COMPUTATIONAL METHODS IN SCIENCE AND TECHNOLOGY **18(1)**, 11-24 (2012)

Assembler Encoding Improved

Tomasz Praczyk

Polish Naval Academy, Institute of Naval Weapon, Gdynia, Poland
t.praczyk@amw.gdynia.pl

(Received: 03 February 2012; revised: 17 March and 30 April 2012; accepted: 02 May 2012; published online: 14 May 2012)

Abstract: Assembler Encoding is a neuro-evolutionary method which was used to produce a neural decision system for a team of autonomous underwater vehicles. Since results accomplished during experiments with the classic variant of Assembler Encoding appeared to be unsatisfactory, the method has been appropriately improved. The paper presents modifications to Assembler Encoding and reports experiments whose main goal was to test effectiveness of each of them.

Key words: evolutionary neural networks

I. INTRODUCTION

Assembler Encoding (AE) is a neuro-evolutionary method which originates from the cellular encoding (CE) [8]. Like CE, AE encodes the Artificial Neural Network (ANN) in the form of a program. This program called Assembler Encoding Program (AEP) has, however, a different structure from the CE program. In CE, the programs are in the form of trees, whereas AEPs have a linear structure as programs in Linear Genetic Programming [9, 13]. The consequence of differences in the construction of programs in both related methods is using different evolutionary techniques in both cases. CE uses Genetic Programming to form programs, whereas AE uses Genetic Algorithms (GAs) for the same purpose. The second difference between CE and AE is the method for creating ANNs. In CE, ANNs are created directly[1], i.e. each operation from a program operates (adds or removes neurons or connections) directly on ANN. AE presents a different approach. In this case, ANNs evolve in three separate phases. First, a GA is used to produce AEPs. Next, each AEP creates and fills up a Network Definition Matrix (NDM) including all the information necessary to produce a final ANN. Once the matrix is completely formed, it is transformed into an ANN. The advantage of such approach is that a single operation in AEP can modify many elements of a network. For example, modification of a single row in NDM modifies all synaptic weights of a neuron. Meanwhile, in the CE program, a single operation usually changes a single parameter of ANN (weight of interneuron connection, parameter of neuron, etc.). In consequence, effective AEPs can be simpler and shorter than CE programs and thus, they could be found faster than the latter.

Another method which, like AE, uses a three-phase procedure of evolving ANNs is HyperNEAT proposed at more or less the same time as AE by Gauci and Stanley [4]. In this case, first the NEAT method [21] is used to evolve ANNs called Compositional Pattern Producing Networks (CPPNs). Each CPPN is then used to create a special data structure which represents the final ANN. Once the CPPN stops working, the ANN is formed based on the information included in the data structure.

At first glance, AE and HyperNEAT are very similar. However, they differ in two fundamental elements. First, they use various evolutionary techniques to form AEPs and CPPNs. In AE, the evolution proceeds according to Cooperative Co-Evolutionary GA (CCEGA) [14, 15]

[1] In CE, indirect representations of ANNs, i.e. CE programs, directly operate on the networks

which, in contrast to NEAT, is a co–evolutionary, multi–population technique predisposed to evolve modular solutions as AEPs. Second, AEPs and CPPNs work in a completely different way. CPPNs are neural networks which are separately activated for each element of the data structure defining ANN. Outputs of the network are used to update the data structure and thereby determine weights of interneurons connections. Meanwhile, AEPs are programs which use their operations to modify NDMs and, in consequence, parameters of ANNs. The modus operandi of operations depends on both their implementations determined beforehand by a man and values of parameters determined in the evolutionary way. In contrast to CPPNs which can only decide about parameters of ANNs, AEPs have greater potentials. Since AEP operations are partially defined by a man, the programs can in principle perform any actions on NDMs and ANNs. For example, they can also decide about the size of ANNs or control the learning process with application of an external learning algorithm.

To date, AE has been tested in three different testing problems, i.e. in the optimization problem, in the predator-prey problem and in the inverted pendulum problem. In all the tests, the method demonstrated fairly good effectiveness. Noteworthy is the fact that it successfully competed with different direct neuro-evolutionary methods (e.g. CM [11], SANE [12]) and reinforcement learning methods (e.g. Q-learning [2]) in problems which rather prefer the latter [19][2]. Good performance of AE in the testing problems and the fact that it belongs to the indirect class of neuro–evolutionary methods, appropriate for more complex tasks, affected the decision about applying it to construct a decision system (DS)[3] for a team of Autonomous Underwater Vehicles (AUVs). Since a target use of the vehicle team is to protect a warship against various types of underwater objects, preliminary tests with AE were carried out in a variant of the predator-prey problem in which roles of predators and prey were played by AUVs. In the experiments performed in simulation, the main goal of the vehicles-predators, like in the standard form of the predator-prey problem, was to capture a fast vehicle-prey behaving by 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. The experiments revealed that the version of AE used so far has difficulties in generating effective DSs. To improve effectiveness of AE, it was necessary to introduce a few modifications to the method. All the modifications were tested experimentally and compared with the classic form of AE. The results of the tests are presented in the current herein.

The paper is organized as follows: Section II is an introduction to AE; Section III is a description of all the modifications to AE; Section IV is the report on the experiments; Section V is the summary.

II. FUNDAMENTALS OF ASSEMBLER ENCODING

There are two key elements of AE, i.e. AEP and NDM. Each AEP is composed of two parts, i.e. a part including operations and a part including data (memory cells). To build NDM, the operations are run in turn and when working they can use data located at the end of AEP (Fig. 1). In AEPs, various operations can be used. The main task of most operations is to modify NDM. The modification can involve a single element of NDM or a group of elements. Figure 2 presents two example operations used in AE.

The task of `CHG` is to change a single element in NDM. A new value of the element, stored in parameter p_0, is scaled to <–1,1>. An address of the element depends on both parameters p_1, p_2 and registers R_1, R_2. The role of the registers is detailed in the further part of the paper.

