Improving Learning and Adaptation Capability of Agents

Yohannes Kassahun and Gerald Sommer
Christian Albrechts University of Kiel, Institute of Computer Science, Olshausenstr. 40, 24098 Kiel, Germany

Abstract. Neural networks, reinforcement learning systems and evolutionary algorithms are widely used to solve real-world problems. We investigate learning and adaptation capabilities of agents and show that the learning time required in continual learning is shorter than that of learning from scratch under various learning conditions. We argue that agents using appropriate hybridization of learning and evolutionary algorithms show better learning and adaptation capability as compared to agents using learning algorithms only. We support our argument with experiments, where agents learn optimal policies in an artificial robot world.

1 Introduction

When an infant learns how to go and how to stand, it has no explicit teacher, but it does have a direct sensorimotor connection to its environment. From this connection, the infant receives a wealth of information about cause and effect, about consequences of actions, and about what to do in order to achieve goals. This interaction is a major source of knowledge about our environment and ourselves. Learning from interaction is a fundamental idea underlying nearly all theories of learning and intelligence [6]. It is used by agents at the individual level. In this work, we shall investigate agents using learning from interaction. This type of learning is different from supervised learning, which is learning from examples provided by a knowledgeable external supervisor. Supervised learning is an important type of learning but alone it is not adequate for learning from interaction. Moreover, it is usually impractical to obtain examples of desired behavior that are both correct and representative of all the situations in which the agent has to act and learn [5].

At the population level, it is clear that parents have inherited the infants the ability to learn and survive. This inherited ability is developed through evolution. A generation of an organism can only survive or continue to live if the population adapts itself to various situations in the environment. This shows that the learning and adaptation capabilities of agents are also affected by evolution.

Reinforcement learning [3] is one form of learning from interaction. It is learning what to do, how to map situations to actions so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover by itself which actions yield the most reward by trying them. Like the infant, an agent using reinforcement learning learns and adapts itself through interaction with the environment. In this paper, we use Q-learning [6], which is one form of reinforcement learning, to investigate the learning and adaptation of agents at individual level.

Evolutionary algorithms are, on the other hand, flavors of the well known machine learning method called beam search where the machine learning evaluation metric for the beam is called the ”fitness function” and the beam of the machine learning is referred to as the ”population” [1]. Like other machine learning systems, evolutionary algorithms have operators that regulate the size, contents and ordering of the beam (population). We use genetic algorithms (GA), which are one form of evolutionary algorithms, to investigate the learning and adaptation of agents at population level.

In this work, we try to answer the following important questions:

1. Is the learning time required by agents shorter in continual learning at both individual and population levels, and under various learning conditions?

2. Is it possible to improve the learning and adaptation capability of agents by hybridizing learning and evolutionary algorithms?

We will use agents using Q-learning, hybrid of multi-layer perceptron (MLP) and genetic algorithm, and hybrid of Q-learning and genetic algorithm respectively in answering the above questions. The agents live and operate in an artificial robot world.

2 The Robot World (Test Scenario)

A deterministic world of denumerable states is used as a test scenario to investigate the learning and adaption capability of an agent. The agent is assumed to be a point robot with simplified motor actions: `left, forward` and `right` [3]. All actions can be tried in all states. The robot world and its state of transitions as a function of the present state and action taken, are shown in figure 1. The arrows in the cells show the orientation of the point robot when the robot finds itself in these states.

The task of the agent is to reach a given goal state through the shortest path. For reinforcement learning agents, a reward function given any current state, next state and action, s_t, s_{t+1} and a, is given by equation (1).

