Selecting Operations for Assembler Encoding

Tomasz Praczyk
Institute of Navigation and Hydrography, Naval University of Gdynia
ul. ´Smidowicza 69, 81–103, Gdynia, Poland
t.praczyk@amw.gdynia.pl

Abstract

Assembler Encoding is a neuro-evolutionary method in which a neural network is represented in the form of a simple program called Assembler Encoding Program. The task of the program is to create the so-called Network Definition Matrix which maintains all the information necessary to construct the network. To generate Assembler Encoding Programs and the subsequent neural networks evolutionary techniques are used.

The performance of Assembler Encoding strongly depends on operations used in Assembler Encoding Programs. To select the most effective operations, experiments in the optimization and the predator-prey problem were carried out. In the experiments, Assembler Encoding Programs equipped with different types of operations were tested. The results of the tests are presented at the end of the paper.

Keywords: artificial neural networks, assembler encoding program

1. Introduction

Using evolutionary techniques to generate Artificial Neural Networks (ANNs) is usually connected with encoding the latter. There are many ANN encoding methods, e.g. connectivity matrix proposed by Miller, Todd and Hedge [9], Kitano's matrix rewriting encoding scheme [7], Gruau's cellular encoding [5], edge encoding proposed by Luke and Spector [8], Symbiotic Adaptive NeuroEvolution devised by Moriarty and Miikkulainen [10], the schemes proposed by Nolfi and Parisi [11] and Cangelosi, Parisi and Nolfi [2] or Cooperative Synapse Neuroevolution proposed by Gomez et al. [4]. Assembler Encoding (AE) [15-18] is one of such methods. It originates from the cellular and edge encoding but it also has features common with Linear Genetic Programming presented, among other things, in [12]. AE represents ANN in the form of Assembler Encoding Program (AEP) whose structure is similar to the structure of a simple assembler program. The task of AEP is to create Network Definition Matrix (NDM) containing all the information necessary to produce ANN. The process of ANN construction consists of three stages. First, Genetic Algorithm (GA) is used to produce AEPs. Next, each AEP creates and fills up NDM. Finally, NDM is transformed into ANN (Fig. 1).

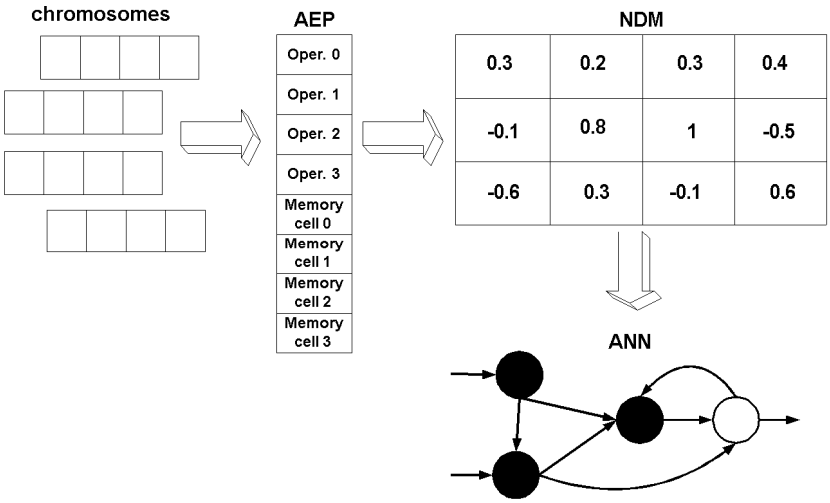


Figure 1. Diagram of AE

To date, AE has been tested in three different testing problems, i.e. in an optimization problem, in a predator-prey problem and in an inverted pendulum problem. In addition to simple testing problems, the method was also applied in a real problem in which a task was to generate a neuro-controller for a team of autonomous underwater vehicles (results of the experiments with the vehicles will be presented in another paper). In all the tests, the method demonstrated fairly good effectiveness. Noteworthy is the fact that it successfully competed with different direct neuro-evolutionary and reinforcement learning methods in problems which rather prefer the latter of them [18]. However, all the tests also showed that effectiveness of AE strongly depends on operations used in AEPs. There were situations when it was not able to produce any ANN in satisfactory time. But, it was enough to replace one or two operations with other ones or to add additional operations to a set of operations available to AEPs to improve the situation.

In AE, many different variants of operations can be used. The operations can be simple and represented in the form of short chromosomes. Another solution is to use operations with complex behavior and with a longer genotypic representation. AEPs can be built based on many different operations or based on a few the most effective ones. To form AEPs four parameter operations or three-parameter ones (both types of operations are described further) can be used. To test all the possibilities specified above, experiments in the optimization and in the predator-prey problem were carried out. The main goal of the experiments was to select the most effective operations for AE. In the experiments, different types of operations with different behavior and with different form of genotypic representation were tested.

The paper is organized as follows: section 2 is a short introduction to AE; section 3 is the report on the experiments, and section 4 is the summary.

2. Fundamentals of AE

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

NDM determines 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. Elements of NDM that are unimportant from the point of view of the process of ANN construction, for example because of the assumed feed-forward topology of ANN, are neglected during ANN building. Apart from the basic part, NDM also possesses additional columns that describe parameters of neurons, e.g. type of neuron (sigmoid, radial), bias etc.

