GA-based Action Learning

Satoshi Yonemoto
Graduate School of Information Science, Kyushu Sangyo University, Fukuoka, Japan

Keywords: Genetic Algorithm, Action Learning and Action Series Learning.

Abstract: This paper describes a GA-based action learning framework. First, we propose a GA-based method for action learning. In this work, GA is used to learn perception-action rules that cannot be represented as genes directly. The chromosome with the best fitness (elitist) acquires the perception-action rules through the learning process. And then, we extend the method to action series learning. In the extended method, action series can be treated as one of perception-action rules. We present the experimental results of three controllers (simple game AI testbed) using the GA-based action learning framework.

1 INTRODUCTION

This paper describes a GA-based action learning framework. First, we propose a GA-based method for action learning. And then, we extend the method to action series learning. This work focuses on the following problem: Action learning method is certainly suitable for estimating reactive actions. Obstacle avoidance is one of such reactive actions. In this case, the preceding action occurs just as the direct result of 'avoidance'. However, just a perception-action rule does not always solve more complicated tasks. In general, a task (for one agent) should be completed by performing the combination of action series. For solving this problem, we propose a new method that can include action series under the same GA-based learning framework.

Action learning framework is often used in artificial intelligence for games (called *Game AI*) (*Togelius, 2010*) (Shaker, 2011) (*Yannakakis, 2012*), and in autonomous robot simulations (*Gu*, 2003) (*Brooks, 2014*) (*Siegwart, 2011*). Recently, human-like character (avatar) in computer games can be controlled by various AI engines. Autonomoous car is also controlled by AI engine, using the same learning framework. The above mentioned, complex action is not performed by just one action associated with a perception data. For example, human-like character performs manipulation tasks in virtual environment, their action rules are often hand-coded (Kuffner, 1998), and it is difficult to learn such action series by perceiving real human activities. Therefore, we tackle to develop a new method to acquire action series from the percaptual data. GA-based learning is one way to acquire action rules in non-parametric manner. In the other learning approach, a convolutional neural network is often used (*Mnih, Volodymyr, et al., 2015*). In the GA based learning, the model is perception-action rules (i.e., table), learned with the evolution process. GA based learning has the following merits. Action rule for each perception is not formed as a block box. Therefore, large and sparse table can be slimmed after the learning process.

2 GA-BASED LEARNING

2.1 Genetic Algorithm

Genetic Algorithm (GA) is an optimization technique that globally estimates model parameters.
GA is one of stochastic search methods which have been applied with success to many types of complex problems such as image analysis and control problems. In general, genetic algorithm is performed the following steps:
1. Generate an initial population (N individuals) by randomly choosing.
2. Calculate the fitness of the population as an evaluation process.
3. Select a subset of population under pre-defined selection criteria.
4. Recombine the selected population as a new population (genetic operation such as mutation and crossover).

5. Repeat 2-4 until an elitist condition is satisfied.

In this work, GA is used to learn action rules that cannot be represented as genes directly. In this approach, the purpose of GA-based evolution is to construct a policy for perception-action pair.

Given a perception s, an action a, and a policy π, the action-value of the pair (s, a) under π is defined by $\pi(s, a)$. This corresponds to compute the function values $\pi(s, a)$ for all perception-action pairs (s, a), using evolutionary computation.

2.2 Genetic Encoding for Action Learning

Figure 1 illustrates our genetic encoding. Chromosome represents n perception parameters (*n bit* binary code). For example, such perception includes sensor grids around game characters and range sensors for autonomous robots. In this encoding, the binary code indicates a number (index) for the action-value table, π (see Figure 1).
The desirable action a for a perception s is given as the value of the table. The action is also defined by *k bit* binary code. The genetic algorithm is performed in three fundamental steps, that is, evaluation, selection and genetic operation. During the evaluation process, the fitness of the population is calculated. The chromosome with the best fitness (elitist) guides properly the best perception-action rules π.

3 EXTENSION TO ACTION SERIES LEARNING

3.1 Problem in Action Learning

Although GA-based learning is one of effective solvers for action learning, there is still another consideration. This approach is suitable for estimating reactive actions such as obstacle avoidance. However, it is just a one-step solver using perception-action rules $\pi(s, a)$. In general, a task should be completed by performing the combination of action series.
Action series learning is an extended version of GA-based action learning where the evaluation is performed by several action series. This means that the perception-action table includes action series $a_1, \cdots, a_m$ for a perception s. After performing several action series, the evaluation function is calculated.

