Assembler Encoding Versus Connectivity Matrix Encoding in the Inverted Pendulum Problem with a Hidden State

Tomasz Praczyk

The Naval University, ul. Śmidowicza 69, Gdynia, Poland
e-mail: T.Praczyk@amw.gdynia.pl

Abstract: Assembler Encoding is the Artificial Neural Network encoding method. To date, Assembler Encoding has been tested in the optimization problem and in the so-called predator-prey problem. The paper reports experiments in a next test problem, i.e. in the inverted pendulum problem. To compare Assembler Encoding with other Artificial Neural Network encoding methods in the experiments, two direct encodings were also tested.

Key words: evolutionary neural networks

I. INTRODUCTION

Artificial Neural Networks (ANNs) are created by means of different methods, including Genetic Algorithms (GAs) [4, 5, 12]. GA processes a population of genotypes that typically encode one phenotype, although encoding several phenotypes is also possible. In the neuroevolution, genotypes are encodings of corresponding ANNs (phenotypes). The evolutionary procedure involves selecting genotypes (encoded ANNs) for reproduction based on their fitness, and then by introducing genetically changed offspring into the next population. Repeating the whole procedure over many generations causes the population of encoded ANNs to gradually evolve into individuals corresponding to high fitness phenotypes (ANNs).

In principle, all the existing ANN encoding methods [3, 7, 8, 11, 13-15] can be divided into two main classes, i.e. direct encodings and indirect encodings. In the direct encodings, all the information necessary to create ANN is directly stored in chromosomes. Thus, to encode larger ANNs larger chromosomes are necessary, which is the main drawback of the direct encodings. In the indirect encodings, chromosomes are recipes how to create ANN. Such encodings can be used to create larger ANNs by means of relatively short chromosomes.

The paper presents a new indirect method to encode ANNs. The method is called Assembler Encoding (AE) [21-23]. AE originates from the cellular [7] and edge encoding [11], although it also has features common with Linear Genetic Programming presented, among other things, in [9, 16]. AE represents ANN in the form of a program called the Assembler Encoding Program (AEP). The structure of AEP is similar to the structure of a simple assembler program. AEPs are formed by means of GAs. The task of each AEP is to create the Network Definition Matrix (NDM) which stores all the information necessary to create ANN. In AE, the process of ANN construction consists of three stages. First, GA is used to produce AEPs. Next, each AEP creates and fills up NDM. Then, the matrix created is transformed into ANN.

To compare AE with other methods, several experiments have been carried out so far. The experiments involved the optimization problem [21, 22] and the predator-prey problem [21, 23]. The current paper gives a report on tests in another field. The next test-bead for AE was the inverted pendulum problem. In the tests, for comparison purposes, two simple direct encodings based on the Miller et al. connectivity matrix (CM) [13] were also used.

The paper is organized as follows: Section II is a short presentation of AE; Section III is a short presentation of compared methods and a report on the experiments; and Section IV is the summary.

II. ASSEMBLER ENCODING

Since AE was already described in [23], this section contains only a short presentation of the method. In AE, ANN is represented in the form of AEP which is composed
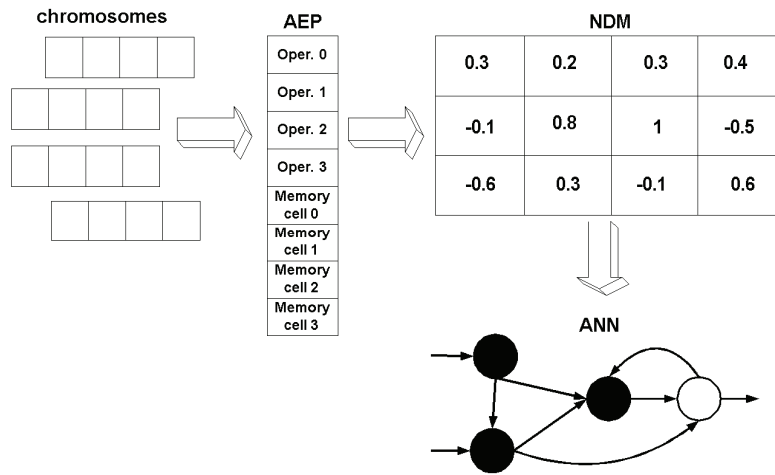
Fig. 1. Diagram of AE

