Diverse Neural Architectures in Assembler Encoding

Tomasz Praczyk

Institute of Naval Weapon, Polish Naval Academy
81-103 Gdynia, ul. Śmidowicza 69
E-mail: t.praczyk@amw.gdynia.pl

Received: 27 January 2014; revised: 22 March 2014; accepted: 22 March 2014; published online: 26 March 2014

Abstract: The paper presents a neuro-evolutionary method called Assembler Encoding (AE) and proposes its several modifications. The main goal of the modifications is to ensure AE greater freedom in generating diverse neural architectures. To compare the modifications with each other and with the original method the particular case of the predator-prey problem has been discussed.

Key words: evolutionary neural networks

I. INTRODUCTION

In recent years, an increasing interest has been noticed in two domains of the artificial intelligence, i.e. in evolutionary computation and artificial neural networks (ANN). Evolutionary techniques are usually used as global optimization methods, while ANNs are applied in such problems as approximation, identification, feature extraction, and reinforcement learning. Successes in both domains have promoted a new combined domain called neuro-evolution (NE) which uses the evolutionary approach to search for effective ANNs. The evolution of ANNs uses many evolutionary mechanisms of genetic evolution. This means that every network is represented in the form of a genotype, i.e. a chromosome or a set of chromosomes. The chromosomes include all the information necessary to create an ANN. Chromosomes representing different ANNs are concentrated in one or more populations. During evolution, the chromosomes are replaced with their genetically modified offspring arisen as a result of executing various genetic operators on parental chromosomes. Using the rule whereby the genetic material of better chromosomes, i.e. chromosomes encoding better ANNs, has a greater chance to survive than the genetic material of worse chromosomes, leads to better and better ANNs generated within evolution.

There are numerous NE methods (e.g. [2, 7, 8, 11, 12, 13, 14, 25]). In principle, all the existing methods can be divided into two main classes, i.e. direct and indirect methods. As for the direct ones, all the information necessary to create an ANN (e.g. weights, number of neurons, number of layers) is directly stored in chromosomes. This way, to encode complex networks complex chromosomes are necessary, which is the main drawback of the direct methods. As regards the indirect ones, we deal with chromosomes which are recipes how to create a network. Such methods can encode complex neural architectures by means of relatively short chromosomes.

The paper presents an indirect NE method called Assembler Encoding (AE). AE originates from the cellular [7] and edge encoding [11], although, it also has features common with Linear Genetic Programming presented, among other things, in [9, 15]. In AE, an ANN is represented in the form of Assembler Encoding Program (AEP) whose structure is similar to that of the structure of a simple assembler program. The AEP is composed of two parts, i.e. a part including operations and a part including data. The task of each AEP is to create a Network Definition Matrix (NDM) which includes all the information necessary to create a network. During operation, the AEP runs individual operations which gradually form NDM. When working, each operation can use data located at the end of AEP. In AE, the process of

ANN construction consists of three stages (Fig. 1). First, a genetic algorithm (GA) is used to form a population of AEPs, next, each AEP creates and fills up its own NDM, and finally, the matrices are transformed into ANNs.

AE is a general NE method which can be used to solve any problem requiring ANNs (or only matrices). Even though a target application of AE is to form complex ANNs, the method proved that it can successfully be used also to problems solvable with simpler networks [20, 23]. To date, AE has been applied to solve three different testing problems, i.e. the optimization problem [18], the predator–prey problem [18, 19, 22], and the pole balancing problem [20, 23]. In all the tests, the method demonstrated fairly good effectiveness successfully competing with such state-of-the-art NE and reinforcement learning methods as: Q-learning [1], Adaptive Heuristic Critic, Genitor [29], Symbiotic Adaptive Neuro–Evolution (SANE) [13], Connectivity Matrix [12], Neuro Co–Evolution (NCoE) [22], Fuzzy Expert System [24], Neuro–Evolution of Augmenting Topologies (NEAT) [25, 26], Enforced Subpopulations (ESP) [5], and Cooperative Synapses Neuro–Evolution (CoSyNE) [6].

To be able to solve different problems, AE has to be a complete method, i.e. it should be capable of producing each possible ANN. Since, in AE, ANNs are represented in the form of NDMs, the ability to produce any ANN can also be viewed as the potential to form any matrix. In AE, the easiest method to produce any matrix of an assumed size is to apply an AEP with a single operation whose task is to cover all the elements from NDM with data. Since in the case of such a program the content of NDM only depends on the data, by changing them in a variety of ways any matrix can be produced (any matrix for which there exists a counterpart in a genotype space). However the global range of the program above, i.e. access to all elements in NDM, has one serious consequence. Since the zero value encoded in the data is only one out of many possibilities, copying them directly to all elements of NDM predominantly leads to matrices with no zeros. Further, since individual elements of NDM, in most cases, define weights of interneuron connections, the program above mainly produces fully–connected ANNs with a maximum number of neurons.

To increase the ability of AE to form other architectures than the one mentioned above, the current version of AE uses operations whose operational range depends on their parameters. The operations can work both on larger and smaller areas of NDM. A single operation may update one, two, or three elements of the matrix but also all of them. Such solution enables AEPs to form NDMs with many untouched fields, which is a necessary condition for ANNs with diverse topologies to evolve (all unmodified elements of NDM are equal to zero). The ability to produce different network topologies allows AE to adjust both parameters and topologies of ANNs to a problem. It is a very important feature of AE which has been confirmed experimentally.

However, the tests which on the one hand showed that AE has a great ease in generating various effective network architectures, on the other hand also revealed that most NDMs in spite of differences in construction share some common features. Generally, it appeared that most NDMs are zero matrices with oblong (horizontal or vertical) continuous clusters including values different from zero. Matrices with loosely scattered content turned out to be very rare.

As before, the cause is construction of operations used in AE. Since each of them modifies elements of NDM which create one cohesive cluster (e.g. a fragment of a column), matrices with many isolated values different from zero have generally a very little chance to be generated. Since difficulties with creating such matrices may limit applicability of AE or reduce its effectiveness in solving some problems, in the paper, a number of modifications to the method are proposed.

To increase diversity of NDMs and ANNs, the mentioned modifications apply two different solutions. The first solution is an operation with the potential to operate in separated areas of NDM scattered over all the matrix. As in the case of operations used so far, a new operation copies selected data into NDM. The second solution introduces a new class of operations responsible exclusively for defining topology of ANNs. In this case, the information how to connect individual neurons is included in one operation whereas connection weights between neurons are fixed by other operations contained in AEP.

To prove that the ability to produce matrices with loosely scattered content is a key element in AE and to estimate completeness of the modifications to AE whose main feature which differentiates them from the original method is greater ease in evolving such matrices, two types of experiments were carried out. First, all the modifications were used to evolve neuro-controllers (NCs) for a team of Autonomous Underwater Vehicles (AUVs) whose common goal was to capture an escaping AUV behaving by a simple deterministic strategy (predator–prey problem). The first attempts to combine AE with AUVs are reported in [22]. The experiments reported in the current paper were performed in almost the same conditions as those described in [22]. The only difference was application of an AUV–prey with a more advanced escaping strategy and shortening time which the AUVs–predators had to capture the AUV–prey. The additional elements increased the complexity of the problem solved by ANNs comparing to the previous research.

In the second phase of the experiments, the ability of the methods to form diverse neural architectures were estimated. However, since AE represents ANNs in the form of matrices, the task of each method in this phase was not to produce ANNs but matrices of varied construction.