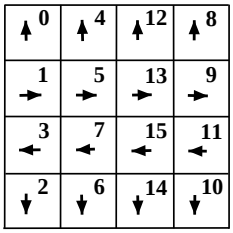
$$R^a_{s_t,s_{t+1}} = \begin{cases} 0 & \text{if} \quad s_{t+1} \neq s_t \\ 1 & \text{if} \quad s_{t+1} = \text{goal state} \\ -1 & \text{if} \quad s_{t+1} = s_t \end{cases} . \tag{1}$$

The negative numerical reward in equation (1) discourages agents attempting an action against the world boundary. This action does not change the state of the environment. For genetic algorithm, a fitness function given by equation (2) is used.

$$f\,(n) = \gamma^n. \tag{2}$$

The quantity f represents the fitness value of an individual, γ is a constant laying in the interval $[0, 1)$, and n is the number of steps taken by the point robot from a given start state to a given goal state. Equation (2) encourages those individuals that go from the start state to the goal state through a shortest path. The dynamics of the robot world, which is described by the state transitions table and the reward function, is not known to the agents a priori.

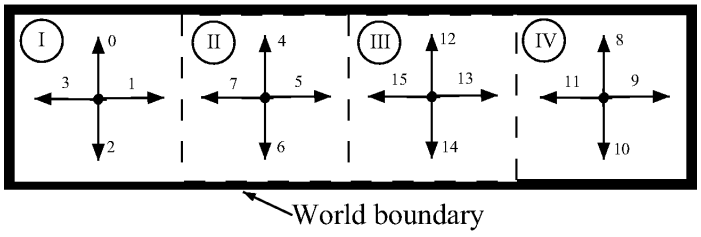
The robot world is a very highly simplified scenario of a real robot world. First, it is impossible to think a dimensionless robot or completely distinguishable states. Second, it is



(a)

Start State	Resulting State		
	Left	Forward	Right
0 4 12 8	3 7 15 11	0 4 12 8	1 5 13 9
1 5 13 9	0 4 12 8	5 13 9 9	2 6 14 10
3 7 15 11	2 6 14 10	3 3 7 15	0 4 12 8
2 6 14 10	1 5 13 9	2 6 14 10	3 7 15 11

(b)



(c)

Figure 1: A two-dimensional robot world (a) The point robot must find the shortest path from any start state to the goal state. (b) The state transitions table that governs the motion of the point robot. (c) Interpretation of the robot world. It consists of four positions. In each of these positions, the robot can take one of the four orientations. A robot in state 0 or a robot in position I with orientation north will bump against the world boundary if it executes a `forward` action. In this case, the state of the robot world will not change. If it executes a `right` action, then it changes its orientation to east or goes to state 1.

not possible to throw the details of low level control and deal with only simplified motor actions. Even though these assumptions are unrealistic, we base our experiments on artificial robot world due to the following justifiable reasons:

1. The experiments have to be carried out for a large number of times for different conditions of learning and adaptation experiments. This requires a lot of time and energy to execute all the experiments on real robot until one gets agents with satisfactory behaviors.

2. There is a danger of coming up with wrong conclusions with experiments on real robots. This is because of the fact that noise and error makes certain parts of the agent's policy to fluctuate.

A more efficient and inexpensive method is, therefore, to run the experiments on an artificial robot world that needs much less experimental effort and yet to come up with domain free results.

3 What to Learn?

In this work, the agent learns on-line through interaction with the environment either the optimal policy for perceived states or the action values of the states of the environment. A policy defines the learning agent's way of behaving at a given time. It is a mapping from perceived states of the environment to actions to be taken when in those states. An action value of a state shows "how good" it is for an agent to perform a given action in a given state.

By optimal policy, we mean a policy that enables the agent to go from a given start state to a given goal state with minimum number of actions or steps. With genetic algorithm, the agent learns directly the optimal policy without having to learn the model of the environment. In Q-learning, the agent learns the action values and saves them in a Q-table [3]. It can generate the optimal policy for perceived states from the Q-table.

4 Experimental Setup

The following test cases are selected for the experiments. Each of the cases shows the level of the knowledge of the agent about what is going to be learned.

