Towards Behavior Control for Evolutionary Robot Based on RL with ENN

Regular Paper

Jingan Yang1,2*, Yanbin Zhuang1,3 and Chunguang Li3

1 Changzhou Key Laboratory of Software Technology & Applications, Jiangsu Prov., P. R. China
2 School of Computer & Information Sciences, Hefei University of Technology, Hefei, Anhui Province, China
3 School of Computer & Information Engineering, Changzhou Institute of Technology, Changzhou, Jiangsu Province, China

* Corresponding author E-mail: yja431008@gmail.com

Received 4 Jul 2012; Accepted 2 Oct 2012

© 2013 Yang et al.; licensee InTech. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract This paper proposes a behavior-switching control strategy of an evolutionary robot based on Artificial Neural Network (ANN) and genetic algorithm (GA). This method is able not only to construct the reinforcement learning models for autonomous robots and evolutionary robot modules that control behaviors and reinforcement learning environments, and but also to perform the behavior-switching control and obstacle avoidance of an evolutionary robot in the unpredictable environments with the static and moving obstacles by combining ANN and GA.

In comparison with the conventional behavior network and the adaptive behavior method, our algorithm simplified the genetic encoding complexity, improved the convergence rate $\rho$ and the network performance.

Keywords Evolutionary Neural Network, Reinforcement Learning, Simulated Binary Crossover, Behavior Switching, Robotic Adaptability, Simulated Robotic Agents

1. Introduction

1.1 Evolutionary Robotics

Evolutionary robotics has been an active research area that applies artificial evolution to construct control systems for autonomous robots. This is also a promising methodology for the autonomous development of the robots, in which their behaviors are obtained as a consequence of the structural coupling between the robot and the environments [39][27]. In evolutionary robotics, the evolved control mechanism determined a robot’s perception-action loop [33]. Many works have proposed evolutionary robotics control systems using evolutionary...
adaptations of artificial neural network [26][8][22][1], genetic programming [34][15], and a learning classifier [5]. We commonly use artificial neural networks to evolve a robot controller [11][31] and apply an evolutionary algorithm to design and/or train NN for solving a given task. Many attempts for robot control and an evolutionary algorithm have focused on developing autonomous robots inspired by animals and humans that have robust adaptation and stable behavior in unpredictable environments [17][35]. Recent researches emphasized on the adaptive fitness functions are used in evolutionary robotics [25][29][4]. In order to enable the robots to operate normally in unpredictable environments, the robots must have the ability to evolve their behaviors. In this paper, we establish the modules for obstacle avoidance behavior, behavior-switching control, and goal approach behavior based on combination of artificial neural network (ANN) and genetic algorithm (GA) which allow the robots itself to sense their environments and have self-adaptable function.

1.2 Evolutionary Reinforcement Learning and Adaptation of Behavior Control

In a natural system, animals do not only adapt to environmental changes, but they can also accumulate adaptations. They can store "knowledge" about a previously encountered environments and use it to alter their behaviors when faced with a specific environments again. This process is called learning when it occurs in a lifetime and evolution when it occurs in a lineage [7]. In other words, the agent adapts itself appropriate for different environments by evolving different behaviors.

Robot learning is a continuous dynamic mechanism. ER is one of a host of machines learning methods that rely on interaction with, and feedback from, a complex dynamic environment to drive synthesis of controllers for autonomous agents [25]. Only by constant study can the robot improve better its own adaptability, and acquire knowledge by relying on constant interaction with the dynamic environments. Finally, the robot can learn to move in the unknown environment by repeatedly adjusting the environmental module and its own module. This is what Brooks called behaviorism thought [38][36]. He thought that the effective way to design an intelligent robot should not be like traditional artificial intelligence, i.e., completely based on the symbol inference "top-down", but should be like the evolutionary mechanisms of the biological organisms who use "bottom-up" approach based on perception-action, and learning by interacting with the environments. In this work, we designed a simulation environment of robot navigation and an evolutionary robot module. Strategies for obstacle avoidance and the approach of the robot’s navigation [16] in the unknown environments are regarded as the robot’s basic behavior.

The ability of robotics to jointly apply many basic behaviors to its environments is called combined behavior. The process that the robot activates some basic behavior at some moment is called behavior switching. The robot’s behavior and its behavior switching are controlled by using FFNN (Feed Forward Neural Network) and PNN (Probabilistic Neural Network) [10][32][6][13][23]. ANN-based controllers consist of many basic network modules. Each module is a subnetwork with the standard unified structure. This method may effectively reduce the network computing complexity and the encoding complexity, and also speed up the evolutionary rate. We then directly carry on the encoding evolution to various modules based on genetic algorithm. This evolutionary process may be regarded as a reinforcement learning process of the robots. Finally, we demonstrate by the results of the simulated experiments that these novel modules do not require the complete environmental knowledge, and their structures are simple and can be easily extended to more complex structures, because the robots have a self-adaptive ability to learn from their experiences and environments.

1.3 Main challenges and contributions of this paper

One of the main challenges is to discover and model different adaptation mechanisms. Most of the works is to consider the artificial evolution of neuro-controllers as one of these adaptation mechanisms [27] and develop a feasible methodology of the autonomous agents that can reveal conscious abilities.

Major contributions of this paper are summarized as follows:

1. Proposed a behavior-switching control strategy of an evolutionary robotics based on ANN and GA for implementing robotics navigation in the unknown environments (Section 4).
2. Constructed behavior-switching learning modules of an evolutionary robot by reinforcement learning.
3. The algorithm proposed can successfully avoid local minima by crossover and mutation, perform behavior switching and obstacle avoidance effectively by evolutionary reinforcement learning.
4. Simplified the genetic encoding complexity using the neural network modules.
5. The algorithm proposed improved the convergence rate, solution variation, dynamic convergence behavior, and computational efficiency than the path planning method based on the real-coded genetic algorithm with elitist model.

