CMST 21(3) 123-139 (2015) DOI:10.12921/cmst.2015.21.03.004

Assembler Encoding with Evolvable Operations

Tomasz Praczyk

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

Received: 23 June 2015; revised: 25 August 2015; accepted: 27 August 2015; published online: 04 September 2015

Abstract: Assembler Encoding is a neuro-evolutionary method which represents a neural network in the form of a linear program. The program consists of operations and data and its goal is to produce a matrix including all the information necessary to construct a network. In order for the programs to produce effective networks, evolutionary techniques are used. A genetic algorithm determines an arrangement of the operations and data in the program and parameters of the operations. Implementations of the operations do not evolve, they are defined in advance by a designer. Since operations with predefined implementations could narrow down applicability of Assembler Encoding to a restricted class of problems, the method has been modified by applying evolvable operations. To verify effectiveness of the new method, experiments on the predator-prey problem were carried out. In the experiments, the task of neural networks was to control a team of underwater-vehicles-predators whose common goal was to capture an underwater-vehicle-prey behaving by a simple deterministic strategy. The paper describes the modified method and reports the experiments.

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. evolutionary computation and artificial neural networks (ANN). Evolutionary techniques are usually used as global optimization methods, while in turn, ANNs are applied in such problems as, for example, approximation, identification, feature extraction, and reinforcement learning. Successes in both domains have promoted a new combined domain called the neuro-evolution (NE) which uses the evolutionary approach to search for effective ANNs. The evolution of ANNs proceeds as the evolution in humans. 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, leading to better and better ANNs generated within evolution.

There are a lot of NE methods (e.g. [3, 6, 7, 11-14, 24]). 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. In turn, in the indirect methods, chromosomes are recipes how to create a network. Such methods can encode complex neural architectures by means of relatively short chromosomes.

One of the indirect methods is Assembler Encoding (AE). It originates from the cellular [6] and edge encoding [11], although it also has features common with Linear Genetic Programming presented, among other things, in [8, 15].

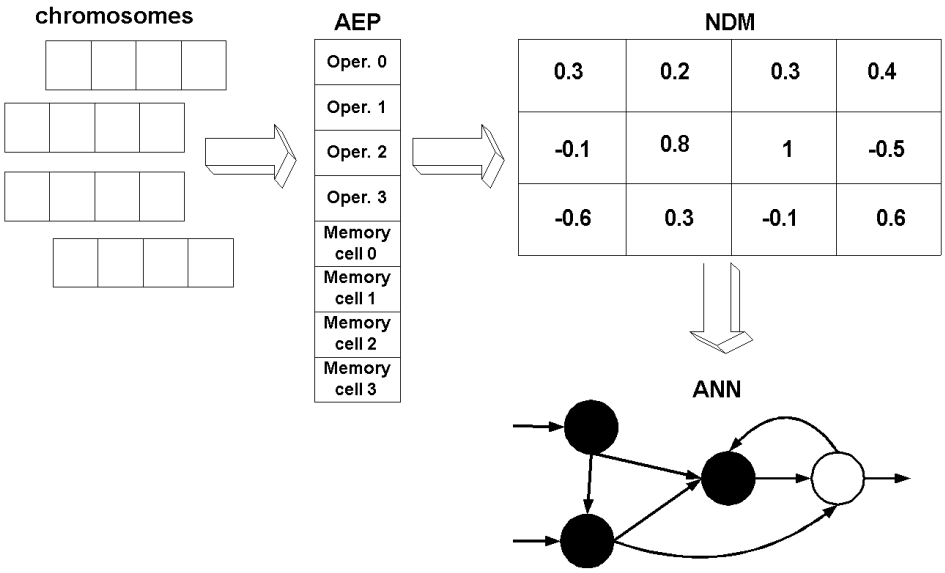


Fig. 1. Using AE to create ANN [20]

In AE, an ANN is represented in the form of an Assembler Encoding Program (AEP) whose structure is similar to 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 produce a network. During operation, the AEP runs subsequent operations which gradually modify 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 (see 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.

In the classic variant of AE (e.g. [19, 21, 22]), implementations of operations which can be used in AEPs do not evolve, they are defined beforehand by a designer. Operations can modify rows, columns of NDM, define ANNs with a layered architecture, produce feed-forward, recurrent, fully or sparsely connected ANNs. Moreover, they can be general or adjusted to a problem. Since all the operations are parametrized, each of them can be used to construct many different neural architectures. Usually, the parameters of the operations define topologies of ANNs to be built (which items in NDMs are modified) and indicate data used by the operations to determine weights of interneuron connections (which data are copied into NDMs).

The operations with hand-made implementations are used during the evolutionary process to form AEPs. A genetic algorithm decides which operations to combine together, how to arrange selected operations, which values to assign to input parameters of each operation, and which data to use in individual operations. Such an approach appeared to be effective in constructing simple and medium complexity neural networks, which has been confirmed during series of experiments (e.g. [19, 21, 22]). Ie same experiments showed, however, that performance of AEPs with hand-made operations strongly depends on which operations are available to the programs during the evolution. Since a designer who selects and implements the operations usually has no a priori knowledge of that which architectures of ANNs are appropriate for a problem to be solved, in most cases he also cannot determine which operations should be used in AEPs to effectively form the networks. In this case the only solution is to experimentally tune the method by testing different configurations of operations and selecting the one with the highest effectiveness. Since, however, there are a lot of possible configurations of operations, the tuning process is usually laborious and time-consuming.

To avoid such a long-lasting tuning process, Assembler Encoding with Evolvable Operations (AEEO), modification of the classic AE which is the subject of the paper, instead of applying hand-made operations, uses operations whose behavior undergoes evolution. In AEEO, the operations take the form of simple ANNs, say ANNs-operations. As in the classic variant of AE, the task of ANNs-operations is to modify the content of NDM. They are run in different areas of the matrix determining weights of interneuron connections and other parameters of a resultant ANN. To fix the weights of the connections (or values of other parameters), output signals of ANNs-operations supplied with addresses of corresponding items in NDM are used. As in the classic variant of AE, the whole set of ANNs-operations evolves according to Cooperative Co-Evolutionary GA (CCEGA) [16, 17]. However, in this case the influence of the evolution on behavior of operations does not limit to defining their param-

```
Parameters:
P_max - maximum number of neurons in resulting ANN
MaxValue - scaling value
Input:
p0, p1, p2, p3 - all operations have four parameters encoded in chromosome
Output:
updated fragment of NDM
```

```
column = abs(p0) mod (P_max + 2)
//2 - two extra NDM columns for ANN parameters
rowInit = abs(p1) mod P_max
numberOfIterations = abs(p2) mod (P_max − rowInit)
FOR i = 0 , i ≤ numberOfIterations
    row = rowInit + i
    NDM_{row,column} = D_{(abs(p3)+i) mod D_length} / MaxValue
END
```

Fig. 2. Example operation used in AE ($NDM_{i,j}$ is an element of NDM, where $i = 1..P_{max}$, $j = 1..P_{max} + 2$, $MaxValue$ is a scaling value which scales all elements in NDM to the range $< -1, 1 >$, D_i is i^{th} element of data, D_{length} is the length of data)

eters, in AEEO, a modus operandi of ANNs-operations is completely defined in the evolutionary way.

The solution above is based on Hypercube NeuroEvolution of Augmented Topologies (HyperNEAT) proposed by Gauci and Stanley in [4]. In HyperNEAT, ANNs are produced by other ANNs called Compositional Pattern Producing Networks (CPPNs) which evolve according to NEAT [24]. To generate an ANN, its neurons are first placed in n-dimensional space. Then, weights of connections between all neurons in the space are determined by a single CPPN. To fix a weight between a pair of neurons the CPPN is supplied with coordinates of the neurons. An output signal of the CPPN indicates the weight of the connection. In order for the CPPNs to be able to generate a diverse neural architectures they can use different neurons, e.g. Gaussian, sigmoid, sinusoid, absolute.

