Biuletyn WAT
Vol. LVIII, Nr 2, 2009

Organization of the evolutionary process responsible for creating neural networks in assembler encoding

TOMASZ PRACZYK

Naval University, 81-103 Gdynia, Śmidowicza 69, Poland

Abstract. Assembler Encoding (AE) represents Artificial Neural Network (ANN) in the form of a simple program called Assembler Encoding Program (AEP). The task of AEP is to create the so-called Network Definition Matrix (NDM) including all the information necessary to construct ANN. AEPs and in consequence ANNs are formed by means of evolutionary techniques.
To make AE an effective tool for creating ANNs it is necessary to appropriately organize all the evolutionary processes responsible for generating AEPs, i.e., it is necessary to properly select values of different parameters controlling the evolutionary process mentioned. To determine optimal conditions of the evolution in AE, experiments in a predator-prey problem were performed. The results of the experiments are presented at the end of the paper.
Keywords: evolutionary neural networks
Universal Decimal Classification: 007

1. Introduction

ANNs constitute a sub-domain of artificial intelligence that is broadly used to solve various problems in different fields (e.g. pattern classification, function approximation, optimization, image compression, associative memories, robot control problems, etc.). The performance of ANNs highly depends on two factors, i.e., network's topology and a set of network's parameters (typically weights). Therefore, to develop an appropriate network it is necessary to determine the architecture and parameters. There are many different ANN learning algorithms (e.g. BackPropagation) that change values of parameters leaving the structure completely intact. In such a case, the process of searching for the proper network's topology is the task of a network's designer who arbitrarily chooses the network's

structure, starts network learning and finally puts the network to a test. If the result of the test is satisfactory, the learning process is stopped. If not, it is continued further. The designer manually determines the next potential network's topology and runs the learning algorithm again. Such loop — topology determination and learning is repeated until the network which is able to carry out a dedicated task at an appropriate level is found. At first glance, it is apparent that such a procedure could be very time-consuming and, what is worse, in the case of more complex problems, can lead to a situation when all chosen and trained networks would be incapable of solving the task.

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

Genetic Algorithms (GAs) are the next technique that has been successfully applied to search for optimal ANNs [4, 5, 11] for the recent years. GA processes a population of genotypes that typically encode one phenotype although encoding several phenotypes is also possible. In evolution of ANNs genotypes there are encodings of corresponding networks (phenotypes). The evolutionary procedure involves selecting genotypes (encoded networks) for reproduction based on their fitness, and then by introducing genetically changed offspring (mutation, crossover and other genetic operators) into next population. Repeating the whole procedure over many generations causes the population of encoded networks to gradually evolve into individuals that correspond to high fitness phenotypes (ANNs).

There are a lot of ANN encoding methods [3, 7, 8, 10, 12, 13, 14]. In principle, all the existing encoding methods can be divided into two main classes, i.e., direct encodings and indirect encodings. As for the direct methods, all the information necessary to create ANN (e.g. weights, number of neurons, number of layers) is directly stored in chromosomes. Thus, to encode larger networks, larger chromosomes are necessary, which is the main drawback of the direct methods. As regards the indirect methods, we deal with chromosomes which are recipes how to create a network. Such encodings can be used to create larger neural architectures by means of relatively short chromosomes.

The paper presents a new indirect encoding method called Assembler Encoding (AE). AE originates from the cellular [7] and edge encoding [10]. However, it also has features common with Linear Genetic Programming presented, among other things, in [9, 15]. AE assumes that ANN is represented in the form of a program (Assembler Encoding Program — AEP) whose structure 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 NDM which determines synaptic weights between neurons in ANN designed. In AE, the process of ANN construction consists of three stages. At first, GA is used to produce AEPs. Next, each AEP creates and fills up NDM. Then, the matrix created is transformed into ANN.

AE has many variants. The individual variants differ in the method used to encode AEP in the form of genotype, in the method used to create modular ANNs, in types of operations used in AEPs, and in types of GAs used to process individual parts of AEPs. To define one final version of AE, different experiments were carried out. The results of the experiments mentioned are presented among other things in [19, 20, 21, 22]. The current paper is a continuation of prior works and its main goal is to define the key element of AE, i.e., the evolutionary process responsible for creating AEPs and ANNs. In AE, to produce AEPs and in consequence ANNs, genetic algorithms (GAs) are used. The evolution of AEPs takes place in many different populations. Each population includes a different element of AEP. To create AEP, elements from each population have to be combined together. In order for AEP to be effective representation of ANN, all its elements have to cooperate. To accomplish the cooperation between individual elements of AEP, the evolution in each population has to be appropriately organized, i.e., it is necessary to properly determine different parameters controlling the evolution in each population. The main goal of the paper is to answer the following question: how to organize the evolution in AE, so as to make ANNs created possibly the most effective? To answer the question, the experiments were carried out. During the experiments, the task of each ANN was to control a team of cooperating agents-predators. The common goal of the predators was to capture a fast moving agent-prey behaving by a simple stochastic strategy.

The paper is organized as follows: section 2 is a short presentation of AE; section 3 is a detailed description of the model of evolution used in AE; section 4 illustrates the results of the experiments; section 5 is the summary.