In all the experiments, the original variant of AE was used as the only point of reference for all the modifications, and there was not any comparison with other methods. As mentioned above, AE has already been compared numerous times with many different methods, hence it seems to be currently a well tested, sensible point of reference, particularly in the predator–prey problem which has been widely used so far as a test bed for AE. Moreover, the main purpose of the first phase of the experiments was not to test how effective ANNs can be produced by means of the modifications proposed in the paper but to prove that matrices with loosely scattered content and the capacity to evolve them is a key factor in AE. To this end, the modifications were compared with their mother method which, as mentioned above, has difficulties with evolving such matrices, so comparisons with other methods were unnecessary in this instance. In turn, in the second phase, the goal was only to estimate completeness of the modifications. The test problem applied in this case was adjusted to the purpose of the experiments and matrix–representation of all the compared methods. Because all the tests were carried out on matrices, the problem to be solved was relatively simple and, in consequence, highly inappropriate for any comparisons with other methods.

The paper is organized as follows: section 2 is a presentation of compared methods, section 3 and 4 are reports on the experiments, and section 5 is a summary.

II. ASSEMBLER ENCODING AND ITS MODIFICATIONS

This section presents Assembler Encoding and its six modifications. The modifications as well as the original variant of AE share three common features. First, they represent ANNs in the form of NDMs. Second, to produce NDMs, each method uses AEPs which evolve according to Cooperative Co–Evolutionary GA (CCEGA) [16, 17]. Third, all the methods can build regular neural architectures with replication of different units. To this end, each method makes it possible to use the same fragments of AEPs many times.

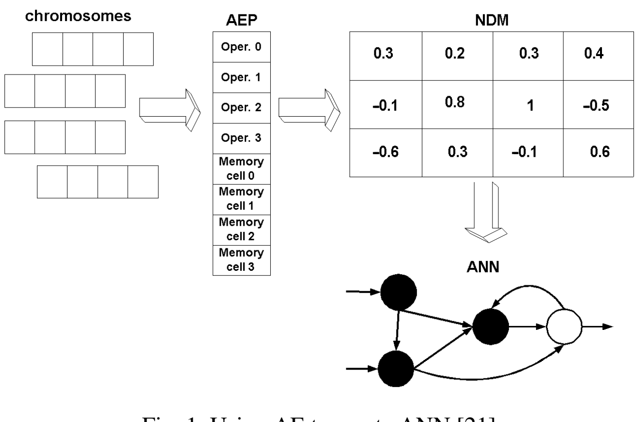
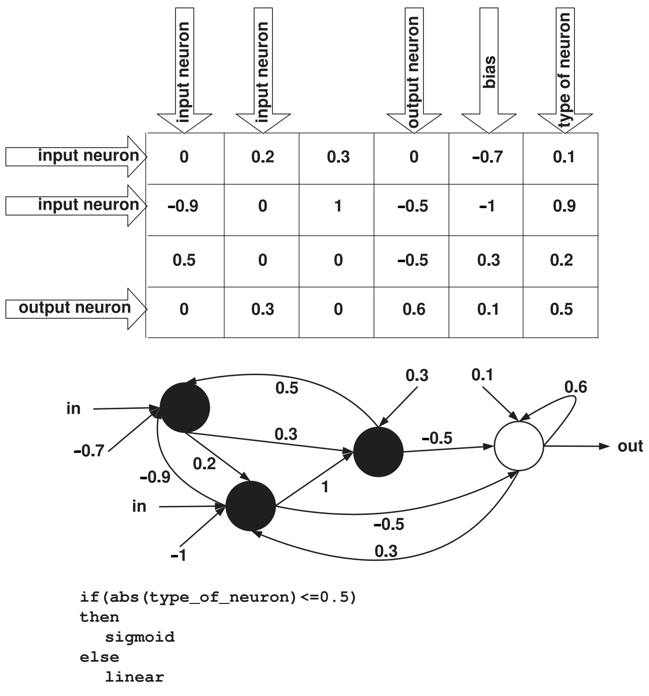


Fig. 1. Using AE to create ANN [21]

NDM is, in principle, a real valued Connectivity Matrix defined in [12]. It stores all the information necessary to construct a network. This information is included both in the size and individual elements of the matrix scaled always to the range $\langle -1, 1\rangle$. The size of NDM determines the maximum number of neurons in ANN whereas individual elements of the matrix define weights of interneuron connections, i.e. component$_{i,j}$ determines a link from neuron i to neuron j. Apart from the basic part, NDM also contains additional columns that describe parameters of neurons, e.g. type of neuron (e.g. sigmoid, radial, linear), and bias (Fig. 2).



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

Fig. 2. NDM as Connectivity Matrix [20]

AEP is an ordered set of predefined operations and optionally data. The task of each AEP is to create NDM and fill in it with values. To this end, the operations are executed in turn, one after another. Each operation modifies some fragment of NDM dependent on type and parameters of the operation (initially, all elements in NDM are set to 0; this means that there are not any connections between neurons). Each AEP is shaped in the evolutionary way. The evolution decides about order of the operations (and optionally data) and about values of their parameters. Implementations of the operations do not evolve, they are defined beforehand.

The evolution of AEPs proceeds according to CCEGA proposed by Potter and De Jong [16, 17]. In CCEGA, each part of a solution evolves in a separate population. To form a complete solution, selected representatives of each population are combined together (usually, each individual from one population is combined with the best individuals from the remaining populations).

Application of this evolutionary scheme in relation to AEPs consists in separate evolution of operations located at various positions in the programs. The same applies to data which also evolve in their own population. For example, AEP consisting of n operations and a sequence of data evolves in n populations with operations and one population with data (Fig. 3). During the evolution, AEPs expand gradually. Initially, all AEPs include some initial number of operations and optionally a sequence of data. When the evolution stagnates, i.e. lack of progress in fitness is observed over some period, a set of populations containing the operations is enlarged by one population. It extends all AEPs by one operation [19].

In each single population, the evolution proceeds according to Canonical GA [4]. Individuals from each population (either the operations or the data) are encoded in the form of binary strings. Each chromosome-operation includes binary encoded parameters and optionally code of the operation (e.g. 01000|11000|01000|00000|00100 represents the following operation: CHGC0|-1|1|0|2, see Fig. 4). The code is only used when more than one type of operation can evolve in a single population. Chromosomes–data are strings including binary encoded data.

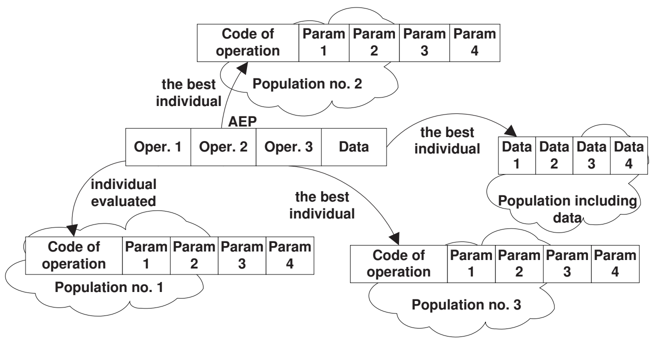


Fig. 3. Evolution in AE for $n = 3$ [21]

II. 1. Assembler Encoding

In the original variant of AE, AEPs are executed according to the order fixed by the evolution, there are no predefined loops or jumps. All operations have maximally four parameters and when working they can use shared data located at the end of each AEP. Generally, in AE, there are three groups of operations. The main task of operations belonging to the first group is to modify the content of NDM. In principle, the operations from this group can work in any manner. AE does not impose the way of constructing the operations and their modus operandi. The operations can be adjusted to a problem, e.g. they can form exclusively ANNs with a layered architecture, or they can be completely general. An example operation used in AE is CHGC0 presented in Fig. 4.