In spite of strong resemblance between HyperNEAT and AEEO, there are also differences between both methods. First, HyperNEAT uses a single CPPN to form an ANN whereas in AEEO a network is produced by many simple, cooperating ANNs-operations. Second, each CPPN evolves according to NEAT while ANNs-operations are constructed with CCEGA. Besides that, there are also other minor differences between the methods. They mainly relate to the construction of CPPNs and ANNs-operations and interpretation of their input and output signals.

To test performance of AEEO, experiments on the predator-prey problem were carried out. During the experiments, the task of AEEO was to produce neuro-controllers responsible for controlling a team of Autonomous Underwater Vehicles (AUVs) whose common goal was to capture an escaping AUV behaving by a simple deterministic strategy. To obtain a point of reference for AEEO, in the experiments, AE and NEAT [1] were also used. The choice of NEAT was mainly dictated by the fact that it seems to be currently the most successful NE method and therefore a sensible reference point. Moreover, it is widely applied [1, 2, 10, 24, 25] and well understood, which means that it can serve as an indication of the complexity of the testing problem used in the experiments which is not a standard problem because of AUVs.

The paper is organized as follows: section 2 is a presentation of the original variant of AE, section 3 is a description of AEEO, section 4 is a short presentation of NEAT, section 5 is a report on the experiments, and section 6 is a summary.

II. ASSEMBLER ENCODING

As already mentioned, in AE, ANNs are represented in the form of AEPs and NDMs. The AEP is an ordered set of predefined operations and data. The task of each AEP is to create an NDM and to fill in it with values. To this end, the operations are executed in turn, one after another. Each operation modifies some fragment of the NDM dependent on type and parameters of the operation (initially, all items in NDM are set to 0; this means that there are no connections between neurons). Each AEP is shaped in the evolutionary way. The evolution decides about an arrangement of the operations and data, and about values of operation parameters. Implementations of the operations do not evolve, they are defined beforehand [19].

The 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

[1] C++ implementation of the method available on http://www.cs.ucf.edu/ kstanley/neat.html

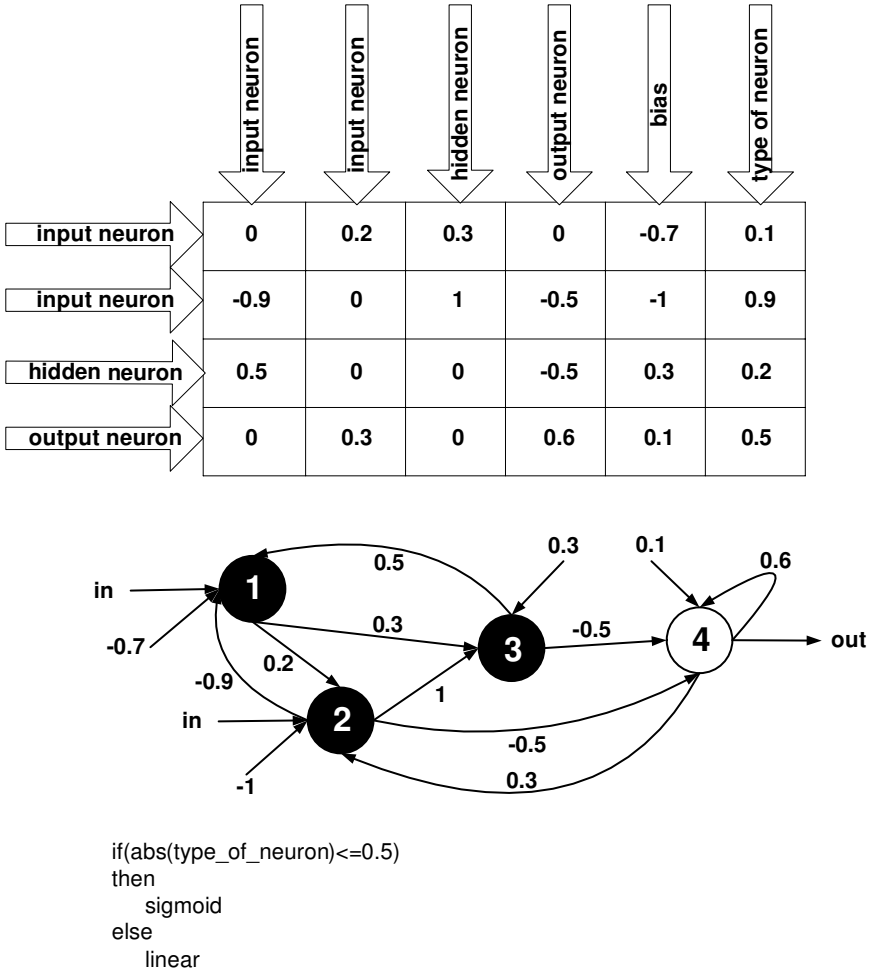


Fig. 3. NDM as Connectivity Matrix [19]

the range $< -1, 1 >$. 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. a component$_{i,j}$ determines a link from neuron i to neuron j. Apart from the basic part, the NDM also contains additional columns that describe parameters of neurons, e.g. type of neuron (e.g. sigmoid, radial, linear), and bias (see Fig. 3) [19].

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 (usually the best ones) of each population are combined together. 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, an AEP consisting of n operations and a sequence of data evolves in n populations with operations and one population with data (see Fig. 4). During the evolution, AEPs expand gradually. Initially, all AEPs include an initial number of operations and 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 [18].

In individual populations, the evolution proceeds according to Canonical GA [5]. 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 code of the operation (e.g. 01000|11000|01000|00000|00100 represents the following operation: `CHGC0`|-1|1|0|2). Chromosomes-data are strings including binary encoded data [20].

During evolution, each individual is combined with selected individuals from the remaining populations to form a complete AEP. The program produces an NDM and, in consequence, an ANN which is then put to an evaluation test. The result of the test is used to fix fitness for all components of the AEP. After the evaluation of all individuals from a population a tournament selection is used to choose

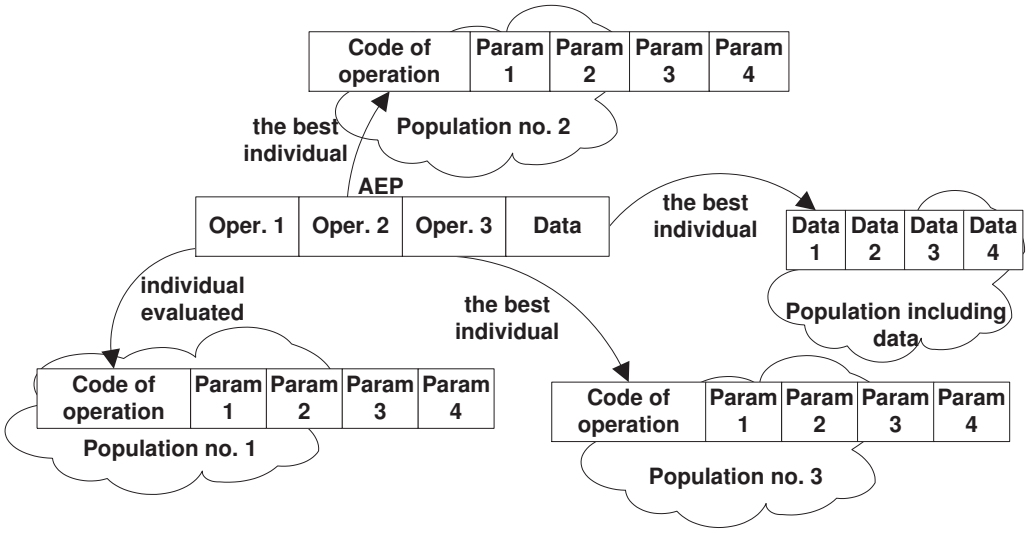


Fig. 4. Evolution in AE for $n = 3$ [20]

parental individuals for reproduction. To form offspring individuals, the parental ones are subject to three different genetic operations, i.e. one-point crossover, mutation and the so-called cut-splice whose task is to change the length of the chromosomes-data.