Test Case A

In this test case, we assume that the states of the policy that is going to be learned are completely contained in the previously learned optimal policy. For example, one of the optimal policies from start state *7* to goal state *15* contains the states $\{7, 4, 5, 13, 12, 15\}$. The sequences of actions that are contained in the policy are $\{$*right, right, forward, left, left*$\}$.

Assuming that the previously learned optimal policy is this policy, any policy with start state $s_{start} \in \{7, 4, 5, 13, 12, 15\}$ and goal state $s_{goal} = 15$ can be considered as a test policy, since it is known from Bellman optimality equation [3, 6] that an optimal policy with $s_{start} \in \{7, 4, 5, 13, 12, 15\}$ and goal state 15 has its states completely contained in one of the optimal policies with start state 7 and goal state 15.

Test Case B

Here it is assumed that the previously learned optimal policy and the policy which is going to be learned have common states. A policy with states $\{3, 0, 1, 5, 4, 7\}$ generated by sequence of actions $\{$*right, right, forward, left, left*$\}$ and a policy with states $\{2, 1, 5, 13, 12, 15\}$ generated by actions $\{$*left, forward, forward, left, left*$\}$ are good examples of policies having common states $\{1, 5\}$.

Test Case C

The previously learned optimal policy and the policy which is going to be learned have no common states. Examples of optimal policies which have no common states are $\{1, 5, 13, 9, 8, 11\}$ generated by actions $\{$*forward, forward, forward, left, left*$\}$ and $\{15, 7, 3, 0\}$ generated by actions $\{$*forward, forward, right*$\}$.

5 Experiments and Results

For all the experiments, the start and goal states $\{7, 15\}$, $\{3, 11\}$ and $\{15, 0\}$ are selected for the previously learned optimal policy for the test case A, B and C respectively, and the start and goal states $\{5, 15\}$, $\{7, 15\}$ and $\{1, 11\}$ are selected for the optimal policy which is going to be learned for the test case A, B and C respectively. Each of the plots of the results of the experiments in this section shows the average taken over 10 experiments by using different random seed values for the Q-learning and genetic algorithm.

5.1 Learning and Adaptation at Individual Level

In this section, we use Q-learning in investigating learning and adaptation of agents at individual level. Q-learning [6] is a reinforcement learning algorithm which is used whenever there is no explicit model of the environment. In Q-learning, the agent learns the action values of the states of the environment and it can generate the optimal policy for the perceived states from the learned action values. The algorithm works by maintaining an estimate of the expected return for each state-action pair (Q-values) and adjusting these values based on actions taken and rewards received. The current Q-value is updated by a change,

$$\Delta Q\left(s,a\right)=\alpha\left[r+\gamma\max_{a}Q\left(w,a\right)-Q\left(s,a\right)\right] \tag{3}$$

where w is the next state. The parameter α controls the learning rate and the parameter γ , $0\leq\gamma\leq1$, determines the present value of the future rewards. A reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received after executing an action. In order to balance the exploration and exploitation in the Q-learning, we have used the simple $\epsilon-$ greedy action selection method [3].

In the experiment the parameters α and γ are set to 0.3. For learning from scratch, we have initialized the Q-table randomly and saved the learned Q-values for later use in the continual learning.

