Rev. Roum. Sci. Techn.– Électrotechn. et Énerg.
Vol. **62**, *2*, pp. 209–214, Bucarest, 2017

LEARNING ONLINE SPATIAL EXPLORATION BY OPTIMIZING ARTIFICIAL NEURAL NETWORKS ASSISTED BY A PHEROMONE MAP

BOGDAN-FLORIN FLOREA[1], OVIDIU GRIGORE[1], MIHAI DATCU[1,2]

Key words: Autonomous agents, Cooperative systems, Genetic algorithms, Intelligent robots, Mobile agents.

This paper addresses the problem of online spatial exploration by using reflex agents controlled by neural networks (multilayer perceptrons) optimized using a form of genetic algorithms in combination with a pheromone map that acts as information storage and exchange medium. We have used a fitness function which also contains information about the structure of the problem and progressively changes with the number of generations, ranging from an emphasis on basic behavior related to obstacle avoidance and moving towards the exploration frontier in the early generations to an emphasis on the exploration performance as the number of generations' increases. We have shown that the reflex agents optimized using the technique proposed in this paper are capable to solve the exploration problem even with a small number of neurons.

1. INTRODUCTION

In this paper we address the problem of online spatial exploration by using reflex agents optimized with genetic algorithms. The model-based reflex agents [1] that we propose in this paper are controlled by a feed forward neural network and they use a pheromone map as a form of information storage medium.

The neural network which controls the agent's behavior is optimized using genetic algorithms which evaluate the basic behavior (obstacle avoidance and movement towards the exploration frontier) at the beginning of the optimization process and then the emphasis is shifted towards higher level measures of the exploration performance.

In the last period, even as the technology becomes more and more complex, we keep being amazed by the elegant solutions found by the nature for solving notoriously hard problems, solutions which still have yet to be matched by the modern technology in terms of efficiency, size and quality. This fact embolden us to take an approach based on natural inspiration techniques for solving the spatial exploration problem.

The technique that we propose uses several concepts and techniques of natural inspiration, by using simple reflex agents similar to ants, a pheromone map as information storage and communication medium and a controller based on a neural network optimized using genetic algorithms. The innovation consists into using agents controlled by feed-forward neural networks (multilayer perceptrons) in combination with a pheromone map that acts as information storage and exchange medium during the exploration process.

The problem of online spatial exploration has known a renewed interest with the advent of the autonomous robots, with applications ranging from consumer products and toys to unmanned aerial vehicles and extraterrestrial exploration robots.

First, we present the exploration problem and the formalism used through this paper, then we show the model of the intelligent agent and the evolutionary algorithm used to solve the problem. Finally, we provide a review of the performance of this exploration algorithm benchmarked against some popular algorithms from the scientific literature.

2. THE EXPLORATION PROBLEM

We have modeled the terrain as a discrete 2D grid which is basically a graph (V, E) where V is the set of vertices and E is the set of edges. The vertices corresponding to the cells in the grid are 4-connected to their neighbors, as shown in Fig.1

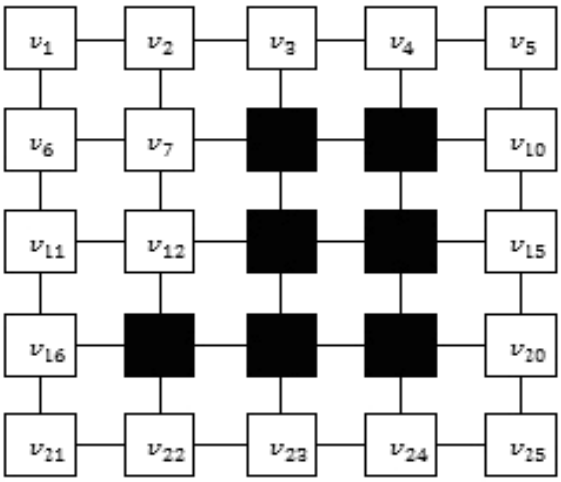

Fig. 1 – An example of the graph model of a 5×5 terrain. The vertices corresponding to obstacles are colored in black.