This behavior-switching controller can easily be extended to other applications by adjusting control parameters of ANN, GA, and physical constraints.
2. RL-ENN and Evolutionary Robotics

2.1 Reinforcement Learning with ENN

The objective of reinforcement learning is to solve decision-making tasks through trial and error interactions with the environment. Actually, the reinforcement learning [21] is a learning method for agent to map the environment states to actions in order to maximize the total reward and obtain the maximum cumulative reward value of state-action pair. Formally, this is defined as a Markov decision process using the state space \( S \), action \( A \), and the rewards \( R \).

**Definition 2.1** A Markov decision process is denoted as \((S, A, \tau, r, p)\), where \( S = \{s_1, s_2, \ldots, s_n\} \) is the state space, \( A = \{a_1, a_2, \ldots, a_n\} \) is the action space, \( r: S \times A \rightarrow R \) is the reward function of the agent, \( \gamma \in (0, 1) \) is the discount factor which reflects the notion that rewards depreciate by factor \( \gamma < 1 \), and \( p: S \times A \rightarrow \Delta \) is the transition function, where \( \Delta \) is the set of probability distributions over state space \( S \).

The closer the discount factor \( \gamma \) is to 1 the greater the weight is given to future reinforcements.

**Definition 2.2** A fitness function is a particular type of objective function that prescribes the optimality of a solution (for example, a chromosome in a genetic algorithm) so that the particular chromosome may be ranked against all the other chromosomes. Optimal chromosomes, or at least chromosomes which are more optimal, are allowed to breed and mix their datasets for producing a new generation that will hopefully be even better.

**Definition 2.3** For any policy \( \pi \) and any state \( s \), the value of policy \( \pi \) in states is denoted as \( V^\pi(s) \) which is the expected discounted cumulative reward the agent gets and is formulated as follows:

\[
V^\pi(s) = \sum_{k=0}^{M} \gamma^k E[r_{t+k} | \tau, s_t = s]
\]

**Definition 2.4** For any policy \( \pi \) and any state \( s \), the expected return of taking action \( a \) in state \( s \) and following policy \( \pi \) afterward is denoted as \( Q^\pi(s,a) \) which is the expected discounted cumulative reward the agent gets:

**Definition 2.5** The immediate reward, for all states in the state space is defined as:

1. The reward for non-terminal states: \( r=0 \).
2. The reward for terminal states:
   a. The reward for the goal state: \( r=1 \).
   b. The reward for invalid states: \( r=0 \).
   c. The reward for obstacle states: \( r=-1 \).

In state \( s \in S \), an action \( a \in A \) is executed. The agent experiences a state transition \( s_t \rightarrow s_{t+1} \) and obtain a reward \( r_{t+1} \in R \). The objective of the agent is to maximize its reward \( R_t \) defined by [30]

\[
R_t = \sum_{i=t+1}^{T} r_i
\]

where \( T \) is the time reaching the final state. The states \( s_0, s_1, \ldots, s_T \) is called an episode.

The decision of which action to take in a certain state is determined by the policy

\[
\pi(s,a) = p(a|s) \quad \forall s \in S
\]

which denotes the probability of taking action \( a \) in state \( s \). Q-function for a policy \( \pi \) is defined as

\[
Q^\pi(s,a) = E_a[R_t | s_t = s, a_t = a]
\]

which denotes the expected reward of taking action \( a \) in state \( s \) and following policy \( \pi \) afterward.

The task of a reinforcement learning robot is to learn a policy \( \pi : S \rightarrow A \) for selecting actions based on current observed states \( \pi(s_j) = a_j \). Learning the optimal policy \( \pi^* \) for producing the greatest cumulative reward over time is denoted as follows:

\[
\pi^*(s) = \arg \max_{a \in A} \left[ r(s,a) + \gamma V^*(\delta(s_j, a_j)) \right]
\]

where the optimal action in state \( s \) is \( a \) that maximizes the sum of the immediate reward \( r(s,a) \) and the value (discounted by \( \gamma \) ) by following the optimal policy. The \( \delta(s_j, a_j) \) (next state given \( s_j \) and \( a_j \)) is defined as a state transition function. \( V^* \) is the maximum discounted cumulative reward that the agent can obtain starting from state \( \delta(s, a) \).

When an agent is at some environmental state, the reinforcement learning method does not inform the agent to adopt the corresponding correct actions, but the selected action’s quality is evaluated by using the reinforcement signal provided by the environment, and the optimal strategy is obtained through trial-and-error unceasingly to get the best mapping from state to action. The reinforcement learning structure and the state-action pair are shown in Figure 1.
The environment for an agent is described as the state set: \( S = \{ s_i | s_i \in S \} \), the action set that an agent performs is expressed as: \( A = \{ a_i | a_i \in A \} \). Under the current state \( s_i \), an agent selects actions \( a_i \) and performs it. As soon as the state \( s_i \) transfers to \( s_{i+1} \), a robot obtains the reinforcement signal \( r \) from the environment immediately. Task of agent’s reinforcement learning is to obtain the maximum cumulative reward using a control policy \( \pi \).

Assume that an agent is a point robot with simplified motor actions: forward, left, right, and backward. All actions can be tried in all states. The robot world and its state of transitions are regarded as a function of the present state and action taken. The task of the robots is to reach a given goal state via the shortest path. For reinforcement learning robots, a reward function given any current state, next state, and action, \( s_i, s_{i+1}, \) and \( a_i \) is given by equation (6).

\[
 r_{s_i,s_{i+1}} = \begin{cases} 
 0 & \text{if } s_{i+1} \neq s_i \\
 1 & \text{if } s_{i+1} = \text{given goal state} \\
 -1 & \text{if } s_{i+1} = s_i (\text{obstacle state})
\end{cases} \quad (6)
\]

The negative numerical reward in equation (6) discourages agents attempting an action against the world boundary. This action does not change the state of the environment.