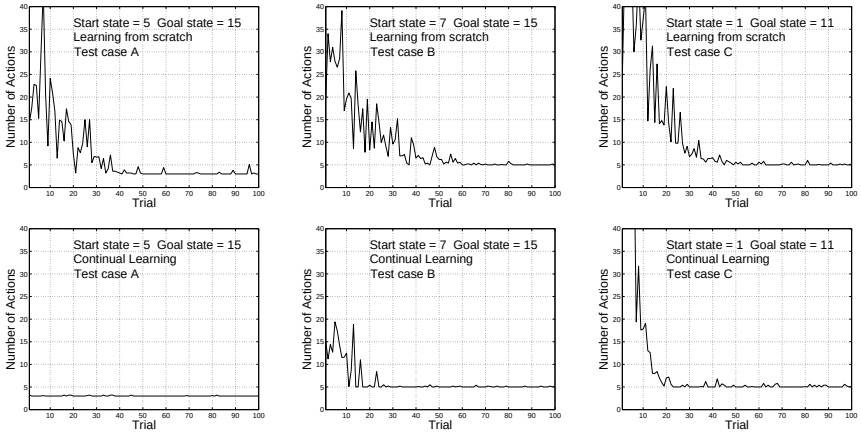


Figure 2: Result obtained for Q-learning. The top row shows from left to right results of learning from scratch for test case A, B and C respectively. The bottom row shows the corresponding result in continual learning.

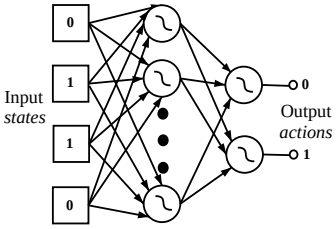
From the result shown in figure 2, one can conclude that the learning time is shorter in continual learning for all test cases. In the figure, a trial is made up of a sequence of actions performed by an agent in an attempt to reach a goal state from a given start state. In a trial, an agent may or may not reach the goal state. For test case A, one can see that the agent does not need to learn the optimal policy in continual learning at all. Test case B shows that the learning time of the agent is decreased in continual learning as compared to the learning time in learning from scratch. In test case C, one can see that even though the previously learned policy and the policy which is going to be learned have no common states, the agent needs shorter learning time in continual learning. This is possible due to the fact that the agent has collected information about other states which are not contained in the previously learned policy while learning it.

5.2 Learning and Adaptation at Population Level

Genetic algorithms (GA) are used in investigating the learning and adaptation of agents at population level. Genetic algorithms [7] are computational models inspired by natural evolution. They encode a potential solution of a given problem on a simple chromosome-like data structure and apply genetic operators to these structures so as to preserve critical information. A genetic algorithm starts with a population of chromosomes which are randomly generated. Chromosomes are then evaluated and given reproductive opportunities according to the result of their evaluations. Those chromosomes which represent a better solution to the target problem are given more chances to reproduce than those chromosomes which are poorer solutions.

In this section, we have used multi-layer perceptrons (MLPs) to represent the optimal policy. A population of MLPs with two layers forms a population of controllers. The structure of the networks and the number of hidden units is fixed but the weights are determined directly by the genetic algorithm. The MLP controls the point robot in the robot world. One should note that the MLPs have no any capability of learning through interaction. That means, learning and adaptation occurs only at population level and not at individual level. The genetic algorithm lets each individual to control the point robot and evaluates and selects an individual (controller) that moves the point robot from a given start state to a given goal state with minimal number of steps. It then applies genetic operators to generate the next population of MLPs for predefined number of trials.

Figure 3 shows an example of an MLP used in this experiment and the encoding of the states and actions. A binary code is used for both input (states) and output (actions) of the neural network.



(a)

State	Code
0	000
⋮	⋮
14	1110
15	1111

(b)

Action	Code
left	00
right	01
forward	10
“Don’t care”	11

(c)

Figure 3: The MLP used (a) and the encoding of the states (b) and actions (c).

A fitness function given by equation (2) is used to evaluate the individuals. An example of a chromosome representing an MLP (an individual) is shown in figure 4. The parameters of the neural network and genetic algorithms are given in table 1.

$Wo_{1,1}$	$\cdots$	$Wo_{1,N}$	$Wo_{2,1}$	$\cdots$	$Wo_{2,N}$	$Wh_{1,1}$	$\cdots$	$Wh_{1,4}$	$Wh_{N,1}$	$\cdots$	$Wh_{N,4}$

Figure 4: A chromosome encoding an MLP. Wo’s show the synapses going to the output units and Wh’s show synapses going from input to hidden units. N is the number of hidden units. In this experiment, we used two output and four input units, and six hidden nodes.

