CMST 19(3) 145-155 (2013) DOI:10.12921/cmst.2013.19.03.145-155

Modularity and Regularity in Neural Networks Produced with Assembler Encoding

T. Praczyk

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

Received: 18 February 2013; revised: 21 April 2013; accepted: 11 July 2013; published online: 5 September 2013

Abstract: The main focus of the paper is on the ability of the neuro-evolutionary method called Assembler Encoding to repeatedly use the information included in a genotype and to construct modular and/or regular neural networks. It reports experiments whose the main goal was to test whether the method is capable of adjusting topology of neural networks to a modular and regular problem. In the experiments, the task of Assembler Encoding was to evolve neuro-controllers responsible for balancing two or three inverted pendulums instaled on separate carts. Since both the carts and the pendulums were identical the task of neuro-controllers could be performed by means of modular/regular neural networks.
Key words: artificial neural networks, evolution, modularity/regularity

I. INTRODUCTION

Artificial Neural Networks (ANNs) are used to solve wide range of problems. They play more and more substantial and responsible roles and solve more and more difficult problems. Since to solve complex problems, appropriately sophisticated tools are usually required, it is also necessary to apply proper methods to build complex ANNs [20]. The neuro-evolution is a technique that is more and more frequently used for this purpose. So far, many neuro-evolutionary methods have been proposed. Most of them are direct methods which store topology and parameters of ANNs directly in chromosomes (e.g. SANE [13], NEAT [24], ESP [7], CoSyNE [8], CM [12]). The main drawback of such an approach is its poor scalability. In the direct methods, complex ANNs require complex chromosomes which could be a serious obstacle in generating effective ANNs. An alternative class of neuro-evolutionary methods are indirect methods whose genotypes do not include parameters of ANNs but recipes how to create them.

Assembler Encoding (AE) presented in the paper is an example of such a method [21]. AE originates from Cellular Encoding (CE) [4, 5] and Edge Encoding (EE) [11], although, it also has features common with Linear Genetic Programming (LGP) presented, among other things, in [10, 15]. In AE, an ANN is represented in the form of a program (Assembler Encoding Program – AEP) whose structure is similar to that of a simple assembler program. The task of AEP is to create a Network Definition Matrix (NDM) containing all the information necessary to produce an ANN. The process of ANN construction consists of three stages. First, a Genetic Algorithm (GA) is used to produce AEPs. Next, each AEP creates and fills up NDM. Finally, NDM is transformed into an ANN [20].

To date, AE has been tested on three different testing problems, i.e. the optimization problem [19], the predator-prey problem [22, 23] and the inverted pendulum problem [21]. In all the tests, the method demonstrated fairly good effectiveness. Noteworthy is the fact that it successfully competed with different direct neuro-evolutionary methods and reinforcement learning methods in tasks of little or medium complexity which rather prefer simpler solutions than AE [21].

However, the target application of AE is not to solve simple problems but those with a greater complexity. Since such problems are usually modular and have regularities, solving them in an effective way requires from AE the ability to construct modular/regular neural architectures. Such architectures can be built in two ways. When the structure of a problem to be solved is known in advance it is also

known the high-level architecture of a neural solution to the problem, i.e. the number of modules, connectivity between modules, repetitiveness of modules. In AE, such knowledge can be utilized by adjusting AEPs or NDMs to a problem. In the former case, AEPs can operate exclusively on items in NDM which correspond to elements of modular/regular neural solution whereas in the latter one AEPs work without any limitations, however, ANNs are constructed solely based on items in NDM defining individual modules and connectivity between modules. A more interesting case is, however, when the structure of a problem to be solved is unknown. In such a case, the only way is to use methods with the ability to automatically decompose a problem and to form modular/regular neural structures.

CE mentioned above is one of such methods which was confirmed by experiments reported in [5]. During the tests, neuro-controllers were generated, composed of sub-controllers of similar structure, each of which was responsible for its own task. Obtaining such an architecture of the controllers was possible thanks to two things. First, CE represents an ANN in the form of a tree-shape-program and provides mechanisms for repeated use of the same fragment of code. Second, the programs in CE directly operate on an ANN, adding or removing from it neurons or connections. Combination of both elements is a base for modular/regular ANNs to arise.

As already mentioned, AE originates from CE. It also represents an ANN in the form of a program and it also provides tools for repeated execution of code. However, operations from AEPs construct ANNs in an indirect way. They operate not on ANNs, but on NDMs. In consequence, mechanisms which enable CE to form modular/regular ANNs, in AE, may be applied to construct modular/regular NDMs built with repeated blocks of similar shape and content. The ability to generate such matrices could be useful when solving complex decomposable optimization problems in which a solution can be represented in the form of a matrix. However, the question is whether the same ability is also useful when the goal is to create ANNs, not optimal matrices. Other important issues requiring an answer are: whether mechanisms applied in AE especially to obtain repetitiveness (regularity) in NDMs will be used at all, whether usage of these mechanisms will really cause modular/regular NDMs to be created, and how, if at all, modularity/regularity in NDMs translates into modularity/regularity in ANNs.