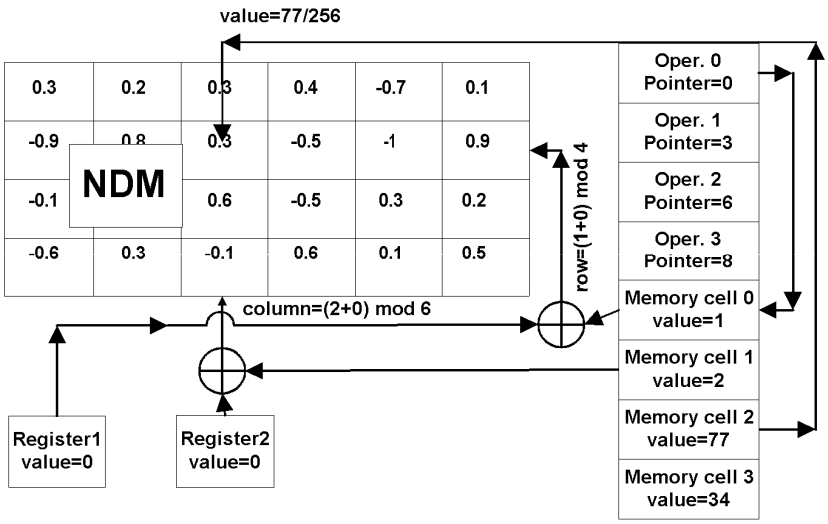


Figure 2. Example AEP and its NDM (AEP presented on the right includes four operations and four memory cells. Operation 0 updates a single entry in NDM. To this end, it uses three consecutive memory cells. The first two cells store an address to the element in NDM being updated. To determine the final address of the element mentioned, values of registers are also used. The third memory cell used by Operation 0 stores a new value for the element. The value is scaled before NDM is updated. A pointer to the memory part of AEP where three cells used by Operation 0 are located is included in Operation itself.)

2.1. Operations

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 AE, we deal with two types of operations, i.e. with the four-parameter and three parameter operations. The four-parameter operations can have maximum four parameters. Their characteristic is that they always change elements of NDM that constitute some tight integrity. Each operation works in a similar way, i.e. the first element to change is chosen at the beginning, then the neighbor of the chosen element is altered, then the neighbor of the chosen element neighbor is updated and so on. Fig. 3 and Fig. 4 present two example four-parameter operations. The task of CHG presented in Fig. 3 is to change a single element in NDM. The new value for the element, stored in parameter p_0, is scaled to <-1,1>. An address of the element depends on both parameters p_1, p_2 and registers R_1, R_2. A role of the registers is detailed in the further part of the paper.

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

Figure 3. CHG operation

CHGC0 presented in Fig. 4 modifies elements of NDM located in the column indicated by the parameter p_0 and the register R_2. The number of elements updated by the operation is stored in the parameter p_2. An index of the first updated element is located in the register R_1. To update elements of NDM, CHGC0 uses data from AEP. An index to a memory cell including the first data used by CHGC0 is stored in p_1.

```
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;
     }
}
```

Figure 4. CHGC0 operation changing part of column in NDM

The characteristic of operations presented above is that they always change a block of neighboring elements in NDM. It is impossible for the altered elements to be located in different remote fragments of the matrix. The three-parameter operations which always have three parameters, can perform their task in the way the four-parameter ones are unable to perform, i.e. they can modify separate areas of NDM (a genotypic representation is another difference between the three and four-parameter operations). `CHG_MEMORY` presented in Fig. 5 is the example of three-parameter operation. It modifies elements of NDM indicated in `list1` and `list2`. The lists mentioned include numbers of columns and rows of NDM which, in turn, indicate elements of the matrix updated by the operation. All possible combinations of columns and rows considered in both lists determine a set of elements that are modified by the operation. The parameter p_0 indicates a place in the memory part of AEP where new values for updated elements can be found.

```
CHG_MEMORY(p0,list1,list2)
{
for(i=0;i<list1.length;i++)
     for(j=0;j<list2.length;j++)
          {
          row=(list1[i]+R1)mod NDM.width;
          column=(list2[j]+R2)mod NDM.height;
          NDM[row,column]=
          D[(abs(p0)+i*list2.length+j)mod D.length]/Max_value;
          }
     }
```

Figure 5. CHG_MEMORY operation changing elements of NDM indicated in list1 and list2

In addition to the operations whose task is to modify a content of NDM, AE also uses a jump operation denoted as `JMP`. The jump makes it possible to repeatedly use the same code of AEP in different places of NDM. It is possible thanks to changing values of the registers once the jump is run. An example use of the jump is demonstrated in Fig 6. The program presented in the figure proceeds as follows. First, both registers are initiated to 0. Then, the first two operations are carried out, the result of which is visible in the top left corner of NDM. In the next step, the jump, denoted in the figure as `JMP(0,2,0,*)`, is run. It first updates values of the registers and then control goes back to the first operation of AEP. R_1 is set to 0 (Memory cell 0) whereas R_2 to 2 (Memory cell 1). At this point, the two operations preceding the jump are carried out once again. This time, however, both operations update a different fragment of NDM. Since the jump is run twice, each time with different values of the registers, the first two operations of AEP are executed in three different areas of NDM.

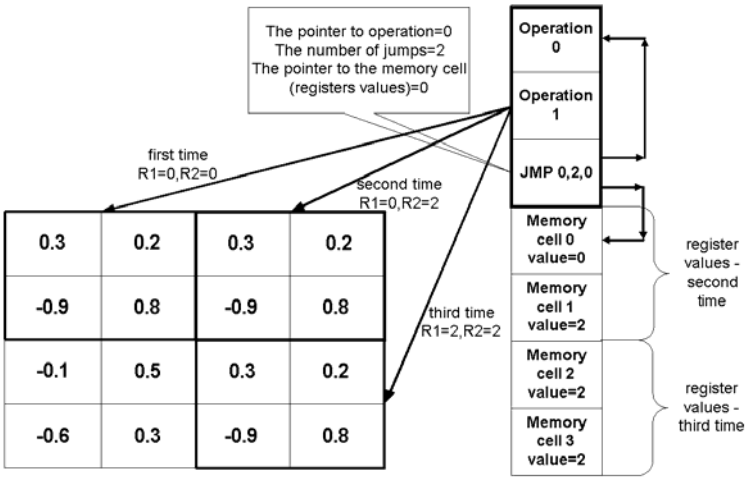


Figure 6. JMP operation

An additional group of operations, used in AE, are operations whose task is to change the size of NDM. In AE, the initial size of NDM is encoded in the chromosome with data (Fig. 7). Then, each AEP has a potential to modify the size of NDM through using 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 the neuron practically takes place through removing a corresponding row and column from NDM.