On the set of vertices, we define the obstacle function, a function which tells whether a node corresponds to a cell occupied by an obstacle or not as follows:

$$f_0 : V \to \{0,1\},\ f_0(v) = \begin{cases} 1, & \text{if } v \text{ is an obstacle} \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

We also define an accessibility function, which takes a value of 1 when the vertex v is accessible from the starting vertex v_0 or 0 otherwise:

$$f_a(v) = \begin{cases} 1, & \neg f_o(v) \wedge \left(\exists u \in V : \left((u \downarrow v) \wedge f_a(u)\right) \vee (u = v_o)\right) \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where: $u \downarrow v$ means vertex u is adjacent to vertex v.

The goal of the exploration algorithm is to find a path so that the agent visits each vertex at least once with a cost as low as possible:

[1] University "Politehnica" of Bucharest, Department of Applied Electronics and Information Engineering, 030018, Romania, bogdan.florea@ai.pub.ro, ovidiu.grigore@ai.pub.ro, mihai.datcu@dlr.de
[2] Remote Sensing Technology Institute, German Aerospace Center (DLR), Oberpfaffen, Weßling, 82234, Germany

$$P = \underset{\substack{(v_0, v_1, v_2 \cdots v_{n-1}) \\ v_i \downarrow v_{i-1}, i=\overline{1,n-1} \\ f_a(v_i)=1, i=\overline{1,n-1}}}{\operatorname{argmin}} \sum_{i=1}^{n-1} f_c\left(e_{i-1,i}\right), \quad (3)$$

where: $P = (v_0, v_1, v_2 \cdots v_{n-1}) \in V^n$ is the path; $e_{i-1,i} = \{v_{i-1}, v_i\}$ is the edge which connects the vertex v_{i-1} to v_i; $f_c : E \to \mathsf{R}$ is the cost function which returns the cost for moving from one vertex to another along an edge.

3. THE MODEL OF THE INTELLIGENT AGENT

The intelligent agent is modeled as a reflex agent with a visibility horizon of one cell, considering the terrain modeled as a 4-connected grid. The agent has 4 obstacle sensors, one for each of the directions in which it can move.

In addition to the obstacle sensors, each agent also has 4 pheromone sensors, one for each direction, which are used to sense the pheromone levels from the environment.

The behavior of the intelligent agent is controlled by a feed-forward neural network (a multilayer perceptron) with $S_0 = 8$ inputs, $S_2 = 4$ outputs and a single hidden layer with $S_1 = 8$ neurons. The number of neurons from the input layer is equal to the number of variables taken as inputs by the neural network. We use 4 neurons as inputs for the obstacle sensors and another 4 neurons as inputs for the pheromone map values. In the output layer we use one neuron for each possible direction. The output neuron with the highest activation level indicates the direction where the agent should move at the next step. Finally, for the hidden layer, we use a small number of neurons in order to avoid overfitting and to make the optimization process computationally tractable. A high number of neurons would make the neural network prone to learning the structure of the maps used in the training set instead of generalizing.

The neural network takes as input the values provided by the obstacle sensors from within the visibility horizon and the corresponding normalized values of the pheromone map for the neighboring cells.

Each layer of the neural network is connected to the next layer by synapses of variable strength modeled by a weight matrix. The weights for the neurons from the hidden layer are contained by the matrix $\mathbf{W}_h$ and the weights for the output neurons are contained by the matrix $\mathbf{W}_0$.

Each of the neurons from the hidden layer and from the output layer contains a summation block for computing the activation potential and then it applies the activation function to the activation potential.

The neurons from the hidden layer and the neurons from the output layer use sigmoid activation functions, more specifically, the logistic function.

This particular choice of architecture for the neural network has been motivated by the fact that a multilayer perceptron with at least one hidden layer and sigmoidal activation functions is a universal approximator [2].

Each of the output neurons corresponds to a neighboring cell and it indicates that the agent wants to move to that cell. If it is not possible to move the desired cell (i.e., due to an obstacle), then the agent chooses the action corresponding to the next neuron. The neurons are then ordered in descending order by the strength of their output.

When the intelligent agent visits a cell, the pheromone map is automatically incremented in order to provide some kind of memory.

4. THE EVOLUTIONARY ALGORITHM

In this section we present the chromosome that we use for encoding the possible solutions for the optimization problem, then we present the initialization method and the mutation and crossover operators. Then, we present the fitness function that we use to evaluate the quality of a chromosome (solution candidate), the selection function that we use to select the parents that participate in the crossover and the replacement function.