The second group of operations contains only a jump operation denoted as JMP. The jump makes it possible to repeatedly use the same code of AEP in different places of NDM. This way, the same neural structures may appear many times in a resultant ANN. However, since many experiments with AE showed a little effectiveness of AEPs including JMP, the operation is currently used very rarely.

The last group of operations used in AE are operations whose task is to change the size of NDM. In AE, an initial size of the matrix is encoded in a chromosome with data, then, each AEP has the potential to modify the size by using operations ADDN and DELN. ADDN adds new rows and columns to NDM which corresponds to adding new neurons to ANN, neurons unconnected with the rest of the network. Addition of new neurons does not destroy connections established earlier. The task of DELN is to remove a single neuron from ANN which practically takes place through removing the corresponding row and column from NDM [19].

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

Fig. 4. CHGC0 operation – it modifies elements of NDM located in a column indicated by p_0. Index of the first element to be updated is located in p_1 whereas the number of elements to be updated is stored in p_2. To modify elements of NDM, CHGC0 uses data from AEP. Index to the first element of data used by CHGC0 is stored in p_3 (NDM[i,j] is an element of NDM, where i=1..NDM.width, j=1..NDM.height, MaxValue is a scaling value which scales all elements in NDM to the range $< -1, 1 >$, D[i] is ith element of data, D.length is the length of data)

During experiments reported further, a general variant of A0 with operations specified in Table 1 were used. Since, in the experiments, the maximum number of neurons in ANNs was assumed to be constant, operations ADDN and DELN were unnecessary and thereby they were omitted.

In AE, different types of operations can evolve in a single population. Since all the operations have maximally four parameters, to represent them, constant length chromosomes including five segments with binary genes are used. The first segment determines a code of operation while the remaining segments contain a binary representation of its parameters. Chromosomes encoding data are vectors including binary encoded integers, each of which encodes a single element of data. The chromosomes-data can change the length during the evolutionary process. Example AEP produced with AE and its encoded form are presented in Fig. 5.

Tab. 1. List of operations used in experiments

Operation	Description
`CHGC0` (`CHGR0`)	update of certain number of elements in column (row), new values for column (row) elements are in data part of AEP
`CHGC1` (`CHGR1`)	like `CHGC0`, but new value for column (row) elements, same for all elements, is located in parameter of operation
`CHGM0`	update of certain number of elements in continuous rectangle-shaped fragment of NDM, new values for elements are in data part of AEP
`CHGM1`	like `CHGM0`, but new value for each element is sum of its current value and parameter of operation
`CHGFF`	update of all elements above diagonal of NDM, new values for elements are in data part of AEP

```
CHGM0|59|-11|53|37
CHGM1|48|15|30|-3
CHGM1|-32|-20|29|7
Data:-34|-8|-46|46|58|-57|-4|-48|-1|52|-12|
-54|-7|40|35|23|-47|1|47|38|32|-46|55|-22|17
```

(a)

```
Operations:
0011000 0110111 1110100 0101011 0101001
1101110 0000011 0111100 0011110 1110000
0101000 1000001 1001010 0101110 0111000
Data:
1010001 1000100 1011101 0011101 0010111
1100111 1001000 1000011 1100000 0001011
1001100 1011011 1111000 0000101 0110001
0111010 1111101 0100000 0111101 0011001
0000001 1011101 0111011 1011010 0100010
```

(b)

Fig. 5. Phenotypic (a) and genotypic (b) representation of an example AEP in AE

II. 2. Modification no. 1

The main difference between AE and modification no. 1 (M1) concerns operations used in AEPs. The remaining elements of both methods are the same. In AE, AEPs can be built with all the seven operations specified in Table 1. The characteristic of all the operations is that they always modify some continuous fragment in NDM. Modification of many isolated fragments of the matrix is impossible in this case.

Unlike AE, M1 uses only one type of operation, say, `CHGM2`. Construction of the operation enables it, however, to modify separate areas of NDM. To accomplish such effect, `CHGM2` introduces successive data into NDM leaving empty spaces time to time. Segments of NDM filled in with data and segments remained intact occur in the matrix alternately. The length of each segment depends on parameters of the operation, the first parameter determines the length of all the segments with data whereas the second parameter indicates the length of all the empty segments. The remaining two parameters determine the number of iterations (the number of successive empty segments) and address of an element in NDM where the operation starts to work.

In M1, like in AE, AEPs evolve in one population with data and at least in one population containing `CHGM2`s. Since `CHGM2` uses a total of four parameters it is encoded in the form of a constant length chromosome consisting of four binary segments, one per each parameter. The code of the operation is unnecessary in this instance because each population includes exclusively `CHGM2`s.

II. 3. Modification no. 2

In modification no. 2 (M2), the main operation responsible for shaping NDM is `CHGM3`. Each AEP includes one or more `CHGM3`s situated at the very start of the program. Each `CHGM3` modifies consecutive elements of NDM, row by row, using data for that purpose. The same data can be used many times by a single `CHGM3`. The operation has four parameters. The first two parameters indicate an element in NDM where the operation starts to work (indexes of column and row), the third parameter determines the number of elements to be modified whereas the fourth parameter is a pointer to the first element of data used by the operation.

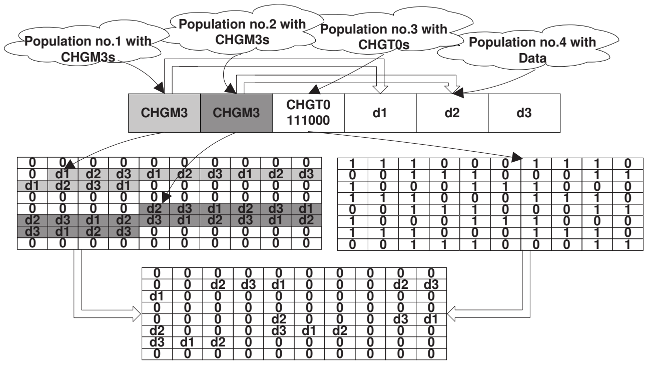

Population no.1 with CHGM3s
Population no.2 with CHGM3s
Population no.3 with CHGT0s
Population no.4 with Data
CHGM3
CHGM3
CHGT0 111000
d1
d2
d3

Fig. 6. Example NDM produced with M2

The last operation in each AEP is CHGT0 whose task is to determine the final architecture of ANN. This operation does not fix weights of interneuron connections as CHGM3 but only indicates which connections exist in the network and which do not. CHGM3s determine an initial architecture of ANN which is then reduced by means of a single CHGT0. To this end, CHGT0 reads all the elements from NDM and eliminates connections which should not appear in a resultant ANN. Information of which connections should be eliminated and which whould not is stored in an input whose length can be various for different operations. When the number of bits in the vector is lower than the number of all items in NDM, information included in the vector is used again, until the whole ANN architecture is defined. Like shared data for different operations, such a solution enables AE to repeatedly use the same information memorized in AEPs and, in consequence, to represent complex ANNs by means of simple programs.

