Using Assembler Encoding to Solve Predator-Prey Problem

Tomasz Praczyk

***Abstract*—**The paper presents a neuro-evolutionary method called Assembler Encoding. The method was tested in the predator-prey problem. To compare Assembler Encoding with another neuro-evolutionary method, in the experiments, a co-evolutionary version of simple connectivity matrix was also applied.

***Keywords*—**evolutionary neural networks, predator-prey problem.

I.Introduction

Artificial neural networks (ANNs) constitute a sub–domain of artificial intelligence that is broadly used to solve various problems in different fields (e.g. pattern classification, function approximation, optimization, image compression, associative memories, robot control problems, etc.). The performance of ANNs highly depends on two factors, i.e. network's topology and a set of network's parameters (typically weights). Therefore, to develop an appropriate ANN it is necessary to determine the topology and parameters. There are many different ANN learning algorithms that change values of parameters leaving the structure completely intact [16]. In such a case, the process of searching for the proper ANN topology is the task of a designer who arbitrarily chooses the ANN structure, starts ANN learning and finally puts ANN to a test. If the result of the test is satisfactory, the learning process is stopped. If not, it is continued further. The designer manually determines the next potential topology and runs the learning algorithm again. Such loop – topology determination and learning is repeated until ANN which is able to carry out a dedicated task at an appropriate level is found. At first glance, it is apparent that such a procedure could be very time-consuming and, what is worse, in the case of more complex problems, can lead to a situation when all chosen and trained ANNs would be incapable of solving the task.

In addition to the learning concept presented above, there exist other approaches that can be called constructive and destructive. The constructive ones use a learning philosophy that consists in incremental development of ANN starting from small architecture. At the beginning, ANN has a small number of components to which next components are gradually added until a resultant ANN fully meets the requirements imposed. On the other hand, the destructive ones prepare a large fully connected ANN and then try to remove individual elements of the network, such as synaptic connections and neurons.

Genetic Algorithm (GA) is a next technique that has been successfully applied to search for optimal ANNs [3],[4],[7],[15] for the recent years. GA processes a population of genotypes that typically encode one phenotype although encoding several phenotypes is also possible. In ANN evolution, genotypes are encodings of corresponding ANNs (phenotypes). The evolutionary procedure involves selecting genotypes (encoded ANNs) for reproduction based on their fitness, and then by introducing genetically changed offspring (mutation, crossover and other genetic operators) into a next population. Repeating the whole procedure over many generations causes the population of encoded ANNs to gradually evolve into individuals corresponding to high fitness phenotypes (ANNs).

There are a lot of ANN encoding methods. Several of them are briefly presented further in the paper. In principle, all existing encoding methods can be divided into two main classes, i.e. direct encodings and indirect encodings. As for the direct methods, the whole information necessary to create ANN (e.g. weights, number of neurons, number of layers) is directly stored in a chromosomes. Thus, to encode larger ANNs larger chromosomes are necessary, which is the main drawback of the direct methods. As regards the indirect methods, we deal with chromosomes which are recipes how to create ANN. Such encodings can be used to create larger neural architectures by means of relatively short chromosomes.

The paper presents a new indirect ANN encoding method called Assembler Encoding (AE) [13]. AE originates from the cellular encoding [5], although, it also has features common with Linear Genetic Programming (LGP) [11]. In AE, the process of ANN construction consists of three stages (Fig. 1). First, GA is used to produce Assembler Encoding Programs (AEPs). Next, each AEP creates and fills up Network Definition Matrix (NDM) which includes all the information necessary to create ANN. Then, once AEP stops working the matrix is transformed into ANN.

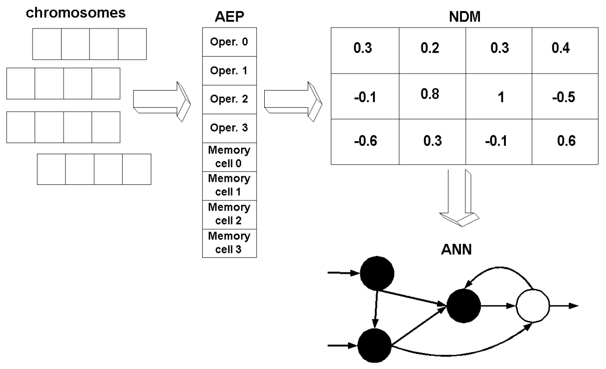


Fig. 1 Diagram of AE

To test AE simple experiments in the predator-prey problem were carried out. In the experiments, the task of AE was to generate ANNs controlling a set of cooperating predators whose common goal was to capture a fast moving prey. Apart from AE, for the purpose of comparison, a modified version of CM was also used in the experiments. The results of the experiments are presented at the end of the paper.

The paper is organized as follows: section 2 reviews related research; section 3 presents AE; section 4 illustrates the results of the experiments; section 5 is the summary.

II.RELATED WORK

For the recent years many attempts have been made to define genotypes for ANNs and to describe the genotype into phenotype mapping process. One of the earliest concepts was proposed by Miller, Todd and Hedge [8]. In their approach ANN is represented in the form of the Connectivity Matrix. Each element of the matrix informs about existence of connection between two neurons or about lack of such connection.