2. Assembler encoding — fundamentals

In AE, ANN is represented in the form of a program called AEP. AEP is composed of two parts, i.e., the part including operations (the code part of AEP) and the part including data (the memory part of AEP). The task of AEP is to create and to fill in NDM with values. To this end, AEP uses the operations. The operations are run in turn. When working, the operations use data located at the end of AEP (Fig. 1). Once the last operation finishes its work the process of creating NDM is completed. NDM is then transformed into ANN.

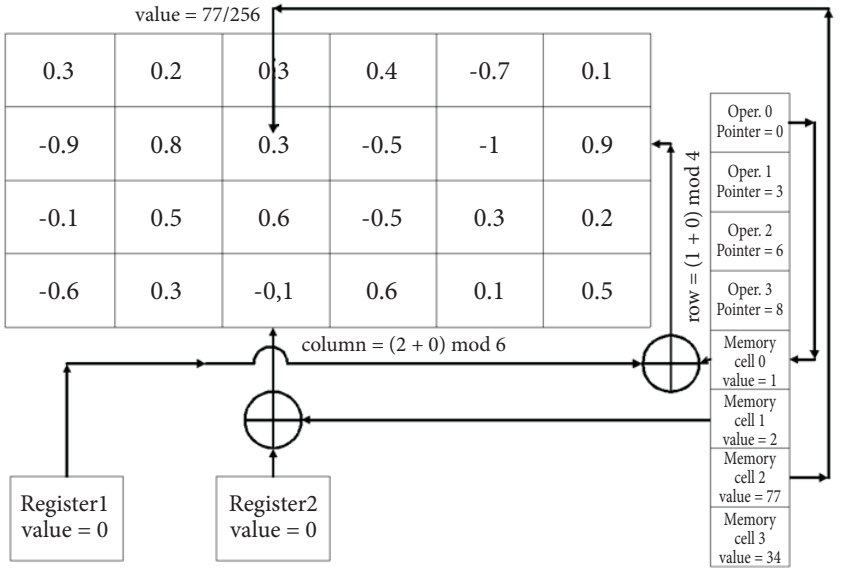

Fig. 1. Diagram of AE (AEP presented on the right includes four operations and four memory cells. Operation 0 changes a single element of NDM. To this end it uses three consecutive memory cells. The first two cells store an address to the element of NDM being updated. To determine the final address of the element mentioned values of registers are also used. The third memory cell used by the Operation 0 stores a new value of the element. The value is scaled before NDM is updated. A pointer to the memory part of AEP, where three cells used by the Operation 0 are located, is included in the Operation itself.)

AEPs can use various operations. The main task of most of the operations is to modify NDM. The modification can involve a single element of NDM or a group of elements. In Fig 2 an example operation used in AE is presented.

CHGC0 operation presented in Fig. 2 modifies values of elements of NDM located in the column indicated by the parameter p_0 and the register R_2. The number of elements being updated is stored in the parameter p_2. The index of the first element being updated is located in the register R_1. To update elements of NDM CHGC0 uses data from AEP. The index to a memory cell including the first data used by CHGC0 is stored in p_1.

In addition to the operations whose task is to modify elements of NDM, AE also uses operations changing the size of NDM. AE assumes that the initial size of NDM is encoded in the chromosome with data (Fig. 3). Then, each AEP has a potential to modify the size of NDM through 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. The addition of new neurons does not destroy connections established in ANN. The task of DELN is to remove a single neuron from ANN. The elimination of a neuron practically takes place through removing corresponding row and column from NDM.

```
CHGC0(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]/Max_value;
      }
}
```

Fig. 2. CHGC0 changing a part of column of NDM (*D*[i] — the i[th] data in AEP, *D.length* — the number of memory cells)

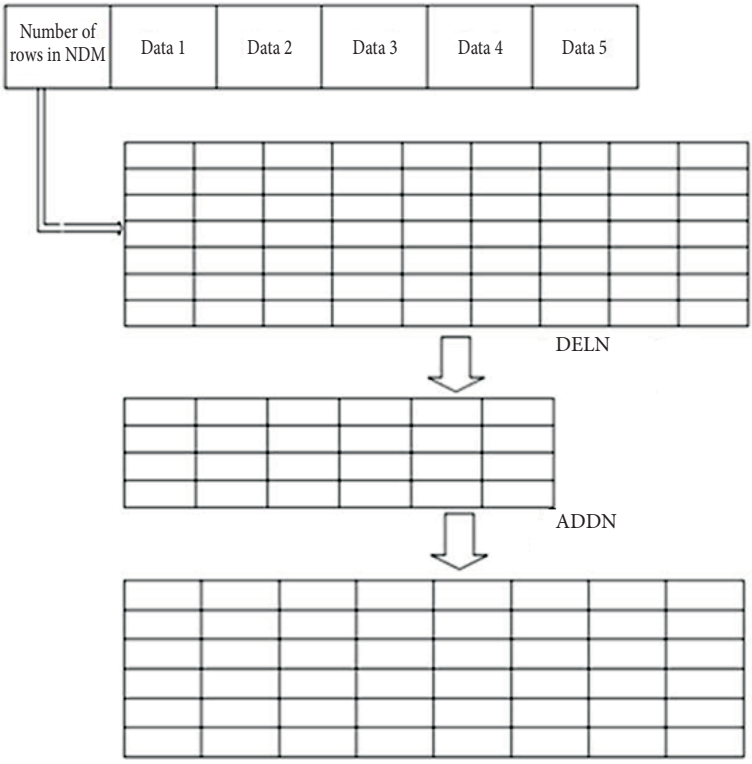

Fig. 3. Using ADDN and DELN by AEP

In addition to the operations presented above, AE also uses a jump operation denoted as JMP. This operation 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 performed.