In M2, AEPs evolve at least in three populations. One population includes CHGM3s, the next one CHGT0s, and the last one sequences of data. The number of populations with CHGM3s automatically grows when it is necessary to add the next CHGM3 to all AEPs. Since all the populations with operations are homogeneous in terms of operations they include, the only component of the operations which has to be encoded in chromosomes is a list of their parameters. As a result, each CHGM3 is encoded in the form of a constant length chromosome consisting of four binary segments, one per each parameter. In turn, CHGT0 is a variable length binary string in which each "1" means connection whereas "0" no connection.

II. 4. Modification no. 3

Modification no. 3 (M3) is in principle modification no. 2 without shared data. Lack of the data in AEPs results in three changes of M3 in relation to M2. First, instead of four parameters CHGM3, M3 uses CHGM4 with a variable list of parameters which instead of data are transferred into NDM. A modus operandi of CHGM4 is exactly the same as CHGM3. Second, AEPs evolve at least in one population with CHGM4 and in one population with CHGT0. There is no population with data. Third, whereas CHGM3s evolve as constant length binary strings, CHGM4s, at the genotypic level, are represented in the form of variable length chromosomes.

II. 5. Modification no. 4

Modification no. 4 (M4) uses two operations, i.e. CHGM5 and CHGT0. Like CHGM4, CHGM5 copies input parameters directly into NDM. The order of modified elements in NDM is the same as in CHGM4 and CHGM3. However, in CHGM5 there is no parameter which decides about the number of copied parameters, and CHGM5 simply copies all its parameters. Moreover, in CHGM5, there are no parameters which indicate the element of NDM where the operation should start to work, this element is pointed out by AEP.

Construction of AEP is different from all the previous variants of AE. Like in M3, each AEP consists of two parts. The first part includes a sequence of CHGM5s whereas the second part is a single CHGT0. In this case, however, the first part is performed in a loop. In M4, CHGM5s work like relay runners, i.e. the first CHGM5 starts to work in an initial fragment of NDM whereas each subsequent operation continues work of its predecessor in a place up to which it came. The whole sequence of CHGM5s is run in a loop up to the point when all the elements in NDM have an assigned value. Like in M3, the last operation in AEP is CHGT0 which determines the final architecture of ANN.

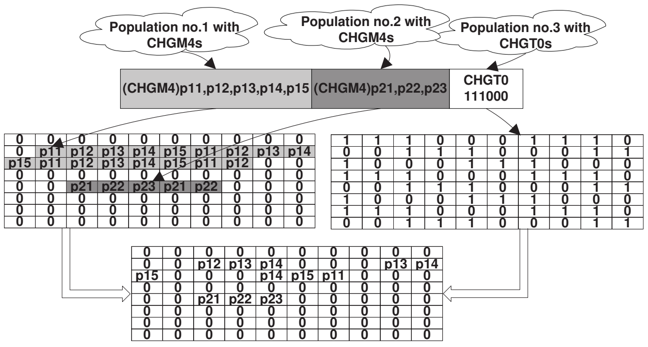

Population no.1 with CHGM4s
Population no.2 with CHGM4s
Population no.3 with CHGT0s
(CHGM4)p11,p12,p13,p14,p15
(CHGM4)p21,p22,p23
CHGT0 111000

Fig. 7. Example NDM produced with M3

In M4, AEPs evolve in almost identical way as in M3. The only difference between both modifications concerns encoded forms of CHGM4 and CHGM5. CHGM4, in addition to parameters copied into NDM, needs three other parameters which define its area of activity in NDM, while CHGM5 does not need such information. It copies all the parameters into NDM, starting from a point where its predecessor left off. In consequence, chromosomes encoding CHGM5 are shorter by three binary segments than chromosomes encoding CHGM4.

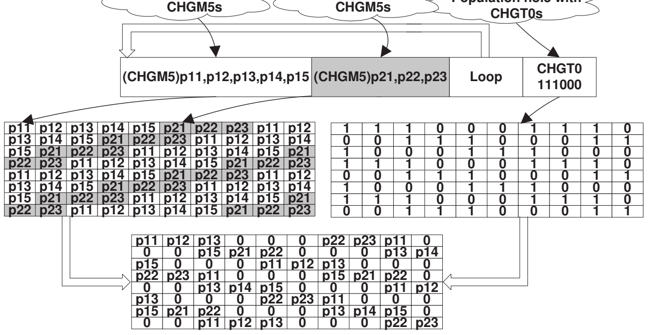

Population no.1 with CHGM5s
Population no.2 with CHGM5s
Population no.3 with CHGT0s
(CHGM5)p11,p12,p13,p14,p15
(CHGM5)p21,p22,p23
Loop
CHGT0 111000

Fig. 8. Example NDM produced with M4

II. 6. Modification no. 5

This modification (M5) is a counterpart of M3 in which roles of CHGM4 and CHGT0 are swapped. M5 uses two operations, i.e. CHGT1 and CHGM6. A single CHGM6 is always located at the very start of each AEP and its role is to fill in all NDM with values. To this end, the operation copies input parameters directly into the matrix, row by row. Transfer of the parameters into the matrix is repeated up to the point when all items in NDM have an assigned value.

After CHGM6, a sequence of CHGT1 starts. Each CHGT1 has three parameters determining area of activity of the operation (the address of the first element and the number of consecutive elements in NDM analyzed by the operation) and one binary vector indicating which connections should exist in ANN and which should not. CHGM6 produces NDM representing a fully–connected recurrent ANN whose connectivity is then reduced by successive CHGT1s.

In M5, AEPs evolve in one population with CHGM6s and at least in one population with CHGT1s. At the genotypic level, both operations are represented as varied length binary strings.

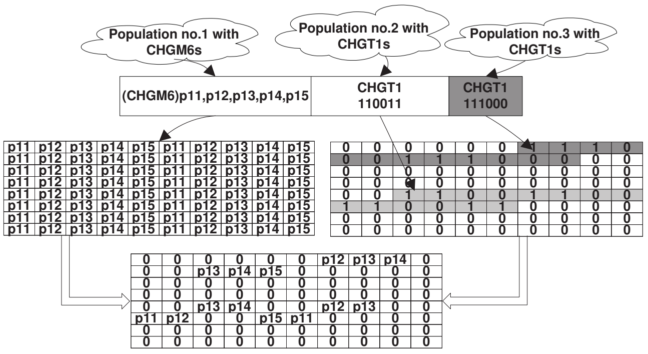

Population no.1 with CHGM6s
Population no.2 with CHGT1s
Population no.3 with CHGT1s
(CHGM6)p11,p12,p13,p14,p15
CHGT1 110011
CHGT1 111000

Fig. 9. Example NDM produced with M5

II. 7. Modification no. 6

Modification no. 6 (M6) is a combination of M4 and M5. It uses two operations, i.e. CHGT2 and CHGM6. Like in M5, each CHGM6 is situated at the very start of each AEP and its task is to fill in all NDM with values. In the further part of AEP occurs a sequence of CHGT2s that is looped like in M4. Successive CHGT2s establish or remove connections in consecutive fragments of NDM starting work from a point where predecessors left off. Each CHGT2 operates in a fragment of NDM of the length equal to the length of its input vector. The sequence of CHGT2 is repeated until the last element in NDM is reached.

Evolution of AEPs in M6 proceeds like in M5. The only difference is that chromosomes encoding CHGT2s do not contain first three binary segments indicating area of activity of the operation. Since this area is determined by the number of bits in the input vector and by predecessors of each operation, the information contained in these segments is unnecessary to produce CHGT2.