The basic idea of reinforcement learning is on-line learning which combines together the control process with the learning process.

2.2 Evolutionary Neural Network

Here we designed the hierarchical network architecture of ENN based on the biological evolutionary genetic algorithm. The network connection weights, the network topology, the excitation function of each node in the network, and learning rules can be evolved for performing the global search widely in all solution space for network weight training until we found the optimal solution of the network architecture with least square error between the network output and goal output of the given training set for avoiding a local minimum caused by gradient descent learning. The evolution of learning rules can be regarded as a process of “learning how to learn” in ANN where the adaptation of learning rules is achieved through evolution [39].

2.3 Evolutionary Robotics Based on Reinforcement Learning with ENN

Evolutionary robotics (ER) is to create automatically autonomous robots and a sub-field of behavioral robotics. It is concerned with the application of evolutionary computation methods to the area of autonomous robotics control systems. One of the central goals of ER is to develop automated methods that can be used to evolve complex behavior-based control strategies.

In the design process for an intelligent robot, some core questions are: 1) how to realize the behaviorism idea. 2) how to learn the behavior and actions by interacting with the environments. 3) how to acquire knowledge from the environments and their experiences. 4) how to control the topological evolution trend of network modeling and complexity in network model effectively.

The reinforcement learning is to learn what to do, how to map situations to actions for maximizing a numerical reward signal. This is usually based on the idea that the robot receives rewards (feedback signals), which are positive or negative scalar values, based on its performance [3][2][28][40][20]. Depending on the reward, the agent reinforces (positive reward) or decreases (negative reward) its confidence on the correctness of its current behavior. The rewards are given by an external coach or by an evaluation function (reinforcement function) internal to the robot controller. When the robot chooses one act in the environment and after the environments accept this action, the condition and state are changed. One reinforced signal reward (or penalty) is created simultaneously and is fed back to the robot. The robot chooses next action again according to the reinforcement signals and the current environmental
state. The principle of the choice is to make the probability of receiving a positive reward increase. The action chosen not only affects the current reward value, but also affects the environmental conditions and states for next choice, as well as the final reward value [24][18]. A meaningful combination of the principles of neural networks, reinforcement learning, and evolutionary computation is useful for designing agents that learn how to solve a complex task and adapt to their environment through interaction with the environment. Here, we consider a mobile robot with four options of actions: forward, left, right, and backward. Their codes are: 0=forward, 1=left, 2=right, 3=backward. We perform the behaviors and actions of a robot by changing speeds of the left and right wheels based on reinforcement learning with neural network. Figure 2 shows neural network architecture including 8 infrared sensors data inputs, 5 hidden neurons with sigmoid activation function \( f_i \), and two output neurons representing wheels commands. Table 1 illustrated encoding of the states and the actions.

![Neural Network Architecture for Obstacle Avoidance](image)

**Figure 2.** Neural network architecture for the obstacle avoidance using neural modeling of the robot behaviors.

<table>
<thead>
<tr>
<th>The states</th>
<th>0</th>
<th>...</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding of the states</td>
<td>0000</td>
<td>...</td>
<td>1110</td>
<td>1111</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>The actions</th>
<th>forward</th>
<th>left</th>
<th>right</th>
<th>backward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encoding of the actions</td>
<td>00</td>
<td>01</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

**Table 1.** Encoding of the states and the actions

### 2.4 Relationship Between GA and Simulator

Figure 3 illustrates the relationship between the genetic algorithm (GA) and the simulator. Obviously, the simulator and the GA have the different functions. The GA is responsible for transferring genotype to the simulator, and decoding genotype as a neural network and/or input sensor information (phenotype). The phenotype mainly describes an organism’s traits and the organism’s genotype describes its genetic makeup. Two organisms may have the same phenotype, but have different genotype if one is homozygous dominant and the other is heterozygous.

![Genetic Algorithm Simulators Interaction](image)

**Figure 3.** Relationship between genetic algorithm and simulator

Suppose the fitness values of individuals \( 1, 2, \ldots, n \) be \( f_1, f_2, \ldots, f_n \) respectively, then the probability of an individual selected for reproduction based on roulette wheel selection (see Fig.3) is calculated by following equation [12]:

\[
p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}
\]  

(7)

The simulator uses this information to execute simulation process, and transfers the simulated results, namely, the robot’s fitness values, to the GA processing. The interactive process between the robot and the environment is detailed as follows.

1. Fitness=0
2. \( t=0 \)
3. while( \( t \leq N_u \))do
   1. reads the sensor inputs
   2. \( t=t+1 \)
   3. compute and return to Fitness

where \( t \) is the length of time step and \( N_u \) is the upper limit of the simulation running time steps. The fitness
function computation has many different options in the different experiments. For each generation of the GA, each individual in the population is required to perform the simulated process above. By interacting with the environments, the robots read the sensor input data for computing the output of the control network and changing their positions according to the output of the control network.

3. Strategies for Obstacle Avoidance

We have designed the behaviors for avoiding movable obstacles which get nearer from various positions and directions using the behavior network and the proposed method.