III. ASSEMBLER ENCODING WITH EVOLVABLE OPERATIONS

In AEEO, AEPs consist only of operations, the programs do not use data. As mentioned above, the operations take the form of ANNs which in AEEO are called ANN-operations. The architecture of ANN-operations is determined by a designer as well as in the evolutionary way. The number of input and output neurons is adjusted to the task of ANN-operations, the number of hidden neurons is determined by a designer whereas topology and weights of interneuron connections are defined in chromosomes. In order to enable the ANN-operations to construct diverse neural architectures, they can include various types of neurons, e.g. sigmoid, radial, linear, and sinusoid.

As in AE, evolution of the AEPs proceeds according to CCEGA, complete programs are sequences of ANN-operations from different populations. An AEP consisting of n ANN-operations evolves in n separate populations, the first ANN-operation comes from the first population, the second operation from the second population, and so forth.

At the genotypic level, each ANN-operation is represented in the form of a variable length chromosome consisting of two parts. The first short part defines topology of the ANN-operation whereas the second part includes its parameters. Construction of an ANN-operation proceeds in three phases. First, topology of the ANN-operation is determined based on the information contained in the first part of the chromosome. To this end, binary values from this part are directly copied into NDM, a single bit corresponds to a single item in the matrix. When the number of bits is insufficient to completely fill in the matrix, the whole sequence of bits is used again. In the next phase, the parameters from the second part of the chromosome are successively introduced into NDM. In this case, only items equal to one are modified. The remaining items, i.e. items equal to zero, remain intact. As before, transfer of the parameters is performed in a loop until all the elements in NDM have an value assigned (see Fig. 6). In the last phase, NDM is transformed into an ANN-operation.

```
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 constructed with AE

In the paper, three different variants of AEEO are proposed, say AEEO1, AEEO2 and AEEO3. In all the variants there is no communication between ANN-operations, they simply modify the same NDM, one item of the matrix after another, and they receive the same fitness for their work. Since, as it turned out in the experiments reported in [23], the ability to produce NDMs with a content loosely scattered throughout the matrix (such NDMs represent ANNs with a sparse connectivity) is a very important property of AEPs, AEEO has been especially designed to have such ability. In AEEO1 and AEEO3, ANN-operations move through all items in NDM and each time they decide whether to update a given item or not, initially, as in AE, all items are set to 0, which means that there are no connections between neurons. In turn, in AEEO2, each ANN-operation "jumps" only to selected items assigning a value to each of them, the items unvisited by ANN-operations remain 0. A detailed description of all the three variants of AEEO is given in the following sections.

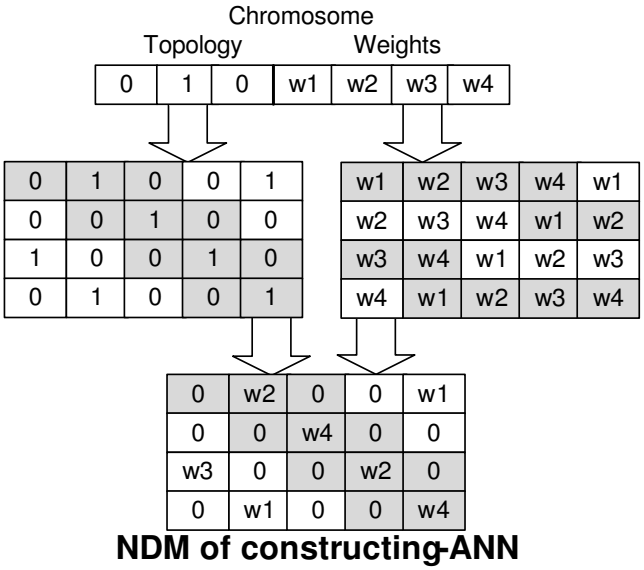


Fig. 6. Encoding ANN-operation in chromosome

III. 1. AEEO1

In AEEO1, ANN-operations are activated one after another as operations in AE. Each ANN-operation is separately run for each item in NDM. Each of them has two inputs, two outputs and an assumed number of hidden neurons. The inputs indicate an item in the matrix to be modified (number of row and column) whereas the outputs determine its value. The first output decides whether to modify the item or not whereas the second output determines a new value for the item. The value from the second output is copied into NDM if the signal from the first output is greater than an assumed threshold (see Fig. 7). A detailed algorithm of AEEO1 is presented in Fig. 9.

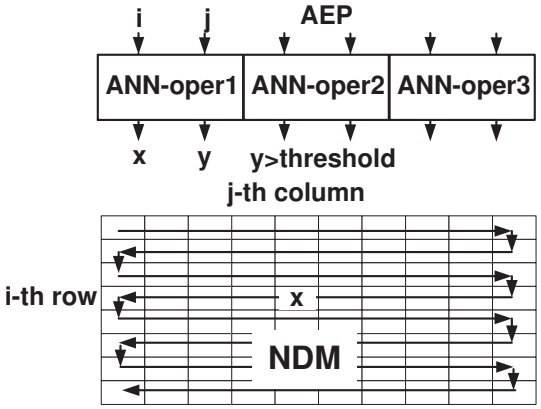


Fig. 7. Example operation of AEP in AEEO1 (ANN-oper1 modifies $NDM_{i,j}$, value x is inserted into the matrix because $y > threshold$, the operation tries to modify all items in NDM, i, j – inputs, x, y – outputs of ANN–oper1)

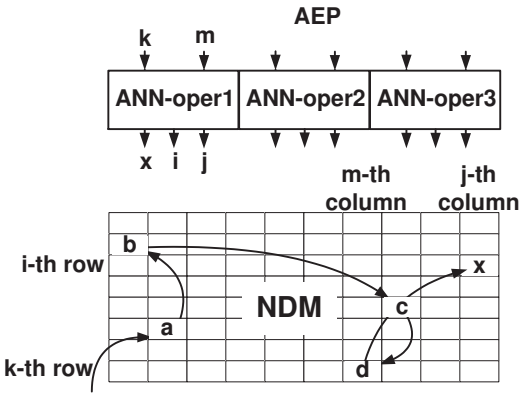


Fig. 8. Example operation of AEP in AEEO2 (ANN-oper1 modifies $NDM_{i,j}$, value x is inserted into the matrix, the operation modifies only selected items in NDM, k, m indicate an item modified in the previous step, i, j, x – outputs, k, m – inputs of ANN-oper1)

III. 2. AEEO2

In AEEO2, ANNs-operations are activated one after another as operations in AE and AEEO1. In contrast to the previous solution, each ANN-operation is run exclusively for selected items in NDM, "jumping" from one to the other and modifying their values. To select items for modification, two separate outputs of an ANN-operation are used. Each ANN-operation has two inputs and three outputs. The outputs indicate items for modification and determine their values whereas the inputs indicate items modified in previous steps. Initially, the inputs of each ANN-operation are set to zero. After activation of the network, the first two outputs point out the first item to be updated whereas the third output determines a new value for the item. In the next step, the

```
Parameters:
T - threshold
P_max - maximum number of neurons in resulting ANN
Input:
set of ANN-operations
L_ANN-oper - number of ANN-operations
Output:
NDM - network NDM
```

```
InitiateNDM(0) //all items in NDM are set to 0
FOR l = 1 , l ≤ L_ANN-oper
    FOR i = 1 , i ≤ P_max
        FOR j = 1 , j ≤ P_max + 2
        //2 - two extra NDM columns for network parameters
            IF ANN_2^l(i,j) > T
            //ANN_k^l(i,j) - kth output of lth ANN-operation
            //i and j are inputs to network
                NDM_{i,j} = ANN_1^l(i,j)
            END
        END
    END
END
```

Fig. 9. Using ANN-operations to form NDM in AEEO1