III. EXPERIMENTS ON THE PREDATOR–PREY PROBLEM

The main goal of the experiments on the predator–prey problem was to verify the following hypothesis. Generally, there are considerably more neural architectures which can be represented in the form of NDMs with a distributed content than other architectures (for a given maximal topology of an ANN and NDM, hence we can create many other ”lighter“ topologies by eliminating some elements from NDM), and there are also more effective neural architectures represented in the form of such matrices. In consequence, the ability to produce the above matrices is crucial for effectiveness of each method which uses a matrix representation of ANNs – methods which have such ability are more effective than other methods.

In the beginning of the experiments, each method presented in the previous section was tuned to the problem. To this end, each of them was run many times for different parameter settings. For one setting thirty evolutionary runs were carried out. Results obtained for the best settings were then used to compare the methods. The comparison was made based on two criteria. First, a learning ability, i.e. the ability to evolve effective neural solutions, was tested. To measure this ability, the average fitness of ANNs being the final effect of each evolutionary run was used. Second, ANNs were also analyzed in terms of their capability to generalize knowledge acquired during the evolutionary process. To this end, each ANN was tested on tasks which were not presented to them before.

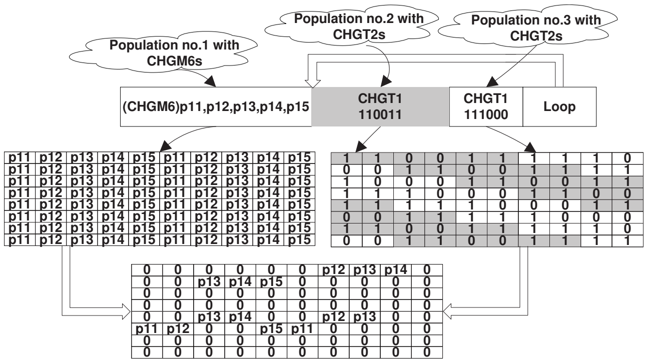

Population no.1 with CHGM6s
Population no.2 with CHGT2s
Population no.3 with CHGT2s
(CHGM6)p11,p12,p13,p14,p15
CHGT1 110011
CHGT1 111000
Loop

Fig. 10. Example NDM produced with M6

The experiments were carried out in simulation and in the configuration with one prey and three chasing predators. Both the predators and the prey were implemented as Remotely Operated Vehicle (ROV) ”Ukwial“ [10] (Fig. 12; the vehicle with a control system become Autonomous Underwater Vehicle – AUV). The behavior of all the vehicles was simulated by means of a discrete time model defined in [22].

In the experiments, the predators and the prey lived in a common artificial environment. To represent the environment, a square of size 100×100 meters was used (see

Fig. 11, to simplify calculations, both the predators and the prey moved on a horizontal surface under the water). The environment did not contain any obstacles. In order to ensure infinite space for the predators and the prey and for their struggles, the environment was open at each side. Thus, every attempt to move beyond upper, lower, right or left border of the square caused the object making such an attempt to move to the opposite side of the environment.

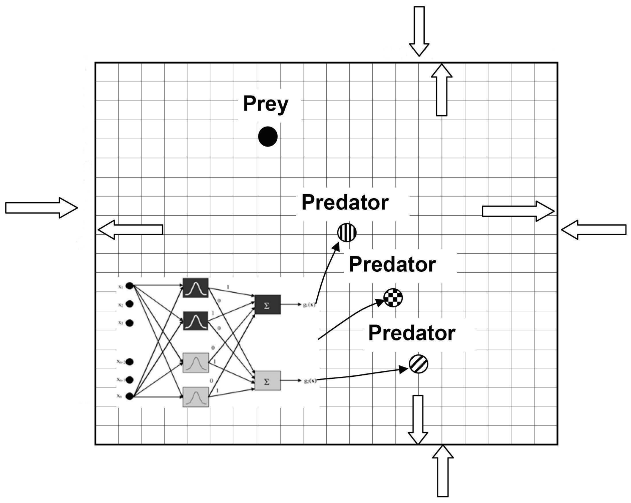


Fig. 11. Artificial world for predators and prey [19]

The predators were controlled by a single neuro-controller (NC) whose task was to determine a movement direction for each of them. At each time step, NC decided about change of the current course of each vehicle. The course could be changed by: 0, 5, 10, . . . , 355 degrees (it is necessary to note that decisions of NC determined only a final state of the vehicles which they ultimately should reach, a real course after the maneuver and duration of the complete maneuver depended on current parameters of each vehicle). The speed of the predators was constant during the tests and amounted to 0.5 m/s.

In the experiments, two types of prey were used, i.e. a simple and an advanced prey. The strategy of the simple prey forced it to stand still when no predator was closer to it than its range of vision (the range of vision of the prey amounted to 50 meters) and to move directly away from the nearest predator otherwise. In contrast to the simple prey, the advanced one always took into account all visible predators. As before, it started to move only when some predators were noticed. To determine a direction of a next move, the first activity of the advanced prey was to calculate a single position representing all predators being in its close proximity. The closer the predator was to the prey, the greater its influence on the calculated position was. In the following step, the "common" position of the predators was treated as the position of the closest (virtual) predator and the strategy of the simple prey was used thereafter.

When moving each prey could select the same actions as the predators. The speed of the preys also amounted to 0.5 m/s. Since the speed of the predators was the same as the speed of the escaping prey, they could not simply chase it to grasp it. We assumed that the prey was captured if the distance between it and the nearest predator was shorter than 10 meters.

In all the experiments, NCs had six inputs and three outputs. The number of outputs corresponded to the number of predators. In turn, the number of inputs was twice the number of predators. Each output gave commands to one predator. In turn, each input informed about vertical or horizontal distance between the prey and one of the predators.

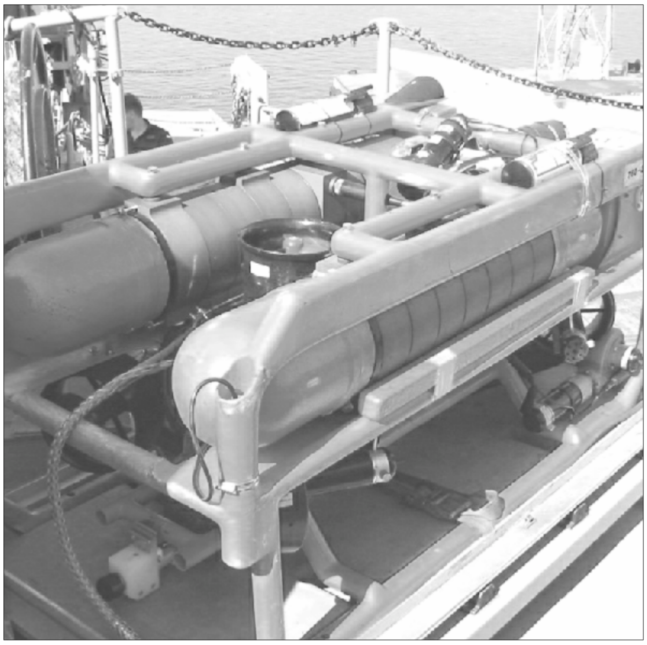


Fig. 12. Vehicle ”Ukwial“ [22]

III. 1. Evaluation of NCs

In order to evaluate NCs, ninety different testing scenarios were produced. The first thirty scenarios were used to learn NCs. The remaining ones were applied in a generalization phase to evaluate prepared NCs in terms of their capability to generalize knowledge acquired during the learning phase. Individual scenarios differed in the initial position of the prey, the number of steps the predators could make to capture the prey, and the type of the prey (simple or advanced). Consecutive scenarios were more and more difficult. Initially, the predators had to capture the simple prey and they could make 40 steps, which means 40 decisions of NCs (400 seconds for a chase, one step took 10 seconds). The predators which passed the first exam, had to do the same but in 30 steps. In the next scenario, the maximum number of steps which the predators could make to capture the prey was decreased once again, this time to 20 steps. The predators which captured the prey in all the previous scenarios had to face the advanced prey. As before, they had to perform their task first in 40, then in 30, and finally in 20 steps. In all the scenarios, starting positions of all the three predators were the same and situated in position (0, 0). All the scenarios are described in Table 2.