3.2 Genetic Encoding for Action Series Learning

For chromosome represented by n perception parameters (n *bit* binary code), m action series are assigned. Each action a_i is defined by k *bit* binary code. The desirable action series $a_1, \cdots, a_m$ for a perception s are given as the value of the table. Figure 2 illustrates the genetic encoding for this approach.
After action series are performed, the evaluation function is calculated. Naturally, before performing all of m action series, the task can be completed. Therefore, maximum length of action series must be determined in advance. This approach provides a capable of effectively compressing for candidates of actions. That is, both action time series and several candidates of actions can be described in a unified manner.

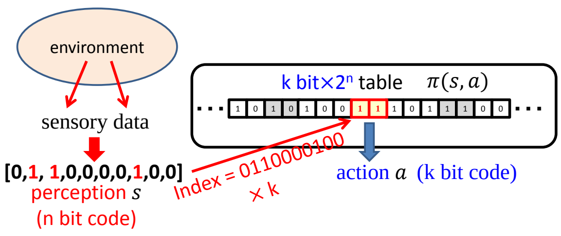


Figure 1: Genetic encoding for action learning.

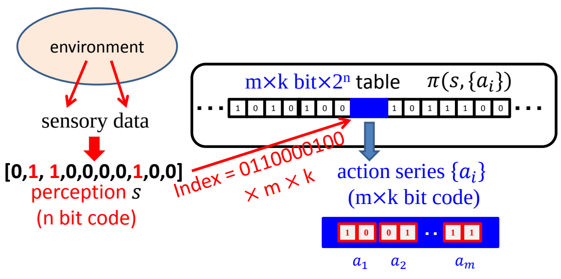


Figure 2: Genetic encoding for our extended approach.

4 EXPERIMENTS

To evaluate our GA-based action learning framework, three autonomous controllers (simple game AI testbed) are implemented. These programs are written in Java. To evaluate an essential GA evolution, game rules are simplified. Common genetic parameters are shown in Table 1.

4.1 Case 1: Vehicle Controller

First, our GA-based action learning is applied for a simplified vehicle control problem. Figure 3 illustrates the screenshots.

Table 1: Common genetic parameters.

Population size	100
Crossover rate	0.9
Mutation rate	0.05
Max generations	100

4.1.1 Game Rules

The following rules are applied:

- Vehicle: The vehicle can select three directions: go straight, turn left (-30[*deg*]) and turn right (+30[*deg*]). The vehicle has five range sensors to detect the neighbour obstacles (see Figure 4).
- Scene: Static obstacles (set of line segments) are arranged into the scene.
- Game over: The vehicle touches one of the obstacles.
- Task (goal): The vehicle keeps a drive during several laps or until over pre-defined distance.

4.1.2 GA Rules

For genetic encoding, the following rules are defined:

- Perception: Five range sensors are attached to detect obstacles. The sensor is defined by *5 bit* binary code. When the sensor i detects an obstacle (i.e., crossing a line segment), output is 1, (otherwise output is 0).
- Action: Three actions (turn left, turn right and go straight) are encoded by *2 bit* binary code.
- Fitness (f): If the vehicle touches an obstacle (a line segment), then $\mathrm{f} = \varepsilon$. If the vehicle keeps a drive during several laps (or pre-defined distance driving), then $f = 1.0$.

- Genetic encoding: Perception is defined by *5 bit* binary code (n=5). And action is defined by *2 bit* binary code (k=2). The code indicates a number (index) of $2^5 \times 2$ action-value table. The learning process is equal to fill the table, satisfying the fitness $\mathrm{f} = 1.0$.

4.1.3 Results

Figure 5 illustrates the results of trajectory of the elitist. In this case, the trajectory is represented as a loop line. We demonstrated for several courses, and we found obstacle avoidance with static scene is relatively easy. In practical use, solution of obstacle avoidance which exists dynamic objects is needed. This solution will show the next Case 2.

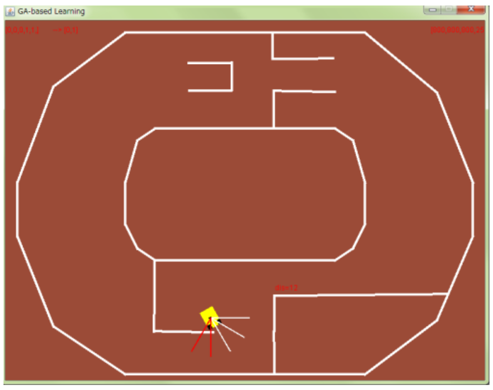


Figure 3: Screenshots of vehicle controller.