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

In addition to the operations whose task is to modify the content of NDM, AE also uses a jump operation denoted as `JMP`. The jump makes it possible to repeatedly use the same code of AEP in different places of NDM. It is possible thanks to changing values of the registers once the jump is carried out. An example use of the jump is demonstrated in Fig. 4. The program presented in the figure proceeds as follows. First, both registers are initiated to 0. Then, the first two operations are run, the result of which is visible in the top left corner of NDM. In the next step, the jump denoted in the figure as `JMP`(0, 2, 0) is executed. It

[2]AE was also compared with state-of-the-art neuro-evolutionary methods: NEAT [21, 22], ESP [6] and CoSyNe [7]. The experiments reported in the work [16], unpublished so far, confirmed high effectiveness of AE

[3]The task of DS is to provide high-level decisions concerning direction and velocity of move for each vehicle

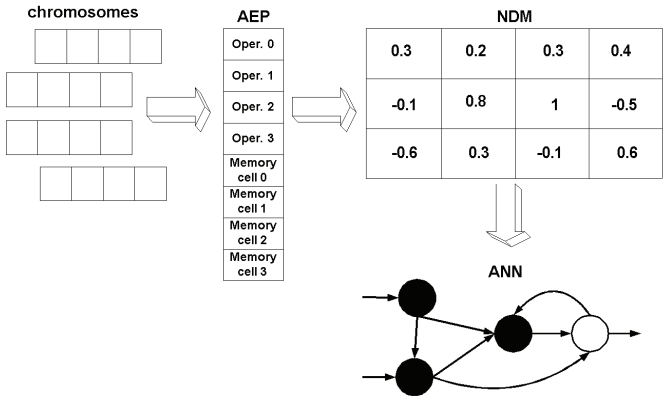


Fig. 1. Using AE to create ANN

```
CHG: :run(p0,p1,p2)
{
row=(abs(p1)+R1)mod NDM.width;
column=(abs(p2)+R2)mod NDM.height;
NDM[row,column]=p0/MaxValue;
}
```

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

Fig. 2. `CHG` and `CHGC0` operations (p_0, p_1, p_2 are parameters of operation, `D[i]` is *ith* element of data, `D.length` is the length of data sequence, `MaxValue` is a scaling value, `R1,R2` are registers)