To answer the above questions, experiments on the inverted pendulum problem were carried out. Like in Gruau's experiments reported in [5], the testing problem was configured so that it was solvable by means of a single ANN consisting of a number of separate sub-ANNs. In the experiments, each ANN dealt with two or three poles installed on separate wheeled carts. The task of each ANN was to protect all the poles from falling down and to keep the carts within track boundaries. Since all the poles and their carts were identical, the task could be solved by means of two or three the same or similar sub-ANNs.

In the experiments, apart from AE, a classical neuro-evolutionary method known as Connectivity Matrix (CM) [12] was also applied. CM is a direct method which encodes all parameters of an ANN in a single chromosome. The main reason for using this method during the experiments was to test how a technique which does not provide any tools to create modular/regular ANNs is able to deal with the problem in which the most natural solution is a modular ANN consisting of two or three separate sub-ANNs of the same or similar structure.

The paper is organized as follows: section 2 is a short presentation of AE; section 3 is a description of AE mechanisms that can be used to create modular/regular ANNs; section 4 is a report on the experiments; and section 5 is a conclusion.

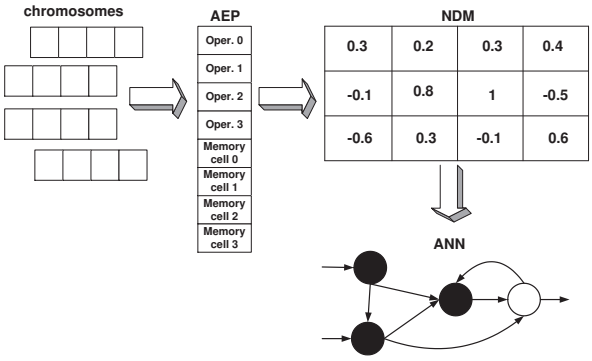


Fig. 1. Using AE to create ANN [22]

II. ASSEMBLER ENCODING FRAMEWORK

Since AE is widely described in [23], in this section only an outline of the method is given. In AE, an ANN is represented in the form of a linear program (Assembler Encoding Program – AEP) consisting of operations with predefined implementations and data. The task of the AEP is to create a Network Definition Matrix (NDM) defining a single ANN (see Fig. 1). To form the AEP and, in consequence an ANN, Cooperative Co-Evolutionary Algorithm (CCEGA) [17, 18] is used. CCEGA divides a solution into parts, each of which evolves in a separate population. A complete solution is formed from 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. 2). During 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 evolution stagnates, i.e. lack of progress in fitness is observed over some period, the set of populations containing the operations is enlarged by one population. This procedure extends all AEPs by one operation [20].

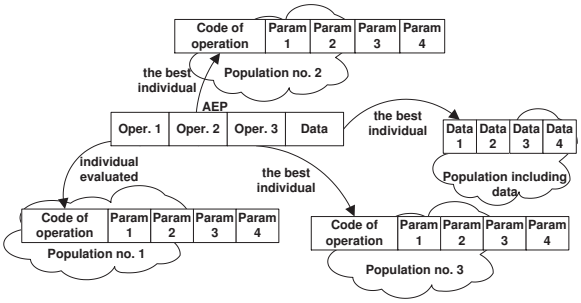


Fig. 2. Evolution of AEPs for $n = 3$ [22] – three operations and one sequence of data

In individual populations, the evolution proceeds according to Canonical GA (CGA) [6]. Individuals from each population (either the operations or the data) are encoded in the form of binary strings. Each chromosome-operation includes binary encoded parameters and optionally code of the operation (e.g. 01000|11000|01000|00000|00100 represents the following operation: `CHGC0|-1|1|0|2` - update of a fragment of a column from NDM). Chromosomes-data are strings including binary encoded data.

During evolution, each individual is combined with selected individuals from the remaining populations to form a complete AEP. The program produces a NDM and in consequence an ANN which is then put to an evaluation test. The result of the test is used to fix fitness for all components of the AEP. After evaluation of all individuals from a population the tournament selection is used to chose parental individuals for reproduction. To form offspring individuals, the parental ones are subject to three different genetic operators, i.e. one-point crossover, mutation and the so-called cut-splice (changing length of chromosome).

NDM stores all the information necessary to construct a network. This information is included both in the size and individual items of the matrix, scaled always to the range $< -1, 1 >$. The size of NDM determines a maximum number of neurons in an ANN whereas individual items of the matrix define weights of interneuron connections, i.e. an item NDM$[i, j]$ determines a link from neuron i to neuron j. Apart from the basic part, NDM also contains additional columns that describe parameters of neurons, e.g. type of neuron (e.g. sigmoid, radial, linear), and bias.

III. MODULAR AND REGULAR ANNS IN AE

In the paper, a modular ANN is loosely understood as an ANN segmented into modules, each of which is responsible for a separate task. Such ANNs have been of a great interest of scientists and practitioners for many years (e.g. [2, 3, 9, 14, 16]). The cause of their popularity is that they are usually the simplest solution to many technical problems of a great complexity. Typically, the problems mentioned are decomposable into sub-problems which are computationally simpler to solve individually and whose solutions can be combined to produce a solution to the entire problem. Such problems are ideal area of application for modular ANNs in which each module corresponds to a separate sub-problem. A lower complexity of sub-problems causes ANNs composed of interconnected modules to be usually simpler solution than monolithic ANNs.