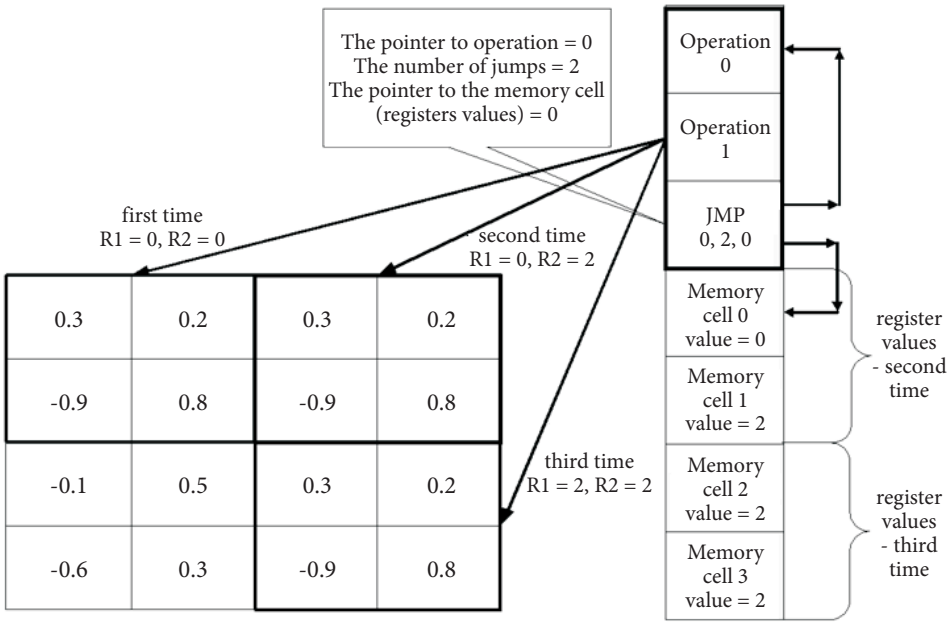

Fig. 4. JMP operation

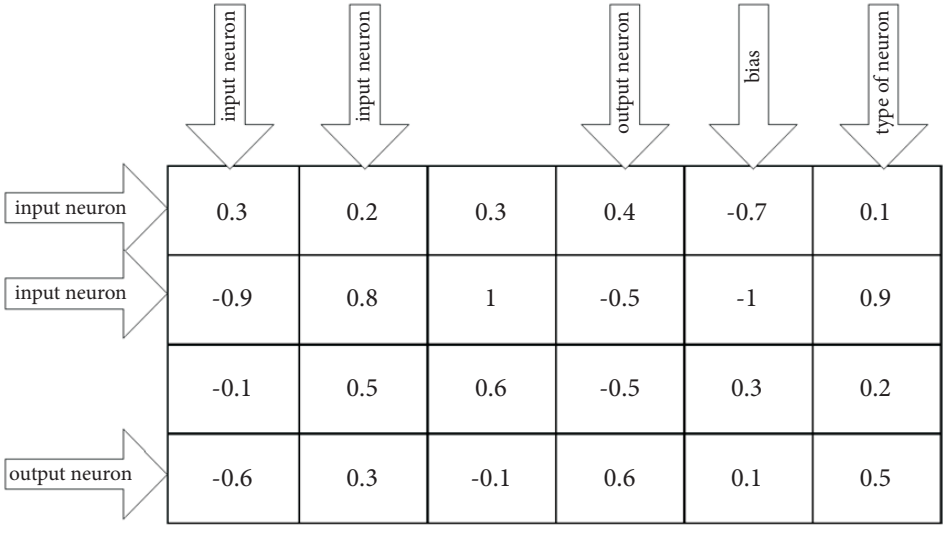

Fig. 5. NDM as Connectivity Matrix

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 the network. When we wish to create the same skeleton of ANN, i.e., ANN without weights of interneuron connections determined, NDM can take the form of the classical connectivity matrix (CM) [12], i.e., a square, binary matrix of the number of rows and columns equal to the number of neurons. The value "1" in the i^{th} column and the j^{th} row of such a matrix means the connection between the i^{th} neuron and the j^{th} neuron. In turn, the value "0" means lack of the connection between these neurons. When the purpose is to create complete ANN with determined values of weights, types of neurons, parameters of neurons, NDM should take the form of a real valued variety of CM, with extra columns or rows containing definitions of individual neurons. The example of such NDM is presented in Fig. 5.

