Control and Cybernetics

vol. **42** (2013) No. 1

Using Assembler Encoding to build neuro-controllers for a team of autonomous underwater vehicles*

by

Tomasz Praczyk[1], Piotr Szymak[2]

Polish Naval Academy,
[1]Institute of Naval Weapon,
[2]Institute of Electrical Engineering and Automatics
Gdynia, Poland
t.praczyk,p.szymak@amw.gdynia.pl

Abstract: The paper compares a neuro-evolutionary method called Assembler Encoding with two other methods from the area of neuro–evolution. As a testbed for the methods a variant of the predator–prey problem with Autonomous Underwater Vehicles (AUV) operating in an environment with the sea current was used. In the experiments, the task of vehicles–predators controlled with evolutionary neural networks was to capture a vehicle–prey behaving according to a simple deterministic strategy. All the experiments were carried out in simulation, and in order to simplify calculations in the two–dimensional environment – AUVs moved on a horizontal surface under the water.

Keywords: evolutionary neural networks, autonomous underwater vehicles

1. Introduction

Artificial Neural Networks (ANNs) are a subdomain of artificial intelligence broadly used to solve various problems in different fields (e.g. pattern classification, function approximation, optimization, image compression, associative memories, robot control problems etc.). To build an ANN it is necessary to determine its topology and parameters (typically weights). There are many different ANN learning algorithms (e.g. BackPropagation) that change the values of parameters, leaving the topology completely intact. In such a case, the process of searching for a proper network topology is the task of a network designer, who arbitrarily chooses the network structure, starts network learning and finally puts the network to a test. If the result of the test is satisfactory, the learning process is stopped. If not, it is continued further. The designer manually determines the next potential network topology and runs the learning

*Submitted: October 2010; Accepted: January 2013.

algorithm again. Such a loop - topology determination and learning - is repeated until a network capable of carrying out a dedicated task at an appropriate level is found. At a first glance, it is apparent that such a procedure could be very time-consuming and, even worse, in the case of more complex problems, can lead to a situation when all chosen and trained networks would be incapable of solving the task.

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

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

There are a lot of NE methods (e.g. Cangelosi, Parisi and Nolfi, 1994; Gruau, 1994; Kitano, 1990; Luke and Spector, 1996; Miller, Todd and Hedge, 1989; Moriarty, 1997; Nolfi and Parisi, 1992; Stanley, 2004). 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. Hence, to encode larger networks larger chromosomes are necessary, which is the main drawback of the direct methods. As regards the indirect methods, we deal with chromosomes which are recipes how to create a network. Such encodings can be used to create larger neural architectures by means of relatively short chromosomes.

The paper presents a new indirect NE method called Assembler Encoding (AE). AE originates from the cellular (Gruau, 1994) and edge encoding (Luke and Spector, 1996), although, it also has features common with Linear Genetic Programming presented, in particular, in Krawiec and Bhanu (2005) and Nordin, Banzhaf and Francone (1999). In AE, ANN is represented in the form of a program (Assembler Encoding Program - AEP) whose structure is similar to that of the structure of a simple assembler program. AEPs are formed by means of GA. The task of each AEP is to create a Network Definition Matrix (NDM) which includes all the information necessary to create a network. In

AE, the process of ANN construction consists of three stages. First, GA is used to produce AEPs, next, each AEP creates and fills up NDM, and finally, the matrix is transformed into ANN.

To date, AE was tested in three different problems, i.e. in the optimization problem (Praczyk, 2007), in the predator-prey problem (Praczyk, 2007, 2008, 2010), and in the pole balancing problem (Praczyk, 2009). In all the tests, the method demonstrated fairly good effectiveness. It is worth noting that it successfully competed with different NE and reinforcement learning methods in problems which rather prefer the latter of them (Praczyk, 2009). All the prior tests did not, however, show the true abilities of the method to form complex neuro-controllers. To obtain a deeper knowledge of the capabilities of AE in this respect, the decision was taken to apply AE to produce neuro-controllers for a team of cooperating autonomous underwater vehicles (AUVs). Since the AUV controllers have to take into account not only the common goal of the vehicles but also such factors as their inertia, maneuverability, and the current in the see, their construction is a complex problem which can constitute an attractive testbed for such methods as AE.

The first attempts to combine AE with AUVs are reported in Praczyk and Szymak (2011). The paper mentioned presents experiments in which AUVs dealt with an ideal sea environment without the sea current. In such conditions, it appeared that AE have no serious problems with evolving effective AUV controllers. To test AE in a more realistic and more difficult conditions, that is, in presence of a variable sea current in the environment, subsequent experiments were carried out, whose results are shown in the current paper. Except for the influence of the sea current on AUVs, other settings of the experiments were almost the same as those in Praczyk and Szymak (2011). That is, the task of AE was to produce a Decision System (DS*) for a team of AUV-predators whose common goal was to capture an escaping AUV-prey behaving according to a simple deterministic strategy. Since the speed of each predator was lower than or equal to the speed of the prey, the predators had to cooperate to accomplish the goal. As in Praczyk and Szymak (2011), to compare AE with other NE methods, two variants of the classical Connectivity Matrix (CM), Miller, Todd and Hedge (1989), were also used in the experiments.