With regard to regularity, it applies to ANNs with repetition or similarity of units. Like modularity, regularity is also a very significant feature of ANNs. Since each repetition in a regular ANN can be defined only once, representations of such ANNs can be significantly smaller in volume then the ANNs themselves. This is important when solving complex problems with regularities. In this case, compact representations of ANNs significantly facilitate building effective neural solutions.

When the units of an ANN are tightly integrated into a single structure then we deal with a regular monolithic ANN. In turn, an architecture with loosely connected repeated units, each of which has its own function is both modular and regular. Such architectures can also be implemented in the neuroevolution. CE and EE mentioned in the previous section are examples of methods which enable forming ANNs including sub-ANNs of the same or similar structure. To this end, the methods above allow the same fragment of a program memorized in a chromosome-tree to be run many times. Since operations from the program act directly on an ANN, their repeated execution in different places of this ANN results in an architecture with many the same sub-ANNs.

AE is a next method which also provides tools for building modular and regular neural architectures. However, in AE, the way to accomplish such architectures is different from that used in CE and EE. Since, in AE, ANNs are represented in the form of NDMs, the only way to accomplish modularity and/or regularity in ANNs is modularity and regularity in NDMs. For example, in order for an ANN to include redundancy, an NDM defining the ANN should contain the same values replicated in different locations. To achieve such an effect, mechanisms are necessary which would enable AEPs to repeatedly introduce the same values into NDMs.

The first method which can be used for that purpose is a jump operation. Each jump makes it possible to repeatedly use the same code of AEP in different places of NDM (details of the jump are presented in [20]). A next solution which enables AEPs to form modular/regular neural architectures is repeated use of the same data by operations. When working the operations often have to use the information placed in the memory part of each AEP. Because the data are common for all operations included in the same AEP, different operations can use the same data. This means that the information contained in the data part of AEP can be used many times to alter various fragments of NDM. In effect, NDM can include the same elements in many locations, which is the base for modular/regular neural architectures to arise.

It is also possible to repeatedly use the same data by a single operation, e.g. CHGM0 presented in Listing 1. The task of this operation is to modify a fragment of NDM. Elements of the matrix are updated in columns, one after another, starting from the element pointed by parameters p_0 and p_1. The number of updated elements and a place in the memory where new values for the elements are located are determined in parameters p_2 and p_3. Generally, the operation can modify all elements in NDM (see variable iterations, Listing 1). To this end, the operation uses data from AEP. However, AEP usually does not include enough data to assign a different value to each element of NDM. In consequence, the operation very often uses the same data many times. This is possible thanks to the application of mod operator (see the last line in CHGM0) [20].

Listing 1. Implementation of CHGM0

```
CHGM0(p0,p1,p2,p3)
{
rowInit=abs(p0);
columnInit=abs(p1)-1;
iterations=abs(p2)mod(NDM.width*NDM.height);
for(i=0;i<=iterations;i++){
  sumRow=i mod NDM.height;
  if(sumRow == 0)
    columnInit++;
  row=(rowInit+R1+sumRow)mod NDM.height;
  column=(columnInit+R2)mod NDM.width;
  NDM[row,column]=D[(abs(p3)+i)mod D.length]/MaxValue;}
}
```

In AE, modular/regular ANNs can also be established by means of operations especially constructed for that purpose. For example, to create an ANN with two the same neurons, it is only enough to apply a special operation defining a single neuron but in two different places of NDM. Introducing the same values into many columns or rows is another example of the operation specialized in producing modular/regular ANNs. AE makes it also possible to easy form ANNs with a predefined modular architecture, e.g. ANNs with a layered organization. To obtain such ANNs, appropriately constructed operations are only necessary. The task of each of them could be to form a separate layer or module.

Generally, AE provides the whole range of mechanisms to build ANNs consisting of many repeated components. Noteworthy is the fact that such ANNs, may be theoretically obtained by means of very simple AEPs. For example, to produce an ANN with, say, three layers, an AEP with three simple operations is sufficient (the task of each operation would be to update elements in NDM which correspond to weights of neurons included in the same layer). It seems that achieving a similar effect by means of such methods as CE, EE, and others based on tree-structured-chromosomes would require applying more complex programs. However, in order for AE to produce ANNs with repeated components, mechanisms alone are insufficient. The method has to know how to use them. To test the true ability of AE to create modular/regular ANNs, experiments were carried out whose results are presented in the following section.

IV. EXPERIMENTS

The task of each ANN evolved during the experiments was to control two or three carts with inverted pendulums. Since both the carts and the pendulums were identical, to control them, neuro-controllers including two or three separate sub-controllers of the same structure seemed to be the most natural. However, the information about preferred topology of neuro-controllers were not accessible to the method building ANNs. There were not any hints suggesting which ANNs should be built and which should not. Thereby, ANNs of any topology could appear during the experiments. The problem was, however, whether AE is able to pick an ANN with a specific modular and regular architecture out from many other possible neural solutions. Types of possible modules/units (neurons, sub-ANNs) and their size (number of NDM cells necessary to define a module/unit) were other important unknowns during the tests.