3.1 Fitness Function for Obstacle Avoidance

The fitness function is the heart of an evolutionary computing application [25]. The fitness function is responsible for determining which individual (controller in the case of ER) is selected for reproduction. If there is no fitness function, the individual is randomly selected. Successful evolution of the autonomous robot controllers is ultimately dependent on the formulation of suitable fitness functions that are capable of selecting successful behaviors. The performance for each evolved controller is evaluated by using the different fitness functions. Therefore, for the autonomous robots interacting with the environment, they are given an external reward or penalty \( r = r_i \) in some state \( s_i \). For obstacle avoidance behaviour \( r = -1 \) upon collision with an object and \( r = 0 \) otherwise. In order to get an optimal policy, the reward function should be fixed. For example, a robot that cannot avoid an object soon become immobilized when its path is blocked, the robot controller would obtain a negative fitness value.

The internal reinforcement signal \( r^* \) is derived that represents the immediate reward assigned to the robotics system in terms of correctness of the actions executed so far and also is the prediction error of the total reward sum between two successive steps:

\[
r^* (t) = r + \gamma V(x_{t+1}) - V(x_t) \tag{9}
\]

where \( r \) represents the direct effect of action on the transition, \( \gamma V(x_{t+1}) - V(x_t) \) is the estimation of the improvement of states. The \( r^* \) is used as the error to train the neural networks mentioned above.

If the desirability of a state is associated with a certain action \( Q(x_t, a) \) the equation (5) for the prediction error becomes:

\[
r^* (t) = r + \gamma Q(x_{t+1}, a_{t+1}) - Q(x_t, a_t) \tag{10}
\]

where the parameter \( \gamma (0 \leq \gamma \leq 1) \) determines the present value of the future rewards. Main differences between the two reinforcement learning approaches are to calculate 1) state evaluations \( V(x_t) \) and 2) state-action evaluations \( Q(x_t, a_t) \).

3.2 Network Architecture and Encoding

An autonomous robotics can acquire signals from the goal sources using approach sensor for locating the goal sources, and then should approach and reach the goal sources as soon as possible. This behavior is just opposed to obstacle avoidance behavior. The network architecture design of the approach behavior is also relative to the network architecture design of obstacle avoidance behavior. Behavior control of the robots is performed by using a neural module network. The whole network consists of multiple neural network modules (g0-g7). All modules have a similar topological architecture including two input nodes, one hidden layer with two nodes, and one output node. Reasonable selection of the node number in the hidden layer should synthetically take complex degree of the network architecture and permissible errors into account. If the number of nodes in the hidden layer is too small, the network may not be trained at all or the network performance gets very poor. If the number of the nodes in the hidden layer is too large, this can reduce the errors of the network system, but may lengthen training time of the network at the same time.

![Network Architecture and Encoding](Image)

**Figure 4.** The network architecture and encoding of approach behavior control of the robot.

On the other hand, the training of neural network is easy to fall into the local minimum not to be able to obtain the optimum solution which also is one of main reasons for over-fitting in neural network training and learning.

Our preliminary simulation experiments have also proved that the performance of the crossover mapping method was not satisfied. When a robot goes forward directly in free space, it will reduce the speed of the
motor on the side of the sensor approaching in order to turn to examine some goal source. Therefore, our algorithm directly maps the left sensor input into the left motor and the right sensor input into the right motor, namely, only connection weights \(w_{01}\) between \(g_0\) and \(g_1\), \(w_{03}\) between \(g_0\) and \(g_3\), \(w_{21}\) between \(g_2\) and \(g_1\), \(w_{23}\) between \(g_2\) and \(g_3\), \(w_{45}\) between \(g_4\) and \(g_5\), \(w_{47}\) between \(g_4\) and \(g_7\), \(w_{65}\) between \(g_6\) and \(g_5\), and \(w_{67}\) between \(g_6\) and \(g_7\) are remained. The whole network structure of the approach sensor is similar to that of the touch sensor. But the main difference is that the input of the approach sensor makes use of the difference between the left sensor signal and right sensor signal (as shown in Figure 5). In the experiments, we not only use chromosome with 56 bits (6×8+8×56 bits), but also add 6 bits for evolving the related sensor parameters. 3 bits of them are for encoding the exploratory ranges of the sensors, and other 3 bits for encoding the intervals between the sensor ends. A ci in Figure 4 represents the encoding of \(w_{ij}\) \((j=1,3,5,7)\).

3.3 Parameter Setup for Obstacle Avoidance

Here, the GA algorithms apply a single-point crossover to artificial chromosomes with 56 bits. The algorithm uses a special bit mutation pattern so that only one bit in the chromosome turns over. First we choose one chromosome to execute mutation based on the mutation rate, and then stochastically choose one bit to turn over in the selected chromosome. Therefore, mutation rate that must be divided by the chromosome bits can be equal to the mutation rate normally used in a GA algorithm. Elitism was implemented by finding the fittest individuals \(n\) in the current generation and copying them across into the new generation before the breeding process has begun. Elitist selection always preserves the fittest individuals from the population to the next generation. Therefore, elitist selection increases a convergence rate of GA, but elitist selection may lead to premature convergence to be trapped in local optima in the GA search process since the inherited best individuals might not be close to global optima. For example, if we preserve too many individuals from one generation to the next generation or if a ‘super’ individual (an individual with a far better fitness than the rest of the population) occurs, then elitism can greatly increase the rapidly replicated chances of the best individual within a few generations and fill the whole population with identical ‘siblings’. Once this happens, the population will tend to stagnate to local minima. Our studies have shown that one way to overcome this is to take a higher mutation rate. The effect of low mutation rate on the population reflects few variations to respond to sudden environmental changes. This indicates the population is slower to adapt to changing circumstances. A higher mutation rate may damage more individuals, but the individuals could increase the speed at which the population can adapt to changing circumstances by increasing variations in the population. Here we have embodied the Elitist selection in the following aspects.