The paper is organized as follows: Section 2 is a presentation of the compared methods, Section 3 is a description of conditions of the experiments, Section 4 is a report on experimental results, and Section 5 is a summary.

2. Methods

2.1. Assembler Encoding

Because the detailed description of AE is given in Praczyk (2010), here, only a short outline of the method is presented. In AE, ANN is represented in the form

*the task of DS is to provide high-level decisions concerning direction and velocity of move for each vehicle

of a program called Assembler Encoding Program (AEP). AEP is composed of two parts, i.e. a part including operations and a part including data. The task of AEP is to create and fill in Network Definition Matrix (NDM) with values. To this end, AEP uses the operations whose implementations are defined by the designer. The operations are run in turn. They can use data located at the end of AEP (Fig. 1). Once the last operation terminates, the process of creating NDM is completed. NDM is then transformed into an ANN.

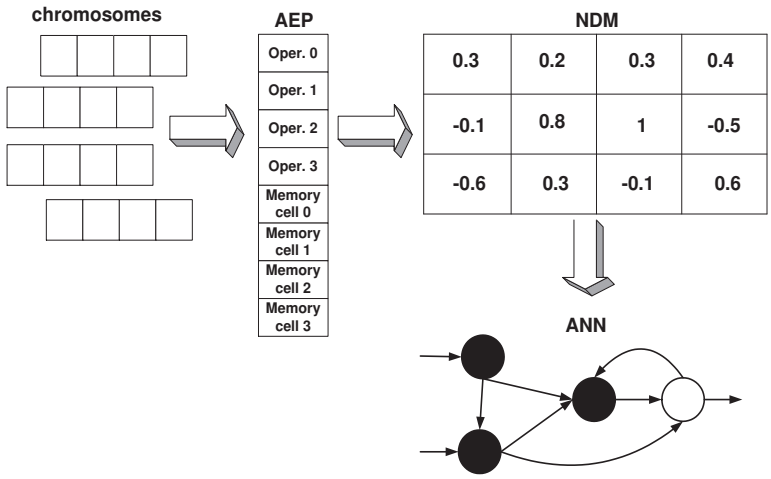


Figure 1. Using AE to create ANN (Praczyk, 2011)

AEPs can use various operations with predefined implementations. The task of most of them is to modify NDM by introducing new values from the data part of the program into the matrix. The modification can involve a single element of NDM or group of elements, e.g. a fragment of a column or row. In addition to the operations whose task is to modify the content of NDM, AEPs can also be equipped with a jump operation which makes it possible to repeatedly use the same code of AEP in different places of NDM. An additional possibility for AEPs is to use operations whose task is to change the size of NDM, and, in consequence, the size of the resultant ANN.

Once AEP finishes its work, the process of transforming NDM into an ANN is started. To make it possible to construct ANN based on NDM the latter has to include all the information necessary to create ANN. When we wish to create only the skeleton of ANN, i.e. ANN without determined weights of interneuron connections, NDM can take the form of the classical connectivity matrix (CM), Miller, Todd and Hedge (1989), i.e. a square, binary matrix with the number of rows and columns equal to the number of neurons. The value "1" in i^{th} column and j^{th} row of such a matrix means a connection between i^{th} neuron and j^{th} neuron. In turn, "0" means lack of connection between these neurons. When the purpose is to create a complete ANN with determined values of weights,

types of neurons, parameters of neurons, then NDM should take the form of a real valued variety of CM with extra columns or rows containing definitions of individual neurons. The example of such a matrix is presented in Fig. 2.

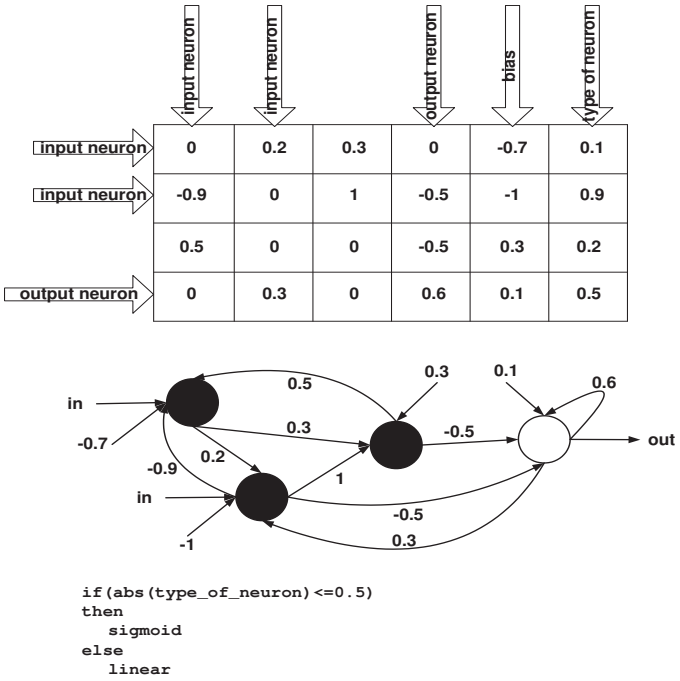


Figure 2. NDM as Connectivity Matrix (Praczyk, 2009)

The evolution of AEPs proceeds according to Cooperative Co-Evolutionary Genetic Algorithm (Potter, 1997; Potter, De Jong, 2000). It assumes a division of evolutionarily created solution into parts. Each part evolves in a separate population. A complete solution is formed out of selected representatives of each population. In AE, an 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 one operation and a sequence of data. The operations and the data come from two different populations. When the evolution stagnates, the set of the populations containing the operations is enlarged by one population. This procedure extends all AEPs by one operation.