of a part including operations and a part including data. The task of AEP is to create and fill NDM with values. To this end, AEP uses operations that are run in turn. When working, the operations can use data located at the end of AEP. Once the last operation ends its work, the process of creating NDM is completed and the matrix is transformed into ANN (Fig. 1).

NDM determines the architecture of ANN. Each element of the matrix determines synaptic weight between corresponding neurons. For example, component$_{i,j}$ defines the link from neuron i to neuron j. Apart from the basic part, NDM also includes additional columns that describe parameters of neurons, e.g. type of neuron (sigmoid, radial), bias, etc.

AEPs can use various operations. The main task of most operations is to modify NDM. The modification can involve a single element of NDM or group of elements. In addition to the operations whose task is to modify the content of NDM, AE also uses the jump operation denoted as JMP. The jump makes it possible to repeatedly use the same code of AEP in different places of NDM.

In AE, the evolution of AEPs proceeds according to a scheme proposed by Potter and De Jong [17-20]. To create AEP, the mentioned scheme combines operations and data from various populations. Each population including chromosomes-operations has a number assigned determining the position of the operation from the population in AEP. In this approach, the number of operations corresponds to the number of populations including chromosomes-operations. Each population delegates exactly one representative to each AEP. In the beginning, AEPs have only one operation and a sequence of data. Both the
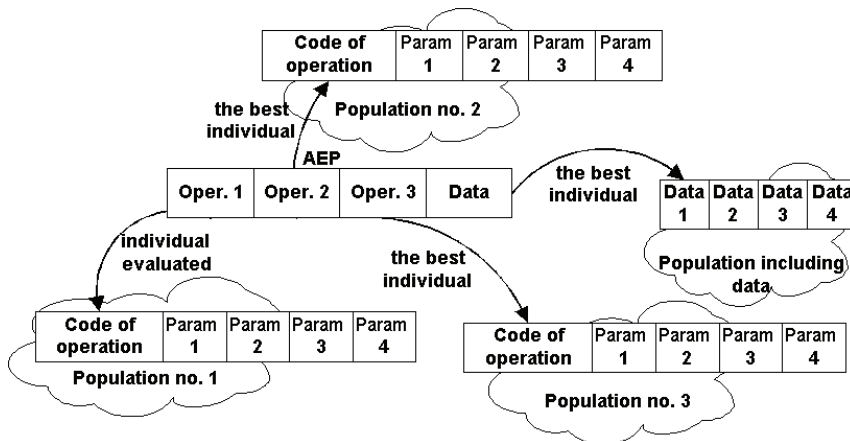

Fig. 2. Evolution of AEPs

operation and data come from two different populations. Further populations of operations are successively added if generated AEPs cannot accomplish progress in performance over the assumed number of co-evolutionary cycles. Populations with operations and data can also be replaced by newly created populations. This can happen if the contribution of a population (contribution of operations from population) to creating AEPs is considerably smaller than the contribution of the remaining populations.

III. EXPERIMENTS

The main goal of the experiments was to compare AE with other neuroevolutionary methods in the inverted pendulum problem. In the paper, AE was compared to two different variants of classical CM. The main idea behind comparing AE with CM was that both solutions use a matrix to represent ANN. Both methods differ only in the approaches to creating the matrix. While in the classical solution GA is used directly to form CM, AE uses AEPs formed in the evolutionary way.