Moriarty and Miikkulainen [9] proposed a Symbiotic Adaptive NeuroEvolution (SANE). Their concept assumes that information necessary to create ANN is included in two types of individuals, i.e. in blueprints and in encoded neurons. Both types of individuals evolve in separate populations. The task of the blueprints is to record the most effective combinations of neurons. Each blueprint specifies a set of neurons that cooperate well together. The population of neurons includes individuals encoding hidden neurons of two-layered feed-forward ANN (FFANN). Each individual from the population of neurons defines connections of the neuron with input and output neurons and the strength of each connection.

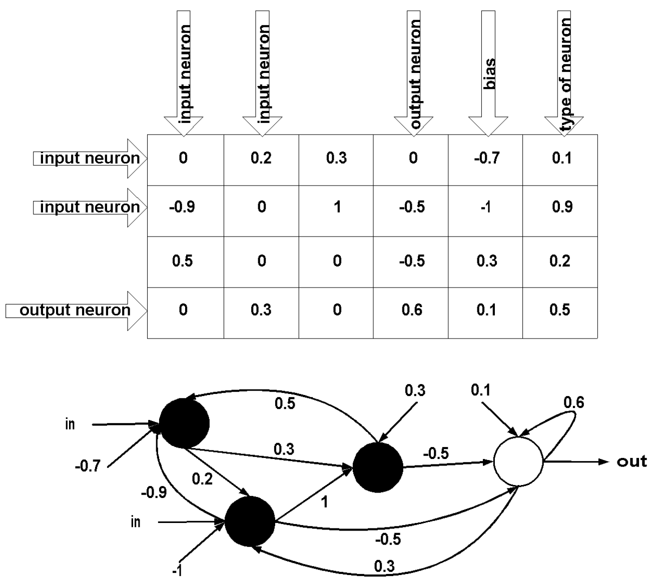
Kitano [6] defined the matrix rewriting encoding scheme. Initially, the method assumes 2x2 matrix that contains non-terminal elements. These elements are subsequently substituted for matrices including other non-terminal elements or terminal elements. This process is repeated until the resultant enlarged matrix contains only the terminals that indicate either existence of connection between neurons or lack of such connection.

In the Nolfi and Parisi model [10], the genotype defines the location of each neuron in a two-dimensional space and growth parameters of each neuron’s axon. Neurons that are on the left part of the space are considered to be input neurons and the ones placed on the right are considered to be output neurons. The remaining neurons are hidden neurons. After the location phase, axons of neurons start to grow further according to an assumed procedure. The connection between neurons is established if the branching axon of a source neuron reaches another neuron.

Chromosome in Gruau’s cellular encoding [5] contains a set of instructions that are applied to ANN consisting initially of one hidden node. ANN evolves towards larger structures during successive executions of individual instructions. The instructions are organized into a tree and include such operations as: node duplication, node division, removal of connectivity and many others. A very important feature of the cellular encoding is its potential to build modular ANNs consisting of similar elements located in various places of a network. This potential is a result of applying a set of trees (with instructions) instead of applying a single tree, and repeated execution of instructions grouped in each of them. The result of such a procedure is analogous to the multiple procedure execution in the main body of a structural program. Another crucial characteristic of the cellular encoding is the form of chromosome – a tree. Due to this feature the only evolutionary technique, which is applicable to process individuals constructed in this way, is genetic programming.

III. ASSEMBLER ENCODING - FUNDAMENTALS

In AE, ANN is represented in the form of AEP, which is composed of two parts, i.e. a part including operations (the code part of AEP) and a part including data (the memory part of AEP). The task of AEP is to create and to fill in NDM with values. To this end, AEP uses predefined operations which are run one by one. When working, the operations use data located at the end of AEP. Once the last operation finishes its work, the process of creating NDM is completed. The matrix is then transformed into ANN.



	input neuron	input neuron		output neuron	bias	type of neuron
input neuron	0	0.2	0.3	0	-0.7	0.1
input neuron	-0.9	0	1	-0.5	-1	0.9
	0.5	0	0	-0.5	0.3	0.2
output neuron	0	0.3	0	0.6	0.1	0.5

```
if(abs(type_of_neuron)<=0.5)
then
  sigmoid
else
  linear
```

Fig. 2 NDM used as the CM

A. Network Definition Matrix

NDM, as the same name implies, is the matrix defining ANN. It stores all the information necessary to create and to functioning ANN. In principle, NDM can have any structure, i.e. it can define ANN in any way. Individual elements of NDM can inform about synaptic weights of interneuron connections, although any other interpretation is also possible. The way of representing ANN by means of NDM always depends on the type of ANN we want to obtain. In the experiments, two types of ANNs were used, i.e. ANNs whose architecture was permanently fixed as a result of evolutionary process as well as dynamic ANNs with Hebb learning, whose weights underwent changes during ANNs's "life" [4]. To define complete architecture of ANN, i.e. weights, topology, and transfer functions, NDM can take the form of the classical CM. In turn, to represent ANN with Hebb self-organization somewhat different construction of NDM is necessary. Two forms of NDM, used in the experiments, are described below.