3. The model of the evolution used in AE

In AE, AEPs and in consequence ANNs are created by means of GAs. The evolution of AEPs proceeds according to Cooperative Coevolution GA (CCGA — Fig. 6), i.e., the scheme proposed by Potter and De Jong [17, 18]. The scheme assumes a division of the evolutionarily created solution into parts. Each part

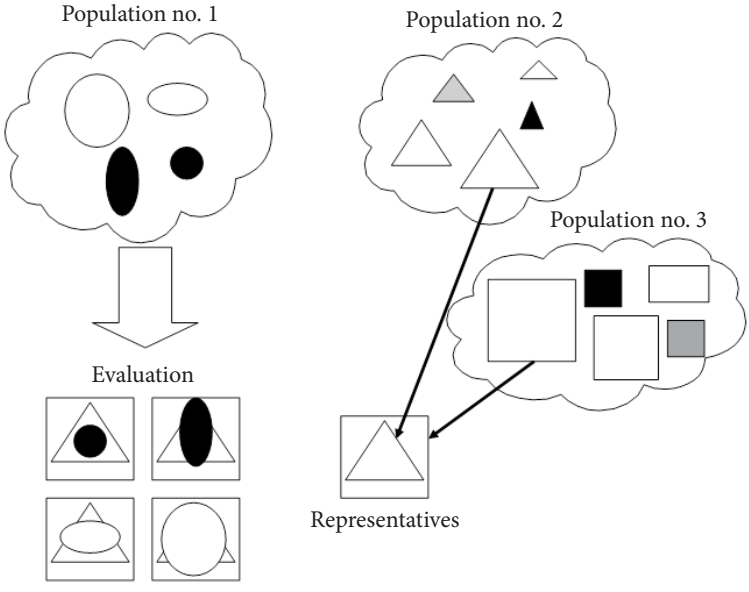

Fig. 6. Evaluation in CCGA

evolves in a separate population. The 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 (Fig. 7). In the case of AEPs, the division is natural. The operations and data make up natural parts of AEPs. Since the evolutionary scheme chosen assumes the evolution of each part in a separate population, AEP consisting of *n* operations and a sequence of data evolves in *n* populations with operations and one population with data. During the evolution, AEPs expand gradually. Initially, all AEPs include one operation and a sequence of data. The operations and the data come from two different populations. When the evolution stagnates, i.e., lack of progress in fitness of generated solutions is observed over some period, a set of the populations containing the operations is enlarged by one population. This procedure extends all AEPs by one operation.

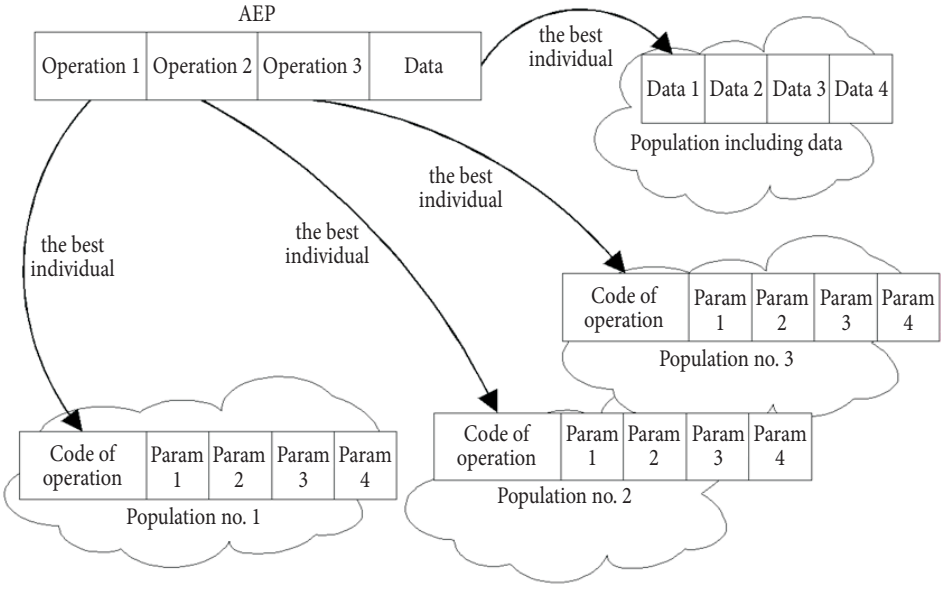

Fig. 7. Evolution of AEPs

In AE, the operations and data are usually encoded in the form of binary strings (in AE, two classes of operations are used, i.e., four-parameter operations encoded as binary strings and three-parameter operations encoded as strings including zeros, ones and the so-called *don't cares* — "#" [2, 6]. In the experiments reported in the paper only four-parameter operations are used). 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 chromosomes-operations have the same length. Chromosome-data can change the length during the evolutionary process.