2.2. Conventional neuro-evolution

CNE is a classic direct NE method in which a single chromosome includes all the information necessary to create an ANN (the location of a gene in the chromosome strictly determines a parameter of the ANN encoded by the gene). ANNs

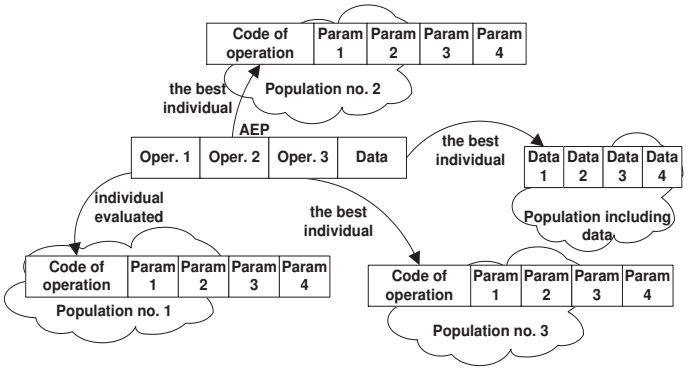


Figure 3. Evolution in AE for $n = 3$ (Praczyk, 2011)

evolve in a single population. The evolution in the population proceeds according to Canonical GA (Goldberg, 1989). Chromosomes with encoded ANNs are in the form of binary strings. Each chromosome encodes weights of interneuron connections and parameters of neurons. Evolution of ANNs in CNE is presented in Fig. 4 (Praczyk, Szymak, 2011).

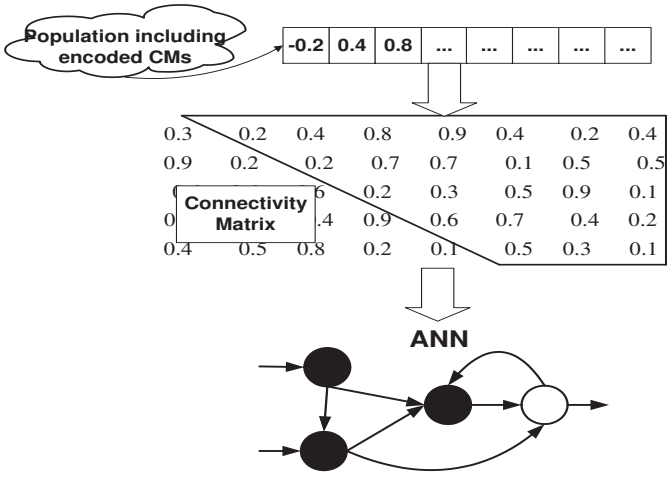


Figure 4. Evolution of ANNs in CNE (evolving elements of CM are enclosed) (Praczyk, Szymak, 2011)

2.3. Neuro-CoEvolution

NCoE is a variant of CNE. In CNE, ANNs evolve in a single population. Each chromosome from the population represents a single ANN. In NCoE, we deal

with a different situation. Each ANN created by means of NCoE evolves in a few different populations. The populations include chromosomes which define different elements of ANNs (weights of interneuron connections and parameters of neurons). As before, Canonical GA is used to evolve ANNs. Each ANN created during the evolution is evaluated once per generation. Evolution of ANNs in NCoE is presented in Fig. 5 (Praczyk, Szymak, 2011).

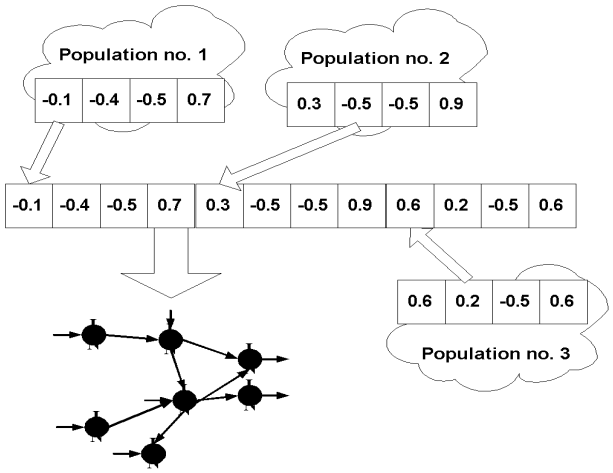


Figure 5. Evolution of ANNs in NCoE (ANNs evolve in three populations)(Praczyk, Szymak, 2011)

3. Experiments

The experiments were carried out in almost the same conditions as those described in Praczyk, Szymak (2011). The only differences were the presence of the current in the environment and application of a prey with a more advanced escaping strategy. The additional elements increased the complexity of the problem solved by ANNs compared to previous research.