NDM used as CM is organized as follows. Each element of NDM determines synaptic weight between corresponding neurons. For example, component$_{i,j}$ defines the link from neuron i to neuron j. Elements of NDM unimportant from the point of view of the process of ANN construction, for example because of assumed feed-forward structure of ANN, are neglected during building ANN. Apart from the basic part, NDM also contains three additional columns that describe parameters of neurons, i.e. type of neuron (sigmoid, radial, linear), parameter of neuron and bias.

NDM used to represent a dynamic ANN with Hebb self-organization is defined as follows. It includes H rows and Z=2M+2 columns where H denotes the number of hidden and output neurons whereas M is the number of all neurons in ANN. Extra two columns, as in the previous case, include additional information about neurons, i.e. bias and value of a single parameter of a neuron (in this case only sigmoid neurons are considered). The main part of NDM consists of two sub-matrices of equal size (HxM). The first sub-matrix determines the topology of ANN, i.e. it indicates which connections exist in ANN and which do not. Each element of this sub-matrix unequal to zero informs about a connection between neurons. A sign of this element determines a sign of the connection while a value of the element determines a type of Hebb rule assigned to the connection. For example, the value -0.2 of the element $NDM[n,m]$ (n=1..H, m=1..M, neurons are indexed from 0 to M) informs both about the negative connection between m^{th} and $[n+(M-H)]^{th}$ neuron and about a plain Hebb rule assigned to that connection. In the experiments, described further, five types of Hebb rules were used [4]:

1. *Plain Hebbian rule*: strengthens the synapse proportionally to the correlated activity of the pre- and post-synaptic neurons.
$$\Delta w = (1 - w)xy \qquad (1)$$
2. *Postsynaptic rule*: behaves as the plain Hebbian rule, but in addition it weakens the synapse when the postsynaptic neuron is active but the presynaptic is not.
$$\Delta w = w(-1 + x)y + (1 - w)xy \qquad (2)$$
3. *Presynaptic rule*: weakening occurs when the presynaptic neuron is active but the postsynaptic is

not.

$$\Delta w = wx(-1 + y)y + (1 - w)xy \tag{3}$$

4. *Covariance rule*: strengthens the synapse whenever the difference between the activations of the two neurons is less than half their maximum activity, otherwise the synapse is weakened. In other words, this rule makes the synapse stronger when the two neurons have similar activity and makes it weaker otherwise:

$$\Delta w = \begin{cases} (1 - w)\Psi(x, y) \text{ if } \Psi(x, y) > 0 \\ w\Psi(x, y) \text{ otherwise} \end{cases} \tag{4}$$

where $\Psi(x, y) = \tanh(4(1 - |x - y|) - 2)$ is a measure of a difference between presynaptic and postsynaptic activity. $\Psi(x, y) > 0$ if the difference is higher or equal to 0.5 and $\Psi(x, y) < 0$ if the difference is smaller than 0.5.

5. *"Zero" rule*: a synapse does not change strength during "life" of ANN.

$$\Delta w = 0 \tag{5}$$

The second sub-matrix of NDM incorporates learning rates necessary to update the strength of each synaptic weight. For example, *NDM*[*n,m*]=-0.2, where *n*=1..*H* and *m*=*M*..2*M*, informs that learning rate applied to update the connection between [*m-M*]th and [*n*+(*M-H*)]th neuron amounts to |-0.2|. If there exists a connection between neurons but the learning rate corresponding to this connection amounts to zero, to update the strength of the connection, a default nonzero value of the learning rate is used (e.g. 0.5).

Hebb rules from the first part of NDM and the learning rates from the second part are necessary to determine changes that take place in each interneuron connection. Each synaptic weight in ANN alters according to the following formula [4]:

$$w_{ij}^{t} = w_{ij}^{t-1} + \eta_{ij}\Delta w_{ij} \tag{6}$$

where w_{ij}^{t}, w_{ij}^{t-1} are synaptic weights between jth and ith neuron, respectively after and before update, and $0 \le \eta_{ij} \le 1$ is the learning rate.

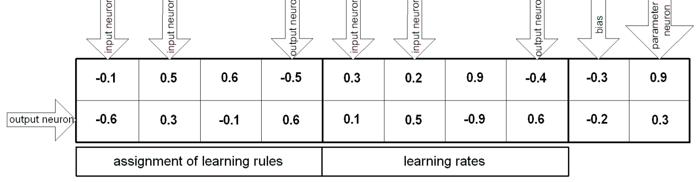


Fig. 3 NDM used to define dynamic ANN

First, once ANN is created, all weights of all nonzero connections are fixed in some assumed manner, for example at random. Then, synaptic weights change according to the formula (6). All synapses can change the strength but they cannot change the sign, which is determined permanently in NDM. The synaptic strength cannot grow indefinitely. All weights range <0,1>. This is possible thanks to application of the self-limiting mechanism in all of Hebb rules mentioned above. An update of each synaptic weight occurs once an input signal is propagated to output neurons, i.e. each time a decision has been taken by ANN.