Selection of a testing problem to measure the ability of AE to produce modular/regular ANNs was dictated by simplicity of analysis of neural solutions in terms of their modularity and regularity. During the experiments, only ANNs which included at least one separate sub-ANN to control a single cart were recognized to be modular. This means that in order for a unit of an ANN to be a module it had to be completely separated from the rest of the ANN and it had to be responsible for controlling its own cart. Regularity in ANNs was measured through analysis of repetitiveness in NDMs. Each ANN whose NDM included repeated elements was recognized to be regular. For an ANN to be both modular and regular it had to have separated modules repeated in NDM at least twice.

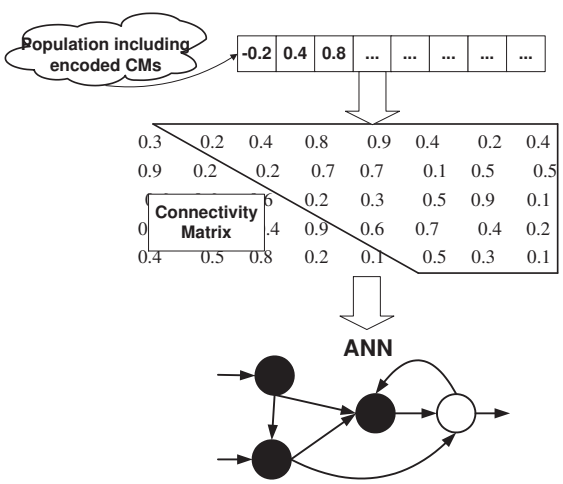


Fig. 3. Evolution of ANNs in CM (evolving elements of CM are enclosed – each chromosome encodes a single ANN and it includes all parameters necessary to build the network; evolution proceeds in a single population, to evolve ANNs Canonical GA is used)

In the experiments, apart from AE, CM was also tested. The main reason for using this method was the need to have

a point of reference for AE. Since CM, in contrast to AE, does not provide any tools for creating architectures with repeated units, using this method in the experiments allowed us to answer the question whether the mechanisms used in AE to create the above architectures are effective and whether AE can effectively use these mechanisms. Parameters of both AE and CM are given in Tab. 2.

Of course, CE and EE, i.e. the methods which like AE support modularity in ANNs, would be the best point of reference for AE. However, implementations of the above methods were inaccessible during the experiments which in effect made comparison between them and AE impossible.

IV. 1. Inverted pendulum problem

In the classic variant of the inverted pendulum problem, we deal with a wheeled cart moving on a finite length track and with a pole installed on the cart. In the problem, the task is to indefinitely balance the pole and to keep the cart within track boundaries. To accomplish the task, the cart is pushed left or right with some fixed force. The decision about the direction and the strength of movement is made based on the information about the state of the cart-and-pole system. The state vector includes such parameters as: the position of the cart (x), the velocity of the cart ($\dot{x}$), the angle of the pole (θ), and the angular velocity of the pole ($\dot{\theta}$). To model the behavior of the cart-and-pole system the following equations can be used [8]:

$$\ddot{x} = \frac{F - \mu_c(\dot{x}) + \sum_{i=1}^{N} \widetilde{F_i}}{M + \sum_{i=0}^{N} \widetilde{m_i}} \tag{1}$$

$$\ddot{\theta}_i = -\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_{pi}\dot{\theta}_i}{m_i l_i}\right), \tag{2}$$

where

$\ddot{x}$ – is acceleration of cart

$\ddot{\theta}_i$ – is acceleration of i^{th} pole

F – is force put to cart

M – is mass of cart

m_i – is mass of i^{th} pole

l_i – is half length of i^{th} pole

μ_c – is coefficient of friction of cart on track

μ_{pi} -is coefficient of friction of i^{th} pole's hinge

g – is gravity

$$\widetilde{F_i} = m_i l_i \theta_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_{pi}\dot{\theta}_i}{m_i l_i} + g\sin\theta_i\right) \tag{3}$$

$$\widetilde{m_i} = m_i\left(1 - \frac{3}{4}\cos^2\theta_i\right) \tag{4}$$

The equations above are used to calculate the state parameters of the cart-and-pole system. The calculations can be performed by means of Euler's method:

$$x = x + t\dot{x} \tag{5}$$

$$\dot{x} = \dot{x} + t\ddot{x} \tag{6}$$

$$\theta = \theta + t\dot{\theta} \tag{7}$$

$$\dot{\theta} = \dot{\theta} + t\ddot{\theta} \tag{8}$$

where t is a step size. Values of the parameters above, (in the experiments, values of all the parameters were scaled to <-1,1>) fixed at each time step, i.e. every t seconds, are used by a neuro-controller, to determine a direction and a strength of a next move. Each decision of a neuro-controller causes the cart-and-pole system to be transitioned to a next state described by other values of state parameters. This procedure is repeated until the pole falls down, the cart accomplishes the end of the track, or the pole is balanced over some assumed number of steps (in the experiments 100000 steps).