a)

```
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
```

b)

```
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. 3. Phenotypic (a) and genotypic (b) representation of example AEP (`CHGM0` and `CHGM1` are example operations used in AE)

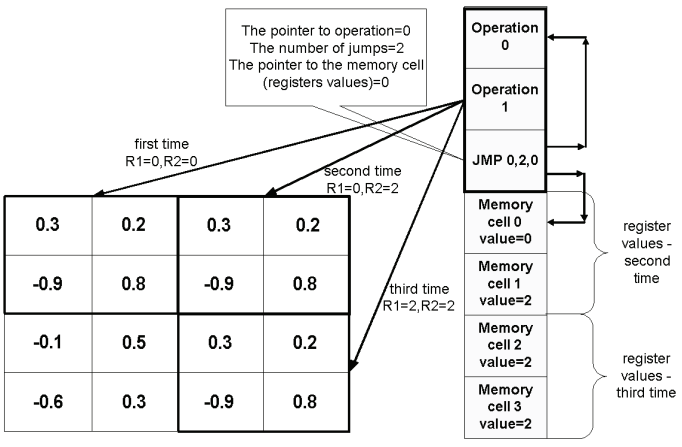


Fig. 4. `JMP` operation

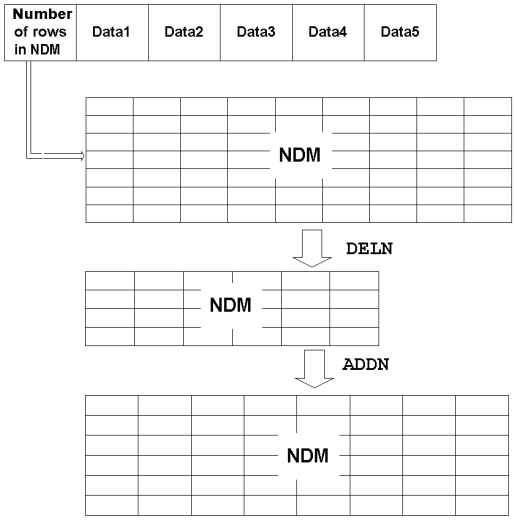
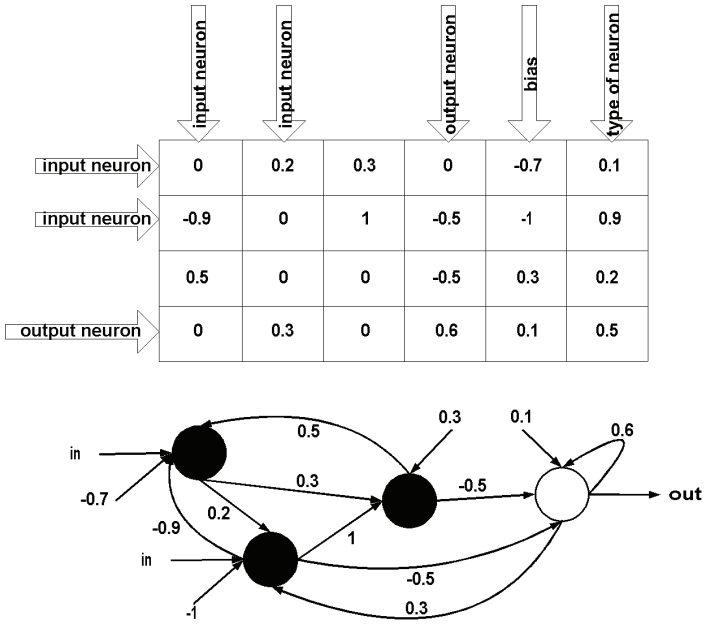


Fig. 5. Using ADDN and DELN by AEP




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

Fig. 6. NDM as Connectivity Matrix

first updates values of the registers and then control goes back to the first operation of AEP. R1 is set to 0 (Memory cell 0) whereas R2 to 2 (Memory cell 1). At this point the two operations preceding the jump are called once again. This time, however, both operations update a different fragment of NDM. Since the jump is run twice, each time with different values of the registers, the first two operations of AEP are executed in three different areas of NDM.

An additional group of operations used in AE are operations whose task is to change the size of NDM. In AE, an initial size of NDM is encoded in a chromosome with data (Fig. 5). Then, each AEP has a potential to modify the size of NDM through the use of operations ADDN and DELN. ADDN adds new rows and columns to NDM. This procedure corresponds to adding new neurons to ANN, neurons unconnected with the rest of ANN. Addition of new

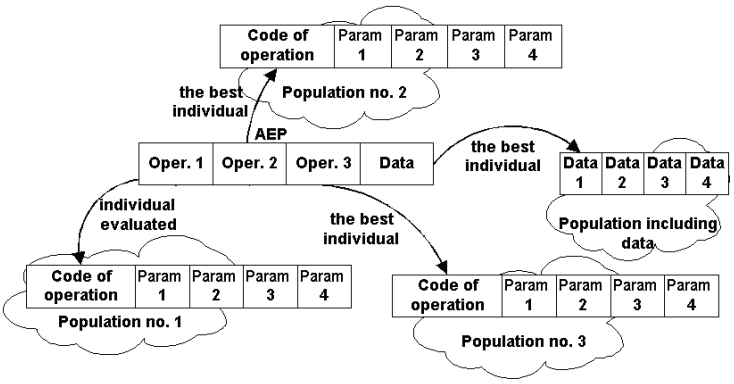


Fig. 7. Evolution of AEPs for $n = 3$

neurons does not destroy connections established in ANN. The task of DELN is to remove a single neuron from ANN. The elimination of the neuron practically takes place through removing a corresponding row and column from NDM.

Once AEP finishes its work, the process of transforming NDM into 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 the same skeleton of ANN, i.e. ANN without determined weights of interneuron connections, NDM can take the form of the classical connectivity matrix (CM) [11], i.e. a square, binary matrix of the number of rows and columns equal to the number of neurons. Value ”1” in the i-*th* column and the *j*-th row of such a matrix means a connection between the *i*-th neuron and the *j*-th neuron. In turn, value ”0” means the 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. An example of such a matrix is presented in Fig. 6.

In AE, AEPs and in consequence ANNs are shaped by GAs. The evolution decides about the arrangement of the operations and data in AEP, and about values of operation parameters. Implementations of the operations do not evolve, they are defined beforehand. The evolution of AEPs proceeds according to CCEGA proposed by Potter and De Jong [14, 15, 17]. In CCEGA, an evolutionarily created solution is divided into parts, and each part evolves in a separate population. A complete solution is formed from selected representatives of each population. To use the scheme above in relation to AEPs, it is necessary to divide them into parts. In the case of AEPs, the division is natural. The operations and data make up natural parts of the programs. Since CCEGA assumes the evolution of each part in a separate population, an AEP consisting of *n* operations and a sequence of data evolves in *n* populations with operations and one population with data (Fig. 7). During the evolution, AEPs expand gradually. Initially, all AEPs include one operation and a sequence of data. The operations and data come from two different populations. When the evolution stagnates, i.e. the lack of progress in fitness is observed over some period, a set of populations containing the operations is enlarged by one population. This procedure extends all AEPs by one operation.

In AE, the operations and data are encoded in the form of binary strings. Each chromosome-operation includes five blocks of genes. The first block determines a code of the operation, while the remaining blocks contain a binary representation of four parameters of the operation (e.g. 01000|11000|01000|00000|00100 represents the following operation: CHGC0|-1|1|0|2). Chromosomes-data are vectors including binary encoded integers. Each integer encodes a single element of data. In AE, all the chromosomes-operations have the same length. The chromosomes-data can change the length during the evolutionary process.

III. MODIFICATIONS TO ASSEMBLER ENCODING

In the experiments reported further, seven different modifications to AE were tested. The first modification (modification No. 1) involves the way of transforming NDM into ANN. The task of the six remaining modifications, i.e. modifications No. 2 to 7, is to streamline the process of evolving AEPs. All the seven modifications mentioned above are shortly described in the following sections.

III.1. Modification 1

As mentioned in Section II, in AE, ANNs are formed based on the information included in NDMs. Each NDM

contains a part defining the connectivity in an ANN and a part defining individual neurons. In the former part, each element determines weight of a connection between corresponding neurons. In turn, in the latter part, each element determines value of one of neuron parameters, e.g. bias, type of neuron, parameter of transfer function. To obtain an effective ANN, AEP has to determine values of appropriate elements in NDM. Elements in the matrix unchanged by the AEP are equal to zero. Parameters of an ANN corresponding to the unchanged elements are in consequence also equal to zero. In the case of the first part of NDM, the zero value means lack of a single connection in ANN. This value does not thwart efforts of AEP in other fragments of NDM. Values inserted in other places of the matrix are not lost. They have influence on the shape of ANN. The same situation is with the second part of the matrix and with such parameters as bias and type of neuron. The zero value of the mentioned parameters also do not make work of AEP in other places of NDM useless. For bias, the value zero only means the lack of "zero" input in a single neuron whereas for the type of neuron it means the assignment of one of the types to the neuron, e.g. sigmoid. In the case of the parameter of transfer function, we deal with a different situation. The value zero of the parameter mentioned means a constant activity of a neuron. Values of other parameters of the neuron, determined by AEP in other fragments of NDM, do not have any influence on its behavior, and in consequence they are lost. In order to prevent such a situation, the definition of a neuron should be complete, i.e. it should involve both connections and parameters. Since, the connections and the parameters of neurons are defined in two separate places in NDM, the complete definition of many neurons may require many operations and be a serious difficulty for AEPs. To make the task of AEPs simpler, modification No. 1 suggests to use a default value for the parameter of transfer function for each neuron. The default value should only be applied when NDM includes the value zero in appropriate place. Otherwise, the value from NDM is used.

III.2. Modification 2