4.1. THE CHROMOSOME

In order to perform the optimization process using genetic algorithms, we use real-valued encoding where each weight which defines the neural network is encoded as a floating point number.

Therefore, the chromosome is a vector of real values:

$$\mathbf{z} = \begin{bmatrix} z_1 & z_2 & \cdots & z_S \end{bmatrix}^{\mathrm{T}}, \quad \mathbf{z} \in \mathsf{R}^S, \quad (4)$$

where: $S=(S_0+1)\cdot S_1+(S_1+1)\cdot S_2$ is the number of genes;

S_0 is the size of the input layer of the neural network;

S_1 is the size of the hidden layer of the neural network;

S_2 is the size of the output layer of the neural network;

$z_1 \cdots z_S$ are the genes corresponding to the weights of the neural network.

4.2. THE INITIALIZATION

The population is initialized by sampling the chromosomes from a zero mean uniform distribution:

$$z_k = U(-b, b), \quad b \in \mathsf{R}, \quad k = \overline{1,S}. \quad (5)$$

4.3 THE CROSSOVER OPERATOR

We perform the crossover using the discrete crossover method for real-valued encoding [3]:

$$z^{(i)} \otimes z^{(j)} = z^{(j)} \odot m + z^{(j)} \odot (1_s - m), \quad (6)$$

where: $\otimes$ is the crossover operator; $\odot$ is the element wise multiplication operator; $\mathbf{z}^{(i)}, \mathbf{z}^{(j)}$ are the parents; $\mathbf{z}' = \mathbf{z}^{(i)} \otimes \mathbf{z}^{(j)}$ is the child obtained from the crossover; $\mathbf{1}_S = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^{\mathrm{T}}$ is an all one vector of size S; $\boldsymbol{m} = \begin{bmatrix} m_1 & m_2 & \cdots & m_S \end{bmatrix}^{\mathrm{T}}$ is a binary vector where each element $m_k \in \{0,1\}$ is sampled from a discrete uniform probability distribution with $p(m_k = 0) = 0.5$ and respectively $p(m_k = 1) = 0.5$.

It makes sense to use the discrete crossover, since it allows for the selection of the weight combinations which perform better than each of the parents.

Using the discrete crossover with real-valued encoding might be seen as a limiting factor, but even so, the number of possible children is 2^S, growing exponentially large with the size of the chromosome.

4.4. THE MUTATION OPERATOR

In order to perform mutations and to allow the weights to cover a big range of values, we use the following biased mutation operator:

$$\mathsf{T}z = z + r, \tag{7}$$

where T is the mutation operator, $r = [r_1 \quad r_2 \quad \cdots \quad r_S]^{\mathrm{T}}$ is a real valued vector where each component is 0 with probability $1 - p_m$ and with probability p_m with sampled from a zero mean normal distribution with a standard deviation of σ_m, p_m is a parameter which controls the mutation probability.

The use of a biased Gaussian mutation operator allows for the fine tuning of the neural network's weights around the current value, hopefully moving closer to the local optimum of the fitness function.

The parameter p_m starts with a value of $p_{m0} = 1\ \%$ and it decreases linearly with the number of epochs

$$p_m = p_{m0}\left(1 - \frac{n_e}{N_e}\right), \tag{8}$$

where: n_e is the number of the current epoch; N_e is the maximum number of epochs.

This decrease of the parameter p_m corresponds to a decrease of the mutation probability, and this inspiration comes from the biology, where simple forms of life (i.e., the viruses) have a higher mutation probability per nucleotide than the more complicated organisms. As the optimization progresses with the number of epochs, the agents expose a more complex behavior, similar to the evolution from simple forms of life to more complex forms of life.

4.5. THE FITNESS FUNCTION

The value returned by the fitness function that we use is a progressive measure of the fitness of each individual for the exploration task, gradually shifting from rewarding basic behavior like obstacle avoidance and gradually moving towards rewarding the actual exploration performance measured by the number of the unexplored cells.