III.1. The inverted pendulum problem

The main goal in the inverted pendulum problem is to protect a pole installed on a card from falling down. In order for the pole to be up, the card has to be pushed left or right. At each time step, the strength and direction of the move has to be determined. During the move the card cannot exceed the limit of a track on which it moves. Usually, to keep the pole up, the following state information is used: the position of the card (x), the velocity of the card ($\dot{x}$), the angle of the pole (θ), and the angular velocity of the pole ($\dot{\theta}$). The behavior of the card-and-pole system under the influence of the force F can be presented by means of the following equations [6]:

$$\ddot{x} = \frac{F - \mu_c \, \text{sgn}(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{M + \sum_{i=1}^{N} \tilde{m}_i} \qquad (1)$$

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

where
$\ddot{x}$ – is acceleration of card
$\ddot{\theta}_i$ – is acceleration of ith pole
F – is force put to card
M – is mass of card
m_i – is mass of ith pole

l_i – is half length of ith pole
μ_c – is coefficient of friction of card on track
μ_{pi} – is coefficient of friction of ith pole's hinge
g - is gravity

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

$$\tilde{m}_i = m_i \left(1 - \frac{3}{4}\cos^2\theta_i \right) \qquad (4)$$

In the experiments reported further, the variant of the inverted pendulum problem with incomplete state information about the card-and-pole system was tested. Each neuro-controller created during the tests did not have the information about velocities of the cart and the pole. Usually, to solve such a problem, recurrent ANNs are used. The task of recurrent connections is to provide ANN with the information about the velocities. In the experiments reported in the paper, only feed-forward ANNs were used. To enable feed-forward ANNs to solve the task, the input layer of each ANN was extended to include an additional neuron per state parameter from the immediately previous time step. Thus, each ANN created in the experiments had the following inputs: x_t, x_{t-1}, θ_t, θ_{t-1}.
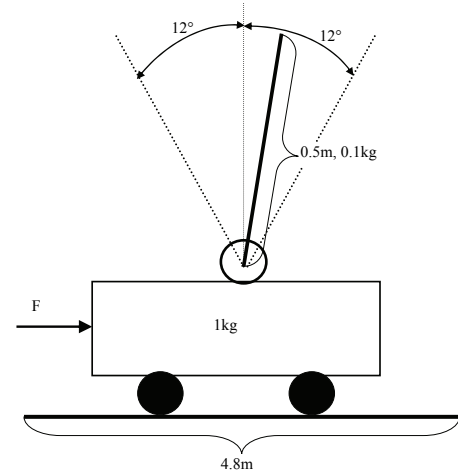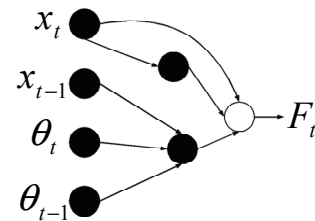

Fig. 3. Card-and-pole system

Fig. 4. Example ANN created in experiments

The remaining parameters essential for the experiments are presented in Tab. 1. Values included in the table are the same as the ones used in the experiments reported in [6, 14].

Table 1. Parameter settings for the inverted
pendulum problem [6, 14]

Param.	Description	Parameter value
l	length of pole	0.5 m
m	mass of pole	0.1 kg
θ	angle of pole	<-12, 12> deg.
	failure angle (pole is considered to be fallen down)	±12 deg.
	initial position of pole	4 deg.
x	position of card	<-2.4, 2.4>m
	initial position of card	0 m
	failure position of card (card is outside track)	±2.4 m
M	mass of card	1 kg
F	force applied to card	<-10, 10>N but no less than ±1/256x10 N
g	gravity	9.8 m/s^2
μ_c	coefficient of friction of card on track	0.0005
μ_p	coefficient of friction of ith pole's hinge	0.000002
t	step size (card is moved every t seconds)	0.02 s
	maximal number of steps pole remained balanced	100 000

III.2. Compared solutions

In the experiments, AE was compared to two methods. Both methods are variants of classical CM. A short description of all the compared methods is presented in the further part of the paper.