```
Parameters:
P_max - maximum number of neurons in resulting ANN
S - maximum number of iterations
Input:
set of ANN-operations
L_ANN-oper - number of ANN-operations
Output:
NDM - network NDM
```

```
InitiateNDM(0)
FOR l = 1 , l ≤ L_ANN-oper
    k = 0
    m = 0
    FOR i = 1 , i ≤ S
            NDM_{ANN_1^l(k,m),ANN_2^l(k,m)} = ANN_3^l(k,m)
            //integer values are assigned to k and m
            k = (integer)P_max ANN_1^l(k,m)
            m = (integer)(P_max + 2)ANN_2^l(k,m)
    END
END
```

Fig. 10. Using ANN-operations to form NDM in AEEO2

first two outputs of the ANN are introduced into its inputs. After the next activation of the network its outputs are used to modify the next item. This process continues for an assumed number of steps (see Fig. 8). A detailed algorithm of AEEO2 is presented in Fig. 10.

III. 3. AEEO3

In AEEO3, each ANN-operation has two inputs and three outputs, the inputs indicate an item in NDM updated by the operation whereas the outputs are used for three different purposes, i.e. to determine a negotiation strength of each ANN-operation, to determine whether the item should

by modified or should not, and to determine a new value for the item. Whereas in AE, AEEO1 and AEEO2 various operations can have influence on the same items in NDM (each operation works independently of other operations), in AEEO3 each item is modified by a different operation. To indicate which ANN-operation should determine the value of a given item, negotiation outputs of all networks are compared. An ANN-operation with the highest output signal is entitled to modify the item. In order for the item to be updated, the second output of the selected ANN-operation is tested. If the output value is greater than an assumed threshold the item gets the value from the third output of the ANN-operation (see Fig. 11). A detailed algorithm of AEEO3 is presented in Fig. 12.

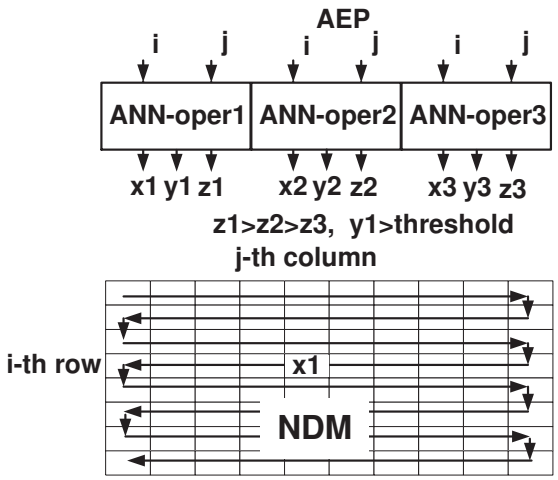


Fig. 11. Example operation of AEP in AEEO3 (ANN-oper1 modifies $NDM_{i,j}$, value $x1$ is inserted into the matrix because $y1 > threshold$ and the negotiation strength of ANN-oper1 – $z1$ is greater than strengths of other ANNs-operations, the operation tries to modify all items in NDM, i, j – inputs, $x1, y1, z1$ – outputs of ANN-oper1)

Node 1 Input	Node 2 Input	Node 3 Hidden	Node 4 Output		
In 1 Out 4 Weight 0.5 Innov 1	In 2 Out 4 Weight 0.8 Innov 2	In 2 Out 3 Weight 0.1 Innov 3	In 3 Out 4 Weight 0.9 Innov 9	In 4 Out 3 Weight 0.3 Innov 11	In 1 Out 3 Weight 0.8 Innov 14

Fig. 13. Structure of chromosome in NEAT [24]

IV. NEUROEVOLUTION OF AUGMENTING TOPOLOGIES

In contrast to AE and AEEO, evolution of ANNs in NEAT proceeds within a single population. Each chromosome from that population is composed of two parts (see Fig. 13) wherein a single ANN is directly encoded. The first part of a chromosome informs about the number of neurons and about the role of each of them, i.e. which neuron is input, output or a hidden neuron. The second part determines connectivity in ANN. Each gene in this part includes the information about the source, destination and weight of a connection. Moreover, it also includes the so-called innovation number indicating a moment of the evolution when the gene is created. The innovation numbers are used for two purposes, i.e. to increase efficiency of crossover so that it could produce meaningful offspring, and to divide chromosomes into species. The main goal of the division into species is to protect innovations appearing during the evolution and to sustain structural diversity. Since new structures compete within their own species they have time to be optimized before they have to compete with more fit individuals from other species. A key feature of NEAT is also increasing complexity of ANNs over the course of evolution. NEAT begins with a population of small, simple ANNs and then adds to them next neurons and connections using structural mutations for that purpose.

V. EXPERIMENTS

In order to test effectiveness of AEEO, experiments on the predator-prey problem were carried out. In the experiments, the task of each ANN was to control a team of AUVs-predators whose common goal was to capture a single AUV-prey behaving by a simple deterministic strategy. The experiments were carried out in simulation and in the configuration with one prey nd three chasing predators. Both the predators and the prey were implemented as AUV "Ukwial" (see Fig. 14) [9]. The behavior of all the vehicles was simulated by means of a discrete time model described in [21].

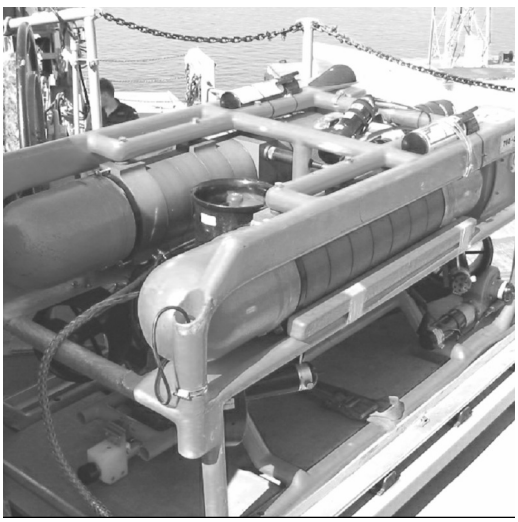


Fig. 14. Vehicle "Ukwial" [21]

```
Parameters:
T - threshold
P_max - maximum number of neurons in resulting ANN
Input:
set of ANN-operations
L_ANN-oper - number of ANN-operations
Output:
NDM - network NDM
```