All the experiments were conducted in simulation with application of mathematical model of a Remotely Operated Vehicle (ROV) of the type "Ukwial" (see Fig. 6), Kubaty, Rowiski (no date) and they were divided into two phases. In the first phase called a construction phase, all the compared methods were used to prepare ANNs. Each NE method was run many times for different parameter settings. For each method many different ANNs were produced. Selected ANNs (thirty most effective ANNs for each NE method) were then tested in the following phase of the experiments, i.e. in a generalization phase. The purpose of this phase was to test the effectiveness of all selected ANNs on tasks which were not presented to them before.

Figure 6. Vehicle "Ukwial" (Praczyk, Szymak, 2011)

3.1. The predator-prey problem

All the tests were carried out in a configuration with one prey and three chasing predators. Both the predators and the prey were implemented as ROV "Ukwial" (the vehicle controlled by ANN became AUV). The behavior of all the vehicles was simulated by means of a discrete time model defined in Section 3.3.

In the experiments, the predators and the prey functioned in a common artificial environment. To represent the environment, the square of 100x100 meters was used (see Fig. 7, to simplify calculations, simulations took place in the two–dimensional environment – AUVs 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 manoeuvres, 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 (Praczyk, Szymak, 2011).

In the experiments, the predators were controlled by a single ANN whose task was to determine movement direction for each of them. At each time step ANN decided about the change of a current course of each vehicle. The course could be changed by 0,5,10, ... ,355 degrees (it is necessary to note that decisions of ANN determined only the final state of the vehicles which they ultimately should reach, the real course after the maneuvre and duration of the complete maneuvre depended on current parameters of each vehicle). The speed of the predators was constant during the tests and amounted to 0.5 m/s (Praczyk, Szymak, 2011).

Unlike in Praczyk, Szymak (2011) the prey behaved according to two strategies: simple and advanced. The strategy of the simple prey, the same as the one

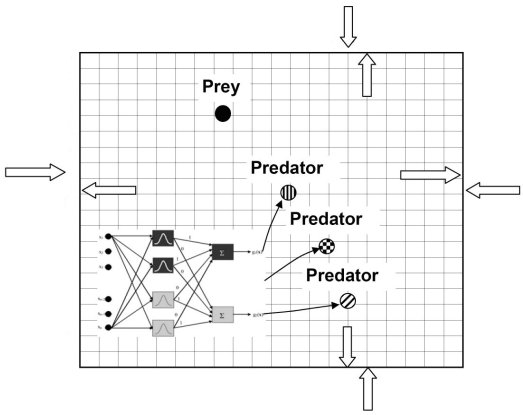


Figure 7. Artificial world for predators and prey (Praczyk, Szymak, 2011)

applied in the previous experiments, 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 the 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 being 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.

When moving, the prey could select the same actions as the predators. The speed of the preys amounted to 0.5, 0.75 or 1 m/s. Since speed of the predators was either lower or at most the same as speed of the escaping prey, they could not simply chase the prey to grasp it. We assumed that the prey was captured if the distance between it and the nearest predator was lower than 5 meters.

In the experiments, the vehicles-predators had to counteract a variable sea current. With regard to the vehicle-prey, the assumption was made that the current does not affect its behavior. In effect, the predators were pushed outside the path determined by ANN whereas the prey always moved accurately according to its strategy. Since the direction of the current was variable, in some testing scenarios described in Section 3.4, the predators were pushed left whereas in other ones they were pushed right (during a single scenario the current did not change the direction). The goal of such a move was to make conditions of the experiments maximally similar to the ones in which real vehicles are used. Of course, the ideal situation for the vehicles is when they always work over the sea areas with the same characteristic. However, they should also be prepared to work in basins in which parameters of the current are variable and depend

on location of the vehicles, season, and time of the day.

To help ANNs to deal with variable current, each of them was supported by the Sea Current Compensation System (SCCS) described in the following section. The task of SCCS was to correct decisions of ANNs taking into account the current occurring in the environment.

Generally, the current always had a main direction which, as mentioned above, was invariable within a single scenario. However, to model random fluctuations of the sea current, its momentary direction slightly differed from the main direction. The consequence of the current randomness was difference in behavior of the predators for the same decisions of ANNs. This led to situations, in which effectiveness of the same decisions taken in the same circumstances could differ. Random effectiveness of individual decisions could also result in random effectiveness of the entire ANNs. In the same scenario, one time, ANN could be effective but some other time it could also completely fail. Such situation made evaluation of ANNs constructed during the evolutionary process, and in consequence the evolution itself very difficult.

3.2. Decision systems

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.