1. If the robot does not do anything, accumulate 0 score.
2. If the robot goes ahead directly, but does not attempt to avoid any obstacle, and finally crosses the environment boundary, then the robot would accumulate one negative score which takes 0 score.
3. If the robot can correctly perform obstacle avoidance by learning, it can accumulate one positive score.
4. If the robot can not only complete obstacle avoidance, but also go forward at full speed in the free space, then it has a higher positive score.

The simulation steps are setup 1500 so that the robot can accumulate one high fitness value in its evolutionary process, and also learn obstacle avoidance behavior correctly. The weighted factor of collision punishment takes a value of 10 (determined after many experiments). If this factor value is too low and the walking distance is far greater than the collision punishment value, the individuals which have poor performances may be regarded as the best individuals which may be involved in the heredity operation in the next generation. When this factor value is too high, this rapidly increases the pressure on the robot to execute obstacle avoidance behavior. Therefore the useful heredity material in the evolutionary process may be lost.

In all experiments to improve the efficiency of the simulation, the size of the population was 40. These simulation experiments have demonstrated that the use of a large population does not enhance the performance of the system. The upper limit of the simulation evolutionary operation takes 40 generations. Another reason is to save time. We observed from our experiments that a community’s average performance enhancement is limited after that community has evolved about 30–40 generations.

3.4 Simulated Results for Obstacle Avoidance

Figure 5 illustrates the simulated experimental results for obstacle avoidance in the different environments and analysis of the results. As well as changing the crossover rate and the mutation rate, we realizes the simulated experiments based on two different environments including 4 obstacles and 10 obstacles. Figure 5(a) and Figure 5(b) show respectively the experimental results.

Evolutionary time for an environment containing many obstacles should be longer. The more obstacles, the less freedom movement opportunity of a robot, and the collision times will increase. This leads the robot to increase its accumulated negative score possibilities. In the environment containing 10 obstacles, the production of the best individual will take 10 generations. However, in the environment containing 4 obstacles, this
evolutionary process needs only 4 generations. The robot’s evolutionary performance become worse in the environments containing more obstacles, because the straight lines of highest score values that obtained by the robot have less opportunity to move forward. Slightly increasing the crossover rate and mutation rate may make the convergence rate slows down. But for 40 generations’ evolution later, the average experimental performance may be accepted. Too high a crossover rate can reduce the GA performance, since its main aim is to have a higher destruction rate to gain the highest fitness value pattern. Too high mutation rate has a remarkable effect of GA performance [40]. In some situations, the system continues to operate for only 10 generations more, and the system performance may get worse. However, the convergence rate and the quality of the solutions are greatly improved by combining parallel GA and crossover mapping.

4. Behavior Switch Control Strategies

4.1 Neural Network Encoding Based on GA

Neural networks are very suitable for training with evolutionary computation-based methods which can be represented by a concise set of tunable parameters [32]. Of varieties of neural network structures, the most commonly used structures are layered feedforward and/or recurrent network architectures. In this paper, we proposed the behavior switching controllers of ER based on the neural network modules realized by combining ANN and GA.

The basic idea of behavior switching control strategy is that behavior-switch controller should be activated immediately and switched to obstacle avoidance behavior and follow_wall behavior once the touch sensor finds the obstacles, and the ERL systems have to change robots’ motion direction rapidly based on the basic behaviors and jump over the region with the local minimum point, and escape from the trap of local minimum and plan their paths to goal again. Furthermore, this paper uses the genetic algorithms to evolve network connection weights for providing a global search method for network weight training and for avoiding local minima caused by gradient descent learning. This is very important to multi-robot system, the simulated robotics, and real robotics soccer games. The approach behavior should also be activated when the robot is walking in the free space. For the basic approach behavior and approach sensor parameters are effective, we can use the GA to evolve the switching network of the basic behaviors directly.

In this paper, we construct the experimental environments with only 10 obstacles and 10 goal sources. This is because the basic behavior and the sensing parameters embedded in the robot evolve on the basis of this environment. Our experiments fix the numbers of obstacles and the goal sources at 10. The positions, the sizes, and the shapes of the obstacles, and the positions of the goal sources are variable. The robots design is basically the same as the one described above, but the approach sensor parameters are fixed, and no longer evolve by the GA encoding. Figure 6 shows the new network structure. The chromosome includes 11 module networks (g0-g10) (each module network has 6-bit encoding; every weight has 1-bit encoding). Therefore, the module networks and their connection weights need 6×11×8×2×4×2×4=94-bit encoding in all. The first 8 module networks are almost the same as the module networks in the several sections above. The main difference is that the output is not directly connected to the motor outputs, but is the input of the second cell network composed by the other two module networks (as shown in Figure 6). The output of the module network 11(g10) determines the basic behavior that will be activated. The output of 0 represents obstacle avoidance, and the output of 1 means approach behavior.

4.2 Parameter Setup of GA Operator

The task of the robots we designed is to reach all goals (10) within 250 time steps. Therefore, the robot which reaches all goals and do not touch any obstacle will obtain the highest fitness value of 2000. However, if this robot touches an obstacle or stops at some time step in the evolution cycle (deadlock), its fitness value should be 0. Fitness values that other robots obtain will lie between two fitness values of 0 and 2000. Their chromosomes may be used as intermediate mediums of the heredity overlapping operation.

In the experiments, we reduced the simulated time steps and correspondingly increased the computation time for each generation. Here we increased the number of individuals up to 50. The number of the evolutionary generations is still 0. The experiments kept the convergence rate \( \rho \) approximately the same as the convergence rate \( \rho \) in our previous experiments. Moreover, we use the Elitism strategy to ensure that the current best individual survives to a next generation.