B. Operations

AEPs can use various operations. The main task of most operations is to modify NDM. The modification can involve a single element of the matrix or a group of elements. Fig. 4 and Fig. 5 present the implementation of two example operations.

```
CHGC0(p0,p1,p2,p3)
{
column=(abs(p0)+R2)mod NDM.height;
numberOfIterations=abs(p2)mod NDM.width;
for(i=0;i<=numberOfIterations;i++)
       {
       row=(i+R1)mod NDM.width;
       NDM[row,column]=D[(abs(p1)+i)mod D.length]
       /Max_value;
       }
```

Fig. 4 `CHGC0` operation changing a part of column of NDM (*NDM*[*i,j*] is element of NDM, R_i *i*=1,2 is value of *i*th register, *Max_value* is scaling value which scales all elements of NDM to <-1,1>, `D[i]` is ith element of data, `D.length` is number of memory cells)

`CHGC0` presented in Fig. 4 modifies NDM elements located in the column indicated by parameter p_0 and register R_2. The number of elements being updated is stored in parameter p_2. The index of the first element being updated is located in register R_1. To update elements of NDM, `CHGC0` uses data from AEP. The index to a memory cell including the first element of data used by `CHGC0` is stored in p_1.

```
CHG_MEMORY(p0,list1,list2)
{
for(i=0;i<list1.length;i++)
      for(j=0;j<list2.length;j++)
            {
            row=(list1[i]+R1)mod NDM.width;
            column=(list2[j]+R2)mod NDM.height;
            NDM[row,column]=
            D[(abs(p0)+i*list2.length+j)
            mod D.length]/Max_value;
            }
```

Fig. 5 `CHG_MEMORY` operation changing elements of NDM indicated in `list1` and `list2`

`CHG_MEMORY` presented in Fig. 5 modifies elements of NDM indicated in `list1` and `list2`. The lists mentioned include numbers of columns and rows of NDM (`list1` includes numbers of rows while `list2` contains numbers of columns) which, in turn, indicate elements of the matrix that are updated as a result of execution of the operation. All possible combinations of columns and rows considered in both lists determine a set of elements that are altered by the operation. p_0 indicates a place in the memory part of AEP where new values for updated elements can be found.

In addition to operations whose task is to modify a content of NDM AE also uses jump operation denoted as `JMP`. The jump makes it possible to repeatedly use the same code of AEP in different places of NDM.

C.Evolution in AE

In AE, AEPs and in consequence ANNs are created by means GAs. The evolution of AEPs proceeds according the scheme which is an adaptation of the idea of evolving co-adapted subcomponents proposed by Potter and De Jong [12]. To create AEP the scheme mentioned combines operations and data from various populations. Each population including chromosomes-operations (each chromosome-operation encodes the type of operation, e.g. `CHGC0`, and parameters of operation; implementations of operations do not evolve) has a number assigned determining the position of the operation from the population in AEP. In this approach, the number of operations corresponds to the number of populations including chromosomes-operations. Each population delegates exactly one representative to each AEP.

At the beginning, AEPs have only one operation and a

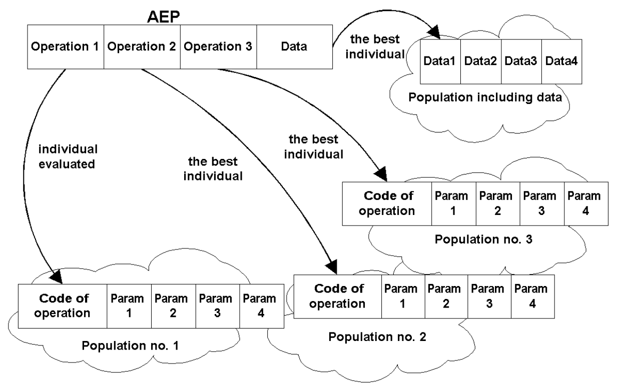


Fig. 6 AEP encoding scheme

sequence of data. Both the operation and data come from two different populations. Further populations including operations are successively added if generated AEPs cannot accomplish progress in performance over an assumed number of co-evolutionary cycles (we use term "co-evolutionary cycle" to differ it from the evolutionary generation that takes place inside a single population with operations and data). Populations with operations and data can also be replaced by newly created populations. This can happen if the contribution of a given population to AEPs is considerably less than the contribution of the remaining populations.

Individual operations in AE can be encoded in two ways. For example, `CHGC0` presented in Fig . 4 is encoded in the form of binary string including five blocks of genes. The first block determines a code of the operation (e.g. binary 00000 indicates that we deal with `CHGC0`), while the remaining blocks contain a binary representation of four parameters of the operation.