In the experiments reported further, ANNs dealt with an extended version of the classic variant of the inverted pendulum problem described above. The task of each ANN was not to control a single cart-and-pole system as is in the classic variant but to move two or three carts with their pendulums. In consequence, ANNs had the following architecture:

1. Two cart-and-pole systems (8 inputs, 4 hidden units, and 2 outputs – inputs: $x_1, \dot{x}_1, \theta_1, \dot{\theta}_1, x_2, \dot{x}_2, \theta_2, \dot{\theta}_2$ outputs: F_1, F_2).
2. Three cart-and-pole systems (12 inputs, 4 hidden units, and 3 outputs – inputs: $x_1, \dot{x}_1, \theta_1, \dot{\theta}_1, x_2, \dot{x}_2, \theta_2, \dot{\theta}_2, x_3, \dot{x}_3, \theta_3, \dot{\theta}_3$ outputs: F_1, F_2, F_3).

IV. 2. Experimental results

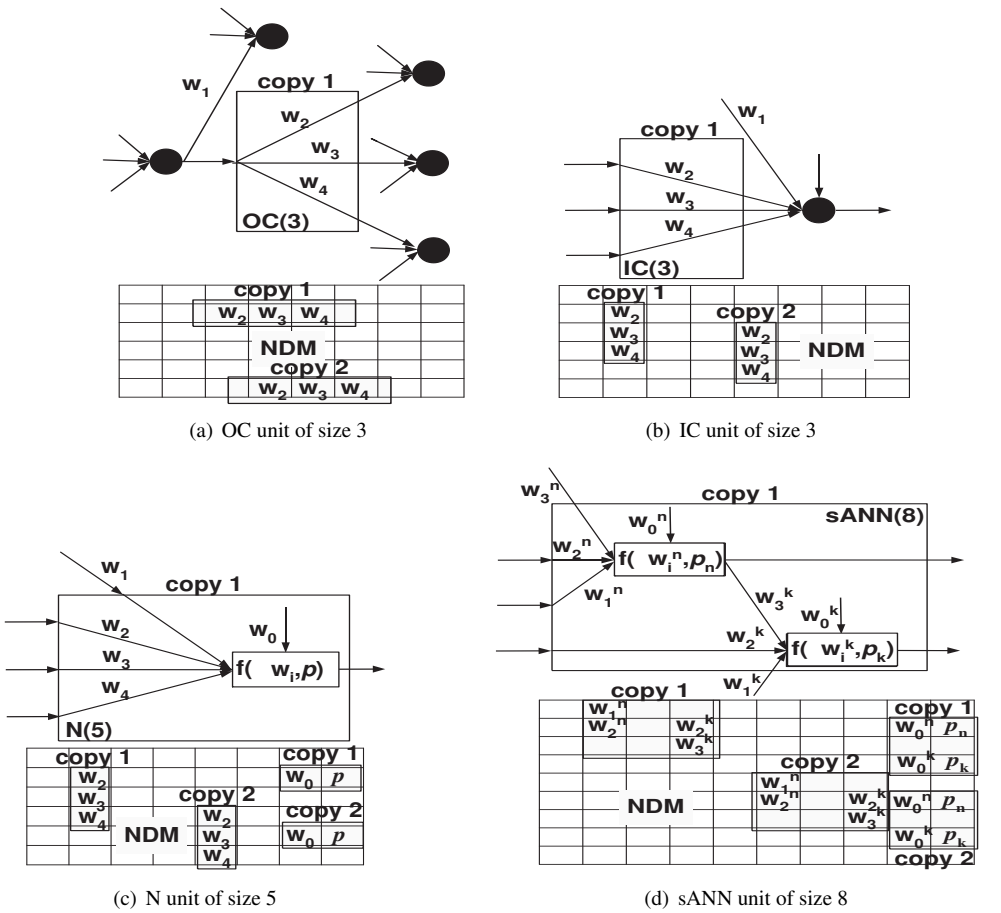
ANNs evolved during the tests were primarily examined in terms of their modularity and regularity. ANNs could be modular, regular, modular and regular or none of them. Moreover, ANNs were also analyzed in terms of type and size of repeated units. The goal of this analysis was to test whether AE is able to evolve ANNs with large-scale regularities, i.e. with repeated sub-ANNs or at least neurons. In order for such ANNs to evolve, it was necessary to fill in a few disjoint fragments of NDM in the same way which was a serious challenge for AEPs. Generally, the following types of units were looked for in ANNs (see Fig. 4):

- output connections (OC units),
- input connections (IC units),
- neurons (N units),
- sub-ANNs (sANN units, it was assumed that such units have to include at least two neurons).