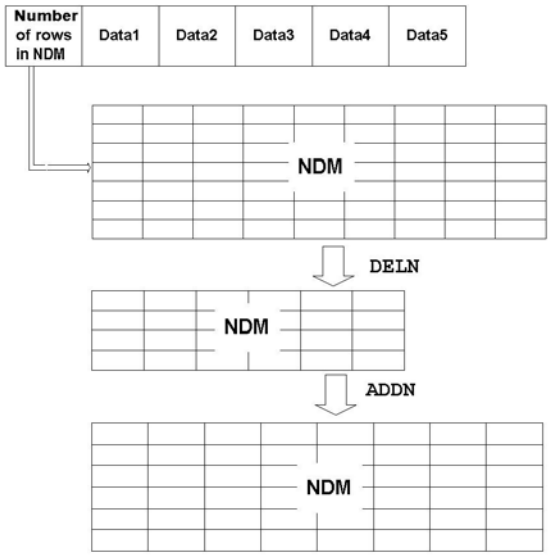


Figure 7. Using ADDN and DELN by AEP

2.2. Evolution in AE

In AE, AEPs, and in consequence ANNs are created by means of GAs. The evolution of AEPs proceeds according the scheme which is an adaptation of the idea of evolving co-adapted subcomponents proposed by Potter and De Jong [13,14]. To create AEP, the scheme mentioned combines operations and data from various populations. Each population including chromosomes-operations (Each chromosome-operation encodes the type of operation, e.g. `CHGC0`, and parameters of operation. Implementations of operations do not evolve) 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 operation and data come from two different populations. Further populations including operations are successively added if AEPs cannot accomplish progress in performance over the assumed number of co-evolutionary cycles (we use term "co-evolutionary cycle" to differ it from the evolutionary generation that takes place inside a single population with operations and data).

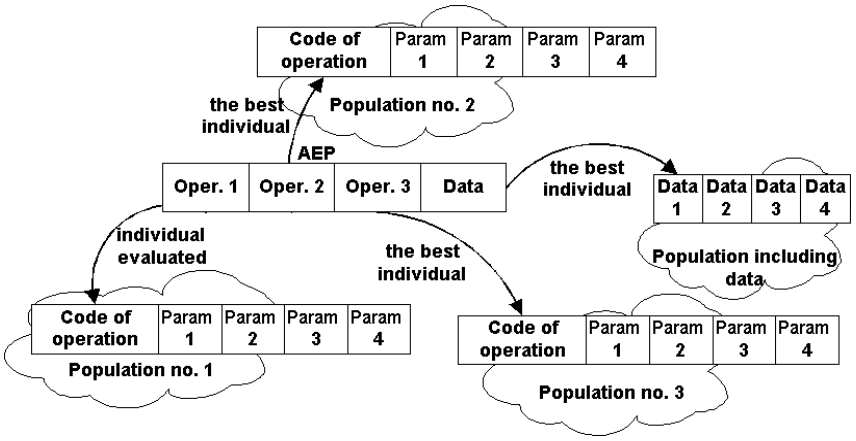


Figure 8. AEP encoding scheme

In AE, the operations can be encoded in two ways. The four-parameter operations are encoded in the form of binary strings. For example, CHG presented in Fig. 3 is encoded in the form of binary string including five blocks of genes. The first block determines a code of the operation (e.g. binary 00000 indicates that we deal with CHG) while the remaining blocks contain a binary representation of four parameters of the operation.

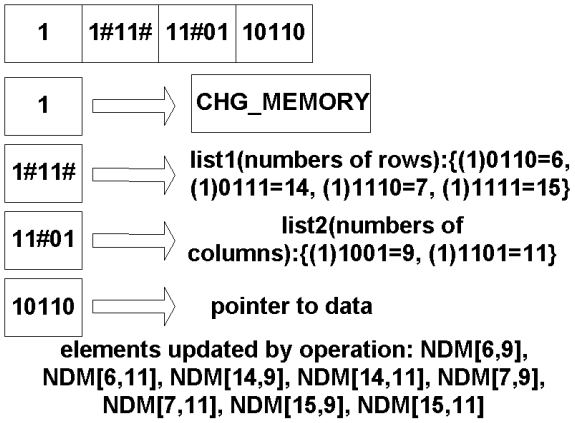


Figure 9. Encoding CHG_MEMORY

The three-parameter operations are represented in a somewhat different way. Their encoded form resembles classifiers from Learning Classifier Systems [1,3,6]. Similarity between classifiers and the three-parameter operations results from the use of the so-called *don't care* symbol "#" in both cases. Each encoded operation consists of four blocks of genes. The first single-bit block determines one of two possible variants of the operation (see Appendix 1). The second and third block indicate location of changes performed by the operation (*don't care* symbol is used for this purpose). The last block specifies the value of an integer parameter of the operation. The example use of *don't care* symbol to locate changes in NDM is illustrated in Fig 9.

3. Experiments

The main goal of the experiments was to give the answer to the following questions:

1. Should AEPs be created based on large set of different operations whether a better solution is to create them based on a small set including exclusively the most effective operations?
2. Should AEPs use simple operations with a short genotypic representation whether a better solution is to use more complex and flexible operations but with a longer genotypic representation?
3. Should AEPs use the three-parameter operations whether a better solution is to use the four-parameter ones?

The tests presented in this section were carried out both in the optimization problem and in the so-called predator-prey problem.