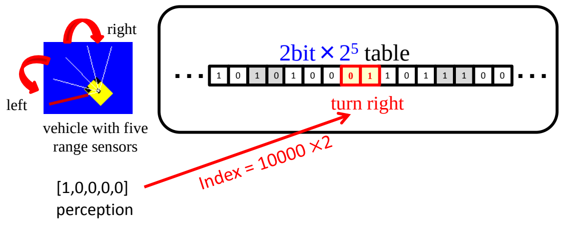


Figure 4: Genetic encoding for vehicle controller.

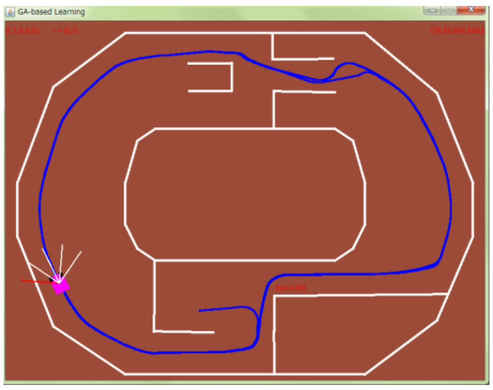


Figure 5: Trajectory of the elitist (vehicle controller).

4.2 Case 2: Action Game Controller

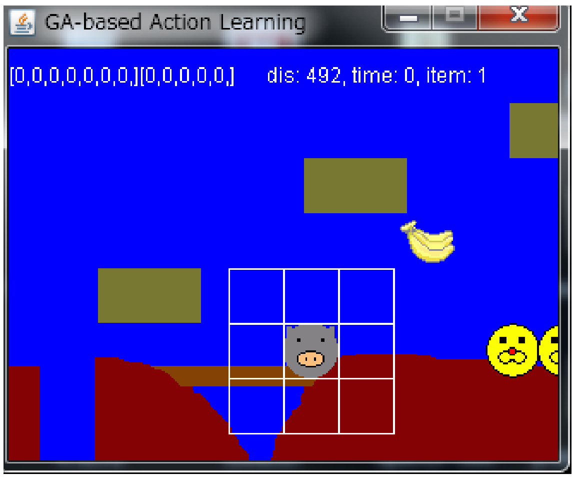
Next, our GA-based action learning is applied for simplified action game, like Nintendo's classic game *Super Mario Bros.* Figure 6 illustrates the screenshots (minimum map size).

4.2.1 Game Rules

The following rules are applied:

- Character: The character can typically move horizontally (left and right) and jump. The character can walk and run to the right and left direction. The character can jump onto obstacles or can jump over holes (gravity acts on the character).

- Scene: Scene length is 5 times screen width. Side scrolling is applied in accordance to the character position. Wall blocks, holes, enemies and flag pole (goal position) are arranged into the scene (according to the character position, they appear at their pre-defined position).
- Game over: The character touches enemies or downs holes.
- Task (goal): The character touches flag pole (reach the goal position).



```
[0,0,0,0,0,0,0,][0,0,0,0,0,]   dis: 492, time: 0, item: 1
```

Figure 6: Screenshots of action game controller.

4.2.2 GA Rules

For genetic encoding, both perception-action rules and the fitness function must be defined.

- Perception: Two sensor grids are used: enemy sensor and obstacle sensor (Figure 7). These sensors are defined as the 3×3 neighbour cells around the character (3×3 is minimum grid size). Enemy sensor is defined by *7 bit* binary code. When enemies or holes are appeared in the neighbour cells, output is 1 (otherwise output is 0). Obstacle sensor is defined by *5 bit* binary code. When wall blocks are appeared in the neighbour cells, output is 1 (otherwise output is 0).
- Action: Three actions (move left, move right and jump) are encoded by *2 bit* binary code.
- Fitness (f): The fitness function is basically defined as the distance to the flag pole (goal). Therefore, if the character reaches the flag pole, then $f = 1.0$. If the character touches one of enemies, then $f = \varepsilon$. If the character downs a hole, then $f = \varepsilon$.
- Genetic encoding: Perception is defined by *12 bit* binary code (n=12). And action is defined by *2 bit* binary code (k=2). The code indicates a number (index) of $2^{12} \times 2$ action-value table. The learning process is equal to fill the table, satisfying the fitness f = 1.0.