4. Experiments

As noted above, AEPs evolve in many separate populations. In order to obtain effective AEPs and in consequence ANNs, individuals from different populations have to cooperate. It involves not only operations and data but also the operations themselves. Operations and data from different populations have to match up to form high quality AEPs. The main goal of the experiments, reported in the paper, was to discover general rules of how to control the evolution in AE so as to produce cooperating operations and data, and to make AEPs and ANNs possibly the most effective. In the experiments, different configurations of parameters supervising the evolutionary process in AE were tested. Each configuration was used thirty times, i.e., thirty ANNs were created for each of them. The task of ANNs created during

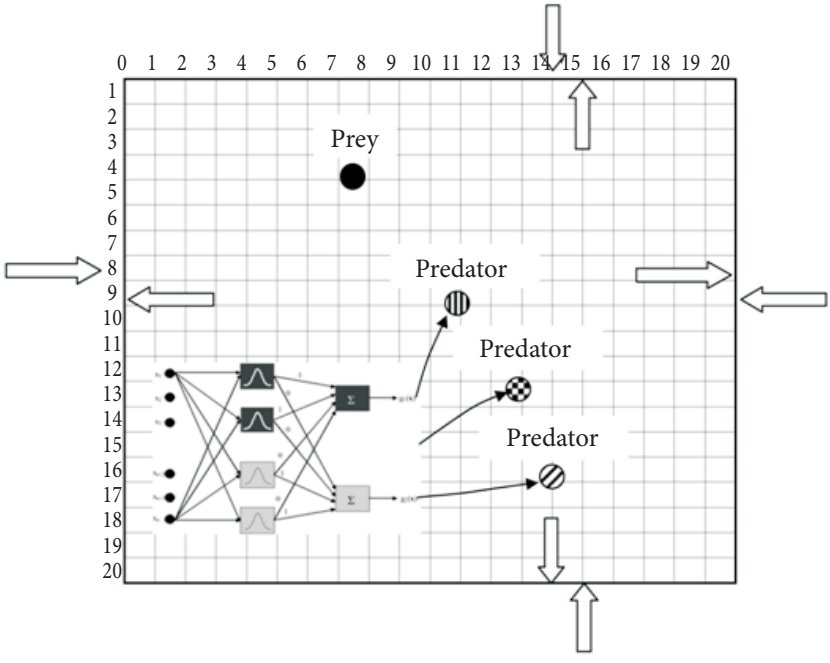

Fig. 8. Artificial world in which task of predators was to capture prey

the tests, was to control a set of cooperating predators whose common goal was to capture a fast moving prey. The experiments were performed in the configuration with one prey and three chasing predators. The prey behaved according to a simple stochastic algorithm whereas the predators were controlled by a single ANN.

4.1. Environment

The predators and the prey lived in common environment. We used 20×20 square without obstacles to represent the environment. In order to ensure infinite space for the predators and the prey and for their struggles, we made the environment open at each side. This means that every attempt to move beyond upper, lower, right or left border of the square caused the object making such an attempt to move to the opposite side of the environment. As a result, a simple strategy of predators consisting in chasing the prey did not work. In such situation, the prey in order to evade predators, could simply escape up, down, right or left.

4.2. Residents of the artificial world

In the experiments, three predators and one prey coexisted in the artificial environment. The predators were controlled by ANN produced by AEP. They could select five actions: to move in North, South, West, East direction or to stand still. The length of the step made by every predator was 1 while the step made by the prey amounted either to 2 or to 1. In order to capture the prey, the predators had to cooperate. Their speed was either two times slower or the same as speed of the escaping prey so they could not simply chase the prey to grasp it. We assumed that the prey was captured if the distance between it and the nearest predator was lower than 2.

In the experiments, we used three types of prey — a simple prey, an advanced prey, and a random advanced prey. The preys, regardless of the type, did not move when no predator was closer to them than 5. Otherwise, the simple prey moved directly away from the nearest predator. Making decision, the advanced prey, unlike its simpler counterpart, always took into consideration the location of all the predators that were situated close to it. Actions performed by the advanced prey always maximized the average distance between the prey and all the predators that were close to it. The only difference between the random advanced prey and its deterministic counterpart is that the random prey, when was running away, sometimes took random decisions. That is, actions performed by the random advanced prey were either random or they maximized the average distance between the prey and all the predators that were close to it. The prey, regardless of the type, when was running away could select four actions: to move in North, South, West or East direction.

4.3. Neural controllers

Feed-forward ANNs used in the experiments contained three types of neurons: radial, sigmoid, and linear neurons. The information about the type of neuron was located in an additional column of NDM. Each matrix included three additional columns. The remaining two columns contained information about bias and value of one parameter of each neuron.

ANNs constructed during the experiments had usually six inputs and three outputs (in some cases ANNs did not require so many inputs to effectively control predators). The number of outputs corresponded to the number of predators. In turn, the number of inputs was twice the number of predators. Each output gave commands to one predator. In turn, each input informed about vertical or horizontal distance between the prey and one of the predators.

4.4. Evaluation process

In order to evaluate ANNs we used thirty different scenarios. The tests proceeded in the following way. At first, each ANN was tested in the scenario no. 1. If the predators controlled by ANN could not capture the prey during some assumed period, the test was stopped and ANN received appropriate evaluation that depended on the distance between the prey and the nearest predator. However, if the predators grasped the prey, they were put to test according to next scenario. During the experiments, we assumed that the predators could perform 100 steps before the scenario was interrupted.