3.1. Experiments in the optimization problem

In AE, ANNs are generated by means of AEPs whose task is to create NDMs. Such course of action makes it possible to take advantage of AE in the optimization problem, in which a solution can be presented in the form of a matrix. Even though searching for optimal matrices is not the target use of AE, experiments in this field should provide useful information about differences in performance between AEPs using various types of operations. This knowledge can be employed in further experiments to reduce the number of tested variants of AE only to those which have produced satisfactory results in the optimization problem.

3.1.1 Objective functions

Three different objective functions were used in the experiments. All the functions were constructed so as to test how AEPs are able to repeatedly use the information included in themselves. To create optimal matrices for the functions, AEPs could use two methods. The first method uses the so-called "brute force", i.e. creates matrices by means of a large number of operations. The other method intelligently uses jumps. All the objective functions are presented below.

$$f_k(\mathrm{NDM}) = -\sum_{i}^{10}\sum_{j}^{10} |A_k[i,j]|, \quad k = 1,2,3 \tag{1}$$

$$A_1[i,j] = \begin{cases} 1 - \mathrm{NDM}[i,j], i,j = 1..5 \\ 1 + \mathrm{NDM}[i,j], i,j = 6..10 \\ 0.4 - \mathrm{NDM}[i,j], i = 1..5, j = 6..10 \\ 0.4 + \mathrm{NDM}[i,j], i = 6..10, j = 1..5 \end{cases} \tag{2}$$

$$A_2[i,j] = \begin{cases} 0.4 - \mathrm{NDM}[i,j], ((i+j) \bmod 2) = 0 \\ 0.4 + \mathrm{NDM}[i,j], \text{otherwise} \end{cases} \tag{3}$$

$$A_3[i,j] = \begin{cases} 0.2 - \mathrm{NDM}[i,j], i = j \\ 0.2 + \mathrm{NDM}[i,j], i = 10 - j + 1 \\ 0.4 - \mathrm{NDM}[i,j], i = 6 \wedge i \neq j \wedge i \neq 10 - j + 1 \\ 0.4 + \mathrm{NDM}[i,j], j = 6 \wedge i \neq j \wedge i \neq 10 - j + 1 \\ \mathrm{NDM}[i,j], \text{otherwise} \end{cases} \tag{4}$$

In all the cases presented above, the task of AEPs was to find 10x10 matrix maximizing a selected objective function. The global maximum for all the test functions was zero. Optimal matrices for each test function are presented in Fig. 10.

3.1.2 Experimental results

In the experiments, the following sets of operations were tested (a description of all the operations specified below is presented in Appendix 1):

- **Set 1** (all four-parameter operations used during the research reported in [15,17]): `CHG, CHGC0, CHGC1, CHGC2, CHGC3, CHGC4, CHGR0, CHGR1, CHGR2, CHGR3, CHGR4, CHGM0, CHGM1, CHGM2, JMP;`
- **Set 2** (the most effective four-parameter operations used in the previous research): `CHGC0, CHGC3, CHGR0, CHGR3, CHGM0, CHGM2, JMP;`
- **Set 3**: simpler variants of operations included in Set 1, the simpler operations, unlike their more complex counterparts, always changed either the whole column or the whole row or the whole matrix, operations from this set had maximally three parameters;
- **Set 4** (the three-parameter operations and jumps): `CHG_VALUE, CHG_MEMORY, JMP.`

$$
\text{a)}\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4\\
1 & 1 & 1 & 1 & 1 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4\\
1 & 1 & 1 & 1 & 1 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4\\
1 & 1 & 1 & 1 & 1 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4\\
1 & 1 & 1 & 1 & 1 & 0.4 & 0.4 & 0.4 & 0.4 & 0.4\\
-0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -1 & -1 & -1 & -1 & -1\\
-0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -1 & -1 & -1 & -1 & -1\\
-0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -1 & -1 & -1 & -1 & -1\\
-0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -1 & -1 & -1 & -1 & -1\\
-0.4 & -0.4 & -0.4 & -0.4 & -0.4 & -1 & -1 & -1 & -1 & -1
\end{bmatrix}
\text{b)}\begin{bmatrix}
0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4\\
-0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4\\
0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4\\
-0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4\\
0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4\\
-0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4\\
0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4\\
-0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4\\
0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4\\
-0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4 & -0.4 & 0.4
\end{bmatrix}
$$

$$
\text{c)}\begin{bmatrix}
0.2 & 0 & 0 & 0 & 0 & -0.4 & 0 & 0 & 0 & -0.2\\
0 & 0.2 & 0 & 0 & 0 & -0.4 & 0 & 0 & -0.2 & 0\\
0 & 0 & 0.2 & 0 & 0 & -0.4 & 0 & -0.2 & 0 & 0\\
0 & 0 & 0 & 0.2 & 0 & -0.4 & -0.2 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0.2 & -0.2 & 0 & 0 & 0 & 0\\
0.4 & 0.4 & 0.4 & 0.4 & -0.2 & 0.2 & 0.4 & 0.4 & 0.4 & 0.4\\
0 & 0 & 0 & -0.2 & 0 & -0.4 & 0.2 & 0 & 0 & 0\\
0 & 0 & -0.2 & 0 & 0 & -0.4 & 0 & 0.2 & 0 & 0\\
0 & -0.2 & 0 & 0 & 0 & -0.4 & 0 & 0 & 0.2 & 0\\
-0.2 & 0 & 0 & 0 & 0 & -0.4 & 0 & 0 & 0 & 0.2
\end{bmatrix}
$$

Figure 10 Optimal matrices for functions f_1 (a), f_2 (b), and f_3 (c)

```
CHGC0(p0,p1,*)
{
column=(abs(p0)+R2)mod NDM.height;
for(i=0;i<= NDM.width;i++)
     NDM[i,column]=D[(abs(p1)+i)mod D.length]/Max_value;
}
```