Tab. 2. Description of scenarios used in the experiments

No. of scenario	Max no. of steps	Type of prey	Initial positions
1-5	40	simple	Fig. 14(a) (learning scenarios)
6-10	30	simple	
11-15	20	simple	
16-20	40	advanced	
21-25	30	advanced	
26-30	20	advanced	
31-40	40	simple	Fig. 14(b) (generalizing scenarios)
41-50	30	simple	
51-60	20	simple	
61-70	40	advanced	
71-80	30	advanced	
81-90	20	advanced	

To measure effectiveness of each NC, the following evaluation functions were used:

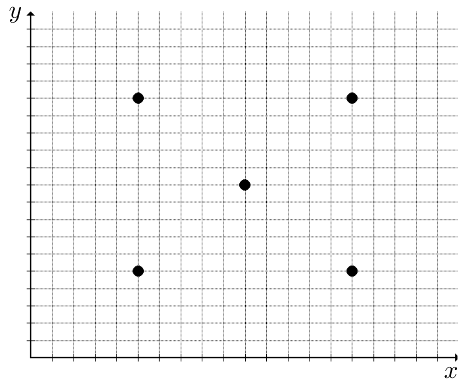
$$f(NC) = \sum_{i=0}^{n} f_i \tag{2}$$

$$f_i^l = \begin{cases} d_{max} - \min\limits_p d_i(p), \\ \qquad \text{prey not captured in } i^{th} \text{ scenario} \\ f_{captured} + (80 - m_i)/a, \\ \qquad \text{prey captured in } i^{th} \text{ scenario} \\ 0, \qquad \text{prey not captured in previous scenario} \end{cases} \tag{3}$$

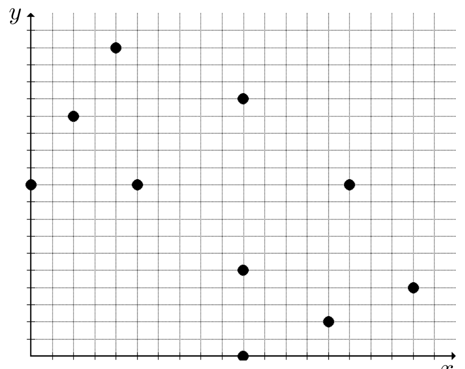
$$f_i^g = \begin{cases} d_{max} - \min\limits_p d_i(p), \\ \qquad \text{prey not captured in } i^{th} \text{ scenario} \\ f_{captured} + (80 - m_i)/a, \\ \qquad \text{prey captured in } i^{th} \text{ scenario,} \end{cases} \tag{4}$$

where

f_i^l	– reward received in i^{th} learning scenario
f_i^g	– reward received in i^{th} generalizing scenario
$d_i(p)$	– distance between prey and predator p in end state of i^{th} scenario
d_{max}	– maximum distance between two points in environment
$f_{captured}$	– extra reward for grasping prey in single scenario ($f_{captured} = 100$)
m_i	– number of steps to capture prey ($m_i \leq 40$, 30 or 20)
a	– this value prevents situation in which partial success is better than success in all scenarios
n	– number of scenarios (learning phase: $n = 30$, generalization phase: $n = 60$)



(a)



(b)

Fig. 13. Starting positions of prey in learning (a) and generalizing (b) scenarios

III. 2. Experimental results

The experiments summarized in Table 3 showed that in terms of the learning ability all the modifications outperform the original variant of AE. Modifications M1, M2, M3, and M5 appeared to be the most effective achieving results at more or less the same level. Somewhat worse efficiency was noticed for modification M4 and its mirror image M6.

Since all the modifications use the same evolutionary scheme as AE, mechanisms responsible for searching a genotype space could not have influence on better performance of the modifications in relation to the original variant of AE. The way for encoding AEPs is very similar in all the cases, which means that this factor also could not affect differences in performance between the modifications and AE.

$$\begin{array}{ccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.634921 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.634921 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -0.634921 & 0 & 0 & 0 \\
0 & 0 & 0.269841 & 0 & 0 & 0 & 0 & 0 & 0 & -0.634921 & 0 & 0 & 0 \\
0 & 0 & 0.15873 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.873016 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.253968 & 0.269841 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.634921 & 0.15873 & 0.714286 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.650794 & 0.666667 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.619048 & 0 & 0 & -0.111111 & 0.619048 & 0.793651 & 0 & 0 & -0.634921 & 0 & 0 & 0 \\
0 & -0.888889 & 0 & 0 & 0.15873 & -0.888889 & -0.031746 & 0 & 0 & -0.634921 & 0 & 0 & 0 \\
0 & -0.968254 & 0 & 0 & 0.126984 & -0.968254 & 0.015873 & 0 & 0 & -0.634921 & 0 & 0 & 0
\end{array}$$

(a) Example pattern matrix generated with M1

$$\begin{array}{ccccccccccccc}
0 & 0 & 0 & 0 & 0 & -0.0952381 & 0 & 0 & 0 & 0.507937 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.793651 & 0 & 0 & 0 & -0.666667 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.936508 & 0 & 0 & 0 & 0.825397 & 0 & 0 & 0 \\
0 & 0 & -0.920635 & 0 & 0 & 0.603175 & 0 & 0 & 0 & 0.269841 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.746032 & 0 & 0 & 0 & -0.380952 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.333333 & 0 & 0 & 0 & -0.365079 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.111111 & 0 & -0.793651 & 0 & 0.0793651 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.698413 & 0 & 0 & 0 & 0.952381 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.015873 & 0 & 0 & 0 & -0.0952381 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -0.444444 & 0 & 1 & 0 & 0.650794 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.746032 & 0 & 0 & 0 & -0.809524 & 0 & 0 & 0 \\
0 & 0 & 0 & -0.920635 & 0 & -0.920635 & 0 & 0 & 0 & -0.793651 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.809524 & 0 & 0 & 0 & 0.746032 & 0 & 0 & 0
\end{array}$$

(b) Example pattern matrix generated with M3

Fig. 14. Example pattern matrices

Therefore, the only justification of the achieved results is that new operations with a special capability to form NDMs with a distributed content make building effective ANNs easier compared to the original variant of AE. The fact that all the modifications in spite of differences in construction appeared to be more effective than AE means that the ability to form the matrices above is a significant factor improving performance of AE. Using operations which can only modify continuous clusters in NDM severely limits this ability. New variants of AE make NDMs with a distributed content easier to generate and thereby they increase effectiveness of AE in forming efficient neural networks.