The scenarios used in the experiments differed in initial position of the prey, in the length of step of the prey and in the type of the prey applied (simple, advanced, random 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 simple prey that was two times faster than the predators. In the next step, the speed of the prey was decreased. However, this time the predators had to face the advanced prey which took better decisions than his predecessor. In the next stage, the predators which coped with all earlier scenarios had to capture the advanced, fast prey. In the last ten scenarios, the predators had to capture the random variety of the advanced prey. In all the scenarios, starting positions of all three predators were the same. The predators always started from position (0, 0). Below, described are all thirty scenarios.

— Scenario no 1, 6, 11, 16, 21, 26: starting position of the prey (5, 5) — position 1;
— Scenario no 2, 7, 12, 17, 22, 27: starting position of the prey (15, 5) — position 2;
— Scenario no 3, 8, 13, 18, 23, 28: starting position of the prey (5, 15) — position 3;
— Scenario no 4, 9, 14, 19, 24, 29: starting position of the prey (15, 15) — position 4;

— Scenario no 5, 10, 15, 20, 25, 30: starting position of the prey (10, 10) — position 5;
— Scenario no 1-5: the simple prey, the prey's step = 1;
— Scenario no 6-10: the simple prey, the prey's step = 2;
— Scenario no 11-15: the advanced prey, the prey's step = 1;
— Scenario no 16-20: the advanced prey, the prey's step = 2;
— Scenario no 21-25: the random advanced prey, the prey's step = 1;
— Scenario no 26-30: the random advanced prey, the prey's step = 2.

To evaluate ANNs, the following fitness function was used:

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

$$f_i = \begin{cases} d_{\max} - \min\limits_{p \in P} d\left(p, s_{100}^{i}\right) & \text{prey not captured in } i^{\text{th}} \text{ scenario} \\ f_{captured} + \left(100 - m_i\right) \Big/ a & \text{prey captured in } i^{\text{th}} \text{ scenario} \\ 0 & \text{prey not captured in the previous scenario} \end{cases}$$

where: f_i — the reward received in the i$^{\text{th}}$ scenario;
$d(p,s)$ — is the distance between the prey and the predator p in state of the environment s;
$d_{\max}$ — maximal distance between two points in applied environment;
s_{100}^{i} — the end state in the i$^{\text{th}}$ scenario;
$f_{captured}$ — the reward for grasping the prey in single scenario (in our experiments $f_{captured}$ amounted to 100);
m_i — the number of steps which the predators needed to capture the prey ($m_i < 100$);
a — this value prevents the situation in which partial success would be better than success in all scenarios;
n — the number of scenarios.

The total fitness of each evaluated neuro-controller is a sum of rewards from the scenarios in which the network considered has taken part. The reward for a scenario depends on the chase result. In the case of success, the neuro-controller obtains an extra fitness for grasping a prey and additionally a reward reversely proportional to the number of steps which the predators had to make to capture the prey. In the case of failure, the controlling ANN obtains fitness proportional to the distance between the prey and the nearest predator.

4.5. Parameters of the evolutionary process

In AE, to process different populations, two types of GAs have been used so far, i.e., Eugenic GA [1, 16, 23] and Canonical GA [6]. In the paper, the experiments are reported in which only the later algorithm was used. It was used to process both types of populations occurring in AE, i.e., populations including operations and populations including data. The algorithm applied in relation to the populations with operations used two classical genetic operators, i.e., crossover and mutation. The algorithm processing the populations including data additionally used a cut-splice operator. The task of the cut-splice was to modify the length of chromosomes-data so that AEPs could use data of varied length. In all cases, the tournament selection was used to select parents for next generations of data and operations.

All chromosomes used in the experiments consisted of 7-bit blocks of genes. Every chromosome-operation consisted of 5 blocks of binary genes (one block for a code of operation and the remaining four blocks for parameters of the operation). The list of applied operations is presented at the end of the paper (Appendix 1). In the experiments we assumed that chromosomes-data could maximally contain 30 data, i.e., 30 7-bit blocks of genes. Each use of excessive number of data caused drastic decrease in fitness of the AEP. Throughout the experiments, we assumed maximal number of operations, which could be possessed by each AEP: 12 operations. Initially, every AEP contained one operation and one set of data from two different populations. Consecutive populations with operations were added every 5000 of co-evolutionary cycles if generated AEPs were not able to achieve progress in performance within this period. Populations of operations and data could be also replaced by newly created populations when the contribution of substituted population to AEPs was considerably less than the contribution of the remaining populations. The contribution of the population was measured as average fitness of individuals belonging to that population.

In the experiments, the influence of the following parameters on the effectiveness of the evolutionary process was tested:

— the number of individuals (in each population);
— mutation probability (in each population);
— the number of individuals taking part in the tournament (in each population).

Such parameters as:

— crossover probability (in each population);
— cut-splice probability (exclusively in the population with data);

remained constant during the tests.