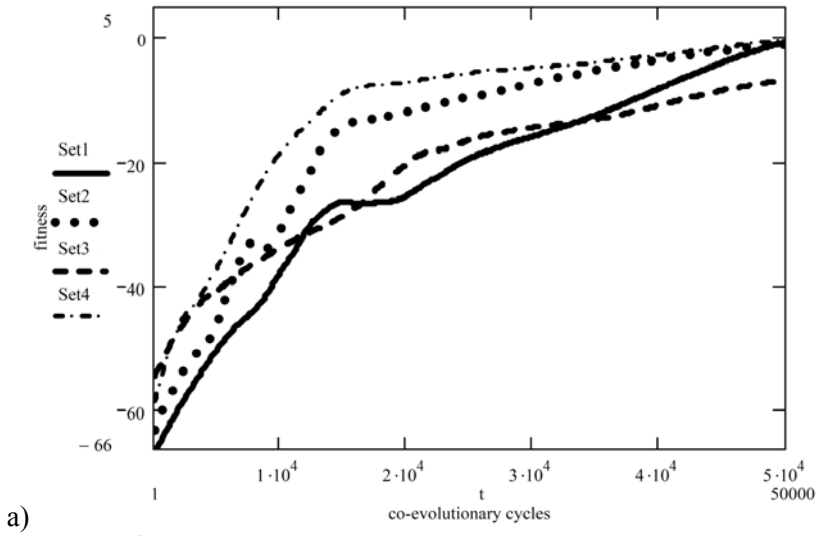
Figure 11 Simpler variant of CHGC0 presented in Fig. 4

Table 1 shows the results of the tests. Each cell in the table includes the following: the average result obtained for 30 evolutionary runs (top value), the result of the best AEP (middle value), and the sum of the number of operations and data in the best AEP (number of operations + number of data; bottom value).

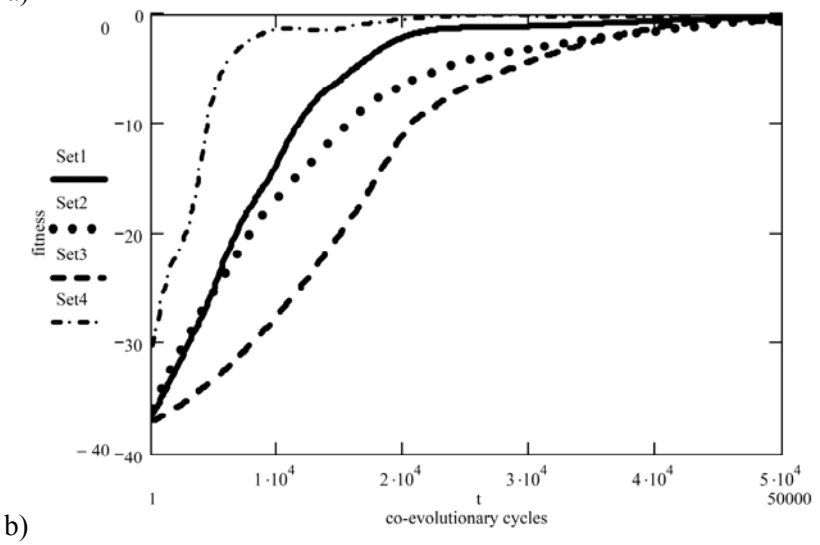
Table 1. Using AEPs with different operations in the optimization problem

	Set 1	Set 2	Set 3	Set 4
F_1	-0.2 0 5+8	-0.42 0 6+8	-7.86 0 7+12	-0.04 0 6+7
F_2	0 0 7+3	0 0 6+5	-0.19 0 5+3	0 0 2+2
F_3	-0.66 -0.6 12+6	-0.94 -0.73 12+7	-2.04 -1.9 12+7	-0.6 0 12+6

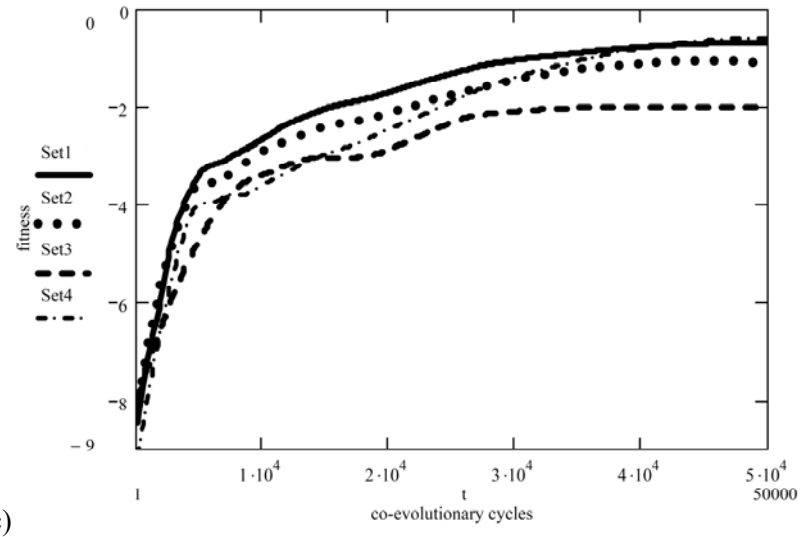
The tests revealed that the best matrices are produced by means of the three-parameter operations (Set 4). In all the cases, AEPs with the three-parameter operations were able to generate the optimal matrices. AEPs including operations from Set 1 and Set 2 achieved somewhat worse results. Using operations from Set 3 turned out to be the worst solution of all. AEPs including operations from this set were considerably worse than the remaining AEPs.



a)



b)



c)

Figure 12 Using AEPs with different operations to solve problem f_1 (a), f_2 (b), and f_3 (c) (all curves are generated from average of 30 evolutionary runs)

3.2. Experiments with ANNs

In the experiments, the task of AE was to produce a single ANN controlling a set of cooperating predators whose common goal was to capture a fast moving prey behaving by a simple, deterministic strategy. The main goal of the experiments reported in this section was to test how AEPs with different operations cope with creating simple ANNs. To find out true capabilities of AE to create ANNs of more advanced architecture, further experiments are required.