The experiments were carried out in two phases. In the first phase, the task of ANNs was to control two carts with inverted pendulums. The second phase was more difficult. In this phase, instead of balancing two poles, ANNs dealt with three poles. To create ANNs, regardless of the phase, AEPs consisting of only two operations were used (the list

Tab. 1. Parameter settings for each cart-and-pole system [8]

Param.	Description	Parameter value
l	length of pole	0.5m
m	mass of pole	0.1kg
θ	angle of pole	(-12,12) deg.
	failure angle (pole is considered to be fallen down)	$\pm$12 deg.
x	position of cart	(-2.4,2.4)m
	failure position of cart (cart is outside track)	$\pm$2.4m
M	mass of cart	1kg
F	force applied to cart	<-10,10>N but no less than $\pm$1/256x10 N
g	gravity	9.8m/s^2
μ_c	coefficient of friction of cart on track	0.0005
μ_p	coefficient of friction of i-th pole's hinge	0.000002
t	step size (cart is moved every t seconds)	0.02s
	maximal number of steps pole remained balanced	100000



(a) OC unit of size 3

(b) IC unit of size 3

(c) N unit of size 5

(d) sANN unit of size 8

Fig. 4. Example modules and their representation in NDM (in brackets next to labels of units the size of each unit is given, parameters included in rectangles repeatedly appear in NDMs and in consequence in ANNs)

Tab. 2. Parameters of AE and CM used in experiments

Parameter	AE	CM
no. of subpopulations	3 (operation1, operation2 and data)	1
size of subpopulations	60 (data), 20 (operations)	100
evals per generation	100	100
no. of data	10..20	
size of ANNs (hidden units)	maximally 4	maximally 4
size of tournament	1 (data), 4 (operations) or 1 (data), 2 (operations)	2 or 4
crossover probability	0.7 in all three populations	0.7
mutation probability	0.04 (data), 0.1 (oper.) or 0.04 (data), 0.05 (oper.)	0.01 or 0.05

of all operations available to AEPs is given in Appendix). In the experiments in both phases, AE as well as CM were run 100 times, i.e. in each phase 100 ANNs were built for each method (each run was represented by the best ANN produced in the run).

Since the main goal of the experiments was not to find effective ANNs as quickly as possible but to test the ability of AE and CM to form modular/regular ANNs, both methods were not tuned to the task and all the tests were carried out for a parameter setting, the most effective in the previous experiments [21].

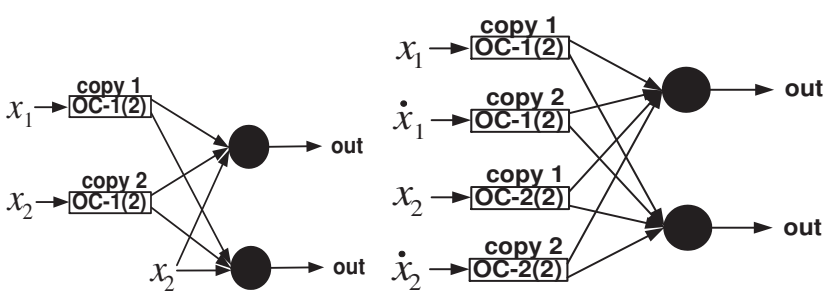
IV. 2. 1. Two cart-and-pole systems

Each ANN created in this part of the experiments was evaluated eight times, i.e. eight balance attempts were performed for each of them. All the attempts differed in initial positions of the poles: (θ_1[deg],θ_2[deg])=(4,4), (4,-4), (-4,4), (-4,-4), (8,8), (8,-8), (-8,8), (-8,-8). Initial values for the remaining parameters in both cart-and-pole systems were the same during each attempt, i.e. x_1=0 m, $\dot{x}_1$=0 m/s, $\dot{\theta}_1$=0 deg/s, x_2=0 m, $\dot{x}_2$=0 m/s, $\dot{\theta}_2$=0 deg/s. The fitness for each ANN was determined by the total number of steps the pole remained balanced in all the attempts. When in a single attempt the pole was up for 100000 time steps, the attempt was interrupted. All the analysis presented below was made on ANNs which successfully protected the pole from falling down in all the eight attempts.

Before the experiments, the most natural solution to the problem with two the same cart-and-pole systems seemed to be a controller consisting of two separate copies of the same sub-controller. The experiments showed something else. A massive majority of ANNs produced with AE (92%) consisted of two different output neurons. Besides, densely connected neural architectures including usually a maximum acceptable number of neurons (see Tab. 2) were also created: CM – 100% of ANNs, AE – 8% of ANNs.

As for ANNs with two output neurons, their architecture seems to mainly arise from differences in initial positions of the poles. Since each pole started from a different position, the problem which had to solve ANNs was portioned into two different sub-problems, each of which was solved by means of a different neuron. Noteworthy is the fact that in spite of many other architectures which could be used to solve the problem, and in spite of the ability of AE to form these architectures, the method suggested exactly such a simple modular solution. As it turned out CM was unable to do the same.