During the experiments, the following configurations of the parameters were tested:

Table 1

Configurations of the parameters used in the experiments

	configuration A (tournament, mutation)	configuration B (tournament, mutation)	configuration C (tournament, mutation)	configuration D (tournament, mutation)
configuration a (populations)	configuration aA	configuration aB	configuration aC	configuration aD
configuration b (populations)	configuration bA	configuration bB	configuration bC	configuration bD
configuration c (populations)	configuration cA	configuration cB	configuration cC	configuration cD

where

- configuration a (populations) — all populations of equal size, 60 individuals in each population;
- configuration b (populations) — more individuals in the populations including operations, 80 individuals in all populations with operations and 40 individuals in the population with data;
- configuration c (populations) — more individuals in the populations including data, 40 individuals in all populations with operations and 80 individuals in the population with data;
- configuration A (tournament, mutation) — small mutation and tournament in all populations, size of tournament = 2, mutation = 0.02 in all populations;
- configuration B (tournament, mutation) — large mutation and tournament in all populations, size of tournament = 8, mutation = 0.1 in all populations;
- configuration C (tournament, mutation) — small mutation and tournament in all populations with operations and large mutation and tournament in the population with data, size of tournament = 2, mutation = 0.02 in all populations with operations, size of tournament = 8, mutation = 0.1 in the population with data;
- configuration D (tournament, mutation) — large mutation and tournament in all populations with operations and small mutation and tournament in the population with data, size of tournament = 8, mutation = 0.1 in all populations with operations, size of tournament = 2, mutation = 0.02 in the population with data;

All the configurations presented above were defined arbitrary by the author.

In the experiments we decided that large size of the tournament should be combined with large mutation and vice versa. Numerous tournament causes fast convergence of Canonical GA to a single solution that is very often far from the optimality. To avoid such situation we combine large tournament with large mutation. In the case when selecting individuals for reproduction is based on the tournament of small size, the competition between individuals is not so rough, divergence in a population is preserved and large mutation is not necessary.

Values of the remaining parameters, essential for the results of the tests, are presented below:

— crossover probability (in each population): 0.7;
— cut-splice probability (exclusively in the population with data): 0.1;
— the number of co-evolutionary cycles: 50 000;
— the maximum size of NDMs: 30 rows and 33 columns;
— the probability of the random prey to select a random action: 0.3.

4.6. Experimental results

The results of the tests are presented in Table 2 and Table 3. During the experiments, thirty evolutionary runs were performed for each configuration of the parameters considered in Table 1 (thirty ANNs were generated for each configuration, i.e., 30 ANNs for configuration aA, 30 ANNs for configuration aB and so on).

Table 2

The best configurations tested in the experiments

	conf. c (populations)	conf. D (tournament, mutation)
	(1)	(2)
Average fitness of ANNs	2602	2826.9
% of successes	60.7%	69%

The results of the experiments presented in Table 2 and Table 3 can make up the base to define general rules of organizing the evolutionary process in AE. The analysis of the results of the tests allows us to formulate the following conclusions. First, as for the size of individual populations it seems that the most beneficial solution to ANNs created is when populations with data contain more individuals than populations with operations (configuration c). It is apparent in the column (1) of Table 2. The results incorporated in this column include all the configurations

in which populations with data contained 80 individuals whereas populations with operations contained 40 individuals. More than 60% of ANNs created in these configurations were successful ANNs, i.e., ANNs which resulted in capturing the prey in all thirty scenarios (such ANNs achieved complete success). The second conclusion involves the probability of mutation and conditions of selecting individuals for reproduction. The results of the tests imply that the best solution in this case is slight mutation and sparse tournament in the population with data and the opposite situation, i.e., large mutation and numerous tournament, in the populations with operations [column (2) in Table 2, configuration D]. In the case considered, almost 70% of ANNs were successful. The similar result was achieved in the configurations with slight mutation and sparse tournament in all the populations — 63% of ANNs were successful in this case (configuration A).

Table 3

The best configurations tested in the experiments

	conf. aA	conf. aD	conf. bD	conf. cA	conf. cC	conf. cD
	(1)	(2)	(3)	(4)	(5)	(6)
Average fitness of ANNs	2755.8	2707.4	2793.3	2788.6	2589.6	2984.8
% of successes	63%	66%	56%	73%	52%	85%

With regard to configurations considered in Table 3, the best of them is the configuration cD (the combined configuration of the configurations c and D). Using this configuration enabled us to generate ANNs that were in 85% successful. What is more, successful ANNs were created, in this case, very fast and based on rather simple AEPs. In this instance, AEPs included on average 2.8 operations and 19.9 data.

5. Summary

AE is ANN encoding method in which a network is represented in the form of AEP. The task of AEP is to create and to fill in NDM with values. Once AEP finishes its work ANN is created based on the information contained in NDM. In AE, to create AEPs and in consequence ANNs, GAs are used.

To make AE an effective tool for creating ANNs, it is necessary to appropriately organize all the evolutionary process responsible for generating AEPs, i.e., it is necessary to properly select values of different parameters controlling the evolutionary process mentioned. To determine optimal conditions of the evolution in AE, the experiments in the predator-prey problem were performed. The task of ANNs

created during the experiments was to control a team of autonomous agents called predators. A common goal of the predators was to grasp a fast moving prey behaving according to a simple stochastic strategy. To succeed the common job of the predators, they had to cooperate.