As already mentioned, the evolution in AE proceeds according to CCEGA. This algorithm assumes that operations and data evolve in many separate populations. It, however, does not introduce any limitations on the type of a GA used inside individual populations. To select GAs that are most suited for AE, a number of experiments had been carried out [18]. In the experiments, three GAs were tested, i.e. Canonical GA (CGA) [5], Steady State GA (SSGA) [25], and Eugenic GA (EuGA) [1, 20]. The experiments have revealed that the best solution for AE is when evolution in each population proceeds according to CGA. In this algorithm, the most popular replacement strategy is to swap all individuals from a population for new offspring individuals being a result of different genetic operators, mainly crossover and mutation, on selected parental individuals. The task of the crossover is a global exploration of the whole space in search of promising solutions, whereas the mutation plays a rather exploitative role, i.e. it carries out a local search in a proximity of a good solution. A local range of the mutation is achieved by introducing into a genotype slight random perturbations. However, the assumption is that small changes in a genotype result in small changes in a phenotype. In AE, we deal with another situation. The mutation may be equally exploratory and destructive as the crossover (see Section III.7). This means that AE has not any tool responsible exclusively for local search of a solution space. In the current variant of the method, this problem is solved by means of a strong selective pressure in populations including operations. Once a promising operation is found a significant portion of the remaining operations from the same population quickly become its copy. Since only a part of mutations results in large modifications of a phenotype and there are many copies of the same good operation (the same situation is in all populations with operations), the area around a promising solution is carefully explored. Such a course of action of the algorithm can be viewed as jumps from one promising region of the phenotype space to another.

Modification No. 2 suggests solving the problem described above in another way, namely by means of elitism. Thanks to the elitism the most fit operations and data should always be preserved. In this case, good solutions should not be lost so fast as in the previous case. This makes a chance for a part of mutations to accurately explore a region in the proximity of a good solution. Moreover, the elitism may also create an opportunity to increase a diversity in each population. Since there is a guarantee that the best solution will not be lost, it does not have to be replicated many times to increase its chance to survive. In consequence, each population may include many different individuals, each of which may become a potential parent for a next good solution.

In the experiments, two different types of elitism were tested, i.e. pure and probabilistic elitism. In the first case, only the best individuals become elite and are introduced without any modifications to a newly created population. In the second case, elite individuals are selected at random. Of course, better individuals have a greater chance to be selected than the worse ones.

III.3. Modification 3

In the current version of AE, the evolution in each population proceeds with constant values of parameters. Probabilities of mutation, crossover and cut-splice (shortening or extending a chromosome; this operator is only used in relation to chromosomesdata which can change their length during the evolution) do not change over time. Such an approach, however, makes effective evolution of AEPs difficult. The greatest problems are with the code part of AEPs and with the mutation of individual operations. For AEPs, a constant mutation of operations means greater and greater modifications of their code parts in subsequent co-evolutionary generations. As mentioned before, the code parts of AEPs change the length with the passage of time. Initially, each AEP has only one operation to which next operations are successively added if the programs cannot accomplish progress in performance over an assumed time. In the initial phases of the evolution, when all AEPs include only one operation, slight modifications introduced to operations do not cause greater changes in behavior of their programs. The situation changes when AEPs have two, three or more operations. This time, not large modifications in the operations can mean huge changes in the activity of their programs. This, in turn, can cause large bounds of fitness of AEPs created in consecutive coevolutionary generations, and in consequence great difficulties in the evolution of effective programs.

In order to prevent such problems, modification No. 3 proposes to adjust the level of mutation performed on operations to the number of operations contained in AEPs. The magnitude of the mutation determined at the beginning of the evolutionary process refers only to programs consisting of one operation. When AEPs have more operations the probability of mutation determined for a single operation AEPs is divided by the number of operations currently used in the programs.

III.4. Modification 4

This modification introduces a similar solution as above, but in relation to data. Chromosomes-data can change their length during consecutive co-evolutionary generations. To this end, the cut-splice operator mentioned above is used. A constant value of per-bit mutation probability, in the case of variable length data, means statistically more mutations in longer chromosomes-data than in the shorter ones. Moreover, assuming that data from longer chromosomes more often occur in NDMs and thereby in ANNs (it is not always the case, the number of data introduced to NDM depends on operations), more mutations in longer chromosomes-data could also mean more changes in ANNs. In order for the changes introduced to chromosomes-data to always induce similar changes in ANNs, regardless of the length of chromosomes, modification No. 4 suggests making mutations in chromosomes-data dependent on their length. Shorter data should be mutated with a greater probability than their longer counterparts.

III.5. Modification 5

As above, in modification No. 5, we also deal with a variable mutation of data. Initially, the standard value of the mutation is used. This value is changed in two situations. Stagnation of the evolution over an assumed period is the first out of the two situations mentioned above. To escape from a point where the evolution is stuck, the mutation of data is gradually augmented. The process of augmenting the mutation is immediately stopped once the first symptoms of exit from the stagnation point are observed. The end of the stagnation is connected with returning to a standard value of the data mutation.

The improvement of the best result achieved so far is the second situation when the standard value of the data mutation is changed. This time, however, we deal with a reduction of the mutation to some an assumed value. The main goal of such a procedure is to meticulously search the area close to the currently best solution. When further improvement of the result does not occur for a longer time the mutation is getting back to the standard value.

III.6 Modification 6

Modification No. 6 suggests introducing restrictions into the process of crossing over the operations. In the version of AE used so far, each operation can be crossed over with any other operation, regardless of the type of both operations. This means that two completely different operations can exchange their genetic material without any hindrance. In modification No. 6, this situation is changed and freedom of the crossover is restricted exclusively to operations of the same type.

III.7. Modification 7

Modification No. 7 suggests altering the way of mutating operations. In the version of AE used so far, the operations are mutated in the same way regardless of the type. They are treated as simple binary strings. Different roles played by individual fragments of chromosomes-operations are not taken into consideration during the mutation. Each bit is mutated with the same probability. In such a case, even a modification of a single bit can lead to enormous changes in both the modus operandi of the operation and the ANN itself. For example, switching a single