We use the following fitness function:

$$f(\mathbf{z}) = f_e(\mathbf{z}) - w_i \cdot c_i(\mathbf{z}) - w_a \cdot c_a(\mathbf{z}) - w_e \cdot c_e(\mathbf{z}), \tag{9}$$

where $\mathbf{z}$ is the chromosome whose fitness we evaluate, $f_e(\mathbf{z})$ represents the number of explored cells, $c_i(\mathbf{z})$ is a cost for the invalid actions that the agent tried to take, $c_a(\mathbf{z})$ is a cost for the failing to avoid obstacle, $c_e(\mathbf{z})$ is a cost for failing to move towards unexplored cells when already near them, w_i, w_a, w_e are the weights allocated to each cost.

The number of explored cells $f_e(\mathbf{z})$ is computed by running a simulation of the agent corresponding to the evaluated chromosome for n steps

$$n = N_0 \cdot \frac{n_e}{N_e}, \tag{10}$$

where n is the number of steps for which we run the simulation, N_0 is the number of steps that we estimate to be necessary for a reflex agent to explore the map, n_e is the number of the current epoch, N_e is the maximum number of epochs for which we ran the optimization algorithm.

The cost for invalid actions $c_i(\mathbf{z})$ is computed by counting the number of times that the neural network controller attempted to indicate an action which was invalid (running into an obstacle for example):

$$c_i(\mathbf{z}) = \sum_{k=1}^{n} \left| \left\{ y_i^{(k)} \middle| y_i^{(k)} > y_f^{(k)} \right\} \right|, \tag{11}$$

where $y_f^{(k)} = \max_{\text{s.t. } f_o(v_{kj})=0} \left\{ y_j^{(k)} \right\}$ is the most confident output neuron whose corresponding action is feasible, $v_{kj} = f_s\left(v_k, a\left(y_j^{(k)}\right)\right)$ is the successor function which returns the state of the agent after taking the action $a\left(y_j^{(k)}\right)$ from the state v_k, $a\left(y_j^{(k)}\right)$ is the action corresponding to the j^{th} output neuron.

The cost for failing to avoid the nearby obstacles $c_a(\mathbf{z})$ is computed by considering the error of each output neuron for the situation when only one neuron should fire unambiguously. This situation corresponds to the scenario when the agent is surrounded by obstacles and it has just one option for moving without running into an obstacle.

This cost is computed by considering the squared error:

$$c_a(\mathbf{z}) = \sum_{j=1}^{S_2} \left\| \boldsymbol{y}\left(\boldsymbol{x}^{(j)}\right) - \boldsymbol{t}_j \right\|_2^2, \tag{12}$$

where

$$\boldsymbol{x}^{(j)} = \begin{bmatrix} 1-\delta_{j1} \\ \vdots \\ 1-\delta_{jS_2} \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{S_2} \end{bmatrix}, \quad \boldsymbol{t}_j = \begin{bmatrix} \delta_{j1} \\ \vdots \\ \delta_{jS_2} \end{bmatrix},$$

$\boldsymbol{x}^{(j)}$ is an input vector corresponding to the scenario described above (and considering the pheromone map filled with zeros), $\boldsymbol{y}$ is the output of the neural network, $\boldsymbol{t}_j$ is the desired output, $\delta_{ij} = \begin{cases} 1, i=j \\ 0, i \neq j \end{cases}$ is the Kronecker delta function, $S_0 = 8$, $S_2 = 4$ since we used a 4-connected grid.

The cost for failing to move towards the unexplored cells $c_e(\mathbf{z})$ is computed by considering the error of each output neuron for the situation when only one neuron should fire unambiguously. This situation corresponds to the scenario when the agent has only one unexplored cell adjacent to its current location (all other neighboring cells have pheromone values greater than zero).

