A REVIEW OF METHODS FOR ENCODING NEURAL NETWORK TOPOLOGIES IN EVOLUTIONARY COMPUTATION

Jozef Fekiač
Tomas Bata University in Zlin,
Faculty of Applied Informatics,
nám. T.G.Masaryka 5555,
760 01- Zlín (Czech Republic)
Email: fekiac@fai.utb.cz

Ivan Zelinka
Faculty of Electrical Engineering
and Computer Science
Technical University in Ostrava
17. listopadu 15
70833- Ostrava-Poruba (Czech Rep.)
Email: ivan.zelinka@vsb.cz

Juan C. Burguillo
E.T.S.E. Telecomunicación
Universidad de Vigo
Campus Universitario de Vigo
36310-Vigo (Spain)
Email: jrial@det.uvigo.es

KEYWORDS

artificial neural network, automata network, evolutionary computation, genetic programming, genetic algorithm, network encoding, graph grammar

ABSTRACT

This paper describes various methods used to encode artificial neural networks to chromosomes to be used in evolutionary computation. The target of this review is to cover the main techniques of network encoding and make it easier to choose one when implementing a custom evolutionary algorithm for finding the network topology. Most of the encoding methods are mentioned in the context of neural networks; however all of them could be generalized to automata networks or even oriented graphs. We present direct and indirect encoding methods, and given examples of their genotypes. We also describe the possibilities of applying genetic operators of mutation and crossover to genotypes encoded by these methods. Also, the dependencies of using special evolutionary algorithms with some of the encodings were considered.

I. INTRODUCTION

Despite of frequent critics of artificial neural networks as a black-box method, they are with no doubt useful in various applications from signal processing and recognition to industrial control.

There are various specialized topologies of networks used to solve different kinds of problems. But it can be assumed that there exist other topologies useful for the types of problems that are not primarily solved by neural networks at this time. But when a new topology is needed, thanks to the almost black-box structure of neural networks, it seems to be almost impossible to manage it by standard analytic or engineering methods. Even the task of finding the number of neurons in the hidden layer of a feed-forward neural network is often only a matter of trying different possibilities. Therefore this looks like an ideal situation to use the heuristics of an evolutionary algorithm.

Evolutionary algorithms often work with direct representation of the solution. Example of this could be genetic programming (GP), which uses program trees as a genotype, but the trees are also solutions. When we try to apply a similar approach to graphs (as neural networks are), there will probably raise a problem in the application of genetic operators. The mutation operator seems easy to apply to any kind of structure, but crossing over two graphs is not so straightforward. Because of that, it seems to be an advantage to separate the genotype and the phenotype. That means to use a simplified representation of solution in chromosomes. This process of converting a network into genes will be called *encoding* in the rest of this paper.

Encoding methods can be divided in three main groups according to the process of creating the network from the encoded genome: direct, parametric and indirect encoding. They are presented in the next subsections. At the end we present the conclusions.

II. DIRECT ENCODING

In direct encoding methods there exists a direct genotype-phenotype mapping for the network. That means that all parameters of the network are clearly understandable from the genes without any repeated process of transcription or growing.

Some authors ([1]) differentiate between *direct* encoding and *structural* encoding, but in this paper we consider them being the same type of encodings. In [1], the main difference between direct and structural encoding is that direct encoding holds not only information about the presence of connections, but also about their weights.

After deeper look at *structural* encodings, authors considered that all of them could be extended to hold also weight information, though should be quite simply interchangeable. However, the encoding of the connection topology is the main concern of this paper.

A. Connection matrix

Probably the simplest representation of a graph or a network is a connection matrix. It is a square matrix $n \times n$, where n is equal to the number of nodes in the network. Then every number in the matrix at coordinates $[i, j]$ is the weight of connection between node i and node j. It is obvious, that on the main diagonal lay the weights of the graph loops and under diagonal lay the weights of recurrent (backward) connections. So, if necessary, connection matrix can be limited to the upper triangular matrix to force the network to be feed-forward (i.e. without backward connections).

Proceedings 25th European Conference on Modelling and Simulation ©ECMS Tadeusz Burczynski, Joanna Kolodziej Aleksander Byrski, Marco Carvalho (Editors)
ISBN: 978-0-9564944-2-9 / ISBN: 978-0-9564944-3-6 (CD)

One of the possibilities when using a connection matrix is to use the numbers on diagonal as identifiers of node types instead of representing the loop weights. That would of course exclude loops, what is probably useful only when designing feed-forward networks.

The numbers in the connection matrix can be of course limited to contain only numbers from set {0, 1}. This simplification does not take the weights into account; it only creates connection between nodes containing number 1 in the matrix. Figure 1 shows the process of transcription of a binary chromosome into a network phenotype.

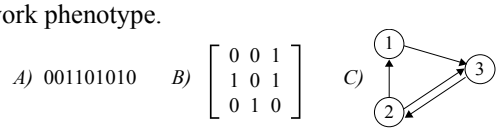

Figure 1. Binary genotype (A) is rewritten into connection matrix row-by-row (B), from which the network is created directly (C).