We have run the experiment for all test cases and obtained the result shown in figure 5. As can be seen in the figure, the population attains a certain average fitness value. The average

Table 1: Parameters of the MLPs and genetic algorithm.

Number of individuals	50
Crossover probability	0.2
Mutation probability per bit	0.05
Selection method	Truncation selection
Number of hidden nodes	6
Number of bits per gene coding a synapse	8
Number of generations	100

fitness value, which is controlled by the genetic operators, shows an equilibrium point of two ”forces”. One of the forces, which is controlled by selection operator, tries to pull the population towards the global maximum fitness value (fitness value of the best individual) and the other force, which is controlled by the crossover and mutation operators, tries to maintain the variation between individuals. The learning time per individual, which is measured in number of generations, required in attaining a certain average fitness value is shorter in continual learning than the learning time in learning from scratch for all test cases. Note that an individual has only one trial per generation in controlling the point robot from a given start state to a given goal state.

We have also run experiments for agents using a hybrid of a recurrent neural network and genetic algorithm for both Elman and Jordan architectures [4]. In the experiments, the weights of the neural networks is determined directly by genetic algorithm.

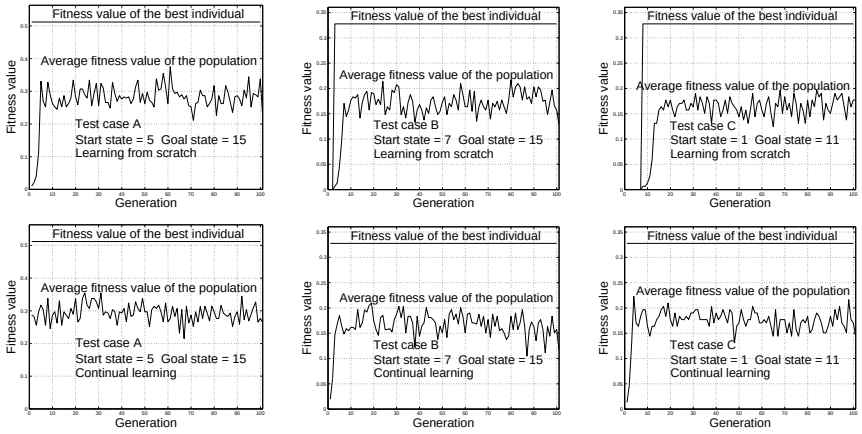


Figure 5: Result obtained for a hybrid of neural networks and genetic algorithm. The top row shows from left to right results of learning from scratch for test case A, B and C respectively. The bottom row shows the corresponding result in continual learning.

5.3 Hybrid of Learning and Evolutionary Algorithm

A population of reinforcement learning agents using Q-learning and whose performance is improved by a genetic algorithm are used to form the hybrid algorithm. In this section, we investigate agents that use the Lamarckian strategy and agents that use the Darwinian strategy [4]. For both agents, the algorithm starts with genetic algorithm, which initializes the Q-tables

of the agents. The agents learn through interaction in their lifetime and change the content of the Q-table as they learn about their environment. At the end of the life of an agent that use the Lamarckian strategy, the collected knowledge which is stored in the Q-table will be written back to the chromosome which encodes it. In other words, the current generation will inherit to the next generation what it has learned about its environment. For agents using Darwinian strategy, the contents of the Q-table will not be written back to the chromosome at the end of the life of the agent. Figure 6 shows the Q-table and the chromosome that encodes it and used in this experiment.

	States				
Actions	0	1	$\cdots$	14	15
`left`	$Q_{0,0}$	$Q_{0,1}$	$\cdots$	$Q_{0,14}$	$Q_{0,15}$
`right`	$Q_{1,0}$	$Q_{1,1}$	$\cdots$	$Q_{1,14}$	$Q_{1,15}$
`forward`	$Q_{2,0}$	$Q_{2,1}$	$\cdots$	$Q_{2,14}$	$Q_{2,15}$

$Q_{0,0}$	$\cdots$	$Q_{0,15}$	$Q_{1,0}$	$\cdots$	$Q_{1,15}$	$Q_{2,0}$	$\cdots$	$Q_{2,15}$

Figure 6: The Q-table and the chromosome that encodes it.