Most ANNs with two output neurons (94%), in addition

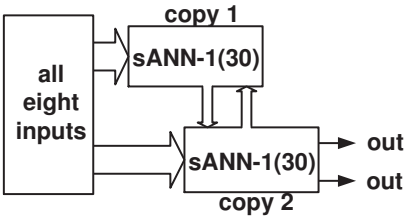


(a) two copies of OC unit of size 2 (b) two OC units of size 2, each with two copies

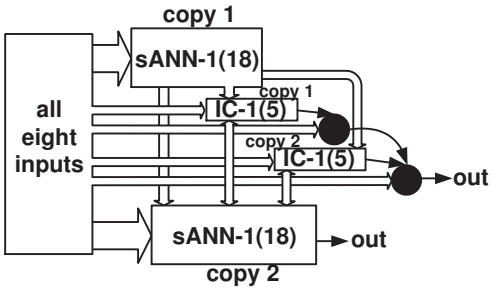
Fig. 5. Examples of typical ANNs created by means of AE in the experiments with two cart-and-pole systems

to heterogeneous modules, included also replicated OC units of size 2. In most cases, the ANNs contained only one pair of such units (Fig. 6(a)) although there were also cases with two pairs (Fig. 6(b)).

As mentioned above, the remaining ANNs produced with AE were densely connected and usually included the maximum acceptable number of neurons. Such construction of ANNs created favorable conditions for many replicated units of different size to arise. sANN units containing two or three neurons were the greatest of them. Usually, a single ANN included many different units of varied size. Very often, the units overlapped with the result that ANNs were decomposable into units in many different ways (Fig. 6).



(a) two copies of sANN unit of size 30



(b) two copies of sANN unit of size 18 and two copies of IC unit of size 5

Fig. 6. Two example ways to decompose the same ANN into units

In contrast to ANNs generated by means of AE, the ones evolving as CMs did not include any modules and replicated units of size larger than 2 (Tab. 4). What is more, the number of ANNs with the units was very small (12%). Noteworthy is the fact that, in the case of AE, the architectures with many neurons and connections included many repetitions of units of different size. Meanwhile, ANNs created by means of CM, in most cases, included no units.

Tab. 3. Modular, regular and modular and regular ANNs in first phase of experiments

	Modular	Regular	Modular and Regular	None
AE	6%	8%	86%	0%
CM	0%	12%	0%	88%

Tab. 4. Type and size of units in first phase of experiments

	Types of units				Size of units		
	OC	IC	N	sANN	Max	Min	Average
AE	√	√	√	√	35	2	3.1
CM	√	√	X	X	2	2	2

IV. 2. 2. Three cart-and-pole systems

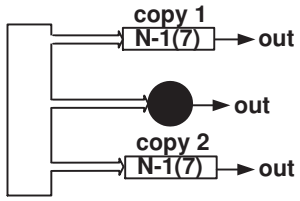
The experiments with three cart-and-pole systems were carried out in the same conditions as the ones with two systems. The number of evaluations of each ANN, parameters of the systems, and the method for calculating fitness were identical as in the previous case. The only new element was the third cart. Since in the previous experiments we did not obtain any ANN consisting of the same, separate sub-ANNs, in the currently presented experiments, we decided to encourage AE to build such architectures by setting the additional pole (the third pole) in the same initial position as the first pole. The starting position of the second pole was as before.

Tab. 5. Modular, regular and modular and regular ANNs in second phase of experiments

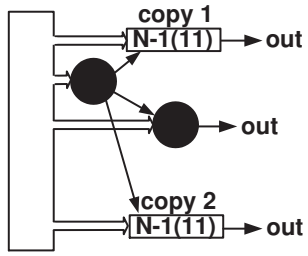
	Modular	Regular	Modular and Regular	None
AE	0%	32%	65%	3%
CM	0%	5%	0%	95%

During the tests, it turned out that all AEPs produce ANNs with a very similar architecture. Again, the modular architecture with only output neurons appeared to be dominant. In all the experiments at this stage, AEPs generated only a few ANNs with hidden neurons. The characteristic of most ANNs produced with AE was also significant similarity or in many cases even identicalness of neurons responsible for controlling the systems staring from the same initial position (Fig. 7).

The architecture of ANNs affected types of modules/units contained therein. Again, it is necessary to take notice of the fact that in spite of many other feasible neural solutions, most ANNs produced with AE (65%) consisted of three separate modules, including one N module in two copies and one module without replication. In addition to repeated N modules, repeated IC units were also a frequent case (they occurred in 30% of ANNs). They usually played a similar role as the N modules mentioned above, i.e. they were responsible for the cart-and-pole systems starting with the same initial position. In contrast to the N modules and IC units, repeated OC units were very rare. They appeared in 2% of ANNs, typically, together with IC units. In the experiments, there were also ANNs without any repeated units. However, the number of such ANNs was very small – 3% (as in the previous case, ANNs produced with CM were exceptional in this respect).



(a) two copies of N module of size 7



(b) two copies of N module of size 11

Fig. 7. Examples of typical ANNs created during experiments with three cart-and-pole systems

Tab. 6. Type and size of units in second phase of experiments

	Types of modules				Size of modules		
	OC	IC	N	sANN	Max	Min	Average
AE	√	√	√	X	11	2	3.4
CM	√	√	X	X	2	2	2

With regard to the size of repeated modules/units (Tab. 6), although most of them were of size 3 or 4, the modules/units of larger size (e.g. 11, 10, 8 or 7) were also generated (see Fig. 7). Generally, the repeated units from this part of the experiments were of larger size than those generated previously. There were, however, no units as large in size as the largest units evolved in the prior stage of the experiments. Previously, the largest replicated units were always a part of densely connected ANNs with many neurons. Since this time, all ANNs were smaller in size than the largest ANNs produced previously, such units had no chance to arise.