In this stage of the experiments, two types of AEPs were used, i.e. AEPs-four and AEPs-three. AEPs-four are programs including operations from Set 1 (the second position in the previous experiments) while AEPs-three are programs composed of three-parameter operations from Set 4 (the first position in the previous experiments).

3.2.1. Environment

The predators and the prey used in the experiments lived in a common environment. 20x20 square without obstacles but with two barriers located on the left and on the right side of the square was used to represent the environment. Both barriers caused the predators as well as the prey to move right or left only to the point at which they reached one of the barriers. Moving further in the barrier direction caused no effect. In order to ensure infinite space for the predators and the prey and for their struggles, the environment was open at the bottom and at the top. This means that every attempt to move beyond upper or lower border of the square caused the object making such attempt to move to the opposite side of the environment. As a result, the simple strategy of the predators consisting in chasing the prey did not work. In such situation, the prey in order to evade predators, could simply escape up or down.

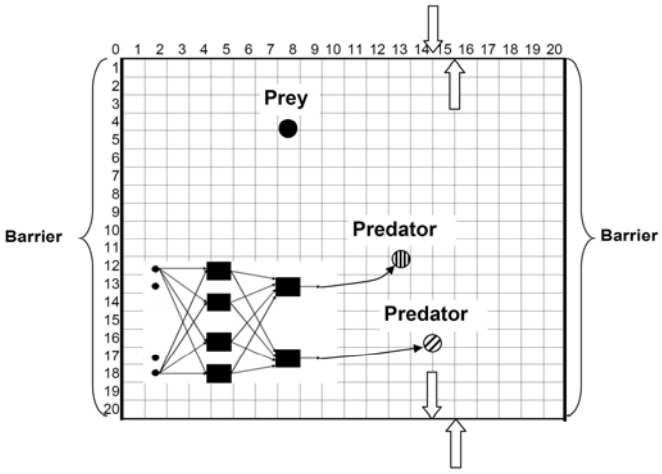


Figure 13. Artificial world in which task of predators was to capture prey

3.2.2. Residents of the artificial world

In the experiments, two predators and one prey coexisted in the artificial environment. The predators were controlled by ANN produced by AEP (because in the experiments a single ANN controlled the set of predators, the problem solved by ANNs can also be viewed as the problem of capturing the prey by one predator with several "arms"). They could select five actions: to move in North, South, West, East direction or to stand still. The length of step of every predator was 1 unlike the step of the prey that amounted to 2. In order to capture the prey the predators had to cooperate. Their speed was twice lower than speed of the escaping prey so they could not simply chase it to grasp it. The prey was captured if the distance between it and the nearest predator was lower than 2.

In the experiments, the predators could see the whole environment. The predators based the decision which action to select on the prey's relative location with reference to each of them. In order to perform the task, ANN controlling the predators had to possess four inputs and two outputs. The outputs provided decisions to the predators whereas inputs informed them about prey's location in relation to each of them.

The prey was controlled by a simple algorithm which forced him to move directly away from the nearest predator but solely in the situation when distance between it and the nearest predator was lower or equal 5. In the remaining cases, i.e. when neither predator was closer to the prey than the assumed distance, the prey did not move. In the situation when selected prey's action would cause hitting the barrier, another move was chosen. The alternative move prevented from hitting the wall, and at the same time, it maximally increased the distance between the prey and the nearest predator. The prey when was running away could select four actions: to move in North, South, West or East direction. The strategy of the prey is presented below.

$$\pi_{\text{simple}}(s) = \begin{cases} \text{StandStill if } \underset{p \in P}{\forall} d(p,s) > 5 \\ \arg\max\limits_{a \in A} D\left(s, a, \arg\min\limits_{p \in P} d(p,s)\right) \text{ otherwise} \end{cases} \quad (5)$$

where

P – set of predators

A – set of actions of prey (A={StandStill, North, South, West, East})

$d(p,s)$ – distance between prey and predator p in state of environment s

$D(s,a,p)$ – distance between prey and predator p in state of environment which is direct consequence of action a performed by prey in state s

3.2.3. Neural controllers

NDMs generated by AEPs usually represented recurrent ANNs. However, in the experiments, the decision was made to use exclusively controllers with feed-forward architecture. In order to obtain such ANNs, elements of NDMs localized in their upper parts were only used. The remaining elements were neglected while constructing ANNs.
ANNs contained three types of neurons, i.e. radial, sigmoid and linear neurons. The information about the type of neuron was located in the additional column of NDM. Each matrix included a total of three additional columns. The remaining two columns contained the information about bias and value of a parameter of a neuron.

3.2.4. Evaluation process

To evaluate ANNs, eight different scenarios were built. They differed in an initial location of the predators and the prey in the environment:

- Scenario no 1: prey(10,10), predator1(0,0), predator2(20,20);
- Scenario no 2: prey(10,10), predator1(20,20), predator2(0,0);
- Scenario no 3: prey(10,10), predator1(0,10), predator2(20,10).
- Scenario no 4: prey(10,10), predator1(10,0), predator2(10,20);
- Scenario no 5: prey(10,10), predator1(0,0), predator2(20,0).
- Scenario no 6: prey(0,20), predator1(10,0), predator2(20,10);
- Scenario no 7: prey(0,10), predator1(10,1), predator2(10,19);
- Scenario no 8: prey(0,20), predator1(0,10), predator2(10,20).