III.2.1. Classical Connectivity Matrix (CM)

In this approach, ANNs evolved in a single population. The mentioned population included chromosomes encoding fragments of CMs above the diagonal (feed-forward ANNs). All created ANNs contained a different number of hidden neurons. To obtain such ANNs, chromosomes were of a varied length. Each chromosome encoded weights of interneuron connections and additionally parameters of neurons.
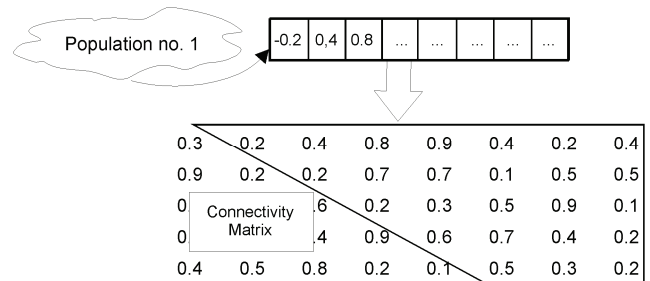

Fig. 5. Evolution of ANNs in CM
(evolving elements of CM are enclosed)

III.2.2. Coevolutionary Connectivity Matrix (CCM)

The evolution of ANNs in CCMs went on in a somewhat different way than in the classical solution presented
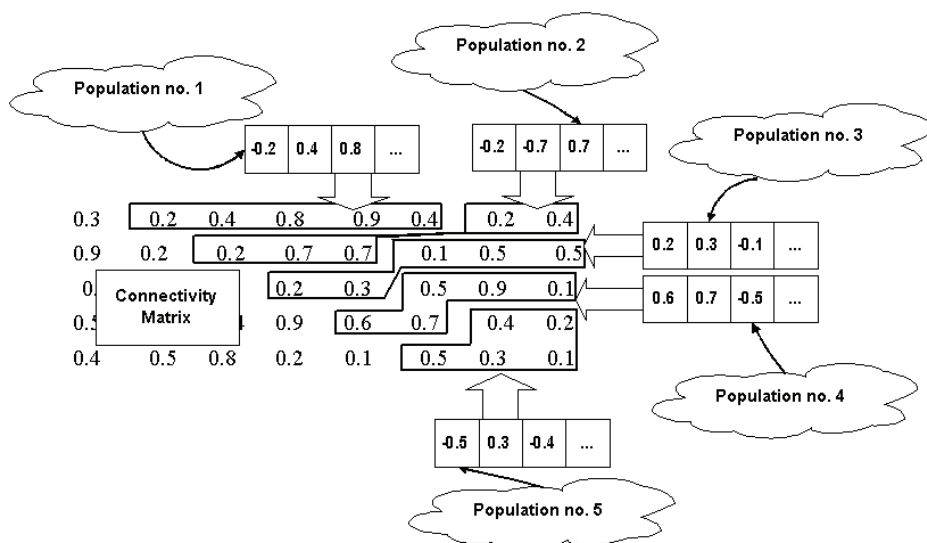

Fig. 6. Evolution of ANNs in CCM (evolving elements of CM are enclosed with five lines; each line encloses elements evolving in the same population)

above. While in the classical approach we dealt with one population of matrices, in CCM the whole CM was divided into parts, and each part evolved in a separate population. As before, to create feed-forward ANNs, only fragments of CMs above the diagonal underwent the evolution. Regardless of the size of ANNs, the evolution always took place in five populations, i.e. CMs were always divided into five parts of more or less the same size.

III.2.3. The version of AE compared to the methods above

In the experiments, AEPs consisting of one operation and a sequence of data were used. AEPs used only one type of operation, i.e. CHGFF. CHGFF updates the part of NDM above the diagonal, i.e. the part defining feed-forward ANN. New values for the elements of NDM are located in the data part of AEP. An example of using CHGFF is presented in Fig. 7.

```
0 0.68254 0.142857 0.730159 0.809524 -0.714286 0.047619 -0.84127 0.269841 0.539683 0.0952381
0 0 0.174603 -0.571429 -0.47619 -0.380952 -0.0793651 -0.777778 -0.603175 0.619048 -0.206349
0 0 0 0.698413 0.111111 -0.142857 0.142857 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.714286 -0.222222 -0.126984
0 0 0 0 0 0 0 0 0.634921 0.555556 0.111111 0.31746
0 0 0 0 0 0 0 0 -0.619048 0.698413 -0.920635
```

Fig. 7. Example use of CHGFF

The remaining parameters for AE and for all the methods compared to AE are presented in Tab. 2 (for all

the methods, different values of parameters were tested; the table includes the best combinations of values of parameters for each method).