<table>
<thead>
<tr>
<th>Functions of RL and genetic algorithm</th>
<th>parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning times (s) of each generation</td>
<td>20-30</td>
</tr>
<tr>
<td>Number of individuals</td>
<td>50</td>
</tr>
<tr>
<td>Crossover probability ( p_C )</td>
<td>0.5-0.7</td>
</tr>
<tr>
<td>Mutation probability ( p_{m1} ) for increasing links</td>
<td>0.1</td>
</tr>
<tr>
<td>Mutation probability ( p_{m2} ) for increasing nodes</td>
<td>0.05</td>
</tr>
<tr>
<td>Weight mutation probability ( p_{m3} )</td>
<td>0.2</td>
</tr>
<tr>
<td>Mutation probability (per gene) ( p_{m4} )</td>
<td>0.1</td>
</tr>
<tr>
<td>Excitation response mutation probability ( p_{m5} )</td>
<td>0.1</td>
</tr>
<tr>
<td>Learning rate ( \eta ) of reinforcement learning</td>
<td>0.3</td>
</tr>
</tbody>
</table>

Table 2. Functions and parameters of RL and genetic algorithm
Figure 5. Comparison between the experimental results in the different obstacle environments: (a) Maximum fitness and average fitness of the individuals at each generation in 4-obstacle environment; (b) Maximum fitness and average fitness of the individuals at each generation in 10-obstacle environment.

Figure 6. The neural network topology of behavior switching control: the output of the module network (g10) has two states: 0 represents obstacle avoidance behavior, and 1 means approach behavior.

4.3 Experimental Results and Evaluation

Figure 7 illustrated comparison curves of average performance for behavior switching after the GA operated 10 times in two different simulated environments. We see from Figure 7 that the robots walk more easily in the environments in which their initial average fitness values are higher, and can maintain this superiority in the extra simulation process. After this robot had operated 10 times in the different experiments, we took the average fitness value, and the obtained result curve is smoother. This expresses a more real learning curve than that in the earlier experimental result from one operation only.

Please note that the robot’s learning curve in the initial phase in this experiment is steeper than the learning curve in the experiment in the previous section. The GA can improve the performance of the robot and establish the correct mapping relations from the sensor inputs to the behavior outputs (actions). One of reasons that this phenomenon occurs is to reduce the number of the system’s behavior outputs. There is only 1 bit which has two kinds of possible outputs in this experiment, but the previous experiments mapped directly 4 bits, namely, 16 kinds of possible outputs, to the motor controller (actions to be executed). The second reason is that the architecture and the connection weights of the neural module networks have been evolved based on crossover and mutation operations which can simplify the complexity of the network architecture with less nodes and the weights preferred.
Moreover, the experimental results have demonstrated that the input of the approach sensors may be reduced to 2 bits, one of them marking goal source on the lefthand side, and the other marking goal source on the right-hand side. If the goal sources exist on both sides of the robot, the system will process goal sources near the robot at first. According to our experiments, when a touch sensor, in particular a front sensor is activated, the robot’s behavior should switch immediately to the obstacle avoidance behavior.

4.4 Comparison with Adaptive Behavior Method

We have tested and evaluated the behavior-switching control strategy of the evolutionary robotics using the evolved ANN controller from the population and the hand-coded knowledge-based controllers. At the same time, we also compared the results with the adaptive behavior method proposed by Hyeun-Jeong Min in [23].

In our experiments, we use the 3D robot simulator and a real Khepera II mobile robot. The simulator environment includes three mobile robots (Robot1, Robot2, and Robot3) which generate behaviors using the proposed method, three goal objects (Goal1, Goal2 and Robot3), and two static circular obstacles. We regard two of the robots as the movable obstacles. These two robot-obstacles can only detect and avoid the walls of the fence, and cannot avoid colliding with other robots. This simulator may change the angles of the view. At the same time, we also use the evolutionary RL for parameter optimization and algorithm verification and for testing performance of the proposed method. At the local minimum points, the robot halted and switched to the collision-avoidance behavior immediately and performed collision avoidance in the different obstacle environments: Robot3 in Figure 8(a) halted and switched to the collision-avoidance behavior immediately, when it found two static obstacles and two movable obstacles in the same direction at its front and avoided two movable obstacles, fence, and two static obstacles respectively. Finally it reached the Goal3 with smaller oscillation of the planned trajectory. Figure 8(b) illustrated the path trajectory of the robot which generates behaviors and two moving robots (Robot1 and Robot2) which act as the movable obstacles in environments (from [23]).

As compared with the conventional behavior network and the adaptive behavior method, our algorithm simplified the genetic encoding complexity and improved the network performances and the convergence rate due to the application of the neural module network. Therefore, the robots could successfully perform obstacle avoidance, goal approach, and behavior-switching control in the dynamic environments with multiple obstacles. Figure 9 illustrated comparison of the collision-avoidance performances and the reward value changes of the robot in the dynamic environments with walls and obstacles in each learning cycle based on evolutionary learning with Neural Networks (ERLNN) and normal RL. It is obvious that the trajectory in the early stage is not smooth and gets smooth by evolutionary learning. At last, the robots could move freely without any collision in the environments. This work demonstrates once again the feasibility of application of the controllers based on ANN and GA to ERL and shows its potentials regarding adaptability and learning behaviors.

5 Applications of proposed algorithm

We have partly applied the results of the algorithm proposed in this paper to our simulated humanoid soccer robotics (as shown in Figure 11) and performed the experiments on both the Webots Pro 6.4.0 simulator and on a real NAO humanoid robot platform (Figure 10) using the parameters described in Table 2.