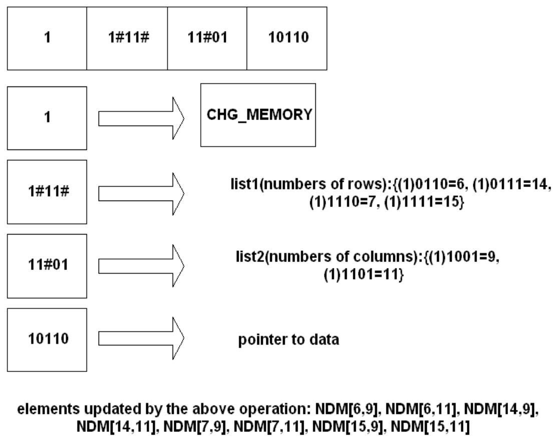

1
1#11#
11#01
10110
1
CHG_MEMORY
1#11#
list1(numbers of rows):{(1)0110=6, (1)0111=14, (1)1110=7, (1)1111=15}
11#01
list2(numbers of columns):{(1)1001=9, (1)1101=11}
10110
pointer to data
elements updated by the above operation: NDM[6,9], NDM[6,11], NDM[14,9], NDM[14,11], NDM[7,9], NDM[7,11], NDM[15,9], NDM[15,11]

Fig. 7 Encoding `CHG_MEMORY` operation

`CHG_MEMORY` (Fig. 5) is represented in a somewhat different way. The encoded form of this operation resembles classifier from Learning Classifier Systems [2]. Similarity between classifier and the encoded operation results from the use of the so called *don't care* symbol "#" in both cases. Each encoded `CHG_MEMORY` consists of four blocks of genes. The first single-bit block determines one of two possible variants of the operation. The second and the third block indicate location of changes performed by the operation (*don't care* symbol is used for this purpose). The last block specifies the value of the integer parameter of the operation. The example use of *don't care* symbol to locate changes in NDM is illustrated in Fig 7.

IV.Experiments

The experiments reported in the paper are only the first step in the whole research process that is necessary to be done to discover full potentials of the encoding method proposed.

The main goal of the experiments was only to learn whether AE can be used to create simple "static" and dynamic ANNs. During the tests, the task of all ANNs was to solve a simple version of the predator-prey problem. Apart from AE, for the purpose of comparison, a modified version of CM was also used in the experiments. The main idea behind comparing AE with the concept of Miller et al. was that both solutions use CM to represent ANN (in AE CM is called NDM). Both methods differ only in the approaches to creating the matrix. While in the classical solution GA is used directly to form CM, AE uses, for the same purpose, AEPs formed in the evolutionary way.

In the experiments, three types of ANNs were used: FFANNs of constant architecture, FFANNs with Hebb self-organization, recurrent ANNs (RANN) with Hebb self-organization. The task of all ANNs created in the experiments was to control a set of cooperating predators whose common goal was to capture a fast moving prey behaving by a simple deterministic strategy. To create ANNs the following ANN encoding methods were used: AEPs consisting of binary encoded operations (AEPs^{01}, e.g. `CHGC0`), AEPs including `CHG_MEMORY` operations (AEPs$^{\#}$), AEPs containing both types of operations (AEPs$^{01\#}$), and the co-evolutionary version of CM. In the experiments, CM was exclusively used to form FFANNs of constant architecture.

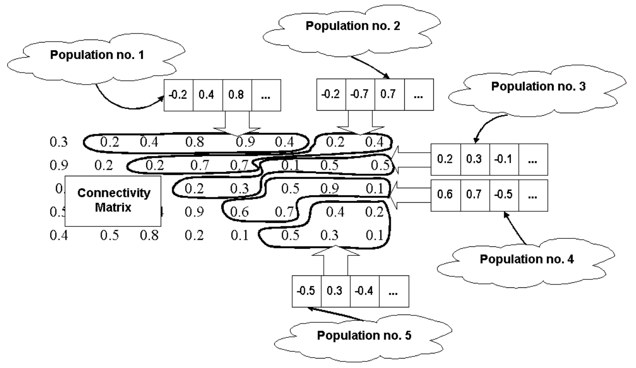


Fig. 8 Method for encoding CM used in experiments

Evolution of ANNs encoded in the form of CMs went on in somewhat different way from the one assumed in the classical solution. While in the classical approach, we deal with one population including matrices, in the solution applied in the experiments the whole CM was divided into parts and each part evolved in a separate population. Since CMs were used to encode FFANNs only fragments of CMs above the diagonal underwent evolution. In all the experiments with CMs, regardless of the size of ANNs, evolution of the matrices took always place in five populations, i.e. the matrices were always divided into five parts of more or less the same size (we decided to divide matrices into five parts because most of the most effective AEPs generated during the experiments also consisted of five components: four operations and a single sequence of data).

A.Environment

The predators and the prey lived in the common environment. We used 20x20 square without any obstacles but with two barriers located on the left and on the right side of the square to represent the environment. Both barriers caused the predators as well as the prey to move right or left only to the point at which they reached one of the barriers. Attempts to move further in the direction of the barrier ended up in failure. In order to ensure infinite space for the predators and the prey and for their struggles, we made the environment open at the bottom and at the top. This means that every attempt of movement beyond upper or lower border of the square caused the object, making such an attempt, to move to the opposite side of the environment. As a result, the simple strategy of predators, consisting in chasing the prey, did not work. In such a situation, the prey, in order to evade predators, could simply escape upwards or downwards.

B.Residents of the artificial world

In the experiments, three predators and one prey coexisted in the artificial environment. The predators controlled by ANN could select five actions: to move in North, South, West, East direction or to stand still. The length of the step made by each predator was 1, while the step made by the prey amounted either to 2 or to 1. In order to capture the prey the predators had to cooperate. Their speed was either two times lower or the same as the speed of the escaping prey so they could not simply chase the prey to capture it. We assumed that the prey could be captured if the distance between it and the nearest predator was lower than 2.