With regard to NDMs, the ones produced by modifications M4 and M6 require special attention. It turned out that all of them represent ANNs with a maximum acceptable number of neurons. This is due to the fact that a resultant NDM in both modifications is a combination of matrices whose all elements have a value assigned by appropriate operations (see Figures 8 and 10). In this case, there is no element untouched by AEP. The matrix including weights of interneuron connections is usually completely filled up with values different from zero whereas matrix defining topology of an ANN includes at least one "1" in each column and row (it is unlikely for a chromosome encoding topology of an ANN to include many zeros in turn). A combination of both matrices leads to a NDM with at least one value different from zero in each column and row. Since no row and column from such a matrix can be omitted when building ANN, a network with a maximum acceptable number of neurons is finally constructed.

Even though the remaining modifications adjust the number of neurons to a problem, their NDMs also seem to be characteristic. This time, we often dealt with matrices whose content took the form of oblong horizontal clus-

Tab. 3. Results of experiments on predator-prey problem (LPh - learning phase, GPh - generalization phase; e.g. fitness 2582.2 means 25 scenarios in which prey was captured, on average)

	AE	M1	M2	M3	M4	M5	M6
Average fitness LPh	2582.2	2757.03	2795.15	2767.15	2716.67	2773.5	2673.01
Average fitness GPh	3399.57	3994.56	3722.16	3878.32	2984.27	3794.29	3382.23

ters with holes (zeros). Such content of the matrices may be the result of the problem to be solved or operations used in AEPs (all new operations modify NDMs row by row). If the operations were mainly responsible for such state of affairs it would mean that a part of modifications may have difficulties in solving problems requiring other NDMs than the ones produced in the experiments. To test it, further experiments were carried out, and their results are reported in the following section.

As for the generalization abilities of ANNs, they in most cases agree with the results achieved during the learning phase. That is, ANNs which performed well in the learning tasks were also effective in the generalization ones and vice versa. ANNs produced by M4 and M6 are exceptions to this rule. In this case, we often dealt with the overfitting phenomenon. As mentioned above, ANNs produced by M4 and M6 were usually much more complex in terms of construction than other ANNs. In consequence, they had difficulties in generalizing knowledge acquired during the learning phase.

IV. EXPERIMENTS ON MATRICES

The main goal of the experiments reported in this section was to estimate and compare the ability of the methods presented in Section 2 to form diverse neural architectures or, in other words, to estimate their completeness. Since, in AE, ANNs are encoded in the form of NDMs, the tests were carried out on matrices, not on ANNs. That is, the task of each method was not to produce ANNs but some pattern matrices. In the experiments, only AE and modifications M1, M2, M3, and M5 were tested. Modifications M4 and M6 were neglected because of their unsatisfactory results in the previous phase of the experiments.

In order for the tests to reliably characterize each method, the pattern matrices had to be as varied as possible. To this end, matrices which represented ANNs in the previous experiments were applied, in total 210 different matrices, 30 per each method. To increase their diversity, half of them were additionally transposed.

Even though the matrices generated in the previous experiments represented ANNs used to solve the same problem, there was a wide diversity range among them. They were very different in terms of how and where they were filled in. Comparing many of them one could have the impression that deals with random matrices. Generally, they were sufficiently diverse to carry out the experiments exclusively based on them. Additional argument for the matrices from the previous phase of the experiments was that they all represented true ANNs. In the case of random matrices, there would not have been any guarantee that they correspond to any ANNs.

During the tests, each method was run 210 times for the same parameter setting as in the experiments on the predator–prey problem (see Table 5 in Appendix). In each run, which took maximally 60000 generations, a single matrix was generated. Fitness of each AEP was measured in the following way:

$$f(AEP) = \begin{cases} \dfrac{\sum\limits_{i,j} f^m_{i,j}}{N^2} & \text{if } \sum\limits_{i,j} f^m_{i,j} > 0 \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$$f^m_{i,j} = \begin{cases} \dfrac{1}{|m^{AEP}(i,j) - m^{P}(i,j)| + 1}, & \\ & \text{if } \begin{array}{l} m^{AEP}(i,j), m^{P}(i,j) \neq 0 \\ \vee\, m^{AEP}(i,j), m^{P}(i,j) = 0 \end{array} \\ -1, & \text{otherwise,} \end{cases} \tag{6}$$

where

$m^{AEP}(i,j)$	– i, j^{th} item in matrix generated by AEP,
$m^{P}(i,j)$	– i, j^{th} item in pattern matrix,
N	– height and width of all matrices.

According to (5), fitness of each AEP was an average over rewards and penalties received while comparing all items from a pattern matrix and the matrix generated by the program. AEPs were rewarded if in both compared matrices either zeros or values different from zero were at corresponding positions, otherwise AEPs were penalized. This way, AEPs were promoted, which produced matrices representing ANNs with the same topology as ANNs represented by means of pattern matrices. Rewards were scaled to the range $(0, 1 >$. Magnitude of the reward depended on the similarity between values from both matrices. Maximum reward amounted to one and was given when the values were identical, while penalties were always equal to minus one.

Tab. 4. Results of experiments on matrices (each cell in the table includes average fitness of AEPs for selected pattern matrices)

	AE	M1	M2	M3	M5
30 AE0 matrices	0.67	0.32	0.91	0.45	0.32
30 M1 matrices	0.37	0.19	0.9	0.19	0.17
30 M2 matrices	0.66	0.39	0.93	0.55	0.38
30 M3 matrices	0.65	0.38	0.92	0.63	0.32
30 M4 matrices	0.1	0.1	0.62	0.09	0.1
30 M5 matrices	0.11	0.11	0.64	0.09	0.1
30 M6 matrices	0.3	0.21	0.85	0.29	0.2
All 210 matrices	0.41	0.24	0.83	0.32	0.23

The experiments summarized in Table 4 showed that modification M2 has the greatest easiness in forming diverse matrices and, as we assume, diverse neural architectures. Regardless of the creator of pattern matrices the ones evolved according to M2 were always closest to them. The remaining methods turned out to be significantly worse than M2. In the case of AE0 and M1, the cause seems to be slight flexibility of both methods. AE0 uses variety of operations whereby it has the potential to produce diverse matrices. However, as already mentioned, all the operations applied in AE0 and, in consequence, AE0 itself have difficulties generating matrices with a distributed content. M1 by using an operation which makes construction of such matrices easier does not have such problems. However, the fact that it is based on only one operation significantly narrows down the set of matrices which can be easily and rapidly generated by means of M1.

As for modifications M3 and M5, it seems that the main cause of their worse performance is the length of chromosomes and a greater genotype space to search. In AE0, M1, and M2, operations use shared data located in a separate chromosome and for that reason they can be encoded in the form of short chromosomes including only four parameters. In spite of a small length, each operation can introduce to NDM many different values. This is because each of them can use any fragment from a long data chromosome. To achieve a similar effect in M3 and M5, their chromosomes encoding operations had to be considerably longer than their counterparts from AE0, M1, and M2. In consequence, AEPs in M3 and M5 had usually more complex genotypes than the ones using shared data. Longer genotypes meant, in turn, a larger search space which affected results of both methods.

One can obviously ask the question why M1, M3, and M5 were so effective in forming neuro-controllers whereas when the task was to produce matrices they completely failed. Note that all the mentioned methods appeared to be even worse than AE0. It seems that the answer is the following. In the predator–prey problem, there were more than one effective solution and the task was only to find one of them. M1, M3, and M5 found efficient ANNs faster than AE0 because they have the ability to form matrices with a distributed content and thereby they have easier access to more effective ANNs than the original variant of AE. Meanwhile, when the task was to generate one specific matrix chances of all the methods became even. In this case, AE0 appeared to be more effective than M1, M3 and M5.