The reward function given by equation (1) is used for the reinforcement learning agents, and the fitness function given by equation (2) is used for the genetic algorithm. The parameters for the genetic algorithm and the reinforcement learning are shown in table 2. The experiment is run for all the test cases and the results are shown in figure 7. As can be seen in the figure, the learning time in continual learning is shorter than the learning time in learning from scratch for all test cases and for both strategies. The average fitness value attained by population of agents using Darwinian strategy is greater than that attained by population of agents using Lamarckian strategy for both learning from scratch and continual learning.

Table 2: The parameters of genetic algorithm and reinforcement learning.

Number of individuals in the Population	50
Crossover probability	0.2
Mutation probability per bit	0.05
Selection method	Truncation selection
Learning rate of reinforcement learning	0.3
Discount rate of reinforcement learning	0.3
Number of bits coding a Q-value	8
Number of generations	100

6 Summary and Analysis of Results

Table 3 summarizes the results of the experiments run in section 5.

The results of experiments clearly show that the learning time required in continual learning is shorter than that required in learning from scratch at both individual and population

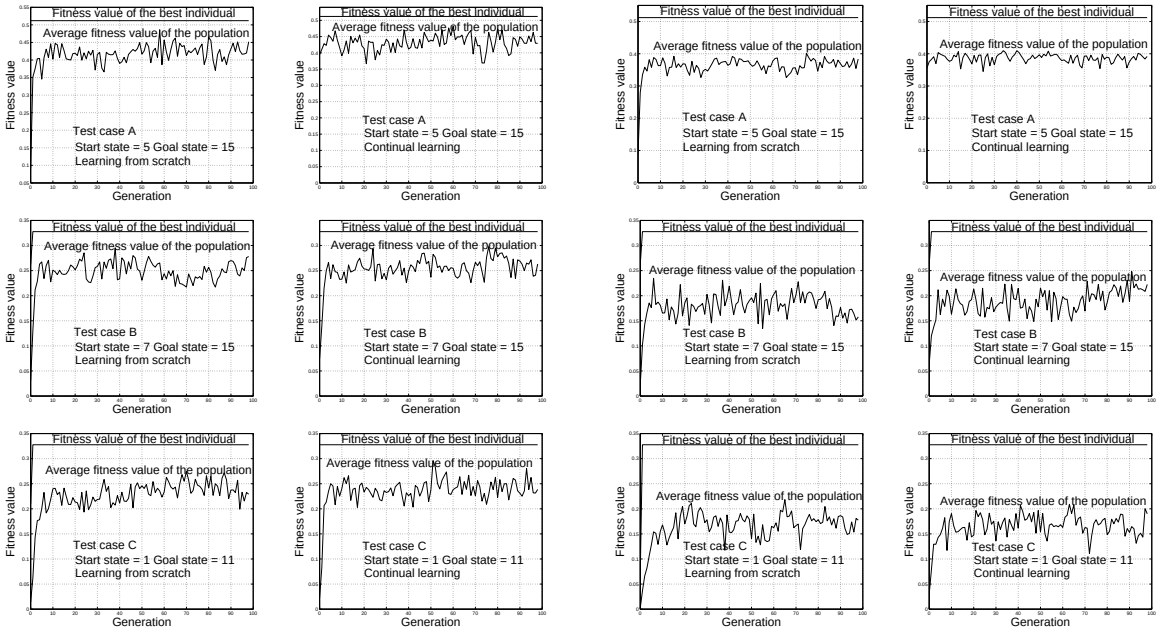


Figure 7: Result obtained for a hybrid of reinforcement learning and genetic algorithm for agents using Darwinian and Lamarckian strategies . The first two columns show results obtained for agents using Darwinian strategy and the last two columns show results obtained for agents using Lamarckian strategy.