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 the 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 the test in the next scenario. In the experiments, the predators could perform 100 steps before the scenario was interrupted. To evaluate ANNs, the following fitness function was used:

$$f(ANN) = \sum_{i=1}^{n} f_i \tag{6}$$

$$f_i = \begin{cases} d_{\max} - \min\limits_{p \in P} d\left(p, s_{100}^{i}\right) & \text{prey not captured in } i^{\text{th}} \text{ scenario} \\ f_{captured} + \left(100 - m_i\right) / a & \text{prey captured in } i^{\text{th}} \text{ scenario} \\ 0 & \text{prey not captured in the previous scenario} \end{cases} \tag{7}$$

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

3.2.5. Experimental results

In order to test different operations, 30 evolutionary runs were performed. As it turned out all the runs were successful, i.e. all they produced successful ANNs. The successful ANNs are ANNs which resulted in capturing the prey in all tested scenarios. Detailed results of the experiments are presented in Table 2.

The experiments showed that AEPs-three are usually shorter than AEPs-four. During the tests, in most cases, AEPs-three included fewer operations and data than AEPs with four-parameter operations. Moreover, to produce AEPs-three, fewer co-evolutionary cycles were necessary than in the case of AEPs-four. Generating effective AEP-four required over three times co-evolutionary cycles more, on average, than generating effective AEP-three.

Table 2. Results of tests in the predator-prey problem

type of AEP	average fitness (best fitness)	average number of neurons in successful ANN (minimal number of neurons)	average length of successful AEP, number of orders + number of data (shortest AEP)	average number of co-evolutionary cycles necessary to generate successful AEP (minimal number of co-evolutionary cycles)
(AEPs-four)	831.09 (865.34)	6 (6)	4.6 + 15.1 (4 + 5)	25670.3 (17277)
(AEPs-three)	833,12 (859.65)	6 (6)	2.8 + 13.8 (2 + 5)	7081.4 (413)

4. Summary

The main goal of the experiments presented in the paper was to select operations for AE. Different types of operations were tested in the optimization and in the predator-prey problem. The experiments showed that the three-parameter operations outperform their four-parameter counterparts. Matrices generated by AEPs-three were usually better than matrices produced by AEPs-four. In the experiments in the predator-prey problem, using both types of operations made it possible to generate successful neural controllers. However, AEPs-three required fewer operations than AEPs-four to generate effective ANNs. To represent ANN consisting of six neurons, the shortest AEP-three required only two operations and five data while the shortest AEP-four, for the same purpose, needed four operations and also five data. The experiments also showed that effective AEPs-three are simpler to generate than effective AEPs-four. Using the three-parameter operations made it possible to generate effective AEPs after seven thousand of co-evolutionary cycles, on average. In the case of the four-parameter operations, almost twenty thousand cycles more were required.

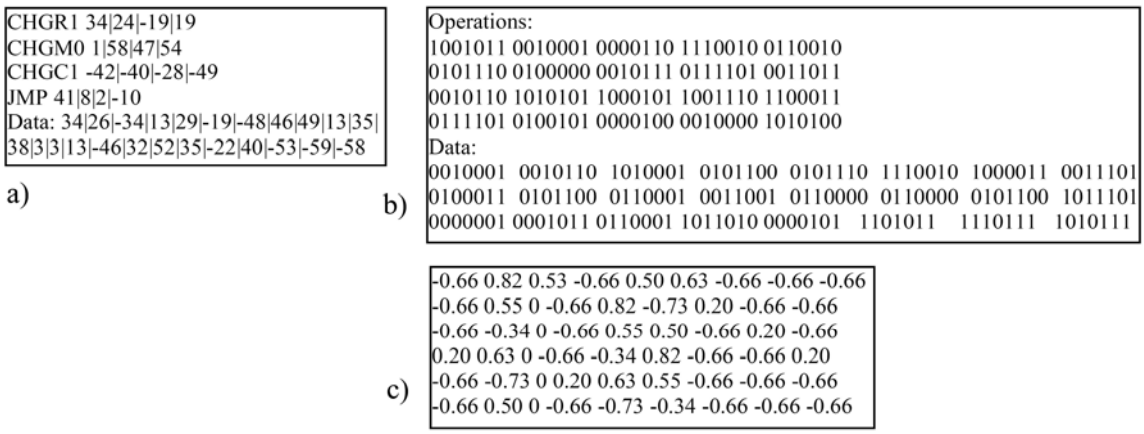


Figure 14. (a) Example AEP-four produced in this stage of experiments, (b) encoded form of AEP, (c) NDM generated by AEP presented in point (a) and (b) (The matrix depicted in point (c) corresponds to ANN consisting of six neurons: four input neurons and two output neurons. Extra three columns determine: types of individual neurons, their bias and values of parameters of neurons, e.g. a shape of radial transfer function)

Even though the experiments showed that four-parameter operations are worse solution than three-parameter ones the decision was made to use both types of operations in the future research to investigate the capability of AE to create more advanced neural architectures. In the research mentioned, we plan to apply three types of AEPs, i.e. AEPs-four, AEPs-three and additionally AEPs-mixed. The latter programs are a mixture of four-parameter and three-parameter operations. When applied in a single AEP, operations of different type may complement each other and as a result, programs better than their homogenous counterparts composed of one-type operations can be produced.