This cost is computed as follows:

$$c_e(\mathbf{z}) = \sum_{j=1}^{S_2} \left\| \boldsymbol{y}\left(\boldsymbol{x}^{(j)}\right) - \boldsymbol{d}_j \right\|_2^2, \tag{13}$$

where

$$\boldsymbol{x}^{(j)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1-\delta_{j1} \\ \vdots \\ 1-\delta_{jS_2} \end{bmatrix}, \quad \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{S_2} \end{bmatrix}, \quad \boldsymbol{t}_j = \begin{bmatrix} \delta_{j1} \\ \vdots \\ \delta_{jS_2} \end{bmatrix}.$$

$\boldsymbol{x}^{(j)}$ is an input vector corresponding to the scenario

described above (and considering that all the neighboring cells are accessible), $\boldsymbol{y}$ is the output of the neural network, $\boldsymbol{t}_j$ is the desired output

4.6. THE SELECTION FUNCTION

In order to select the parents, we have used the rank order selection:

$$p\left(\mathbf{z}^{(i)}\right)=\frac{2-SP+2\cdot(SP-1)\cdot\dfrac{\text{rank}\left(\mathbf{z}^{(i)}\right)}{N_z}}{\sum_{k=1}^{N_z}\left(2-SP+2\cdot(SP-1)\cdot\dfrac{\text{rank}\left(\mathbf{z}^{(k)}\right)}{N_z}\right)}, \qquad (14)$$

where $p(\mathbf{z}^{(i)})$ is the probability of choosing the individual $\mathbf{z}$ as a parent, $\text{rank}\left(\mathbf{z}^{(i)}\right)=\left|\left\{\mathbf{z}\middle|\text{rank}\left(\mathbf{z}\right)\middle|\text{rank}\left(\mathbf{z}^{(i)}\right)\right\}\right|$ is the rank of the chromosome $\mathbf{z}^{(i)}$ [computed considering the value of the fitness function], $SP\in[1,2]$ is the selective pressure.

At each epoch, N_z pairs of parents are selected which are used to produce the children.

4.7. THE REPLACEMENT FUNCTION

We use a genetic algorithm implementation based on steady states and therefore the best of the generated children replace the least fit individuals of the current population at each epoch (only if they are better than the individuals that they replace). At each epoch we generate $N_c = N_z$ children.

5. **THE RESULTS**

In order to evaluate and compare the solutions obtained using genetic algorithms with other exploration algorithms from the literature, we use as a benchmark the number of steps required for exploring 80 – 90 % of the accessible cells from the test maps.

The neural network used to control the behavior of the intelligent agents has been trained using 10 randomly generated maps of the same size as the "Tiny World" map presented in Fig. 2. Good results have been obtained with the following parameters:

$w_i = 0.2$, $w_a = 10$, $w_e = 10$, $SP = 1.5$, $N_z = 24$,
$N_e = 300$, $S_0 = 8$, $S_1 = 8$, $S_2 = 4$, $b = 10$, $\sigma_m = 10$.

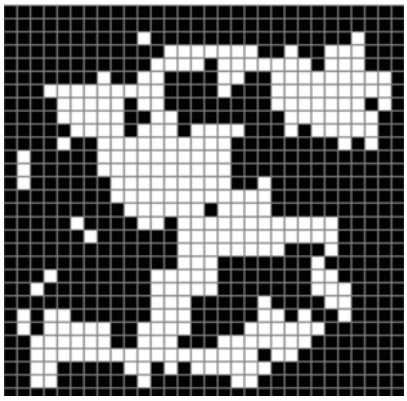

Fig. 2 – The "Tiny World" map.

The obtained solutions have been also tested on a new set of 100 randomly generated maps of the same size as the maps used for training, in order to check if the neural network is able to generalize.

We have compared the exploration speed obtained by the solutions produced by the current algorithm with several standard algorithms from the literature, including the node counting algorithm [4], the exploration algorithm based on intelligent agents controlled using Thrun's rule [5], vertex ant walk which is a smell-oriented exploration algorithm [6] and learning real-time A* with a one cell look-ahead [7, 8].

For the comparison, we used 8 agents, each controlled by a neural network with one hidden layer with 8 neurons, trained using genetic algorithms for 300 epochs with a population size of 24 individuals. From those 8 solutions, we have selected the best performing agent (by testing them on the training maps, which were used for optimization).

The exploration speed results computed by running the best performing agent on the test map set and aiming for 80 % coverage of the accessible cells can be seen in Table 1.

Table 1

Number of steps necessary for 80 % exploration of the map (1 run using the training set of 10 maps)

Algorithm	Step count (median)
Current algorithm	364.50
Node counting algorithm	421.50
Thrun's rule	427.25
Vertex ant walk	409.75
Learning real-time A*	432.00

As expected, the agent has a good performance on the training and outperforms in terms of speed the other exploration algorithms based on reflex agents. Using a small number of hidden neurons, we have decreased the predisposition of the neural network to over fitting.