Table 3: Summary of results obtained after running the experiments for test case A, B and C respectively. For evolutionary experiments, all individuals have exactly one trial per generation.

Learning and evolutionary methods used by agents	**# of trials in learning from scratch**			**# of trials in continual learning**		
	A	**B**	**C**	**A**	**B**	**C**
Q-Learning	40	50	40	0	14	20
Hybrid of MLP and GA	4	9	8	0	4	4
Hybrid of Q-Learning and GA (Lamarckian)	10	5	7	0	3	4
Hybrid of Q-Learning and GA (Darwinian)	5	5	6	0	3	4

levels and under various learning conditions. They also show that the learning time in continual learning depends on the number of states of a policy, which is going to be learned, that are contained in the previously learned optimal policy. The more states the two policies have in common, the shorter will be the time required in continual learning. For test case A, where the states of a policy are completely contained in the previously learned optimal policy, the agent does not need to learn the optimal policy in continual learning. It is also interesting to see that, even though the two policies have no common states (test case C), the time required in continual learning is shorter than the time required in learning from scratch.

By comparing the results of agents using Q-learning and agents using a hybrid of Q-learning and GA, one can see that agents using a hybrid of Q-learning and GA required less number of trials to learn the action values of the states of the environment in both learning from scratch and continual learning than agent using Q-learning algorithm only. This supports our argument that agents using appropriate hybridization of learning and evolutionary algorithms show better learning and adaptation capability as compared to agents using learning algorithms only.

7 Conclusion and Outlook

We have shown that continual learning requires shorter learning time as compared to learning from scratch under various learning conditions. The different test cases of the experiments show that an agent can use a related knowledge to a new situation, which is going to be learned, to adapt itself faster and make the learning time shorter. Furthermore, the adaptation time required by an agent to adapt to a new situation depends on the amount of knowledge it has about the new situation.

Hybridization of various learning algorithms with evolutionary algorithms will give agents two levels of adaptation capabilities. The first is an individual level adaptation capability, and the second is a population level adaptation capability. The individual level adaptation capability depends on the the learning algorithm used. At population level, the adaptation capability is contained in the variation between individuals.

With adequate hybridization of learning algorithms and evolutionary methods (like genetic algorithms, genetic programming and genetic strategies) it is our believe that one can design better agents with better learning and adaptation capability for either lower or higher cognitive levels.

Acknowledgment

This work is sponsored by the German Academic Exchange Service (DAAD) under grant code R-413-A-01-46029 and the VISATEC project under the EC contract no. IST-2001-3422, which are duly acknowledged. We would like also to thank Bodo Rosenhahn for having important discussions with him.

References

[1] Banzhaf W., Nordin P., Keller R.E., Francone F. D. Genetic Programming. An Introduction on the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann Publishers, 1998.

[2] Barto A.G, Bradtke S.J. and Singh S.P. Learning to act using real-time dynamic programming. Department of Computer Science, University of Massachusetts, Amherst MA 01003, 1993

[3] Hailu G. Towards Real Learning Robots. PhD Thesis, Institute of Computer Science and Applied Mathematics, CAU, Kiel, 1999.

[4] Kassahun Y. and Sommer G. Learning and Adaptation: A Comparison of Methods in Case of Navigation in an Artificial Robot World. Technical Report Nr. 0309. Christian Albrechts University of Kiel. November 2003.

[5] Nolfi S. and Floreano D. Evolutionary Robotics, The Biology, Intelligence and Technology of Self-Organizing Machines. A Bradford Book, The MIT Press, 2000.

[6] Sutton R.S. and Barto A. G. Reinforcement Learning. A Bradford Book, The MIT Press, 1998.

[7] Whitley D. A Genetic Algorithm Tutorial. Computer Science Department, Colorado State University Fort Collins, CO 80523, 1993.