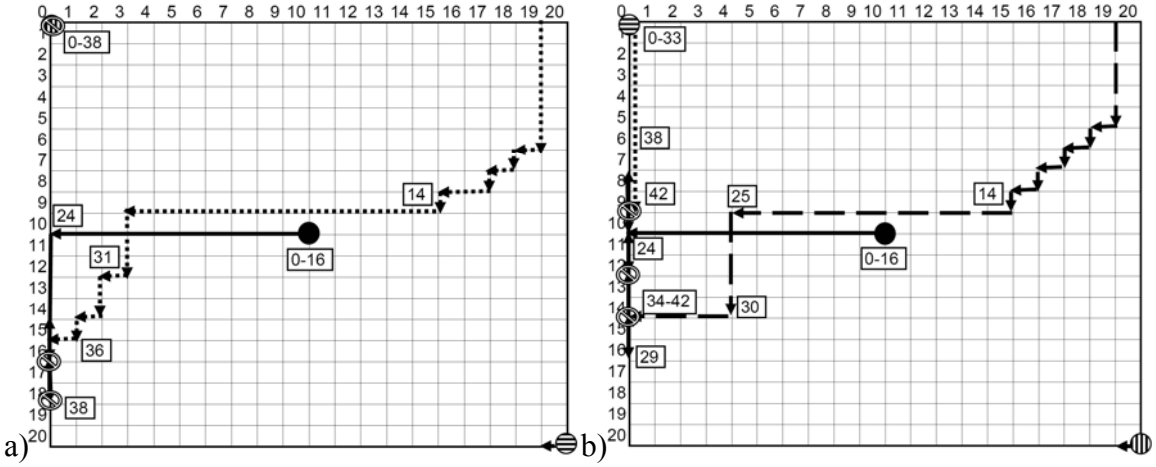


Figure 15 Example behavior of predators and prey in scenarios no. 1 (a) and no. 2 (b). Circles determine initial positions of predators and prey (black circle – prey, circle with vertical stripes – predator no. 1, circle with horizontal stripes – predator no. 2) while arrowed lines indicate their direction of movements (solid line – prey, dashed line – predator no. 1, dotted line – predator no. 2)

5. References

[1] Butz, M. V. (2004). Rule-based Evolutionary Online Learning Systems: Learning Bounds, Classification, and Prediction. (IlliGAL Report No. 2004034). Retrieved from http://citeseer.ist.psu.edu

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

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

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

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

[6] Holland, J. H. (1975). Adaptation in Natural and Artificial Systems, , Ann Arbor, Michigan: University of Michigan Press.

[7] Kitano, H. 1990. Designing neural networks using genetic algorithms with graph generation system. Complex Systems. 4: 461-476.

[8] Luke, S. & Spector, L. 1996. 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 July 28-31 (pp. 117-124). CA, USA: Stanford University, Stanford Bookstore.

[9] Miller, G.F., Todd, P.M. & Hegde, S.U. (1989). Designing Neural Networks Using Genetic Algorithms. In: *Proceedings of the Third International Conference on Genetic Algorithms* (pp. 379-384), of Schaffer J.D.
[10] Moriarty, D. E. & Miikkulainen R. (1998). Forming Neural Networks Through Efficient and Adaptive Coevolution. Evolutionary Computation, 5(4), 373-399.
[11] Nolfi, S. & Parisi, D. (1992). Growing neural networks. In C. G. Langton (ed.), *Artificial Life III*, Reading, MA: Addison-Wesley
[12] Nordin, P., Banzhaf, W. & Francone, F (1999). Efficient Evolution of Machine Code for {CISC} Architectures using Blocks and Homologous Crossover. In L. Spector and W. Langdon and U. O'Reilly and P. Angeline, *Advances in Genetic Programming III*, MIT Press, (pp. 275-299).
[13] Potter, M. (1997). The Design and Analysis of a Computational Model of Cooperative Coevolution. PhD thesis, George Mason University, Fairfax, Virginia
[14] Potter, M. A. & De Jong, K. A. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1), 1-29
[15] Praczyk, T. (2007). Evolving co-adapted subcomponents in Assembler Encoding. *International Journal of Applied Mathematics and Computer Science*, 17(4), 549-563.
[16] Praczyk, T. (2007). Procedure application in Assembler Encoding, *Archives of Control Science*, Volume 17(LIII), No. 1, 71-91.
[17] Praczyk, T. (2008). Modular Neural Networks in Assembler Encoding, *Computational Methods in Science and Technology*, CMST 14(1), 27-38.
[18] Praczyk, T. Using Assembler Encoding to solve the inverted pendulum problem, *Computing and Informatics* (in press).

Appendix 1 – List of operations used in the experiments

The four-parameter operations:

CHG – Update of an element. Both the new value and address of the element are located in parameters of the operation.

CHGC0 – Update of a certain number of elements in a column. Index of the column, index of the first element in the column that will be changed, the number of changed elements and a pointer to data where new values of elements are memorized are located in parameters of the operation.

CHGC1 – Update of a certain number of elements in a column. Index of the column, index of the first element in the column that will be changed, the number of changed elements and a new value for the column elements, the same for all the elements, are located in parameters of the operation.

CHGC2 – Update of a certain number of elements in a column. A new value of every element is a sum of an operation parameter and the current value of this element. The second parameter of the operation is an index of the column. The third and fourth parameter of the operation determine the number of changed elements and index of the first element in the column that will be changed, respectively.

CHGC3 –A number of elements from one column are transferred to another column. Both columns are indicated by parameters of the operation. The number of transferred elements and index to the first element in the column that will be transferred are also included in parameters of the operation.

CHGC4 – Update of a certain number of elements in a column. A new value of every element is a sum of the current value of this element and a 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 the operation.

CHGR0 – like **CHGC0** but an update refers to the row of matrix**.**
CHGR1 – like **CHGC1.**
CHGR2 – like **CHGC2.**
CHGR3 – like **CHGC3.**
CHGR4 – like **CHGC4.**

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

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

CHGM2 – like **CHGM0,** but a new value of each element is a sum of its current value and a value from the memory part of a program. The number of changed elements and a place in the

memory where arguments of individual sums are located are determined by parameters of the operation.

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

The three-parameter operations:

CHG_VALUE – Change of a set of matrix elements indicated by parameters of the operation. All the updated elements have the same value.

CHG_MEMORY – Change of a set of matrix elements indicated by parameters of the operation. Values for the updated elements are located in the memory.