When controlling AUVs, ANNs had to take into account not only a common goal of the vehicles but also their inertia, maneuverability and the sea current. They had to adopt the strategy of AUVs to how fast they are able to perform a given maneuvre and how much place they need for that purpose. When giving commands they should know that AUVs cannot turn at once and they need time to reach a prescribed state.

To cope with the sea current, ANNs were supported by SCCS (see Fig. 8). The task of SCCS was to transform decisions of ANNs made for an ideal environment without the sea current into decisions considering the current. To this end, the system first estimates the parameters of the current, and then corrects decisions of an ANN based on these estimates. The estimates are weighted averages of momentary values of current parameters. Since the current has a variable nature and it can change depending on time of the day, season, or location of vehicle, the system should have the ability to adopt to variable underwater conditions. To accomplish this, the most recent values of the current parameters exert greater influence on the resultant estimates than older values. The momentary values of the current parameters are approximations of true values. They are based on true values of vehicle parameters measured at a given point in time and desirable values which the vehicle should reach at the same point assuming the underwater environment without the current. To calculate the desirable values of the vehicle parameters, the model of the vehicle described

in the following section is used.

```
double SCCS::run(Δψ_d^{t+1}, x^{t+1}, y^{t+1})
    begin
    //Model of vehicle without current is used to calculate
    //expected change in position after maneuver made in t
    (Δx_d, Δy_d) = VehicleModel.getChangeInPosition(ψ_V^t);

    (x_d^{t+1}, y_d^{t+1}) = (x^t, y^t) + (Δx_d, Δy_d);

    ψ_c^t = getCourseFromPointToPoint(x_d^{t+1},y_d^{t+1},x^{t+1},y^{t+1});

    ~ψ_c = (Σ_{i=t-n}^{t} W_i^t ψ_c^i) / (Σ_{i=t-n}^{t} W_i^t), W_i^t = exp(-||t-i||^2 / 2σ^2);

    //⊕ - adds two vectors
    //getCourse(a) - returns course for vector a
    //getVector(a,b) - returns vector of length a and direction b
    ψ_V^{t+1} = getCourse(getVector(V_v, ψ^t + Δψ_d^{t+1}) ⊕ getVector(V_c, -~ψ_c));

    //if Δψ_V > 0 vehicle turns right, otherwise it turns left
    Δψ_V^{t+1} = getChangeInCourse(ψ_V^{t+1},ψ_V^t);

    return Δψ_V^{t+1};
    end
```

Figure 8. Pseudocode of SCCS supporting a single vehicle moving with constant speed; x^t, y^t - position of vehicle at time t, x_d^t, y_d^t - expected position of vehicle at time t, ψ_d^t - desirable course of vehicle at time t, this value is produced by DS and passed on to SCCS, ψ_V^t - course for vehicle after compensation of current at time t, this value is produced by SCCS and passed on to low–level controllers of the vehicle, ψ_c^t - momentary course of current at time t, $\widetilde{\psi_c}$ - estimated course of current ($-\widetilde{\psi_c}$ - opposite course to $\widetilde{\psi_c}$), n - number of successive measurements of current course used to estimate it direction, V_v - velocity of vehicle, V_c - velocity of current

3.3. Vehicles

Since all the experiments were carried out in simulation, behavior of the vehicle "Ukwial" had to be modeled appropriately. Usually, to simulate movement of the vehicle, a nonlinear model described in six degrees of freedom is used (Fossen, 1994). Moreover, to control the vehicle (along a fixed path), several nonlinear controllers of motion parameters are needed see Szymak (2006) (the task of the

controllers is to convert high-level decisions provided by the ANN into low-level ones for propellers of the vehicle). In consequence, the simulation of the vehicle with use of the models and controllers mentioned above inevitably involves time consuming calculations. In the case of experiments with a single vehicle, such an approach seems to be justified. However, when simulating many vehicles, a different solution has to be applied. Since in our experiments NE methods were used, which test many different neural solutions per evolutionary generation, to speed up calculations it was necessary to employ another method for simulating the vehicle. Especially for the purposes of the experiments, a simplified model representing both the vehicle and its controllers was devised, as described in Praczyk, Szymak (2011) and Szymak (2010).

Since the model mentioned above does not take into account the sea current, in the experiments, successive positions of the vehicle fixed with the model were shifted according to a momentary direction (ψ_c) and velocity (V_c) of the current: $x = x + V_c \Delta t_m \cos(\psi_c),\ y = y + V_c \Delta t_m \sin(\psi_c)$, where ($\Delta t_m$) is duration of a single maneuvre of the vehicle.

3.4. Evaluation of ANNs

In order to evaluate ANNs generated during the experiments, 90 different testing scenarios were produced. The first 30 scenarios were used in the ANN construction phase. The remaining ones were applied in the generalization phase to evaluate the prepared ANNs in terms of their capability to generalize knowledge acquired during the construction phase. The evaluation of both prepared and unprepared ANNs proceeded in the following way. At first, each of them was tested in the simplest scenario, say, no. 1. If the predators could not capture the prey during some assumed period (800 s), the test was stopped and ANN received appropriate evaluation that depended on the distance between the prey and the nearest predator. However, if the predators grasped the prey, ANN obtained appropriate reward and a next scenario was run (Praczyk, Szymak, 2011).