In the experiments, we assumed that the predators could see the whole environment. The predators based the decision which actions to select on the prey's relative location with reference to each of them. In order to perform the task ANN controlling the predators had to possess six inputs and three outputs. Outputs of ANN provided decisions to the predators whereas inputs informed them about prey's location in relation to each of them.

In the experiments, we used two types of prey – the simple prey and the advanced prey. The simple prey was controlled by a simple algorithm which forced it to move directly away from the nearest predator but solely in the situation when distance between it and the nearest predator was lower than or equal to 5. In the remaining cases, i.e. when no predator was closer to the prey than the assumed distance, the prey did not move. In the situation when the selected prey's action could cause hitting the barrier another move was chosen. Alternative move prevented from hitting the wall, and at the same time, it maximally increased the distance between the prey and the nearest predator. The prey, when was running away could select four actions: to move in North, South, West or East direction. Making decision, the advanced prey, unlike its simpler counterpart, always took into consideration the location of all predators that were situated close to it. Actions performed by the advanced prey always maximized the average distance between the prey and all predators that were close to it. Other aspects of behavior of the advanced prey, i.e. behavior near the barrier, behavior away from the predators and actions which the prey could perform in each step, were the same as in the case of the simple prey.

C. Parameters of evolutionary process

In the experiments, two types of GAs were used: canonical GA and eugenic algorithm [1],[14]. Canonical GA was used to process data, and fragments of CMs. In turn, eugenic algorithm was used to process operations. In the experiments, AEPs could posses maximum 12 operations. Initially every AEP contained one operation and one set of data from two different populations. Consecutive populations with operations were added every 5000 of co-evolutionary cycles if generated AEPs were not able to achieve progress in performance within this period. Populations including operations and data could be also replaced by newly created populations when the contribution of substituted population to created AEPs was considerably less than the contribution of the remaining populations. The same procedure could also be applied with regard to populations including fragments of CMs. The contribution of the population was measured as average fitness of individuals belonging to that population. The remaining values essential for the experiments are presented below:

- each population size: 20 individuals;
- number of co-evolutionary cycles for one fixed structure of ANN: 50 000 (in the case when even one satisfactory solution was not found during the assumed period all ANNs were expanded by one neuron and evolutionary process started again).

Parameters of Canonical GA:

- crossover probability: 0.7;
- per-bit mutation probability: 0.01;
- cut-splice probability: 0.1 (in the case of chromosomes-data).

Parameters of Eugenic Algorithm:

- selection noise: 0.01, 0.2;
- creation rate: 0.01, 0.2;
- restriction operator: on.

D. Evaluation process

In order to evaluate ANNs ten different scenarios were used. The tests were carried out in the following way. At first, each ANN was tested in the scenario no. 1. If the predators controlled by ANN could not capture the prey during an assumed period, the test was stopped and ANN received appropriate evaluation that depended on the distance between the prey and the nearest predator. However, if the predators grasped the prey, they were put to test according to next scenario. During the experiments, we assumed that the predators could perform 100 steps before the scenario was interrupted.

The scenarios used in the experiments differed in the initial position of the prey, in the length of step of the prey and in the type of the prey applied (simple or advanced). Consecutive scenarios were more and more difficult. At first, the predators had to capture the simple prey that was as fast as them. The predators, which passed the first exam, had to pit against the simple prey that was twice faster than the predators. In the next step, the speed of the prey was decreased once again. However, this time the predators had to face the advanced prey which took better decisions than its predecessor. In the last stage, the predators which coped with all earlier scenarios had to capture the advanced, fast prey. In all the scenarios starting positions for all three predators were the same. The predators always started from position (0,0). Below, described are all eight scenarios:

- Scenario no 1: simple prey (20,5), prey's step = 1;
- Scenario no 2: simple prey(10,8), prey's step = 1;
- Scenario no 3: simple prey (15,3), prey's step = 2;
- Scenario no 4: simple prey (0,10), prey's step = 2;
- Scenario no 5: advanced prey (16,0), prey's step = 1;
- Scenario no 6: advanced prey (2,15), prey's step = 1;
- Scenario no 7: advanced prey (10,19), prey's step = 2;
- Scenario no 8: advanced prey (4,10), prey's step = 2;
- Scenario no 9: advanced prey (10,10), prey's step = 2;

- Scenario no 10: advanced prey (20,0), prey's step = 2.