```
InitiateNDM(0)
FOR i = 1 , i ≤ P_max
    FOR j = 1 , j ≤ P_max + 1
        FOR l = 1 , l ≤ L_ANN-oper
            ns_l = ANN_1^l(i, j) //negotiation strength
        END
        win = Index(ns) //index of winner ANN-operation
        IF ANN_2^win(i, j) > T
            NDM_i,j = ANN_3^win(i, j)
        END
    END
END
```

Fig. 12. Using ANN-operations to form NDM in AEEO3

In the experiments, the predators and the prey lived in a common artificial environment. To represent the environment, a square of 100×100 meters was used (see Fig. 15: to simplify calculations, both the predators and the prey could not submerge under the surface of 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 to make such an attempt to move to the opposite side of the environment [21].

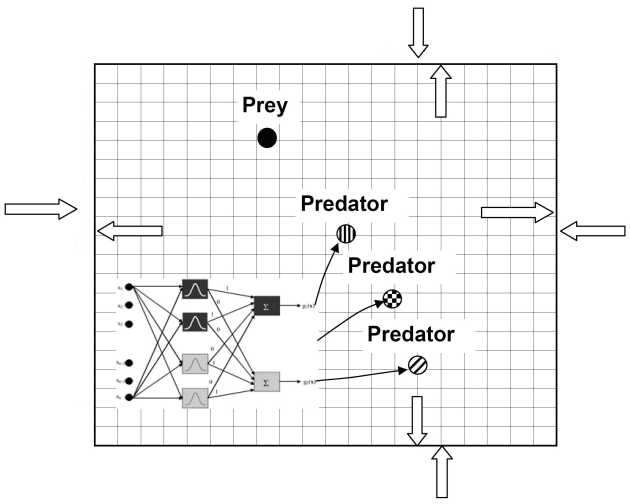


Fig. 15. Artificial world for predators and prey [18]

The predators were controlled by a single ANN whose task was to determine a movement direction for each of them. At each time step, an ANN decided about the change of the current course of each vehicle. The course could be changed by: $0, 5, 10, \ldots, 355$ degrees (it is necessary to note that decisions of ANN determined only a final state of the vehicles which they ultimately should reach, while the 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 the direction of the next move, the first activity of the advanced prey was to calculate a single position representing all predators in its close proximity. The closer the predator was to the prey, the greater its influence was on the calculated position. 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 [21].

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 speed of the predators was the same as 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 lower than 10 meters.

In all the experiments, ANNs 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 [21].

V. 1. Evaluation of ANNs

In order to evaluate ANNs, ninety different testing scenarios were applied. The first thirty scenarios were used to "learn" ANNs. The remaining ones were applied in a generalization phase to evaluate prepared ANNs in terms of their capability to generalize knowledge acquired during the "learning" phase. Individual scenarios differed in an initial position of the prey (see Fig. 16), 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 ANNs (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. The main purpose of gradual reduction of the number of steps which predators could make to capture the prey was to ultimately obtain the chase strategy without unnecessary moves and wandering. The predators should finally be able to quickly organize a chase and effectively cooperate in order to capture the prey.

In all the scenarios, starting positions of all three predators were the same. The predators always started from position (0,0). All the scenarios are described in Tab. 1.

To measure effectiveness of each ANN, the following evaluation functions (or fitness functions) were used:

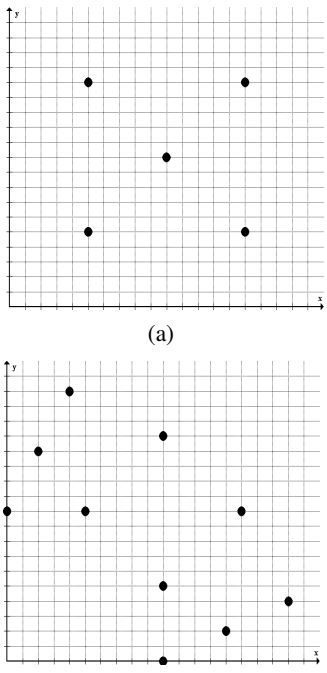
$$f(\text{ANN}) = \sum_{i=0}^{n} f_i \tag{2}$$

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

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

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 ($f_{\text{captured}} = 100$)
- 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. 16. Starting positions of prey in learning (a) and generalizing (b) scenarios

V. 2. Experimental results

In the beginning of the experiments, all the compared methods were tuned to the problem. To this end, each of them was run many times for different parameter settings (see Appendix). For one setting thirty evolutionary runs were carried out. Results obtained for the best setting were then

Tab. 1. Description of scenarios used in the experiments

no. of scenario	max no. of steps	type of prey	initial positions
1-5	40	simple	Fig. 19(c) (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. 19(d) (generalizing scenarios)
41-50	30	simple	
51-60	20	simple	
61-70	40	advanced	
71-80	30	advanced	
81-90	20	advanced	

used to compare the methods. The comparison was made based on three criteria. First, the learning ability, i.e. the ability to evolve effective neural solutions, was tested. 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.

Since the main motivation to design AE and AEEO was the need for the method capable of building complex neuro-controllers for a team of underwater vehicles operating autonomously in order to perform a common task or even a number of tasks, both related methods were also compared in terms of their predisposition to build complex neural architectures. It was assumed that in order for a method to have such a predisposition, first of all, it has to be able to represent elaborate networks in a compact form or in other words, it has to have the potential for phenotypic-genotypic compression. Simply we recognized that it is impossible to evolve complex networks at the level of equally complex genotypes, and to this end, compact representation is necessary. To measure the compression, the following formula was used:

$$\mathrm{Comp(ANN)} = \mathrm{Size(ANN)}/\mathrm{Size(AEP)} \qquad (5)$$

where

$\mathrm{Size(ANN)}$ – number of NDM items necessary to define ANN (since in the experiments all NDMs were of size 13x15 and all neuro-controllers were feed-forward, to define each each of them 91 NDM items were necessary – all the items above the diagonal of NDM),

$\mathrm{Size(AEP)}$ – number of 7-bit integer genes in AEP encoding ANN.

Since NEAT is a direct method which directly maps all parameters of an ANN into a genotype (one-to-one mapping, each parameter of an ANN has a counterpart in a genotype), it was not subject to phenotypic-genotypic compression analysis.

Results of the experiments are summarized in Tab. 2. Generally, they showed that all the variants of AEEO outperform rival methods in terms of both the learning ability and generalization ability of ANNs. Particularly significant is a greater effectiveness of AEEO compared to NEAT which is currently one of the most successful NE methods. Despite the fact that ANNs produced with NEAT were in most cases much more complex than those evolved according to AEEO, they also appeared to be definitely less successful in controlling AUVs than the latter. Even though NEAT-ANNs were gradually expanded within the evolution, which should adjust them to the problem, increase in their complexity, in most cases, did not harmonize with increase in their effectiveness.

Due to similar mechanisms applied in AE and AEEO, ANNs produced in both cases were of similar complexity. A different approach to constructing operations contributed, however, to differences in performance of both related methods. It appeared that the solution in which evolution is mainly responsible for constructing operations is more efficient than hand-made operations. This is confirmed by the results of all the AEEO variants, each of them produces ANNs in a slightly different way, nevertheless, their efficiency is evidently higher than other methods. ANNs-operations have generally a greater freedom in producing diverse neural architectures than their counterparts from AE. Operations used in the classic AE work within the bounds determined mainly by their implementations. They cannot change NDMs in the way different from that imposed by