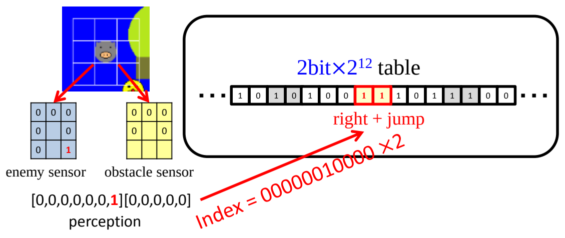


Figure 7: Genetic encoding for action game controller.

4.2.3 Results

Figure 8 illustrates the results of trajectory of the elitist. Although perception represented by 3×3 neighbour cells is a slight clue, avoidance actions with dynamic objects (enemies) are acquired.
Figure 9 illustrates an example of the acquired action. When an enemy closes with the character from right to left, the character can avoid it (i.e., jumping action after rearward moving).

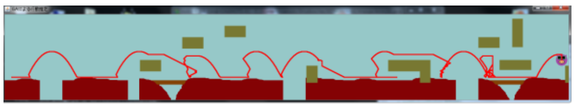
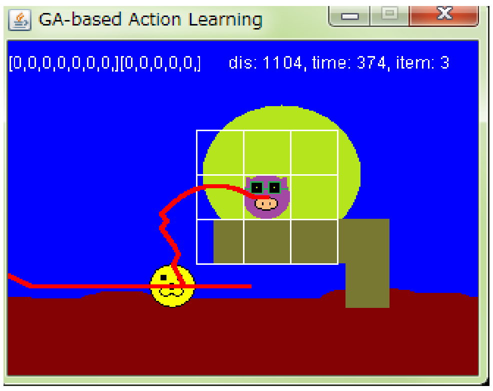


Figure 8: Trajectory of the elitist (action game controller).



```
[0,0,0,0,0,0,0,][0,0,0,0,0,]   dis: 1104, time: 374, item: 3
```

Figure 9: Acquired jumping action after rearward moving.

4.3 Case 3: Map Navigator

Our GA-based learning is extended to learn action series. In this experiment, our method is applied for simplified character map navigation (i.e., path finding problem). Path finding problem in maze is typically realized by another algorithms such as A* algorithm. In this case, it assumes that the character does not have any map information (i.e., a graph) to solve it. This restriction is often used in reinforcement learning. Figure 10 illustrates the screenshots.

4.3.1 Game Rules

The following rules are applied:

- Character: The character can typically move up, right, down and left (four direction) if he is on the road in a map. The character can perceive the direction of an item at a time. The character can collect the item.
- Map: The map is constructed by 20 x 10 wall blocks. Several items are arranged in the map.
- Task (goal): The task is to collect all items in the map (or to visit all check points). One episode corresponds to collect an item.

4.3.2 GA Rules

For genetic encoding, the following rules must be defined:

- Perception: (dx, dy) is defined by a signed vector of the current item direction. They are represented by *4 bit* binary code, (0, 0), (-1,-1), … , (+1,+1). (See Figure 11).
- Action: 4 action series are stored for a perception. Each action is encoded by *2 bit* binary code (move up, right, down and left).
- Fitness (f): If the character can collect an item, then f = 1.0 (then, a new item is set as the next target).
- Genetic encoding: Perception is defined by *4 bit* binary code (n=4). Each action is defined by *2 bit* binary code. In this case, 4 action series are defined (m=4), therefore, action series are defined by $2 \times 4 = 8$ bit binary code. The code indicates a number (index) of $2^4 \times 8$ action-value table. The learning process is equal to the acquisition of the action-value table, satisfying the fitness f = 1.0.

4.3.3 Results

Figure 12 illustrates the results of trajectory of the elitist. Four items were successfully acquired in order (a)-(d) in Figure 12. Moving operations were properly applied according to just item direction (dx, dy).

Figure 13 (a) illustrates a failure in learning process (m=4). However, applying iterated learning process, the character could come near the target item (Figure 13 (b)). We found that augmenting action series eliminates this failure (m is more than 5). Figure 13 (c) shows the successful case (m=5). According to the task level, the length of action series m should be defined.

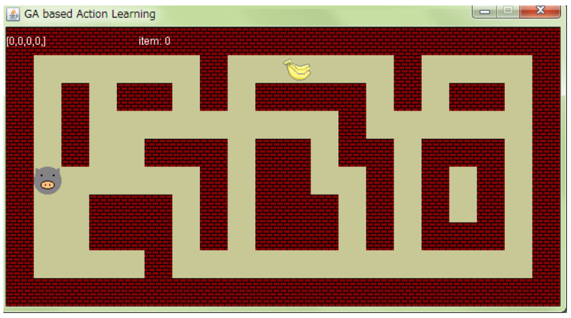


Figure 10: Screenshots of map navigator.