With regard to ANNs evolving as CMs, the results were almost the same as in the previous case. That is, all the ANNs were fully-connected with the maximum acceptable number of neurons, and in total, they included only a few repeated IC and OC units of size at most 2.

V. CONCLUSION

The paper reports the experiments whose the main goal was to test whether AE is able to adjust the architecture of evolved neural solutions to a modular and regular problem. During the tests, it appeared that massive majority of ANNs have the architecture ideally suited to a problem to be solved. They included two or three separated output modules, each of which was responsible for its own cart-and-pole-system. Noteworthy is the fact that even though AE enables forming ANNs with any structure, in the experiments, very simple modular and regular architectures were mostly produced. Modules and repeated units were of varied size and type. In addition to small size modules and units, more complex ones were also noticed. The emergence of larger repeated units should be particularly emphasized because it may be a symptom that AE is able to produce complex ANNs with no less complex repeated sub-ANNs. Of course, to test whether AE really displays such abilities, further experiments are necessary.

CM which like AE has the ability to produce diverse ANNs in terms of the architecture could not achieve a similar result. All ANNs constructed with CM were not adjusted to a task, were fully-connected and included a maximum number of neurons. In all the experiments, the method was unable to propose any other effective architecture. It seems, however, that troubles of CM with producing an appropriate architecture also involve other direct methods. Each direct method strives to attach a value to each parameter of an ANN included in a genotype. To remove an element from the ANN, e.g. a connection, an appropriate gene has to be set to zero. Since the value zero is only one out of many other possibilities, it is very difficult to obtain any other architecture of an ANN than the maximum architecture memorized in a genotype. Therefore, it seems that the direct methods are the most suited for the problems with topology of ANNs known beforehand. In other situations, the indirect methods, like AE, appear to be a valuable alternative.

APPENDIX
List of operations used in the experiments

`CHGFF` – update of the 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 column in NDM

`CHGR0` – update of a fragment of row in NDM

`CHGM0` – Listing 1

`CHGM1` – like `CHGM0`, the difference is that all the updated elements have the same value.

`CHGRR0(CHGCC0)` – like `CHGR0(CHGC0)`, the main difference is that `CHGR0(CHGC0)` updates elements in a single row (column) of NDM whereas `CHGRR0(CHGCC0)` operates in two adjacent rows (columns).

`CHGRR1 (CHGCC1)` – like `CHGRR0(CHGCC0)`, however, `CHGRR1 (CHGCC1)` works in rows (columns) that are not adjacent.

`CHGRR2(CHGCC2)` – modification of more than two rows (columns).

`CHGMM0` – like `CHGM0`, however `CHGMM0` works in two adjacent blocks of NDM.

`CHGMM1` – a variety of `CHGMM0` in which modified blocks can be apart.

`CHGMM2` – like `CHGMM0`, the only difference between the operations is an order in which elements of NDM are updated. In `CHGMM0`, elements are modified in rows whereas in `CHGMM2` they are changed in columns.

`JMP1` – a form of `JMP` in which the maximum number of iterations is restricted to four.

References

[1] K. Balakrishnan, V. Honavar, *Properties of Genetic Representations of Neural Architectures*, Proc. of the World Congress on Neural Networks (WCNN'95), 807-813, (1995).

[2] R. Calabretta, *Genetic Interference reduces the evolvability of modular and non-modular visual neural networks*, Phil. Trans. R. Soc. B **362**, 403-410, doi:10.1098/rstb.2006.1967 (2007).

[3] S. Cho, K. Shimohara, *Evolutionary Learning of Modular Neural Networks with Genetic Programming*, Applied Intelligence **9**, 191-200 (1998).

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

[5] F. Gruau, *Automatic Definition of Modular Neural Networks*, Adaptive Behavior **3**, Issue 2, 151-183 (1994).

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

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

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

[9] V.R. Khare, X. Yao, B. Sendhoff, Y. Jin, H. Wersing, *Co-evolutionary Modular Neural Networks for Automatic Problem Decomposition*, in Proc. of The 2005 IEEE Congress on Evolutionary Computation, **3**, 2691-2698, doi:10.1109/CEC.2005.1555032 (2005).

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

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

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

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

[14] J. Mouret, S. Doncieux, *Evolving modular neural-networks through exaptation*, Proc. of the Eleventh Conf. on Congress on Evolutionary Computation, 1570-1577 (2009).

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

[16] N. NourAshrafoddin, A.R. Vahdat, M.M. Ebadzadeh, *Automatic Design of Modular Neural Networks Using Genetic Programming*, Proc. of the 17th International Conference on Artificial Neural Networks, 788-798 (2007).

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

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

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

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

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

[22] T. Praczyk, *Forming Neural Networks by Means of Assembler Encoding*, Intelligent Automation and Soft Computing **17**, no. 3, 319-331 (2011).

[23] T. Praczyk, *Assembler Encoding Improved*, CMST 18(1), 11-24, (2012).

[24] 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).

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.

DOI:10.12921/cmst.2013.19.03.145-155