```
CHG::mutate(probMutation)
{
mutate(code,0.1*probMutation);
mutate(p0,probMutation);
mutate(p1,0.5*probMutation);
mutate(p2,0.5*probMutation);
}
```

```
CHGC0::mutate(probMutation)
{
mutate(code,0.1*probMutation);
mutate(p0,0.1*probMutation);
mutate(p1,0.8*probMutation);
mutate(p2,0.5*probMutation);
}
```

Fig. 8. Example mutation of CHG and CHGC0. The least mutation is on the operation code, i.e. on the first fragment of chromosome-operation and on parameters which indicate the activity area of the operation in NDM

bit in the code of operation can cause a complete change in its behavior (It is not always the case. In AE, different codes could mean the same operation. For example, for two operations available when creating AEPs, codes 110(3) and 111(7) mean the same operation, i.e. operation No. 1. This is because 3 mod 2 and 7 mod 2 in both cases is equal to 1). To prevent such situations, modification No. 7 suggests making the way of mutating operations dependent on their type. Codes of operations and parameters whose modification results in drastic changes in behavior of the operations (e.g. addresses to data used by operations) should be mutated very rarely. The remaining parameters can be mutated more frequently (Fig. 8).

IV. EXPERIMENTS

The main goal of the experiments was to determine a new and more effective variant of AE; the variant which could successfully be used to evolve efficient neuro-DSs for a system of cooperating AUVs. In the experiments all the modifications presented in Section III were tested. The first step was to examine the variant of AE without modifications. Then, individual modifications were added to the pure variant. In the case of improvement of the result, a modification was accepted and a next modification was introduced to the accepted variant. In order for the problem solved during the experiments to be maximally similar to the one which AUVs will have to solve in the future, decision was made to use the predator-prey problem as a test-bed for neuro-DSs and their AUVs.

To compare individual modifications, their learning abilities, i.e. abilities to produce effective ANNs, were measured. ANNs generated during the experiments were not tested in terms of their generalization performance. The ability of ANNs to generalize knowledge is a quality indicator of the entire method but not individual modifications. The shape of ANNs produced in AE depends on operations used by AEPs. They form NDMs and thereby ANNs. In the experiments, all the modifications used the same types of operations and there is no reason to think that different modifications prefer different operations and thereby produce ANNs with completely different characteristics. A feature differing individual modifications is their speed in creating effective ANNs but not ANNs themselves. For that reason, generalization tests were not carried out.

IV.1. Predator-prey problem

The experiments were carried out in a configuration with one prey and three chasing predators. Both the predators and the prey were implemented as the Remotely Operated Vehicle (ROV) of the type ”Ukwial” (Fig. 9) [10]. The behavior of all the vehicles was simulated by means of a simple discrete time model defined in the following section.

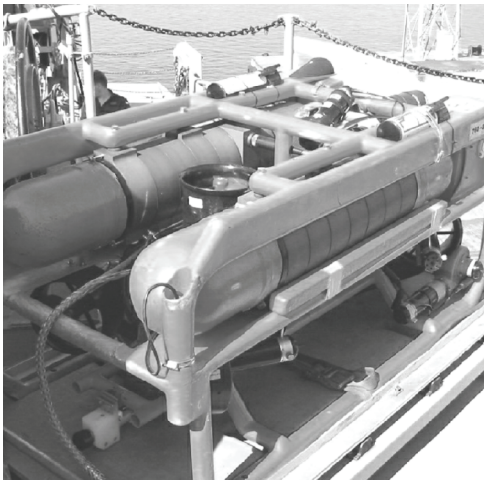


Fig. 9. Vehicle ”Ukwial”

In the experiments, the predators and the prey lived in a common artificial environment. To represent the environment a square of 200 × 200 meters was used (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 the 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.

In the experiments the predators were controlled by a single ANN whose task was to determine movement direction for each of them. ANNs had six inputs and three outputs, the number of outputs corresponded to the number of predators and, in turn, the number of inputs was twice the number of predators. Each output gave commands to one predator (new course, see below) and, in turn, each input informed about vertical or horizontal distance between the prey and one of the predators.

Each predator could move in the following directions: 0, 5, 10, …, 355 degrees. The speed of the predators was constant during the tests and amounted to 0.5 m/s. As for the prey, two different types were used, i.e. simple and advanced prey. The simple prey behaved according to a deterministic strategy which forced it to stand still when no predator was closer to it than 50 meters, and to move directly away from the nearest predator otherwise. When moving, the simple prey could select the same directions as the predators. The speed of the prey amounted to 0.5, 0.75 or 1 m/s.

Unlike its simpler counterpart the advanced prey always made a decision taking into consideration the location of all predators situated close to it (closer than 50 meters). Actions performed by the advanced prey were also deterministic and they directed it toward safe areas of the environment, i.e. areas without predators. Other aspects of behavior of the advanced prey, i.e. its speed, behavior away from the predators, and actions which the prey could perform at each step, were the same as in the case of the simple prey.

Since the speed of the predators was either lower or the same as the speed of the escaping prey, they could not simply chase the prey to grasp it. To achieve the goal, the predators had to cooperate. The prey was assumed to be captured if the distance between it and the nearest predator was lower than 5 meters.