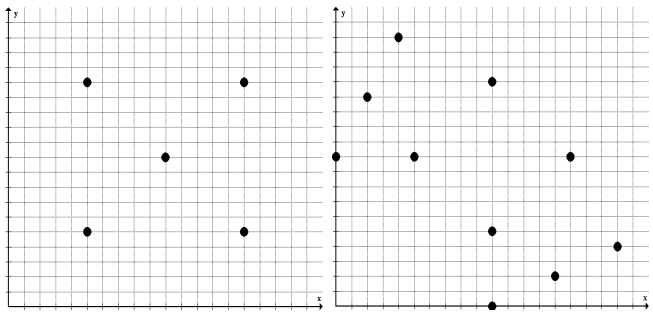
The scenarios used in the experiments differed in initial position, speed (0.5, 0.75 or 1 m/s) and type of the prey (simple or advanced), and additionally in the direction of the sea current affecting only the predators. The range of vision of each prey amounted to 50 m. With regard to the sea current, its speed was constant in all the scenarios and amounted to 0.25 m/s whereas direction of the current was variable and had a random nature. Generally, the current always had a main direction different for individual scenarios and invariable within a single scenario. To model random fluctuations of the sea current, the momentary direction of the current slightly differed from the main direction. The magnitude of the difference was random and it amounted maximally to $\pm 3^\circ$.

Consecutive scenarios were more and more difficult. Initially, the predators had to capture the slowest simple prey. The predators, which passed the first exam, had to pit against the prey one and a half times faster than the predators.

In the next step, the speed of the prey was increased to the maximum value. The predators which captured the prey in all the previous scenarios had to face the advanced prey. In the beginning, the prey moved with the minimum speed. The speed of the prey was increased in the following scenarios. In all the scenarios starting positions of all three predators were the same, namely (0,0). All the scenarios are described in Table 1.

Table 1. Description of scenarios used in the experiments

no. of scenario	prey			sea current	
	speed [m/s]	type	initial positions	speed [m/s]	direction [deg]
1-5	0.5	simple	Fig. 9(a) (learning scenarios)	0.25	0
6-10	0.75	simple		0.25	180
11-15	1	simple		0.25	90
16-20	0.5	advanced		0.25	270
21-25	0.75	advanced		0.25	45
26-30	1	advanced		0.25	315
31-40	0.5	simple	Fig. 9(b) (generalizing scenarios)	0.25	135
41-50	0.75	simple		0.25	225
51-60	1	simple		0.25	20
61-70	0.5	advanced		0.25	160
71-80	0.75	advanced		0.25	200
81-90	1	advanced		0.25	250



(a) (b)

Figure 9. Starting positions of prey used in learning (a) and generalizing (b) scenarios

In all the experiments, the following evaluation function (or fitness function) was used (Praczyk, Szymak, 2011):

$$f(ANN) = \sum_{i=0}^{n} f_i \tag{1}$$

$$f_i = \begin{cases} d_{max} - \min\limits_{p} d_i(p), \text{ prey not captured in } i^{th} \text{ scenario} \\ f_{captured} + (80 - m_i)/a, \text{ prey captured in } i^{th} \text{ scenario} \\ 0, \text{ prey not captured in previous scenario} \end{cases} \quad (2)$$

where
f_i - reward received in i^{th} scenario
$d_i(p)$ - distance between the prey and predator p in the end state of i^{th} scenario
d_{max} - maximum distance between two points in the environment
$f_{captured}$ - reward for grasping the prey in a single scenario ($f_{captured} = 100$)
m_i - number of steps to capture the prey ($m_i < 80$)
a - this value prevents the situation in which partial success is better than success in all scenarios
n - the number of scenarios (the construction phase: $n = 30$, the generalization phase: $n = 60$).

3.5. Variants and parameters of NE methods

In the experiments, different variants of the NE methods were tested. The variants mentioned mainly differed in parameters of the evolutionary process (Table 2). Moreover, individual variants of AE also varied in operations used in AEPs. In this case, two different solutions were applied. The first of them assumed AEPs exclusively with operations `CHGFF`. In the second one, AEPs could use seven different operations, i.e. `CHGFF, CHGC0, CHGC1, CHGR0, CHGR1, CHGM0, CHGM1`. A short description of all the operations is included in Appendix A at the end of the paper.

Table 2. Parameters of AE, CNE and NCoE

Parameter	AE	CNE	NCoE
no. of subpopulations	variable (2..6)	1	2, 4 or 6
size of subpopulations	80 (data), 40 (operations)	120	60, 30 or 20
evaluations per generation	different	120	120
max no. of generations	60 000		
no. of data	maximally 20	X	X
hidden neurons in ANNs	maximally 8		
type of neurons in ANNs	sigmoid		
size of tournament	1 (data), 2 (operations)	2 or 4	2 or 4
crossover probability	0.7		
mutation probability	0.01-0.03 (data), 0.01-0.06 (operations)	0.005-0.1	0.005-0.1

4. Experimental results

The first activity during the tests was to tune each NE method to the problem solved. To this end, each of them was run for different parameter settings (see

Table 2). Individual settings differed in mutation, size of tournament (tournament selection) and number of sub-populations. For each setting fifty evolutionary runs were carried out, each of which produced a single ANN. To find the most effective parameter setting, average fitnesses of ANNs generated for individual settings were compared. A setting with the highest average was regarded as the optimal setting for a given method. During the tuning tests, ANNs dealt exclusively with the learning scenarios.