To evaluate ANNs the following fitness function was used:

$$f(ANN) = \sum_{i=1}^{n} f_i \tag{7}$$

$$f_i = \begin{cases} d_{\max} - \min_{p \in P} d\left(p, s_{100}^i\right) \text{ prey not captured} \\ \qquad \text{in } i^{\text{th}} \text{ scenario} \\ f_{captured} + \left(100 - m_i\right)/a \text{ prey captured} \\ \qquad \text{in } i^{\text{th}} \text{ scenario} \\ 0 \text{ prey not captured} \\ \qquad \text{in the previous scenario} \end{cases} \tag{8}$$

where

f_i–reward received in i^{th} scenario;

d_{max}– maximal distance between two points in applied environment;

s^i_{100}– the end state in i^{th} scenario;

$f_{captured}$–reward for grasping the prey in single scenario (in the experiments $f_{captured}$ amounted to 100);

m_i–the number of steps which the predators needed to capture the prey (m_i<100);

a – this value prevents the situation in which partial success would be better than success in all scenarios;

n – the number of scenarios.

```
Operations:
1 111#11# 1##1##1 0011##0
0 1##100# 1#0#011 00#0#1#
0 1##1010 11000#1 10##0#1
0 1#00011 1000#01 ##00101
1 101#1## 11##111 0010###
1 1#11010 #011##1 11#1#01
Data:
1010110 0011101 0100001 0101101 0011110 0010101
1111000 1010010 1101110 1111110 0100001 1011111
1111000 1110100 0010001 0000111 0011110
```

a)

```
-0.3 -0.06 0 -0.06 0 -0.06 0.47 -0.1 0 -0.06 0 -0.06 0 -0.06 0.66
-0.98 -0.06 0.7 0.66 -0.3 -0.5 -0.3 -0.3 -0.3 -0.3 0.47 -0.1 -0.46 0.52 -0.5
-0.3 0 -0.46 0 0 0 -0.1 0.66 0 -0.1 0 0 0 0 -0.17
0.7 0 -0.98 -0.17 0.88 0.73 0.47 0.52 0 0 -0.1 0.53 0.47 0.52 0.73
-0.3 -0.06 0 -0.06 0 -0.06 0.47 -0.1 0 -0.06 -0.76 -0.76 0 -0.06 0.66
-0.98 -0.06 0 -0.06 0 -0.06 -0.46 0.52 0 -0.06 0 -0.06 0 -0.06 -0.5
0.7 0 0.47 0 0 0 -0.1 0.73 0 0.52 0 0 0 0 -0.17
0.7 0 0 0 0 0 0.47 0.52 0 0 0 0 0 0 0.73
-0.2 0 0 0 0 0 0.47 -0.1 0 0 -0.76 -0.76 0 0 0.66
-0.98 0 0.7 0.66 -0.2 -0.5 -0.3 -0.3 -0.3 -0.3 0.47 -0.1 -0.46 0.52 -0.5
0.88 0 0.47 0 0 0 -0.1 -0.17 0 0.53 0 0 0 0 -0.17
0.7 0 -0.98 -0.17 0.88 0.73 0.47 0.52 0 0 -0.1 0.53 0.47 0.52 0.73
```

b)

Fig. 9 (a) Example of successful AEP$^{\#}$, (b) NDM generated by AEP presented in point (a)

E.Experimental results

30 evolutionary runs were performed for each ANN encoding method and for each type of ANN. The experiments showed that AEPs^{01} outperform other encoding methods. Most ANNs produced by means of AEPs^{01} were successful, i.e. they resulted in capturing the prey in all tested scenarios. ANNs generated by means of AEPs^{01} had fewer neurons than ANNs produced based on other methods. Unlike AEPs^{01} other methods often generated ANNs including maximal acceptable number of neurons. AEPs^{01} were the only method which was able to produce successful ANNs with Hebb self-organization. The remaining methods were only able to generate effective ANNs of constant architecture. The only method which did not produce any successful ANN was CM. Final results obtained in the experiments are presented in Table 1.

Table 1 Results of experiments (column 1 - type of ANN (encoding method); column 2 - average fitness (best fitness); column 3 - average connectivity in successful ANN (100% - fully connected ANN); column 4 - average number of neurons in successful ANN (minimal number of neurons); column 5 - average length of successful AEP, number of operations + number of data (shortest AEP); column 6 - average number of co-evolutionary cycles necessary to generate successful AEP (minimal number of co-evolutionary cycles))

(1)	(2)	(3)	(4)	(5)	(6)
FFANN (AEPs^{01})	1028.14 (1069.75)	84.7%	11.5 (9)	4.9 + 13.8 (2 + 21)	196834.3 (6614)
FFANN (AEPs$^{\#}$)	915.24 (1081.76)	67.3%	14.6 (12)	6.2 + 15.4 (3+14)	335692.4 (158808)
FFANN (AEPs$^{01\#}$)	998.98 (1088.65)	75.9%	13.2 (11)	5+13.3 (3+12)	286951.7 (108456)
FFANN (CM)	682,28 (743.48)				
FFANN Hebb (AEPs^{01})	836,79 (1055.66)	55.8%	14.7 (12)	5.8 + 14.5 (5 + 9)	347053.8 (179808)
FFANN Hebb (AEPs$^{\#}$)	480,41 (563.38)				
FFANN Hebb (AEPs$^{01\#}$)	546,1 (659.27)				
RANN Hebb (AEPs^{01})	728,07 (1065.33)	82.5%	14.5 (12)	5.7 + 15.7 (5 + 14)	339651.4 (183421)
RANN Hebb (AEPs$^{\#}$)	362,04 (441.32)				
RANN Hebb (AEPs$^{01\#}$)	489,59 (648.32)				

V.SUMMARY

The paper presents a new indirect ANN encoding method called Assembler Encoding. In AE, each ANN is represented in the form of a program called Assembler Encoding Program. AEP is composed of operations and data arranged in the linear way. The task of AEP is to create and to fill in Network Definition Matrix with values. To do this, AEP uses the operations. The operations are run in turn. When working the operations use data located at the end of AEP. Once the last operation finishes its work the process of creating NDM is completed. NDM is then transformed into ANN.