Tab. 2. Results of experiments (LPh – learning phase, GPh – generalization phase; e.g. fitness 2888.34 means 28 scenarios in which prey was captured, on average; complete success means that prey was grasped in all learning or generalizing scenarios)

	AEEO1	AEEO2	AEEO3	AE	NEAT
average fitness LPh	2812.24	2831.42	2888.34	2582.2	2598.54
max fitness LPh	3020.25	3020.75	3021.31	3020.43	3018.22
min fitness LPh	1812.1	1911.14	1911.09	1811.34	1309.3
% of evolutionary runs ended with complete success	60%	62%	67%	27%	30%
average fitness GPh	3787.23	4039.21	4243.93	3399.57	2943.45
max fitness GPh	5422.76	5543.63	5856.08	5438.04	4831.86
min fitness GPh	1334.82	1765.16	1574.38	651.12	834.5
% of complete successes in generalization phase	0%				
average compression	3.03	3.51	3.04	2.08	
max compression	7	15.16	10.11	3.37	
min compression	1.02	1.01	0.98	1.38	

1st ANN-operation 0100110 0110101 0011100
2nd ANN-operation 0100110 0011011 1111100 1100100 0001110 0100000 0111100 0001110 1001101
0010001 0010011 1001010 1111101 1100011 0101011 1100011 0001110

(a) AEP including two ANNs-operations (binary chromosomes)

0	0.68254	0	0	0.222222	0.68254	0	0	0.222222	0	0
0.68254	0.222222	0	0	0.68254	0	0	0.222222	0.68254	0	0
0.222222	0	0	0.68254	0.222222	0	0	0.68254	0	0	0.222222
0.68254	0	0	0.222222	0	0	0.68254	0.222222	0	0	0.68254
0	0	0.222222	0.68254	0	0	0.222222	0	0	0.68254	0.222222
0	0	0.68254	0	0	0.222222	0.68254	0	0	0.222222	0
0	0.68254	0.222222	0	0	0.68254	0	0	0.222222	0.68254	0
0	0.222222	0	0	0.68254	0.222222	0	0	0.68254	0	0

(b) NDM representing 1st ANN-operation

0	0.857143	0	0	-0.238095	-0.142857	0	0	0.444444	0	0
0.015873	0.238095	0	0	0.444444	0	0	-0.698413	0.539683	0	0
0.793651	0	0	-0.31746	-0.746032	0	0	-0.777778	0	0	0.84127
-0.777778	0	0	0.444444	0	0	0.857143	-0.238095	0	0	-0.142857
0	0	0.444444	0.015873	0	0	0.238095	0	0	0.444444	-0.698413
0	0	0.539683	0	0	0.793651	-0.31746	0	0	-0.746032	0
0	-0.777778	0.84127	0	0	-0.777778	0	0	0.444444	0.857143	0
0	-0.238095	0	0	-0.142857	0.444444	0	0	0.015873	0	0

(c) NDM representing 2nd ANN-operation

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	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.93074	-0.877612	-0.198116	0.632555	0.999997	0.62831	-0.207981	-0.89048	-0.914514	-0.26231	0.583859	0.998293	0.67454	-0.148002	-0.861104	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	-0.318218	0.622389	-0.851968	0.979442	-0.989532	0.881029	-0.666938	
0	0	0	0	0	-0.659367	0.363511	-0.0240899	-0.318218	0.622389	-0.851968	0.979442	-0.989532	0.881029	-0.666938	
-0.916548	-0.888167	-0.203031	0.632236	0.999997	-0.659367	0.363511	-0.0240899	-0.318218	0.622389	-0.851968	0.979442	-0.989532	0.881029	-0.666938	
0	0	0	0	0	0	0.363511	-0.0240899	-0.318218	0.622389	-0.851968	0.979442	-0.989532	0.881029	-0.666938	
0	0	0	0	0	0	0	-0.0240899	-0.318218	0.622389	-0.851968	0.979442	-0.989532	0.881029	-0.666938	
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	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

(d) NDM representing final ANN

Fig. 17. Example effective AEEO1-AEP

a designer. In the case of ANNs-operations, we deal with a different situation. Their behavior mainly depends on decisions made during the evolutionary process and is adjusted to a problem to be solved. In this case, there are not any rigid operational schemes imposed from the outside. Obviously, ANNs-operations are also restricted with the maximum number of neurons and possible types of neurons, but these limitations seem to be significantly lighter than the ones being the result of one fixed operational scheme.

In addition to implementations defined beforehand, also a limited range of values introduced into NDMs confines classical operations. Before transfer to NDM, each element

1st ANN-operation	0010000 1010111 1000101 1001100 0110000 0101010
2nd ANN-operation	0001001 0000101 1111111 1001101 0110100 0110000 0010110 1100111 0111111
	0010100 1010010 1101111 1100101 0000110 1001111 1110111 0100100

(a) AEP including two ANNs-operations (binary chromosomes)

0	0	-0.920635	0	0	0	0	0	0	-0.634921	0	0
0	0	0	0	-0.190476	0	0	0	0	0	0	0.047619
0	0	0	0	0	0	0.333333	0	0	0	0	0
0	-0.920635	0	0	0	0	0	0	-0.634921	0	0	0
0	0	0	-0.190476	0	0	0	0	0	0	0.047619	0
0	0	0	0	0	0.333333	0	0	0	0	0	0
-0.920635	0	0	0	0	0	0	-0.634921	0	0	0	0
0	0	-0.190476	0	0	0	0	0	0	0.047619	0	0
0	0	0	0	0.333333	0	0	0	0	0	0	-0.920635

(b) NDM representing 1st ANN-operation

0	0	0	0.634921	0	0	-1	0	0	0	-0.698413	0
0	0.174603	0	0	0	0.047619	0	0	0.412698	0	0	0
-0.904762	0	0	1	0	0	0	0.15873	0	0	-0.285714	0
0	0	-0.968254	0	0	-0.650794	0	0	0	0.380952	0	0
-0.952381	0	0	0	-0.936508	0	0	0.142857	0	0	0	0.634921
0	0	-1	0	0	0	-0.698413	0	0	0.174603	0	0
0	0.047619	0	0	0.412698	0	0	0	-0.904762	0	0	1
0	0	0	0.15873	0	0	-0.285714	0	0	0	-0.968254	0
0	-0.650794	0	0	0	0.380952	0	0	-0.952381	0	0	0

(c) NDM representing 2nd ANN-operation

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	0.978583	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	-3.57197	0	0	0	-3.48575	0	0	-3.69786	-3.52408	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	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	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	0.655481	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0.263117	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	0.414901	0	0	0	0

(d) NDM representing final ANN

Fig. 18. Example effective AEEO2-AEP

of data encoded as a binary string is first decoded into an integer and then scaled into the range $< -1, 1 >$. In consequence, repertoire of ANNs which can be produced with AE is restricted only to the ones whose weights of interneuron connections are from the mentioned range.

In the case of ANNs-operations, such a problem does not exist. Since they can also contain linear neurons, the range of values introduced into NDMs is, in this case, in principle unlimited. This, in turn, means that ANNs constructed with ANNs-operations are more "plastic" than the ones produced with the classic operations and in consequence they can be more easily adjusted to a problem than the latter.

In the generalization phase, results of most ANNs, regardless of the constructing method, corresponded to the ones achieved in the learning phase. Better preparation to a task during the evolutionary process resulted in better effectiveness in the generalization tests. There were a number of ANNs which were an exception to this rule, particularly ANNs produced with NEAT. Their complexity was in most cases very high, which seems to be the main cause of their problems with overfitting and a poor generalization.