5.1 Fitness function definition

At first, we define some functions, for example, B is position of ball, O_i represents an opponent, i = 1, 2, ⋯, 5, Dist(X, Y) is referred to the distance between two objects, Dec is to change the binary code of the receiver into algorithm, $Avg(\cdot)$ is the average function. Therefore, the fitness function is defined as:

$$Fit = \frac{\sum_{i=0}^{5} Dist(B, O_i) + Dist(O_i, Dec(c_2c_1))}{Dist(B, Dec(c_2c_1))}$$  (11)

where $k = 1, 2, ⋯, 50$ which is the number of individuals. The $c_2c_1$ is the binary code for the receiver.
Because the operating environments of the soccer humanoid robot system with 11 against 11 agents are dynamic and very complex, the decision-making strategy of the system is extremely important. Therefore evolutionary reinforcement learning (ERL) for performing cooperation and coordination between the soccer multiple-robots is used for learning the decision-making strategy, evolving network architectures and connection weights (including biases) simultaneously, and adjusting parameters of FNN and GA. Furthermore, the residual algorithm is used to guarantee the convergence of the proposed algorithm to the optimal solution and can retain a high learning rate of the direct algorithms.

5.2 Learning decision-making strategies

In this paper, we divided the complex decision-making task into multiple learning subtasks that include dynamic role assignment, action selection including obstacle avoidance, behavior-switching control, goal approach, and action execution which constitute a hierarchical learning system to learn each subtask at the various layers. We have realized the multiple learning subtasks in CIT3D2012.

Definition 5.1 The behavior decision-making function of the multi-agent system are defined as [28]:

\[ f : (I_n) \rightarrow (O_m) \]

where \( I_n \) and \( O_m \) are \( n \)-dimensional input vector group and \( m \)-dimensional output vector group respectively.

Definition 5.2 Let the inputs include three subtask sets \( E, R, \) and \( S \). Of them, \( E \) represents the current environmental subtask set including the ball position, the speed and moving direction of the ball, the position and facing of the bilateral players, and the movement speeds and the directions of the bilateral players and so on. The decision-making system will divide all robots into \( n \) role groups \((n \geq 2)\). \( R \) is the agent current role subtask set, and each robot may have \( n \) role values \( R_1, R_2, \ldots, R_n \), but algebraic sum of their values should be equal to 1. \( S \) is the agent own current objective state parameter set.

Definition 5.3 We define the output as a robot agent behavior choice subtask \( A \) and the role value adjustment subtask set \( \Delta R \). Each robotic agent chooses the appropriate action from its action database according to its behavior choice subtask set \( A \) and executes this action. At the same time, this agent uses the subtask \( \Delta R \) for the role value adjustment to update the subtask set \( R \) of the role value in time. We may describe the entire decision function as:

\[ f (E, R, S) \rightarrow (A, \Delta R) \]

When a robotic agent interacts with the environments at time steps \( t = 0, 1, 2, \ldots, n \), it observes current state \( s_t \) at time step \( t \) and chooses action \( a_t \) according to the role values and executes this action, then the environment responds immediately to this robot and provides it with a reward \( r_t = Q(s_t, a_t) \) that reflects how well the robotic agent is performing this task in the environment and changes its current state to a successor state \( s_{t+1} = \delta(s_t, a_t) \) and corresponding \( \Delta R \). In system’s operation process, each robot agent learns unceasingly by updating its role value \( \Delta R \) and \( n_t \) by interacting with the environments.

Figure 8. A comparison of the collision-avoidance behaviors in the different obstacle environments: (a) An example of simulated movement of the robot with neuro-controller for behavior switching to collision-avoidance behavior in complex dynamic environments with two static obstacles and two moving robots which act as the movable obstacles; (b) the trajectory of the robot and two movable obstacles (moving object1 and moving object2) in the environments (from [23]).
5.3 Reinforcement learning for behavior switching control

The reinforcement learning tasks for robot navigation focus on learning a policy \( \pi : S \rightarrow A \) for selecting actions based on current observed states \( \pi(s_t) = a_t \). The system learns the optimal policy \( \pi^* \) for producing the greatest cumulative reward over time and for using \( \epsilon \)-greedy action selection. When converged to the true state-action values, then the greedy policy for selecting actions is optimal according to the following criterion:

\[
a'(x) = \arg \max_{a \in A} Q(x, a')
\]  

(12)

Algorithm 1 shows the pseudo code of the reinforcement learning for behavior switching control.

---

Initialize ANN and the humanoid robot system

\( qX, qY, qA \) (quantization of \( x, y, a \) configurations)

\( N \leftarrow 40 \) [number of episodes for Q-learning]

\( Q(s, a) \leftarrow 0 \) \( \forall s, a \)

episode \( \leftarrow 0 \) [actual episode]

\( r \leftarrow \{ \text{immediate reward values for all positions} \}

\( \eta \) and \( \gamma \) are set to 0.3 and 0.9 respectively and are encoded in the genome.

repeat

   
   episode \( \leftarrow \) episode+1; initialize \( s' \)

   Get current state

   Obtain \( Q(s, a) \) for each action by substituting current state and action into neural network

   Robot moves and gets current state

   Determine an action by "action=max\( Q(s, a) \)"

   Choose \( a \) from \( s \) using \( \epsilon \)-greedy exploration derived from \( Q(s, a) \).

   \( a'(s) = \arg \max_{a \in A} Q(s, a') \)

   if collision occurred then

      Reinforcement=−1 and back to the position before collision.

      \( Q_{\text{target}}(s, a) = g(s, a, s') + \gamma \max_{a' \in A} Q(s', a') \)

      use \( Q_{\text{target}} \) to train ANN in Fig.1
   end

repeat

   Take action \( a \), observe \( s', r \)

   Update the state-action function \( Q(s, a) \):

   \( Q(s, a) \leftarrow (1-\gamma) \times Q(s, a) + \gamma \max_{a' \in A} Q(s', a') + \text{cost} \)

   \( s \leftarrow s' \)

until \( s \) is terminal

until episode = \( N \)

---

Algorithm 1. RL-learning for BS Control

Here, the \( (s, a) \) is the state-action pair at time \( t \). The \( (s', a') \) is a successor state-action pair at time \( t+1 \). \( \gamma \) is the discount factor \( \gamma \in [0,1] \). \( \eta(s, a) \) is a learning rate parameter of the state-action pair \( (s, a) \) at time step \( t(0 < \eta < 1) \). The \( \Lambda \) is the set of possible actions and \( r \) is the reward the agent receives while taking action \( a \) in state \( s \).