Table 2. Parameters of compared methods

	CM	CCM	AE
Population Size	150	30 (each population)	100 (data), 50 (oper)
Length of chromosomes	maximally 52 floats	10 floats (in popul. no. 1-4), 12 floats (in popul. no. 5)	maximally 40 floats (data), 40 bits (oper)
Number of hidden neurons	maximally 3	3	maximally 3
Genetic Algorithm	Canonical GA	Canonical GA (in all five populations)	Canonical GA (in both populations)
Mutation Probability	0.1	0.1	0.03 (data), 0.1 (oper)
Crossover Probability	0.7	0.7	0.7 (oper and data)
Size of Tournament (tournament selection)	4	1	1 (data), 4 (oper)

III.3. Experimental results

In the experiments, the learning speed of AE, CM, and CCM was tested. Thirty ANNs were generated for each
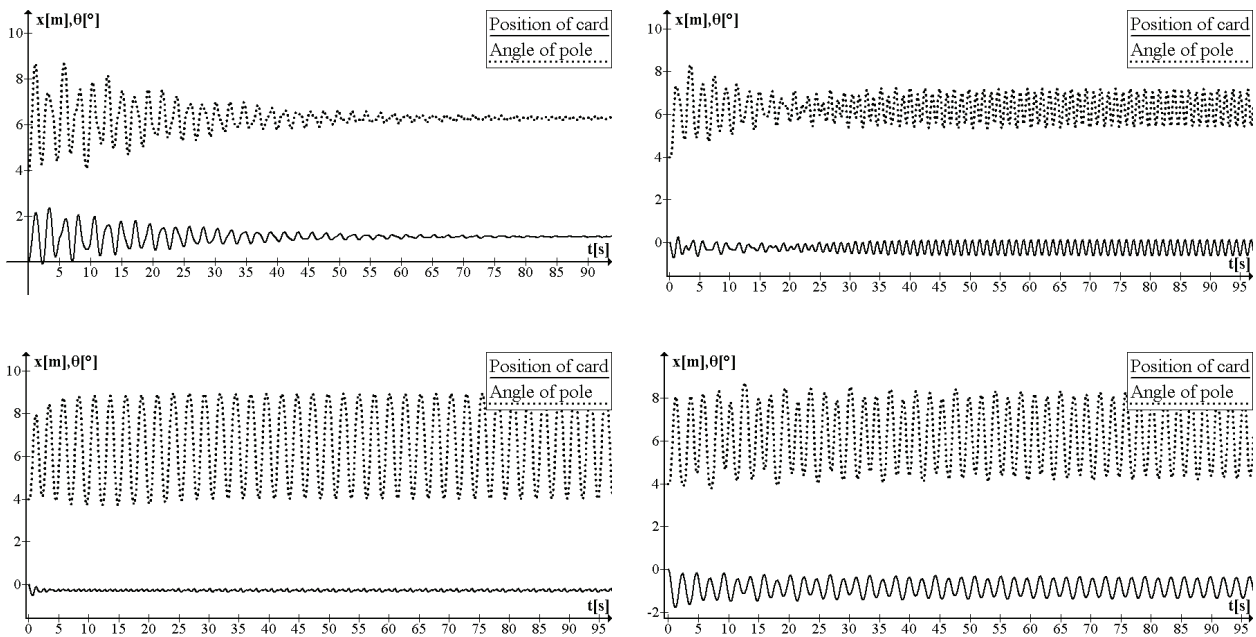

Fig. 8. Example behaviors of the cart-and-pole system (all the figures present the first 100 seconds of balancing the pole; in all the situations presented in the figures the pole was balanced for 100 000 time steps)

method. Each ANN was evaluated once, i.e. a single balance attempt was performed for each ANN. The fitness for each ANN was determined by the number of steps the pole remained balanced. When the pole remained balanced for 100 000 time steps, the test was interrupted. The results of the experiments are presented in Tab. 3. Each cell in the table includes the number of ANNs created up to the point when a successful ANN was formed (the number of evaluations). The successful ANN is the ANN which could balance the pole for 100 000 time steps.