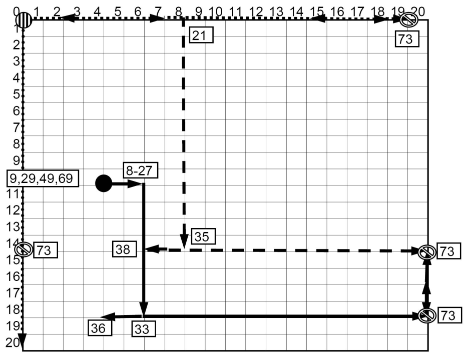


Fig. 10 Example behavior of predators and prey in scenario no. 8 . Circles indicate initial positions of predators and prey (black circle –prey, circle with vertical stripes –predator no. 1, circle with horizontal stripes –predator no. 2), round symbols with diagonal lines denote final positions, arrowed lines indicate directions of movement (solid line –prey, dashed line –predator no. 1, dotted line –predator no. 2) whereas black boxes determine time of occurrence of individuals in a given place.

In order to test AE we made use of it to solve the predator-prey problem. During the tests, the task of AEPs was to generate ANNs controlling a set of cooperating predators whose common goal was to capture a fast moving prey. In the experiments, three types of AEPs were tested, i.e. AEPs^{01} using binary encoded operations, AEPs$^{\#}$ using LCS-classifier like operations and AEPs$^{01\#}$ using operations of both types. To compare AE with another ANN encoding method, in the experiments, a co-evolutionary version of classical Miller et al. CM was also applied. Generally, the experiments showed that AE is able to create simple ANNs. The best ANNs were produced by means of AEPs^{01}. AEPs$^{01\#}$ and AEPs$^{\#}$ turned out to be somewhat worse solutions than AEPs^{01}. The worst ANNs were produced by means of CM.

References

[1] M. Alden, A. Van Kesteren and R. Miikkulainen, *Eugenic Evolution Utilizing a Domain Model*, In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002), (San Francisco, CA, Morgan Kaufmann, 2002).

[2] M. V. Butz, *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction*, University of Illinois, IlliGAL Report No. 2004034, 2004.

[3] D. Curran and C. O'Riordan, *Applying Evolutionary Computation to Designing Networks: A Study of the State of the Art*, National University of Ireland, technical report NUIG-IT-111002, 2002.

[4] D. Floreano and J. Urzelai, *Evolutionary robots with online self-organization and behavioral fitness*, Neural Networks, Vol.13, 2000, PP. 431-443.

[5] F. Gruau, *Neural network Synthesis Using Cellular Encoding And The Genetic Algorithm,*. PhD Thesis, Ecole Normale Superieure de Lyon, 1994.

[6] H. Kitano, *Designing neural networks using genetic algorithms with graph generation system*, Complex Systems, Vol. 4, 1990, pp. 461-476.

[7] B. Kusumoputro, *Mixture Odor Classification using Fuzzy Neural Network and Its Optimization through Genetic Algorithm*, WSEAS Transactions on Systems, Issue 2, Volume 3, 2004, pp.426-431.

[8] G. F. Miller, P. M. Todd and S. U. Hegde, *Designing Neural Networks Using Genetic Algorithms*, Proceedings of the Third International Conference on Genetic Algorithms, 1989, pp. 379-384.

[9] D. E. Moriarty and R. Miikkulainen, *Forming Neural Networks Through Efficient and Adaptive Coevolution*. Evolutionary Computation, 5(4), 1998, pp. 373-399.

[10] S. Nolfi and D. Parisi, *Growing neural networks*, In C. G. Langton, ed., Artificial Life III, Addison-Wesley, 1992. Available: http://citeseer.ist.psu.edu

[11] P. Nordin, W. Banzhaf and F. Francone, *Efficient Evolution of Machine Code for {CISC} Architectures using Blocks and Homologous Crossover*, Advances in Genetic Programming III, L. Spector and W. Langdon and U. O'Reilly and P. Angeline, 1999, pp. 275-299.

[12] M. A. Potter and K. A. De Jong, *Cooperative coevolution: An architecture for evolving coadapted subcomponents*. Evolutionary Computation, 8(1), 2000, pp. 1-29.

[13] T. Praczyk, , "Using assembler encoding to solve inverted pendulum problem," *Computing and Informatics.*, to be published

[14] J. W. Prior, *Eugenic Evolution for Combinatorial Optimization*, Master's thesis, The University of Texas at Austin. TR AI98-268, 1998

[15] M. Rocha, P. Cortez and J. Neves, *Evolutionary Neural Network Learning Algorithms for Changing Environments*, WSEAS Transactions on Systems, Issue 2, Volume 3, 2004, pp.596-601.

[16] R. Sundararajan, A. K. Pal, *A Conservative Approach to Perceptron Learning*, WSEAS Transactions on Systems, Issue 2, Volume 3, 2004, pp.375-380.