As for the comparison of the various variants of AEEO, the experiments showed that AEEO3 is the most effective of them. ANNs produced by AEEO3, both in the learning and generalization phase, coped with their task best. This is particularly evident when comparing percent of evolutionary runs ended with complete success in the learning phase of the experiments. In this case, as many as 67% of ANNs produced by AEEO3 grasped the prey in all the learning scenarios, AEEO1 and AEEO2 achieved a slightly worse result, namely, 60 and 62% of complete successes, respectively. Since all the AEEO variants use the same ANN-operation encoding method, the only explanation of the above result is the difference in the method of cooperation between ANN-operations applied in individual variants. In AEEO3, only one ANN-operation can update an NDM item, the operations do not interfere with each other, none of them can cover the effects of the work of another operation. AEEO1 and AEEO2 are different in this respect, namely, ANN-operations located further in AEPs, and in consequence run later can modify items which were previously modified by ANN-operations preceding them. It means that in AEEO1 and AEEO2 the effect of work of some operations can be lost. In extreme cases, it may even be a situation when some ANN-operations will not have any influence on the shape of a resultant ANN, all their updates will be covered by other operations.

The experiments with all variants of AEEO also confirmed the previous observations on the importance of the ability to construct NDMs with a distributed content. As the

1st ANN-operation 1000001 0010111 0101100 1001011
2nd ANN-operation 0111010 1000101 0011011 1111001 1000100
(a) AEP including two ANNs-operations (binary chromosomes)

0.920635	0	0	0	0	0	0.206349	-0.825397	0	0	0	0
0	0.920635	0.206349	0	0	0	0	0	-0.825397	0.920635	0	0
0	0	0	0.206349	-0.825397	0	0	0	0	0	0.920635	0.206349
0	0	0	0	0	-0.825397	0.920635	0	0	0	0	0
0.206349	-0.825397	0	0	0	0	0	0.920635	0.206349	0	0	0
0	0	-0.825397	0.920635	0	0	0	0	0	0.206349	-0.825397	0
0	0	0	0	0.920635	0.206349	0	0	0	0	0	-0.825397
0.920635	0	0	0	0	0	0.206349	-0.825397	0	0	0	0
0	0.920635	0.206349	0	0	0	0	0	-0.825397	0.920635	0	0

(b) NDM representing 1st ANN-operation

0	-0.634921	0.857143	-0.619048	0	-0.126984	0	0	-0.634921	0.857143	-0.619048	0
-0.126984	0	0	-0.634921	0.857143	-0.619048	0	-0.126984	0	0	-0.634921	0.857143
-0.619048	0	-0.126984	0	0	-0.634921	0.857143	-0.619048	0	-0.126984	0	0
-0.634921	0.857143	-0.619048	0	-0.126984	0	0	-0.634921	0.857143	-0.619048	0	-0.126984
0	0	-0.634921	0.857143	-0.619048	0	-0.126984	0	0	-0.634921	0.857143	-0.619048
0	-0.126984	0	0	-0.634921	0.857143	-0.619048	0	-0.126984	0	0	-0.634921
0.857143	-0.619048	0	-0.126984	0	0	-0.634921	0.857143	-0.619048	0	-0.126984	0
0	-0.634921	0.857143	-0.619048	0	-0.126984	0	0	-0.634921	0.857143	-0.619048	0
-0.126984	0	0	-0.634921	0.857143	-0.619048	0	-0.126984	0	0	-0.634921	0.857143

(c) NDM representing 2nd ANN-operation

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	-2.46364	0	-2.37357	0	0	0	0	0	0	0	0	0	-0.922691	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.203134	0.356482	-0.500732	0.632236	-0.747593	0.843854	-0.918559	0.969801	-0.996271	0.997292	-0.972839	0.923535	-0.850642	0.756019	-0.642085	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.203031	0	0	-4.92232	0	0	0	0	0	-3.2685	0	0	0	0	-3.26456	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.203031	0.356433	-0.50073	0.632236	-0.747593	0.843854	-0.918559	0.969801	-0.996271	0.997292	-0.972839	0.923535	-0.850642	0.756019	-0.642085	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-0.203031	0.356433	-0.50073	0.632236	-0.747593	0.843854	-0.918559	0.969801	-0.996271	0.997292	-0.972839	0.923535	-0.850642	0.756019	-0.642085	
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	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(d) NDM representing final ANN

Fig. 19. Example effective AEEO3-AEP

tuning process of the variants revealed, application inappropriate values for parameters that control frequency of NDM updates by ANN-operations, that is, too small values for a threshold parameter T (see Figs. 9 and 12) and too high value for parameter S (see Fig. 10), resulted in densely connected ineffective ANNs. Only values close to the maximum yielded the expected effect in the form of more efficient ANNs with fewer connections.

Analysis of the phenotypic-genotypic compression showed that both AE and AEEO demonstrate the ability to effectively construct ANNs whose NDMs include more units of information than the corresponding AEPs and their genotypes. Such a situation took place in almost all analyzed ANNs. As it turned out, however, AEEO represented resultant ANNs in a more compact form than the classic variant of AE. The chromosomes representing AEPs were in this case even ten times smaller in size than the corresponding ANNs. It seems that such an effect was possible for two reasons. First, in AEEO, the compression can be two-level. ANNs can be compressed in two different points, i.e. when transforming chromosomes into ANNs-operations (repeated use of information included in chromosomes) and ANNs-operations into NDMs (simple ANNs-operations can produce large NDMs and ANNs). Meanwhile, in AE the compression is only possible when converting AEPs into NDMs (repeated use of information included in chromosomes). The total size of chromosomes-operations and chromosomes-data representing AEPs is always the same as the size of the programs. Second, effective ANNs-operations can usually be represented in the form of shorter genotypes than their counterparts from the pure AE. To obtain an effective ANN, the latter operations need typically many different data. This is because they simply copy data into matrices. NDMs in this case can only contain values which occur in the data part of AEP. The more data is included in the program, the greater is the repertoire of ANNs which can be produced. Meanwhile in AEEO, operations are simple ANNs whose output signals do not depend on the length of their chromosomes and values memorized therein. In consequence, even simple ANNs encoded in short chromosomes can demonstrate very complex behaviour.

A deeper analysis of the architecture of ANNs produced by AE and AEEO also revealed that they, in most cases, contain regularities, that is, repetitions of architectural elements such as neurons and weights (see Figs. 17, 18 and 19). The differences between individual methods relate to grounds of regularity and its frequency. Whereas regularity in ANNs evolved according to AE is strictly connected with the compression and repeated use of the same data by AEPs, in AEEO, regularity is achieved by using sinusoid, cosinusoid and radial neurons in ANN-operations. Comparing roughly the frequency of repetitions in ANNs it is necessary to state that it was greater in ANNs produced in the traditional way. To generate complex ANNs, short classical

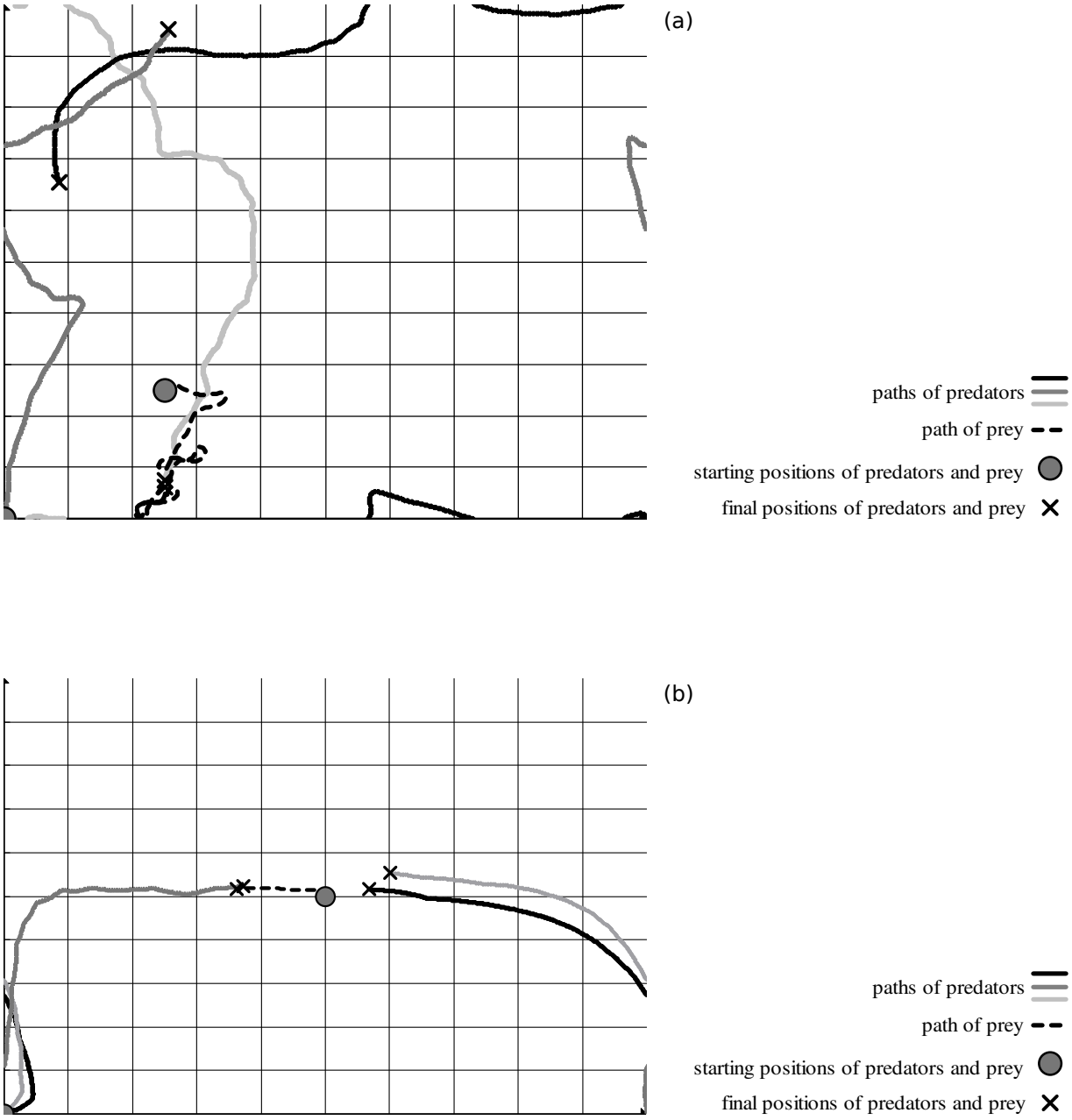


Fig. 20. Example strategies of predators to capture prey

AEPs had to repeatedly use the same data, whereas their counterparts equipped with ANN-operations could achieve the same effect without resorting to repetitions.

As for strategies of the chase implemented in ANNs, it turned out that they are in most cases very similar, regardless of the applied NE method. To capture the prey, the predators successfully cooperated together, their moves were well organized and synchronized. All the compared methods managed to evolve ANNs whose predators performed their task quickly and efficiently, without unnecessary moves and wandering. When chasing the prey the predators were usually, divided into two separate groups. The first two-member group was responsible for pushing the prey towards the second group consisting of one predator. The task of this predator was to ultimately capture the prey.An example of a typical behaviour of the predators is depicted in Fig. 20.