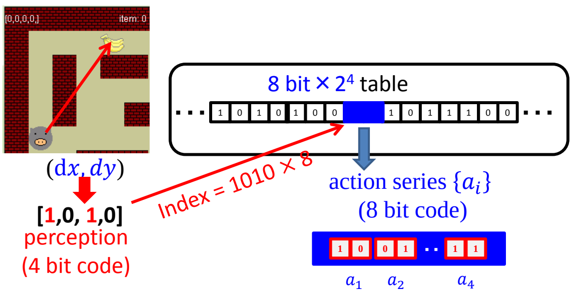


Figure 11: Genetic encoding for map navigator.

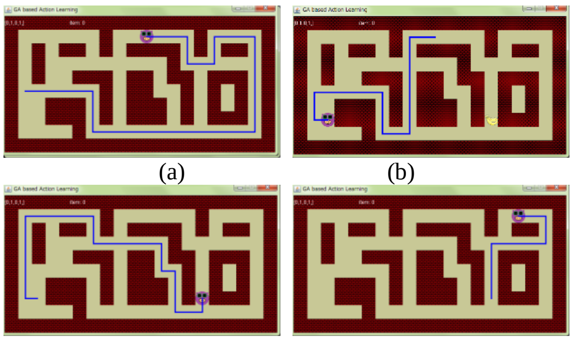


Figure 12: Trajectory of the elitist (map navigator).

5 CONCLUSIONS

We have proposed a GA-based action learning framework. In our framework, GA is used for learning perception-action rules as a table. In the experiments, first, we have demonstrated vehicle controller to solve static object avoidance problem. Next, we demonstrated action game controller for

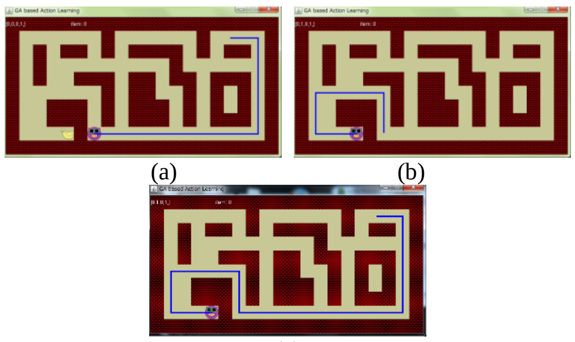


Figure 13: Influence on the length of action series.

dynamic object avoidance. The genetic encoding was performed by constructing a $2^{12} \times 2$ action-value table. Larger table is necessary to extend sensor grid size and to add various actions. However, we think that it can be solved as one of implementation issues. This approach is effective, but is just applicable for such object avoidance problem. Therefore, we proposed a new method that considers action series under the same GA-based learning framework. In the experiments, we showed that map navigator is successfully performed using action series learning. Especially, our extended approach is able to acquire action series (4 set of moving operations) from the slight perceptual data. This approach provides a capable of effectively compressing for candidates of actions.

Future work includes development of practical application and more complex action series acquisition (for action game controller).

REFERENCES

Togelius, Julian, Sergey Karakovskiy, and Robin Baumgarten, 2010. The 2009 mario ai competition, *Evolutionary Computation (CEC),* IEEE Congress on. IEEE.

Shaker, Noor, et al., 2011. The 2010 Mario AI championship: Level generation track, *Computational Intelligence and AI in Games, IEEE Transactions on* 3.4 : 332-347.

Yannakakis, Geogios N, 2012. Game AI revisited. *Proceedings of the 9th conference on Computing Frontiers. ACM.*

Gu, Dongbing, et al., 2003. GA-based learning in behaviour based robotics, Computational Intelligence in Robotics and Automation. *IEEE International Symposium on* 3.

Brooks, Rodney A., 2014. The role of learning in autonomous robots, *Proceedings of the fourth annual workshop on Computational learning theory*.

Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza, 2011. *Introduction to autonomous mobile robots. MIT press*.

Kuffner Jr, James J, 1998. Goal-directed navigation for animated characters using real-time path planning and control, *Modelling and Motion Capture Techniques for Virtual Environments. Springer Berlin Heidelberg,* 171-186.

Mnih, Volodymyr, et al., 2015. Human-level control through deep reinforcement learning, *Nature* 518.7540: 529-533.