The exploration speed results for one run on the training map set for an 80 % coverage of the accessible cells can be seen in Table 2.

Table 2

Number of steps necessary for 80 % exploration of the map (1 run using the training set of 100 maps)

Algorithm	Step count (median)
Current algorithm	376.50
Node counting algorithm	379.50
Thrun's rule	364.00
Vertex ant walk	332.50
Learning real-time A*	351.00

We have also tested the same agent aiming for 90 % map exploration. Since the agent was not capable to explore all the maps up to 90 %, we included only the median in the comparison. The results are presented in Tables 3 and 4.

Table 3

Number of steps necessary for 90 % exploration of the map (1 run using the training set of 10 maps)

Algorithm	Step count (median)
Current algorithm	418.00
Node counting algorithm	532.13
Thrun's rule	543.00
Vertex ant walk	524.13
Learning real-time A*	521.63

Table 4

Number of steps necessary for 90 % exploration of the map
(1 run using the training set of 100 maps)

Algorithm	Step count (median)
Current algorithm	574.50
Node counting algorithm	437.50
Thrun's rule	447.50
Vertex ant walk	439.50
Learning real-time A*	440.00

From Tables 3 and 4, it can be observed that on the test map set, the performance of the agents obtained using genetic algorithms is slightly lower that on the training set. This is expected, since the optimization has been performed on the training map set and therefore the agents are optimized to explore that specific maps, and since the training map set is quite small (due to computational reasons), a small degree of over fitting is inevitable.

As it can be seen from the Table 5, for the successfully explored maps, all agents had similar performance.

We have also analyzed separately the performance of the solution that we obtained on the maps that it was able to explore up to 90 % and we have summarized these results in Table 5.

Table 5

Number of steps necessary for 90 % exploration of the map
(1 run using only the maps explored successfully up to 90%
from the training set of 100 maps)

Algorithm	Step count (median)
Current algorithm	467.00
Node counting algorithm	462.00
Thrun's rule	462.00
Vertex ant walk	481.00
Learning real-time A*	466.00

As it can be observed from the Tables 4 and 5, when we do consider only the maps for which the agent obtained using the current algorithm succeeds to reach 90 % exploration (in the scenarios where the exploration didn't reach 90 % the step count was considered to be infinite), it can be seen that the performance of our algorithm is comparable with the performance of the other algorithms from the literature.

As it can be observed the Tables 1, 2, 3 and 4 the intelligent agent obtained using the technique described by us is capable to generalize and it can complete the map exploration up to a given percent in most of the cases.

The agent that we analyzed was controlled by neural networks whose weights are represented graphically in Fig. 3 and Fig. 4. For convenient interpretation, the weight matrices have been represented as grayscale images, where white corresponds to excitatory weights and black corresponds to inhibitory weights. The weights of each neuron are arranged on the row corresponding to that neuron.

In order to get a glimpse on what neural network has evolved, we have analyzed the weight matrices corresponding to the hidden neurons and to the output neurons.

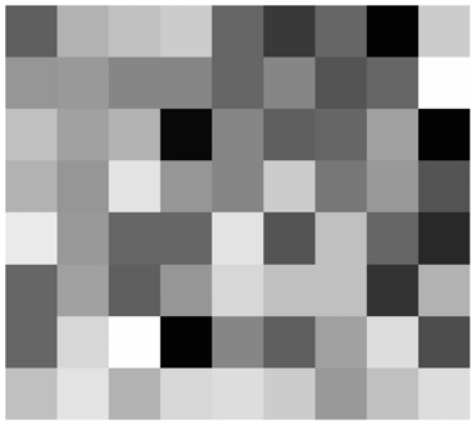

Fig. 3 – The weights of the hidden neurons.