It seems that restrictive conditions of a chase were the main cause of resemblance between strategies evolved by different NE methods. In order for an ANN to obtain maximum fitness it had to perform all the learning tasks. A part

of the tasks required quick decisions and actions from the predators. There was no room for delays and unnecessary operations. Since the predators had to capture the prey as quickly as possible, they in most cases used very similar strategy close to the optimal one.

VI. SUMMARY

The paper presents a new NE method called Assembler Encoding with Evolvable Operations. The method was compared with its prototype method, i.e. AE, and with NEAT, on a variant of the predator-prey problem in which underwater vehicles played the role of the predators and the prey. During experiments, the task of all the compared methods was to evolve ANNs responsible for controlling a team of AUVs-predators whose common goal was to capture AUV-prey behaving by a simple deterministic strategy. The comparison tests showed that AEEO outperforms both rival methods in terms of the learning ability and phenotypic and genotypic complexity. As it turned out, ANNs evolved according to AEEO were both more effective and phenotypically and genotypically simpler.

The experiments also revealed that AEEO3 is the most effective AEEO variant. The method for cooperation between ANN-operations applied in this variant led to the most successful ANNs.

Future research will be focused on the application of AEPs with ANNs-operations as final solutions to a problem. In this case, the task of ANNs-operations will not be to construct NDM representing a single ANN, but to fill in a simple vector which will constitute the output of the whole system. ANNs-operations will collectively decide about the content of the vector based on input information provided from the outside. The above-mentioned solution can be viewed as a modular ANN whose separated modules will not be stiffly associated with specific outputs or inputs. In the considered solution, ANNs-operations will cooperate to perform a task. However, in contrast to many other modular systems, the way of cooperation between the modules will not be imposed by a designer. In this case, it will be the role of the evolution.

APPENDIX – PARAMETER SETTING IN THE EXPERIMENTS

AE and AEEO

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
hidden neurons in ANNs-operations = 4
type of neurons in ANNs-operations = sigmoid, radial, linear, sinusoid,cosinusoid
no. of subpopulations = $2 \div 10$ (AE), $1 \div 10$ (AEEO)
size of subpopulations = data:100, operations:50 (AE), 50 (AEEO)
no. of integer or binary genes in chromosomes = data: $20 \div 40$, operations:5 (AE), $7 \div 30$ (AEEO)
probability of mutation = data:0.01, operations:0.15 (AE), 0.05 (AEEO)

NEAT

number of evolutionary generations=60 000
trait_param_mut_prob=0.5
trait_mutation_power=1.0
linktrait_mut_sig 1.0
nodetrait_mut_sig 0.5
weight_mut_power 1.0
recur_prob 0.0
disjoint_coeff 1.0
excess_coeff 1.0
mutdiff_coeff 7.0
compat_thresh 9.0
age_significance 1.0
survival_thresh 0.4
mutate_only_prob 0.05
mutate_random_trait_prob 0.1
mutate_link_trait_prob 0.01
mutate_node_trait_prob 0.01
mutate_link_weights_prob 0.2
mutate_toggle_enable_prob 0.0
mutate_gene_reenable_prob 0.00
mutate_add_node_prob 0.005
mutate_add_link_prob 0.005
interspecies_mate_rate 0.01
mate_multipoint_prob 0.4
mate_multipoint_avg_prob 0.4
mate_singlepoint_prob 0.0
mate_only_prob 0.2
recur_only_prob 0.0
pop_size 200
dropoff_age 15
newlink_tries 20
print_every 60
babies_stolen 0
num_runs 1

References

[1] T. Aaltonen, et al., *Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF*, Physical Review Letters (2009).

[2] B. Allen, P. Faloutsos, *Complex networks of simple neurons for bipedal locomotion*, In IEEE/RSJ International Conference on Intelligent Robots and Systems (2009).
[3] 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).
[4] J. Gauci, K. Stanley, *Generating large-scale neural networks through discovering geometric regularities*, In Proceedings of the Genetic and Evolutionary Computation Conference, pp. 997-1004 (2007).
[5] D. E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Addison Wesley, Reading, Massachusetts, 1989.
[6] F. Gruau, *Neural network Synthesis Using Cellular Encoding And The Genetic Algorithm*, PhD Thesis, Ecole Normale Superieure de Lyon 1994.
[7] H. Kitano, *Designing neural networks using genetic algorithms with graph generation system*, Complex Systems **4**, 461-476, (1990).
[8] K. Krawiec, B. Bhanu, *Visual Learning by Coevolutionary Feature Synthesis*, IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics **35**, 409-425 (2005).
[9] T. Kubaty, L. Rowinski, *Mine counter vehicles for Baltic Navy*, Internet, `http://www.underwater.pg.gda.pl`.
[10] J. Lehman, K. O. Stanley, *Abandoning Objectives: Evolution through the Search for Novelty Alone*, Evolutionary Computation **19**, 189-223 (2011).
[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, *Modular networks in Assembler Encoding*, Computational Methods in Science and Technology **14**(1), 27-38 (2008).
[19] T. Praczyk, *Using assembler encoding to solve inverted pendulum problem*, Computing and Informatics **28**, 895-912 (2009).
[20] T. Praczyk, *Forming Neural Networks by Means of Assembler Encoding*, Intelligent Automation and Soft Computing **17**(3), 319-331 (2011).
[21] T. Praczyk, P. Szymak, *Decision System for a Team of Autonomous Underwater Vehicles - Preliminary Report*, Neurocomputing, doi:10.1016/j.neucom.2011.05.013.
[22] T. Praczyk, *Solving the pole balancing problem by means of Assembler Encoding*, Journal of Intelligent and Fuzzy Systems **26**, 857-868 (2014).
[23] T. Praczyk, *Diverse neural architectures in Assembler Encoding*, Computational Methods in Science and Technology **20**(1), 21-34, (2014).
[24] K. O. Stanley, R. Miikkulainen, *Evolving neural networks through augmenting topologies*,Evolutionary Computation **10**, 99-127, (2002).
[25] K. O. Stanley, R. Miikkulainen, *Competitive coevolution through evolutionary complexification*, Journal of Artificial Intelligence Research **21**, 63–100 (2004).

Tomasz Praczyk graduated from the Cybernetics Faculty at the Warsaw Military University of Technology in 1996. In 2002, he defends his doctoral thesis on using neural networks to identify ship radio stations. In 2002-2006, he still deals with the identification of the ships and using immune systems for that purpose, moreover, he is also interested in maritime automatic spare positioning systems. The subsequent years were working on a new neuro-evolutionary method called Assembler Encoding and its improvements. The method and previous scientific achievements were the basis for obtaining in 2013 postdoctoral degree at the Cybernetics Faculty of the Military University of Technology.