IV.2. Vehicles

Since all the experiments were carried out in simulation, behavior of the automatically controlled underwater vehicle "Ukwial" (the vehicle with its controllers; the task of the controllers is to convert high-level decisions provided by DS into low-level ones for propellers of the vehicle) had to be modeled appropriately. To simulate movement of the vehicle, a nonlinear model described in six degrees of freedom is usually used [3]. Moreover, to control the vehicle (along a fixed path), several nonlinear controllers of motion parameters are needed [23]. In consequence, simulation of the vehicle with application 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 there is a need to simulate many vehicles, a different solution has to be applied.

Since in the experiments a neuro-evolutionary method which tests many different neural solutions per evolutionary generation was used 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 [24]. The model mentioned above is a reduced discrete counterpart of the nonlinear model. It assumes representation of the vehicle in the form of a matrix $S = [P_{ij}]_{|I|x|J|}$ where I is a set including advance velocities of the vehicle $V_i = i\Delta V$ ($i = 1..|I|$, ΔV is a quantization step of the advance velocity), and J is the number of motion parameters (e.g. course, coordinate x, coordinate y) of the vehicle. $P_{ij} = [\Delta p_{kl}^{ij}]_{|K|x|L|}$ is a matrix including changes of motion parameter Δp_{kl} in response to a desired value of a selected parameter p_k^{set} for V_i and for j-th motion parameter of the vehicle, K is a set including desired values of a selected parameter $p_k^{\text{set}} = k\Delta p^{\text{set}}$ $k = 1..|K|$, Δp^{set} is the quantization step of the desired values of the selected parameter), and L is a set including points in time of parameter registration $t_l = l\Delta t$ ($l = 1..|l|$, Δt is a discretization step).

To test reliability of the simplified model, comparison tests with its nonlinear counterpart were carried out [24]. The tests showed that both models are similar (trajectories of the vehicles modeled by means of both compared models were very similar, the task of the vehicles was to move along a desired trajectory with three turning points). Moreover, the tests also made it possible to determine a degree of speeding up calculations when using the simplified model. It appeared that for the simplified model, simulations of the vehicle motion were 16.500 times shorer than for its more complex equivalent. In both compared cases vehicles moved along the same trajectories. All the simulations were performed on the following computer platform: Intel Core2 DUO and Windows XP/Matlab.

In the experiments reported in the paper, because of flatness of the environment (the vehicles moved on the surface), the following model was used: $S = [Pij]_{6x3}$. The matrix P_{i1} included changes of course $\Delta\psi_{kl}$, the matrix P_{i2} included changes Δx_{kl} of coordinate x, and the matrix P_{i3} included changes Δy_{kl} of coordinate y. All the elements in the matrices above were determined in response to the desired course $\Delta\psi_k^{\text{set}}$, ($k = 1..36$, $\Delta\psi^{\text{set}} = 5^\circ$), in l-th time step (l = 1..60, Δt = 0.5 s), for different V_i (ΔV = 0.25 m/s).

All the parameters in P_{ij} were registered for the vehicle "Ukwial".

IV.3. Evaluation of ANNs

In order to evaluate ANNs generated during the experiments, thirty different testing scenarios (or training tasks) were used. At first, each neuro-DS was tested in the first and the simplest scenario, say, No. 1. If the predators could not capture the prey during an assumed period, the test was stopped and the DS received appropriate evaluation that depended on the distance between the prey and the nearest predator. However, if the predators grasped the prey, they were put to test according to the next scenario. During the experiments, the predators could make 50 steps (50 decisions) before the scenario was interrupted.

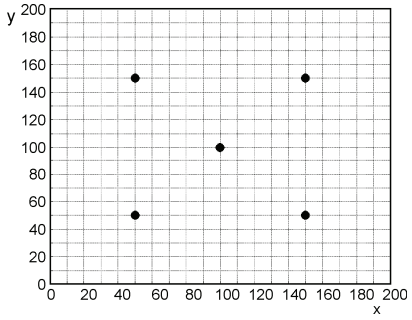


Fig. 10. Starting positions of prey (for one speed and type of prey all five starting positions were tested)

Individual scenarios differed in the initial position (Fig. 10), speed (0.5, 0.75 or 1m/s), and type of the prey (simple or advanced). Consecutive scenarios were more and more difficult. At first, the predators had to capture the simple prey that was as fast as them. The predators which passed the first exam had to pit against the prey that was one and a half time faster than them. 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 taking better decisions than its predecessor. In the beginning, speed of the prey was minimum but it grew in the following testing scenarios. In all the scenarios all the predators always started from position (0,0). All the thirty scenarios are described below.

- Scenarios no 1-5: prey's speed = 0.5 m/s, simple prey,
- Scenarios no 6-10: prey's speed = 0.75 m/s, simple prey,
- Scenarios no 11-15: prey's speed = 1 m/s, simple prey,
- Scenarios no 16-20: prey's speed = 0.5 m/s, advanced prey,
- Scenarios no 21-25: prey's speed = 0.75 m/s, advanced prey,
- Scenarios no 26-30: prey's speed = 1 m/s, advanced prey.

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

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