V. SUMMARY

The paper proposes several modifications to AE and evaluates them in terms of the ability to produce effective diverse neural architectures. The experiments on the predator–prey problem revealed that all the modifications outperform the original variant of AE in generating effective neuro-controllers. They also showed that capability to easily form NDMs with values loosely scattered throughout the matrix is a factor which strongly affects effectiveness of AE. The experiments with matrices showed how the methods cope with building diverse neural architectures. It turned out that modification M2 is unrivaled in this respect.

APPENDIX
Parameter setting in the experiments on predator-prey problem

Tab. 5. Parameter setting (d – data, o – operations; number of genes in chromosomes and probability of mutation were individually fixed for each method during tuning process)

Parameter	All the methods
Number of evolutionary generations	60 000
Probability of crossover	0.7 in all populations
Probability of cut-splice	0.1 in all populations with variable length chromosomes
Size of tournament	1 in all populations
Number of elite individuals	1 in all populations
Hidden neurons in ANNs	maximally 4
Type of neurons in ANNs	sigmoid

Parameter	AE0	M1	M2	M3
Number of subpopulations	2÷10	2÷10	3÷10	3÷10
Size of subpopulations	100 (d) 50 (o)	100 (d) 50 (CHGM2)	100 (d) 50 (CHGM3)	100 (CHGT0s) 50 (CHGM4s)
No. of integer or binary Genes in chromosomes	20÷40 (d) 5 (o)	20÷40 (d) 4 (CHGM2)	20÷40 (d), 5 (CHGM3) 10÷30 (CHGT0)	10÷40 (CHGM4) 20÷40 (CHGT0)
Probability of mutation	0.01 (d) 0.15 (o)	0.03 (d) 0.1 (CHGM2)	0.03 (d) 0.03 (CHGM3), 0.06 (CHGT0)	0.05 (CHGT0) 0.1 (CHGM4)

Parameter	M4	M5	M6
Number of subpopulations	3÷10	3÷10	3÷10
Size of subpopulations	100 (`CHGT0s`) 50 (`CHGM5s`)	100 (`CHGM6s`) 50 (`CHGT1s`)	100 (`CHGM6s`) 50 (`CHGT2s`)
Number of integer or binary genes in chromosomes	10÷40 (`CHGM5`) 10÷40 (`CHGT0`)	10÷40 (`CHGM6`) 10÷40 (`CHGT1`)	20÷40 (`CHGM6`) 30÷50 (`CHGT2`)
Probability of mutation	0.08 (`CHGT0`) 0.05 (`CHGM5`)	0.01 (`CHGM6`) 0.3 (`CHGT1`)	0.03 (`CHGM6`) 0.1 (`CHGT2`)

References

[1] L. Baird III, *Reinforcement Learning Through Gradient Descent*, PhD thesis, Carnegie Mellon University, Pittsburgh (1999).

[2] A. Cangelosi, D. Parisi, S. Nolfi, *Cell division and migration in a genotype for neural networks*, Network: computation in neural systems **5**(4), 497-515 (1994).

[3] T.J. Fossen, *Guidance and Control of Ocean Vehicles*, John Wiley and Sons Ltd. (1994).

[4] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison Wesley, Reading, Massachusetts (1989).

[5] F. Gomez and R. Miikkulainen, *Incremental evolution of complex general behavior*, Adaptive Behavior **5**, 317-342 (1997).

[6] F. Gomez, J. Schmidhuber and R. Miikkulainen, *Accelerated Neural Evolution through Cooperatively Coevolved Synapses*, Journal of Machine Learning Research, **9**, 937-965 (2008).

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

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

[9] K. Krawiec, B. Bhanu, *Visual Learning by Coevolutionary Feature Synthesis*, IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics **35**, 409-425 (2005).

[10] T. Kubaty, L. Rowinski, *Mine counter vehicles for Baltic Navy*, Internet, http://www.underwater.pg.gda.pl

[11] S. Luke and L. Spector, *Evolving Graphs and Networks with Edge Encoding: Preliminary Report*, In John R. Koza, ed., Late Breaking Papers at the Genetic Programming 1996 Conference, (Stanford University, CA, USA, Stanford Bookstore, 1996) 117-124.

[12] G.F. Miller, P.M. Todd, S.U. Hegde, *Designing Neural Networks Using Genetic Algorithms*, Proceedings of the Third International Conference on Genetic Algorithms. 379-384. of Schaffer J.D. (1989)

[13] D. E. Moriarty, *Symbiotic Evolution of Neural Networks in Sequential Decision Tasks*, PhD thesis, The University of Texas at Austin, TR UT-AI97-257 (1997)

[14] S. Nolfi, D. Parisi, *Growing neural networks*, In C. G. Langton, ed., Artificial Life III, Addison-Wesley (1992)

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

[16] M. Potter, *The Design and Analysis of a Computational Model of Cooperative Coevolution*, PhD thesis, George Mason University, Fairfax, Virginia (1997).

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

[18] T. Praczyk, *Evolving co-adapted subcomponents in Assembler Encoding*, International Journal of Applied Mathematics and Computer Science **17**(4) (2007).

[19] T. Praczyk, *Modular networks in Assembler Encoding*, CMST **14**(1), 27-38 (2008).

[20] T. Praczyk, *Using assembler encoding to solve inverted pendulum problem*, Computing and Informatics **28**, 2009, pp. 895-912 (2009).

[21] T. Praczyk, *Forming Neural Networks by Means of Assembler Encoding*, Intelligent Automation and Soft Computing, **17**(3), 319-331 (2011).

[22] T. Praczyk, P. Szymak, *Decision System for a Team of Autonomous Underwater Vehicles – Preliminary Report*, Neurocomputing, Neurocomputing 74(17): 3323-3334 (2011).

[23] T. Praczyk *Solving the pole balancing problem by means of assembler encoding*, Journal of Intelligent and Fuzzy Systems, in press

[24] W. Siler, J. Buckley, *Fuzzy Expert Systems and Fuzzy Reasoning*, John Wiley and Sons Ltd. (2005).

[25] O. Stanley, *Efficient Evolution of Neural Networks Through Complexification*, PhD thesis, Department of Computer Science, The University of Texas at Austin, Technical Report AI-TR-04-314 (2004).

[26] O. Stanley and R. Miikkulainen, *Evolving neural networks through augmenting topologies*, Evolutionary Computation **10**, 99-127 (2002).

[27] P. Szymak, *Using of fuzzy logic method to control of underwater vehicle in inspection of oceanotechnical objects*, Polish Neural Network Society, Artificial Intelligence and Soft Computing, 163-168 (2006).

[28] P. Szymak, *Simplified mathematical model of underwater vehicle and its control system*, in Polish, Pomiary, Automatyka i Robotyka, No. 2 (2010).

[29] D. Whitley, J. Kauth, *GENITOR: A different genetic algorithm*, In Proceedings of The Rocky Mountain Conference on Artificial Intelligence, 118-130 (1988).

Tomasz Praczyk is a senior lecturer at the Institute of Naval Weapon of Polish Naval Academy in Gdynia. He received his MSc degree in computer science in 1996. In 2001, he received his PhD degree; with thesis focused on using artificial neural networks to identify ships. His research interest is in neuro-evolution, artificial immune systems, and reinforcement learning.