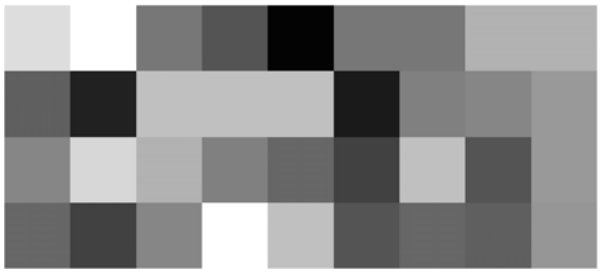

Fig. 4 – The weights of the output neurons.

As it can be seen from Figs. 3 and 4, both the weight matrix of the hidden neurons and the weight matrix of the output neurons are dense matrices, suggesting the fact that the agent is following a complex rule in order to solve the exploration task.

As we have shown through the analysis from the above, the hidden neurons represent complex high level features which characterize the situation encountered by the intelligent agent, and one hidden neuron contributes to the action potential of multiple output neurons.

6. **CONCLUSIONS**

In this paper we have shown that it is possible to obtain good exploration agents by optimizing them using evolutionary algorithms. We have shown that even with a very low number of neurons (only 8 neurons in the hidden layer), the neural network is capable to generalize and to solve the problem on maps unknown to the agent.

Although the results that we have obtained do not outperform the human designed exploration agents on the unknown map, they are still an important step forward because we have shown that is possible to obtain reasonable performance by using automatic design methods instead of human based design.

In this paper we have shown that by optimizing a neural network with genetic algorithms it is possible to obtain complex behavior, which is non-trivial to design by hand.

Our results show that using evolutionary strategies it is possible to solve complex problems, like the exploration problem and to obtain global results by using only local information.

Our research has shown once again that evolutionary algorithms can solve difficult problems which are very different one from each other. We used evolutionary computation to solve the exploration problems, while others used similar algorithms to successfully solve very different problems, like the power consumption optimization problem [9] or the eddy current heating process optimization problem [10].

For solving the exploration problem, we used the multilayer perceptron neural network, but most of the neural networks that are universal approximators should be able to solve this problem given the correct weights for the neurons.

We have also shown in this paper that using the technique proposed by us it is possible to obtain solutions which expose highly complex behaviors, sometimes arriving at solutions similar to what a human designer would implement (like the obstacle edge following agent).

ACKNOWLEDGEMENTS

The work has been funded by the Sectoral Operational Programme Human Resources Development 2007–2013 of the Ministry of European Funds through the Financial Agreement POSDRU/159/1.5/S/132397.

Received on March 19, 2016

REFERENCES

1. Teodor Leuca, Mihaela Novac, *Optimization of Eddy Current Heating Process using Genetic Algorithms*, Part 2, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., **54**, *4*, pp. 231, 2012.
2. G. Cybenko, *Approximations by superpositions of sigmoidal functions*, Mathematics of Control, Signals, and Systems, **2**, *4*, pp. 303–314 (1989).
3. H. Mühlenbein, D. Schlierkamp-Voosen, *Evolutionary computation*, pp. 25–49 (1993).
4. A. Pirzadeh, W. Snyder, *A unified solution to coverage and search in explored and unexplored terrains using indirect control*, Proceedings of the International Conference on Robotics and Automation, pp. 2113–2119 (1990).
5. S. Thrun, *Efficient exploration in reinforcement learning*, Technical Report CMU-CS-92-102, School of Computer Science, Carnegie Mellon University, Pittsburgh (Pennsylvania), 1992.
6. I. Wagner, M. Lindenbaum, A. Bruckstein, *On-Line Graph Searching by a Smell-Oriented Vertex Process*, AAAI Technical Report WS-97-10, 1997.
7. S. Koenig, Y. Liu, *Terrain Coverage with Ant Robots: A Simulation Study*, AGENTS'01, Montreal, 2011.
8. R. Korf, *Real-Time Heuristic Search*, Artificial Intelligence, **42**, pp. 189–211 (1990).
9. Gheorghe Şerban, Laurenţiu Ionescu, Alin Mazăre, *The possibility of optimisation for power supply consumption using evolvable power regulator*, Part 2, Rev. Roum. Sci. Techn. – Électrotechn. et Énerg., **57**, *2*, pp. 222–231, 2012.
10. S. Russel, P. Norvig, *Artificial Intelligence: A Modern Approach* (Ch. 2), 2nd Ed., Upper Saddle River, New Jersey, Prentice Hall, 2003.