So one generation will generate 450 (30*(3+6+6)) offspring.

C. The Results

Fig. 8 shows a typical evolutionary process and the results of one robot player environment. Fig. 8(a) shows an ideal path obtained from an experiment. Fig. 8 (c) shows the paths of the best simulated robot player at 1th, 20th, 50th, and 100th generations. At the 100th generation, the best robot has a traveled path that is almost close to the ideal path shown in Fig. 8(a). Each robot of the initial neural controllers can not find the ball and reach the goal within 80 time steps. In this stage, each robot is controlled by the first term ($R_b(x)$) of Equation (12) in order to close to the ball. At the 20th generation, the best robot has got the ball but it could not reach the goal to kick the ball. In this stage, the robot got a reward, GotBall(.). Then, $R_g(x)$ becomes the main factor to guide the robot to close in upon the goal. In the last stage, the robot closed enough to the goal and the robot shot the ball into the goal at the 50th generation. From the 50th generation to the 100th generation, the robot tries to find a shortest path. Fig. 8(b) shows the corresponding distance between the ball and the

player at 1^{th}, 20^{th}, 50^{th}, and 100^{th} generation in Fig. 8(c).



(a) (b) (c)

Fig. 8. A typical simulated results of one robot player environment. (a) indicates the an ideal action. (b) is the distance between the ball and player in (c) at 1^{th}, 20^{th}, 50^{th}, and 100^{th} generation. (c) shows the paths of the simulated robot player at 1^{th}, 20^{th}, 50^{th}, and 100^{th} generation.

VI Conclusions

The goal of this research is to determine whether neural networks could provide an efficient control mechanism to integrate sequential and learning behavior in autonomous agents. We use a simple neural network and propose a Family Competition Evolution Algorithm (FCEA) to evolve neural networks to integrate different types of behavior in a smooth and continuous manner. Our FCEA approach integrates the techniques of family competition, decreasing-based Gaussian mutation, self-adaptive Gaussian mutation, and Cauchy mutation. Each strategy in FCEA can compensate the shortcoming of each other. For example, self-adaptive strategy can compensate the weakness of decreasing-based strategy in the later search time.

In order to illustrate the power of our approach we apply our approach to two different task domains: the "artificial ant" problem and a sequential behavior problem-an agent learns to play football. In the artificial ant task, networks were evolved by our approach that could learn to generate different sequences of output based on sensory input on ant robot. From the experimental results, we find our approach can evolve the ant agent to travel the trails to eat all 89 food in a given time steps. Our approach outperforms other evolutionary approaches including GAs and EP. In the sequential behavior problem, an agent learns to play football. In the task, networks were trained by our approach that could learn the sequence tasks of scoring the goal. From the experimental results, we also find our approach perform well in the problem.

Our primary conclusion, based on the results form our experiments, is that our approach can evolve neural networks to provide a means of integrating sequencing and learning within a single control system. In summary, we have shown, by using the two complex problems, the power and flexibility of our algorithm. Future work of this research will focus on constructing a real robot to verify our methodology.

References

[1] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent Neural Networks," *IEEE Trans. on Neural Networks*, vol. 5, no. 1, 54-65, 1994.

[2] T. Bäck and H. P. Schwefel, "An overview of evolution algorithms for Parameter Optimization," *Evolutionary Computation*, vol. 1, no.1, 1-23, 1993.

[3] R. A. Brooks, "A robust layered control system for a mobile robot," *IEEE J. Robotics and Automation.* Vol. 2, no. 2, 14-23, 1986.

[4] R. A. Brooks, "Intelligent without representation. Artificial Intelligence," vol. 47, 139-159, 1991.

[5] M. Colombetti and M. Dorigo, "Train agents to perform sequential behavior. *Adaptive Behavior,*" 2 (3), 247-275, 1994.

[6] M. Dorigo and U. Schnepf, "Genetic-based machine learning and behavior based robotics: a new synthesis," *IEEE Trans. on System, Man, and Cybernetics,* vol. 23,141-154. 1993.

[7] L. J. Eshelman and J. D. Schaffer, "Real-coded genetic algorithms and interval-schemata," *In Foundations of Genetic Algorithms 2,* 187-202, 1993

[8] D. Floreano and F. Mondada, 1996. Evolution of homing navigation in a real Mobile robot," *IEEE Trans. on System, Man, and Cybernetic,* vol. 26, no. 6, 1996.

[9] D. B. Fogel, L. J. Fogel, and V. W. Porto, "Evolving neural networks," *Biological Cybernetics*, 487-493, 1990.

[10] Fogel D. B. and Atmar, J. W. 1993. Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetic*, vol, 63, 111-114.

[11] D. E. Goldberg, *Genetic Algorithms in search, Optimization & Machine Learning*, Reading. MA: Addison-Welsley, 1989.

[12] C. Z. Janikow and Z. Michalewicz, "An experimental comparison of binary and floating point representations in genetic algorithms," *In Proceeding of the Fourth Int. Conference. on Genetic Algorithms*, 31-36, 1991.

[13] D. Jefferson, et al., "Evolution as a theme in artificial life: The genesys/tracker system," In *Artificial Life II: Proceedings of the Workshop on Artificial Life*, 549-577, 1991.

[14] J. Koza, "Genetic evolution and co-evolution of computer program," In *Artificial Life II: Proceedings of the Workshop on Artificial Life*, 603-629, 1991.

[15] V. Maniezzo, "Genetic evolution of topology and weight distribution of neural networks," *IEEE trans. On Neural Networks,* vol. 5 no. 1, 39-53, 1994.

[16] L. A. Meeden, "An incremental approach to developing intelligent neural network controllers for robots," *IEEE Trans. on System, Man, and Cybernetics,* vol. 26, no. 6, 474-485, 1996.

[17] D. Parisi, F. Cecconi, and D. Nolfi, "Econets: Neural networks that learn in an environment," *Networks*, vol. 1,149-168, 1990.

[18] J. D. Schaffer, R. A. Caruana, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: A survey of the state of the art," In *Proceeding International Workshop on Combinations of Genetic Algorithms and Neural Networks,* 1-37, 1992.

[19] S. W. Wilson, "Classifier systems and the animat problem," *Machine Learning*, vol. 2, 199-228, 1987.

[20] B. Yamauchi and R. D. Beer, "Integrating reactive, sequential, and learning behavior using dynamically neural networks," *In From Animals to Animats III: Proceedings of the Third International Conference on Simulation of Adaptive Behavior*, 382-391, 1994.