Table 3. Results of experiments

Method	Mean	Best	Worst	SD
CM	27 620	154	393 873	75 409
CCM	29 462	532	95 543	48 093
AE	11 794	487	74 770	15 898

Table 3 shows that on average AE needs fewer ANN evaluations to generate successful ANN than CM and CCM. In the case of AE, the successful ANNs were generated on average after 11 794 evaluations. The result of CM and CCM is almost three times worse. They generated the successful ANNs on average after 27 620 and 29 462 evaluations, respectively.

In the experiments, most ANNs created by means of AE included only one or two hidden neurons. NDMs were of maximal size; however, AEPs usually did not fill them entirely with values (Fig. 7). In the case of CM and CCM, both NDMs and ANNs were of maximal size.

IV. SUMMARY

The paper compares AE with two variants of CM on the inverted pendulum problem. The experiments showed that AE is an effective tool to solve the inverted pendulum problem. Even though AE is an indirect method which seems to be rather suited to solve more complex problems, it achieved better results than two direct methods presented in the paper. To test AE in a more difficult problem, the experiments in the variant of the inverted pendulum problem with two poles installed on the card are planned. In the future experiments, AE will be compared to the next Reinforcement Learning (e.g. Q-learning [1, 2, 10]) and Evolutionary Reinforcement Learning methods (e.g. SANE [14], NEAT [24, 25] and CoSyNE [6]).

References

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

[2] P. Cichosz, *Learning systems*. WNT, Warsaw (2000).

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

[4] D. Curran and C. O'Riordan, *Applying Evolutionary Computation to Designing Networks: A Study of the State of the Art*. National University of Ireland, technical report NUIG-IT-111002 (2002).

[5] D. Floreano and J. Urzelai, *Evolutionary robots with online self-organization and behavioral fitness*. Neural Networks 13, 431-443 (2000).

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

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

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

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

[10] M.L. Littman and C.A. Szepesvari, *Generalized Reinforcement-Learning Model: Convergence and Applications*. In Proceedings of the Thirteenth International Conference on Machine Learning 310-318 (1996).

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

[12] M. Mandischer, *Representation and Evolution of Neural Networks*. In: Albrecht R.F., Reeves C.R., Steele U.C., ed., Artificial Neural Nets and Genetic Algorithms (Springer Verlag, New York, 1993) 643-649.

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

[14] 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).

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

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

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

[18] M. Potter and K.A. De Jong, *Evolving neural networks with collaborative species*. In: Oren T.I., Birta L.G. (Eds.), Proceedings of the 1995 Summer Computer Simulation

Conference, The Society of Computer Simulation 340-345 (1995).

[19] M.A. Potter and K.A. De Jong, *A Cooperative Coevolutionary Approach to Function Optimization*. The Third Parallel Problem Solving From Nature, Jerusalem, Israel, Springer-Verlag 249-257 (1994).

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

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

[22] T. Praczyk, *Procedure application in Assembler Encoding.* Archives of Control Science 17(LIII), 1, 71-91 (2007).

[23] T. Praczyk, *Modular Neural Networks in Assembler Encoding*. Computational Methods in Science and Technology 14(1) (2008).

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

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

TOMASZ PRACZYK. *Education:* Military University of Technology, Warsaw – MSc (1996); Maritime University, Szczecin – PhD (2001). *Activities:* intelligent navigational systems, neural networks, genetic algorithms, neuroevolution, evolutionary reinforcement learning.