Since AEPs evolve in many populations, the organization of all the evolutionary processes requires determining parameters defining the evolution in each population. In the experiments, an influence of the following parameters on the effectiveness of the evolutionary process was tested: the number of individuals (in each population), mutation probability (in each population), and the number of individuals taking part in the tournament (in each population). Such parameters as: crossover probability (in each population) and cut-splice probability (exclusively in the population with data) remained constant during the tests. The experiments were conducted for different configurations of the parameters selected. Each configuration differed from the others in values of the parameters selected. The experiments showed that the most advantageous solution to ANNs is when populations with data contain more individuals than populations with operations. Moreover, it turned out that the most effective ANNs arise when in the population with data we deal with slight mutation and sparse tournament whereas in the populations with operations we deal with the opposite situation, i.e., with large mutation and numerous tournament.

Received July 10 2008, revised December 2008.

Appendix 1 — List of operations used in experiments

CHG — Update of element. Both new value and address of element are located in parameters of operation.
CHGC0 — Update of certain number of elements in column. Index of column, index of first element in column that will be changed, number of changed elements and a pointer to data, where new values of elements are memorized, are located in parameters of operation.
CHGC1 — Update of certain number of elements in column. Index of column, index of first element in column that will be changed, number of changed elements and new value for column's elements, the same for all elements, are located in parameters of operation.
CHGC2 — Update of certain number of elements in column. New value of every element is sum of operation's parameter and current value of this element. The second parameter of operation is index of column. The third and the fourth parameter of operation determine respectively the number of changed elements and index of the first element in column that will be changed.
CHGC3 — The part of elements from one column are transformed to another column. Both columns are indicated by parameters of operation. The number of transferred ele-

ments and index of the first element in column that will be transferred are also included in parameters of operation.

CHGC4 — An update of certain number of elements in column. New value of every element is a sum of current value of this element and respective value from memory of a program. An index of the column, an index of the first element in the column that will be changed, the number of changed elements and a pointer to data, where ingredients of individual sums are memorized, are located in parameters of operation.

CHGR0 — like **CHGC0** but an update refers to row of matrix.

CHGR1 — like **CHGC1**.

CHGR2 — like **CHGC2**.

CHGR3 — like **CHGC3**.

CHGR4 — like **CHGC4.**

CHGM0 — Change of block of elements. Elements are updated in columns, in turn, one after another, starting from element pointed by parameters of operation. The number of changed elements and place in the memory where new values for elements are located are determined by parameters of operation.

CHGM1 — like **CHGM0** but new value of every element is a sum of its current value and parameter of operation.

CHGM2 — like **CHGM0** but new value of each element is a sum of its current value and value from memory part of a program. The number of changed elements and place in the memory where arguments of individual sums are located are determined by parameters of operation.

JMP — Jump operation. The number of jumps, a pointer to next operation and new values of registers are located in parameters of jump operation.

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), San Francisco, CA, Morgan Kaufmann, 2002.

[2] M. V. Butz, *Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction*, University of Illinois, IlliGAL Report no. 2004034, 2004.

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

[4] D. Curran, 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, J. Urzelai, *Evolutionary robots with online self-organization and behavioural fitness*, Neural Networks, vol. 13, 2000, 431-443.

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

[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, vol. 4, 1990, 461-476.

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

[10] S. Luke, 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.

[11] 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.

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

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

[14] S. Nolfi, D. Parisi, *Growing neural networks*, in C. G. Langton, ed., Artificial Life III, Addison-Wesley, 1992.

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

[16] D. Polani, R. Miikkulainen, *Eugenic Neuro-Evolution for Reinforcement Learning*, in Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000), Las Vegas, NV, 2000.

[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, K. A. De Jong, *Evolving neural networks with collaborative species*, in T. I. Oren, L. G. Birta, ed., Proceedings of the 1995 Summer Computer Simulation Conference, 1995, 340-345.

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

[20] T. Praczyk, Procedure application in Assembler Encoding, Archives of Control Science, vol. 17, 53, no. 1, 2007, 71-91.

[21] T. Praczyk, *Using genetic algorithms and assembler encoding to generate neural networks*, Computing and Informatics, 2008 (in press).

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

[23] J. W. Prior, *Eugenic Evolution for Combinatorial Optimization*, Master's thesis, The University of Texas at Austin, TR AI98-268, 1998.

T. PRACZYK

Organizacja procesu ewolucyjnego w kodowaniu asemblerowym

Streszczenie. Kodowanie asemblerowe jest metodą wykorzystującą metody ewolucyjne do tworzenia sieci neuronowych. W kodowaniu asemblerowym sieci neuronowe ewoluują w wielu oddzielnych populacjach. Stworzenie pojedynczej sieci neuronowej wymaga połączenia elementów pochodzących

z różnych populacji. Aby sieci neuronowe tworzone w ten sposób były wysokiej jakości konieczne jest odpowiednie sterowanie ewolucją w każdej populacji. Artykuł prezentuje wyniki badań, których głównym celem było określenie zasad prowadzenia ewolucji w Kodowaniu Asemblerowym.
Słowa kluczowe: ewolucyjne sieci neuronowe
Symbole UKD: 007