The \( g(s, a, s') \) is the cost function. The learning rate \( \eta \) and the discount factor \( \gamma \) are set to 0.3 and 0.9 respectively which are encoded in the genome.

Because the movement error of the ball will be larger with increase of the ball speed and two type noise jamming occurs, we introduce cost function for representing these errors effect. By repeated experiments, we define this function as:

\[
\text{cost} = 0.1 + 0.01 \times V_{t+1} + 0.01 \times \text{Abs} \left( P_{t+1} - P_0 - 0.5 \times \text{margin} \right)
\]

(13)

where \( P_{t+1} \) represents position set of controlling ball region. \( |P_{t+1}| \) is distance from \( p_{t+1} \) to original point, \( V_{t+1} \) is the ball speed set, and \( D_{pt} \), means a distance between the player and ball.

5.4 The solved key problems

In the applications of proposed algorithm, we have focused on solving the following problems:

1. Consider parameter fine tuning, particularly, gait optimization for promoting the speed of the individual robots, the trajectory precision, and the gait stability of the humanoid soccer robots.

2. To find an optimal parameter setup for the various actions that humanoid robots executed based on the ERL techniques with ANN.

3. To promote quickly behavior-switching speed of the humanoid robots from one condition to another suitable for suddenly changing environments, for instance, from FORWARD switching to LEFT or RIGHT. Fast speed switching of robotics actions plays a critical role in the humanoid soccer robotics games.

4. The humanoid soccer robotic agents could avoid successfully local minima and motion deadlock status \( (s_{t+1} = s_t) \), faster converge to optimal solution with various learning rates, and reduce the oscillation of the planned trajectory between the multiple obstacles using various schemes including crossover and mutation, simultaneous learning, and novel modified error function that will be discussed in detail in another article.

Towards Behavior Control for Evolutionary Robot Based on RL with ENN

6. Conclusions and future work

This work has successfully constructed behavior-based control obstacle avoidance, goal approach, and behavior-switching learning modules on an autonomous robot. The novel modules do not need to have complete environmental knowledge and have more simple architectures extended easily to more complex structures. In the experiments, we have conceived the idea of modelling robot behaviors based on NN using evolutionary learning when encountering an obstacle.

In this paper, we have realized the obstacle avoidance behavior, goal-approach behavior, and behaviour switching modules using the evolved NN controller from the population and the knowledge-based controllers. They were not no longer dependent on rules assigned previously, but dependent on the knowledge acquired from the environments by autonomous evolutionary learning. The main contributions of the proposed algorithm include:

1. The RL-ENN algorithm evolved simultaneously network architectures and connection weights (including biases) and emphasized the behavioral links between parents and their offspring in evolution, such as weights training after each architectural mutation and node splitting.
2. To perform the decision-making strategy and parameters adjustment of FNN and GA by learning.
3. To avoid successfully local minima and motion deadlock status \( s_{i+1} = s_i \) of humanoid soccer robotics agents and reduce the oscillation of the planned trajectory among the multiple obstacles by crossover and mutation.
4. To perform effectively behavior switching and obstacle avoidance with behavior-based control using the evolved neuro-controllers.
5. To realize closer cooperation and coordination among the teammate agents by evolutionary learning.

Figure 9. A comparison of the collision-avoidance performances and the reward value changes of the robot in the environments with walls and obstacles in each learning epoch (one epoch = 100 time steps): (a) The collision times of the robot in the environment with walls and obstacles using ERL with NN decrease faster than using normal RL in each learning epoch. At last, the robots could move freely without collision in the environments; (b) Changes in the reward value of using normal RL and ERL with NN.

Figure 10. The experiments have been performed both on the Webots 6.4.0 simulator and on a real NAO humanoid robot with its 21 degrees of freedom using the parameters described in Table 2.

These results have been successfully applied to our simulated humanoid robot soccer team CIT3D which won the 1st prize of the official RoboCup Championship and ChinaOpen2010 (July 2010) and the 2nd place of the official RoboCup World Championship (2011) on 5-11 July, 2011 in Istanbul as shown in Figure 11 and Figure 12.
Figure 11. Some results of the proposed algorithm have been successfully applied to our simulated humanoid robot soccer team CIT3D. Some scene frames for the CIT3D (blue) to participate in official RoboCup ChinaOpen2010 Competition were shown: (a) The player of the simulated humanoid robotics soccer team CIT3D launched an attack on the opponents (red) goal and won by one point at 56.06s; (b) The player of the CIT3D with the ball was intercepted by an opponent player and tried to pass the ball to a teammate at 158.06s; (c) The player of the CIT3D launched an attack and shot at the opponent goal at 26.34s; (d) The player of our humanoid robotics soccer team CIT3D broke through and launched an attack on the opponent goal at 147.74 s.

Figure 12. A player (red) of the real CIT3D launched an attack and shot at the opponent goal in the real environments.
Our future works will focus on developing neurocontrollers with more complex architectures for real mobile robots and humanoid robotics in the real environments and on making further researches on the effect of crossover rate and mutation rate on the best performance and the average performance of the multirobot systems.

7. Acknowledgements

The authors would like to thank the anonymous reviewers and the editors for their helpful suggestions and insightful comments on earlier versions of the manuscript for improving quality of this paper.

8. References