$$f_i = \begin{cases} d_{\max} - \min\limits_{p} d_i(p), & \text{prey not captured in } i\text{-th scenario} \\ f_{\text{captured}} + (50 - m_i)/a, & \text{prey captured in } i\text{-th scenario} \\ 0; & \text{prey not captured in } i\text{-th scenario} \end{cases} \quad (2)$$

where
f_i – the reward received in i-th scenario
$d_i(p)$ – the distance between the prey and a predator p in the end state of ith scenario
$d_{\max}$ – the maximum distance between two points in the environment
f_{captured} – the reward for grasping the prey in a single scenario ($f_{\text{captured}} = 100$)
m_i – the number of steps to capture the prey ($m_i < 50$)
a – this value prevents the situation in which partial success is better than success in all scenarios
n – the number of scenarios ($n = 30$).

IV.4 Parameter settings

In the experiments, all variants of AE (the classic variant and variants with the modifications) were tested many times to tune each of them to the problem. For each variant, values of three parameters were determined, i.e. mutation probability, crossover probability, and size of tournament. The reaming parameters were common for all the variants, and they were not subject to the tunning process (Table 1).

Table 1. Parameter setting common for all the variants

Parameter	AE
No. of subpopulations	from 2 to 6
Size of subpopulations	80 (data), 40 (operations)
No. of data	maximally 20
Size of ANNs (hidden units)	maximally 4

AEPs created within the evolutionary process, regardless of the variant of AE, could use seven different types of operations, i.e. `CHGFF`, `CHGC0`, `CHGR0`, `CHGC1`, `CHGR1`, `CHGM0`, `CHGM1`. All the operations are described in Appendix at the end of the paper.

IV.5. Experimental results

To compare different variants of AE, with and without modifications, two criteria were used, i.e. the percentage of successful ANNs and the average fitness of ANNs. In the paper, the successful ANN means an ANN which successfully controlled the predators in all the thirty scenarios specified in the previous section. Each variant of AE after tunning was run sixty times, i.e. sixty ANNs were generated for each of them. Results of the experiments are presented in Table 2.

The table shows that, in most cases, the modifications proposed in the paper appeared to be beneficial for AE. As a whole they considerably improved performance of the method. The classic variant without modifications generated only 11.6% of successful ANNs whereas the number of successfully ended training tasks was equal to twenty (on average, average fitness > $20f_{\text{captured}}$) in this case. In the preliminary research, such result was recognized as unsatisfactory, the more so because it was achieved in simulated conditions. The modifications allowed AE to construct almost twice as many successful ANNs (23.3% - the last column). Moreover, effectiveness of the chase grew to twenty six successfully ended training tasks, on average.

Modifications No. 1, 2, and 7 had the greatest influence on the final result. In the case of modifications No. 3, 4, and 5, the performance improvement was somewhat smaller. The only modification which appeared to be harmful for AE was modification No. 6. It seems that the main cause of worse results of this modification was the number of different operations used in AEPs. In the experiments, AEPs could use six types of operations (see Section IV.4). As a result, it was a very low chance to select two parental operations of the same type (all the populations with operations included 40 individuals) and to crossover them. A very rare crossover contributed to less effective exploration of the solution space and in consequence to worsening the performance of AE.

To increase the probability of crossovers between operations in modification No. 6, the set of operations available to AEPs was reduced to operations `CHGFF` and `CHGM0`. Even though results achieved in this case were better than for the variant with six operations they turned out to be still unsatisfactory (see Tab. 3).

In the next step, homogeneous AEPs, i.e. AEPs including operations of only one type (either `CHGFF` or `CHGM0`), were applied. Since in such a solution the genetic material was always exchanged between individuals of the same type, all crossovers between operations were permissible and no restrictions were necessary to accomplish the effect assumed in modification No. 6.

Table 2. Results of experiments (The table includes results averaged over 60 evolutionary runs: m 1 is the classic variant with modification No. 1, m 2 is the classic variant with modifications No. 1 and 2, and so on. Since modification No. 6 appeared to be harmful for AE it was not accepted and as a result m 7 is the classic variant with all the modifications except modification No. 6. When analyzing the content of the table it is necessary to remember that the result of each modification can only be compared with the result of its predecessor. This is because individual modifications were successively added to each other and e.g. the result of modification No. 3 is, in principle, the result of modifications No. 1, 2, and 3.)

Criterion	Classic	m 1	m 2	m 3	m 4	m 5	m 6	m 7
% of successful ANNs	11.6%	17.2%	21.2%	22.4%	21.6%	22.7%	20.4%	23.3%
average fitness of ANNs	2023.5	2254.6	2389.5	2415.8	2483.2	2536.5	2405.6	2634.2

Table 3. Results of additional experiments (the table includes results averaged over 60 evolutionary runs; this time m 7 means AE with all the modifications, description of m 6 is as in Table 2

Criterion	m 6			m 7