To compare all the methods, effectiveness of their ANNs in the learning and generalizing scenarios were taken into account. In the first case, learning abilities, i.e. abilities to build solutions effective in learning tasks were compared. In the second case, ANNs being the final product of the "learning" process were tested on tasks which were not presented to them before. It this way, abilities to generalize knowledge acquired during the learning process were measured. To compare the learning abilities of NE methods, fitnesses of ANNs produced for the optimal parameter settings were compared. In the generalization tests, fitnesses of 30 best ANNs generated by each method were compared. The best ANNs were selected out of all ANNs produced by each method. Parameter settings were unimportant in this case.

Table 3. Comparison of learning abilities

	AE	CNE	NCoE
% of fully effective ANNs	22%	8%	16%
average fitness of ANNs	2523.6	2043.1	2543.8
max fitness	3021.5	3024.2	3022.1
min fitness	924.7	1022.2	1233.5

Table 4. Comparison of generalization abilities

	AE	CNE	NCoE
% of fully effective ANNs	13.3%	0%	0%
average fitness of ANNs	3921.2	1132.5	2566.3
max fitness	6038.1	3937.9	4535.2
min fitness	504.0	403.2	306.5

The results of the experiments summarized in Tables 3 and 4 show superiority of AE over the remaining methods. AE turned out to be the most effective in both the learning and generalization testes. This is particularly noticeable when comparing percentages of fully effective ANNs produced by each method. The fully effective ANN is an ANN whose strategy enables the predators to capture the prey in all 30 learning scenarios. In the case of AE and the learning tests,

22% out of all the 50 ANNs appeared to be fully effective. For CNE and NCoE this rate amounts to 8% and 16%, respectively. In the generalization phase, a disproportion between CNE, NCoE and AE is even higher. This time, the percentages of fully effective ANNs are the following: 13.3% - AE, 0% - CNE and NCoE.

In the learning phase, the direct nature of CNE and NCoE seems to be the main cause of their worse performance compared to AE. Since each ANN produced in the experiments could have maximally 8 hidden neurons (see Table 2), CMs representing such networks were of size 17x19 (17=9 inputs and outputs + 8 hidden neurons, 19=17+2 additional columns for parameters of neurons). Given that all ANNs had a feed-forward architecture, to define them, only fragments of CMs above the diagonal were used. Each such fragment included 170 parameters of an ANN, in total. In the case of the direct methods, all the parameters had to be encoded in chromosomes. In CNE, a single chromosome encoded all the 170 parameters. Since during all the experiments, each parameter of an ANN was represented in the form of 8-bit binary string, the length of chromosomes in CNE was 1360 bits. The shortest chromosomes in NCoE (for the evolution proceeded in 6 populations) encoded 28 parameters (to define a single CM, 5 chromosomes of length 28 and 1 chromosome of length 30 were necessary). In other settings, chromosomes in NCoE encoded 42 or 85 parameters. Changing all the figures above into bits we obtain chromosomes of the lengths: 224, 336, and 680 bits. Meanwhile, in AE, all operations were of length 5 (40 bits) whereas data could maximally include 20 elements (160 bits). Generally, in all the experiments, chromosomes in CNE and NCoE were, usually, much longer than those in AE. In effect, the evolution in both direct methods would face a more difficult task than in the case of AE.

As for the generalization abilities of ANNs, it appeared that they mainly depend on the complexity of the networks. ANNs produced by means of CNE and NCoE were usually fully-connected and in addition they included the maximum number of neurons. Meanwhile, ANNs evolved by AE were in most cases simpler in terms of construction, that is, they contained fewer connections and neurons than their rival ANNs. As it turned out, simpler ANNs were, usually, efficient in both phases of the experiments whereas more complex ones, in the second phase, had problems with overfitting, and in consequence with poor generalization.

Again, the direct nature of CNE and NCoE seems to be the main cause of the complexity of their ANNs. Chromosomes in the direct methods include encodings of each parameter of an ANN. In consequence, to eliminate some elements (connections or neurons) from a network, the corresponding genes in the chromosomes have to be equal to zero. Since zero is only one out of many values which can be memorized in the chromosomes, the direct methods often have great problems with evolving simpler architectures than the maximum ones.

In turn, in AE, each operation, usually, modifies only a little fragment of NDM which causes these matrices to often include many elements equal to zero.

Such elements indicate lack of some connections and neurons in the resultant ANNs. Fewer connections and neurons in ANNs makes them, in turn, unaffected by the overfitting problem, and in consequence better adjusted to a task than the fully-connected networks evolving directly as CMs.