	`CHGFF + CHGM0`	`CHGFF`	`CHGM0`	`CHGFF`
% of successful ANNs	21.8%	22.8%	20.8%	24.5%
average fitness of ANNs	2485.3	2622.7	2432.8	2715.6

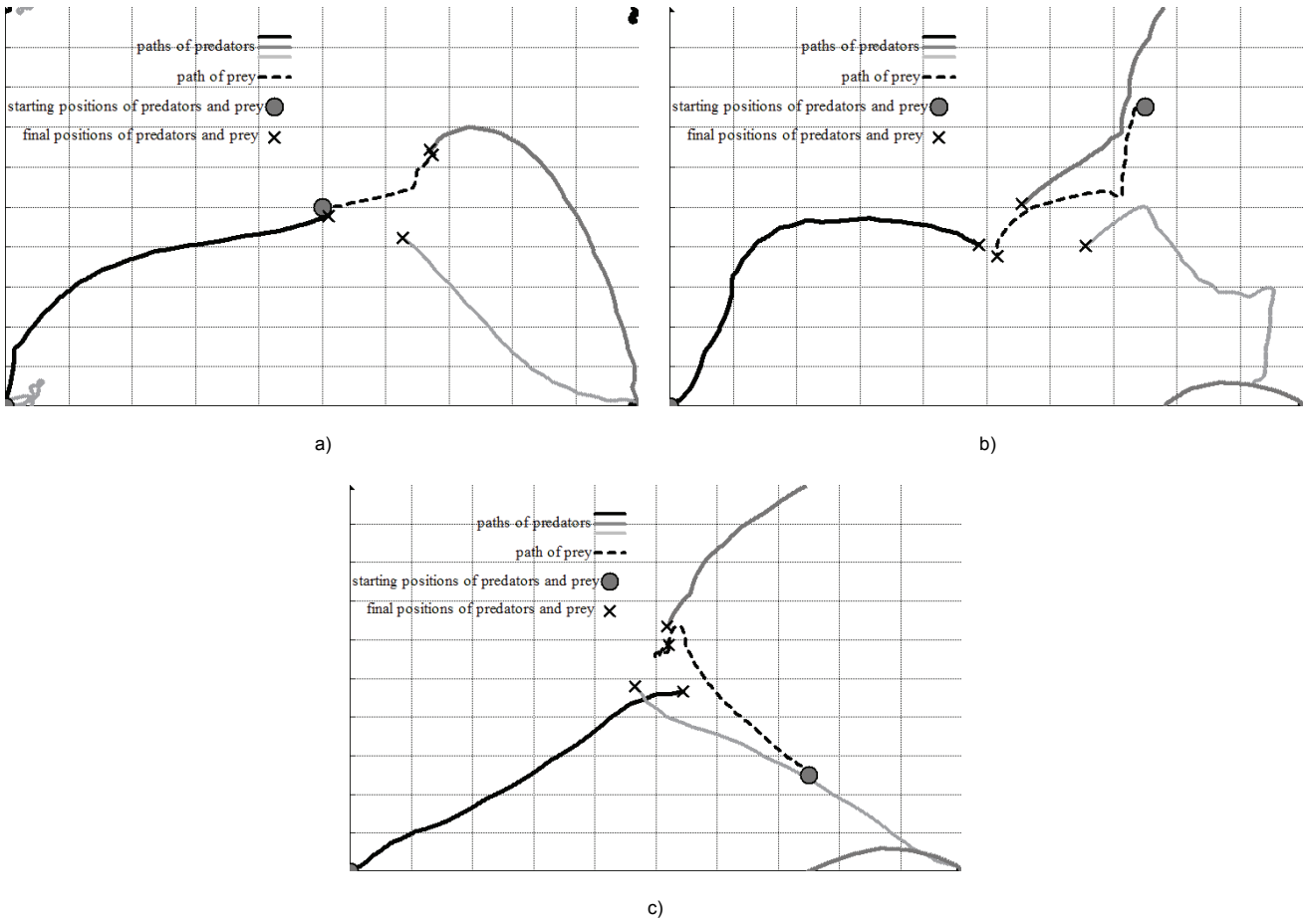


Fig. 11. Example behavior of predators and prey

Reduction of the set of operations to one operation positively affected effectiveness of AE. The results achieved in this case appeared to be better not only for modification No. 6, but also for modification No. 7. Combination of modification No. 7 and homogeneous AEPs yielded the best outcome of the tests (24.5% of successful ANNs and twenty six successfully ended training tasks, on average). However, the performance improvement was only noticed for the variant with `CHGFF`. Application of `CHGM0` was not so effective. This result means that even though modification No. 6 and its homogeneous variant can improve performance of AE, their applicability is limited to cases when we know an operation which guarantees the final success. Otherwise, homogeneous AEPs can even deteriorate effectiveness of AE.

Another disadvantage of the solution with operations of one type may be a poor generalization of ANNs. In AE, the operations can be divided into two groups. The task of operations from the first group is to modify smaller fragments of NDM. An example of such operation is modification of a part of a column. Operations belonging to the second group are able to act in each fragment of NDM defining ANN, e.g. `CHGM0` or `CHGFF`. In order for homogeneous AEPs to be able to influence each parameter of an ANN, they either have to include many operations with a lower operational range or to use more universal operations from the second group. In the latter case, AEPs are usually shorter and could be find faster. However, using exclusively such operations as `CHGFF` or `CHGM0` may result in the situation in which all elements of NDM defining an ANN will be unequal to zero. Such matrices, in turn, correspond to fully connected ANNs which tend to overfit to a problem to be solved.

V. SUMMARY

The paper presents seven modifications to AE and reports experiments whose the goal was to test effectiveness of each of them. The main cause of efforts aiming at modifying and streamlining AE was its unsatisfactory performance in generating effective neural

decision systems for a team of AUVs. Generally, the experiments revealed that the modifications proposed in the paper improve results of AE. The only modification which requires satisfying additional conditions to become a valuable element of AE is the restriction of operation crossovers exclusively to the ones between operations of the same type. In order for the modification mentioned above to result in progress in performance of AE, it appeared that it should be combined with AEPs homogeneous in terms of operations used in the programs.

Generally, as a result of the experiments a new improved variant of AE including modifications No. 1-5, and 7 was developed. With regard to modification No. 6, it needs further research. A particularly interesting idea is to entirely rely AE on one type of operations. Such a solution requires, however, new operations which would enable an AEP to influence each element of ANN and to form ANNs with a sparse connectivity.

References

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

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

[3] T.J. Fossen, *Guidance and Control of Ocean Vehicles*, John Wiley and Sons Ltd., (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. Gomez, R. Miikkulainen, *Incremental evolution of complex general behavior*, Adaptive Behavior, 5: 317-342, (1997).

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

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

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

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

[11] 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).

[12] 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).

[13] 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).

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

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

[16] T. Praczyk, *Solving the pole balancing problem by means of Assembler Encoding*, in review.

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

[18] T. Praczyk, *Selecting genetic algorithms for assembler encoding*, Archives of Control Science, Volume 18(LIV), No. 4, pp. 477-495 (2008).

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

[20] J.W. Prior, *Eugenic Evolution for Combinatorial Optimization*, MSc Thesis, The University of Texas at Austin, (1998).

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

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

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

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

[25] D. Whitley, *The GENITOR algorithm and selective pressure: why rank-based allocation of reproductive trials is best*, In Proceedings of the Third International Conference on Genetic Algorithms and Their Applications, pp. 116-121 (1989).

A APPENDIX

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, but all updated elements have the same value.

CHGR0 – like CHGC0, but update refers to a row of NDM.

CHGR1 – like CHGC1.

CHGM0 – 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 the data part of AEP.

CHGM1 – like CHGM0, but all the updated elements have the same value.

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