With regard to the influence of the sea current on the difficulty of the ANN task, and in consequence, on the results of the compared methods, it seems that it was significantly reduced by application of SCCS. As it turned out, estimates calculated by SCCS were very close to the real values of the current. Since the main direction of the current was invariable within a single scenario, it was quickly and accurately estimated by SCCS. Decisions of ANNs were, in principle, from the very start of each scenario faultlessly corrected by the system. It seems that slight random fluctuations of the momentary current had no greater influence on the effectiveness of ANNs and their AUVs.

5. Summary

The paper compares AE with CNE and NCoE on the predator-prey problem. The experiments reported in the paper revealed superiority of AE over the remaining methods used in the tests. As for CNE, two elements could affect the achieved results, i.e. evolutionary scheme (the evolution of CMs in a single population, very long chromosomes), and the direct nature of the method. NCoE appeared to be more effective than CNE. In the learning tests, NCoE generated almost as effective ANNs as AE. However, in the generalization ones, ANNs produced with NCoE appeared to be considerably less effective than ANNs generated by means of AE. It seems that the main cause of such a situation was the complexity of ANNs produced in both cases. ANNs constructed with NCoE were fully-connected and included the maximum number of neurons which made them overfitted to the learning tasks. Such construction of ANNs resulted from the direct nature of NCoE. Each direct method strives to adjust all the parameters of an ANN included in chromosomes to a problem. It seems that this feature of the direct methods makes them appropriate tool exclusively for constructing ANNs with an architecture known in advance. Using these methods with no knowledge about a desirable architecture of ANNs may cause neural solutions of maximum acceptable complexity to be built usually.

In AE, the architecture of ANNs is dependent on the construction of AEPs. In the beginning, AEPs include a single operation to which next operations are gradually added if AE cannot generate an effective ANN within some assumed period. Thus, AEPs and ANNs increase their complexity with the passage of time and the architecture of ANNs is adjusted to a task in a better way than in the case of the direct methods such as NCoE or CNE.

When comparing AE with the methods presented in the paper it is also necessary to mention the capabilities of AE that the remaining methods do not have. The additional capabilities of AE are a consequence of applying a program to represent an ANN. AEPs can encode not only parameters but also other aspects of ANN functioning. The example is the learning process in an

ANN. AEPs can include not only operations updating NDMs but also operations which can organize training of an ANN (Praczyk, 2008). Another example is the process of growth of an ANN. ANNs like humans can grow from the childhood to the maturity (Elman, 1993; Lang, 2000). In the meantime, they can learn. All the processes mentioned can be organized by AEPs (when to learn, how long, etc.).

6. Reference

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

Elman, J. L. (1993) Learning and development in neural networks: The importance of starting small. *Cognition*, 48, 71-99.

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

Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, Reading, Massachusetts.

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

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

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

Kubaty, T. and Rowiski, L. (no date) *Mine counter vehicles for Baltic Navy*. Internet, `http://www.underwater.pg.gda.pl`.

Lang, R.I.W. (2000) *A Future for Dynamic Neural Networks*. University of Reading, CYB/1/PG/RIWL/V1.0.

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

Miller, G.F., Todd, P.M. and Hegde S.U. (1989) Designing Neural Networks Using Genetic Algorithms. In: J.D. Schaffer, ed., *Proc. of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, 379-384.

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

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

Nordin, P., Banzhaf, W. and Francone, F. (1999) Efficient Evolution of Machine Code for CISC Architectures using Blocks and Homologous Crossover. In: L. Spector et al., eds., *Advances in Genetic Programming III*.

MIT Press, 275-299.

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

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

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

Praczyk, T. (2008) Modular networks in Assembler Encoding. *Computational Methods in Science and Technology, CMST* **14**(1), 27-38.

Praczyk, T. (2009) Concepts of learning in Assembler Encoding. *Archives of Control Science*, 18(3), 323-337 (2008)

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

Praczyk, T. (2010) Searching for optimal size neural networks in assembler encoding. *Control and Cybernetics* **39**(4), 1193-1215.

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

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

Stanley, O. (2004) *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.

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

Szymak, P. (2010) Simplified mathematical model of underwater vehicle and its control system (in Polish). *Pomiary, Automatyka i Robotyka*, 2/2010, Industrial Research Institute for Automation and Measurements, 372-379.

A. Appendix 1 - List of operations used in the experiments

`CHGFF` - Update of a fragment of NDM above the diagonal. New values for the elements of the matrix are located in the data part of AEP

`CHGC0` - Update of a fragment of a column of NDM. As before, new values for the elements of the matrix are located in the data part of AEP

`CHGC1` - like `CHGC0`, the difference is that all the updated elements have the same value

`CHGR0` - like `CHGC0`, the difference is that the update refers to the row of NDM

`CHGR1` - like `CHGC1`

`CHGMO` - Update of a block of elements in NDM. Elements are updated in columns, in turn, one after another. New values for the elements are located in data part of AEP
`CHGM1` - like `CHGMO`, the difference is that all the updated elements have the same value