As can be seen on Figure 2, the genetic operator of mutation can be applied in the way of classical genetic algorithms, when a random bit in the genotype is flipped to its opposite value.

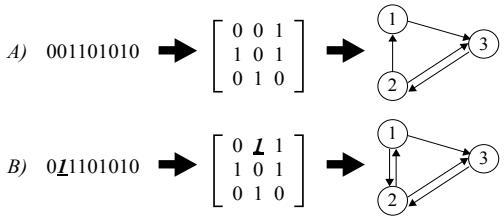

Figure 2. Application of mutation operator applied on connection matrix encoded network. Random bit in original bit string (A) is flipped to reach modified offspring (B).

The crossover operator is also applied in the straightforward way of classical genetic algorithms, as can be seen on Figure 3, where crossover with one cut point can be seen.

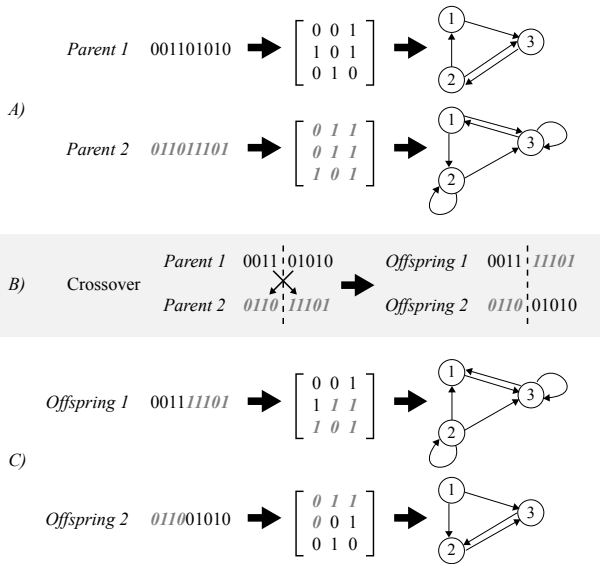

Figure 3. Application of crossover operator on connection matrix encoded network. Parent bit string chromosomes (A) are crossed over at random position (B) to create offspring (C) like in classical genetic algorithm.

B. Node-based encoding

1) Schiffmann node-based encoding

One of the attempts to extend the low flexibility of previous approach is node-based encoding. Instead of describing the network connections by a matrix of all possible connections, node-based encoding enumerates all nodes existing in the network only once, and for each node it enumerates all its inputs. This kind of encoding requires a unique identifier to be assigned to each node. Then these identifiers are used in gene transcription to clearly identify both node and its inputs.

However, this intuitive encoding is only briefly described in [7]. Only the application of the crossover operator was described in detail. In this method, crossing over means swapping parts of the genotype delimited by the borders of node definitions. An example of the encoding and the application of the crossover operator are in Figure 4.

The mutation operator then should be able to add new nodes and connections, or delete existing ones.

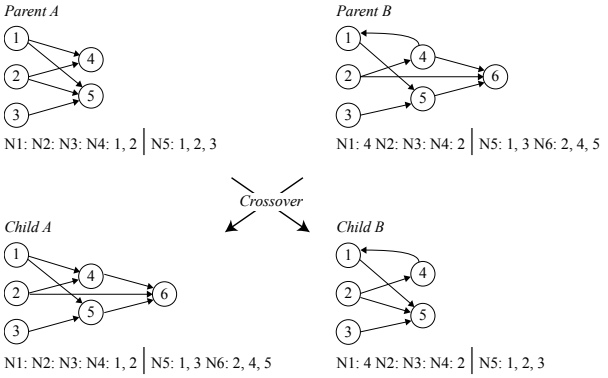

Figure 4. Example of node-based network encoding. Vertical lines in genotype transcription represent crossover points.

2) Koza node-based encoding

Another possibility of node-based encoding is to use genetic programming. Since GP is usually applied to evolve program trees in LISP language, the network in this method is represented as a tree, where the root is the output processing element (neuron) and the leaves represent the input signals. The tree structure contains all hidden nodes and connecting links with weights (Figure 5). When a network with more outputs is needed, then as a root of the genetic tree a LISP function LIST is used. This list should be holding all the outputs which are roots of their respective sub-trees. Details of this method and more possibilities of creating more complex topologies with "defined functions" are described in [6].

The genetic operators are defined by applying the rules of genetic programming, as mentioned in [5]. Mutation is defined as a replacement of a sub-tree with a new randomly generated sub-tree. This random sub-tree, as well as initial random population, has few constraints to produce a well-formed network. As can be seen in Figure 5, under any processing unit (P) there must be a variable amount of weights (W) that represent connections of processing units. Every W has two

arguments. The first of them is a number that represents the actual weight of created link. This can be a float number constant or a numerical expression tree. The second argument of W is the connection source element – input signal or another processing node. It can be easily derived that only float number constants and input signals are allowed as the leaves of a genetic tree.

Crossover genetic operation is defined simply by swapping sub-trees of two genetic trees. The only constraint is that both sub-trees have to be cut at an element of the same type, e.g., the roots of both sub-trees have to be P.

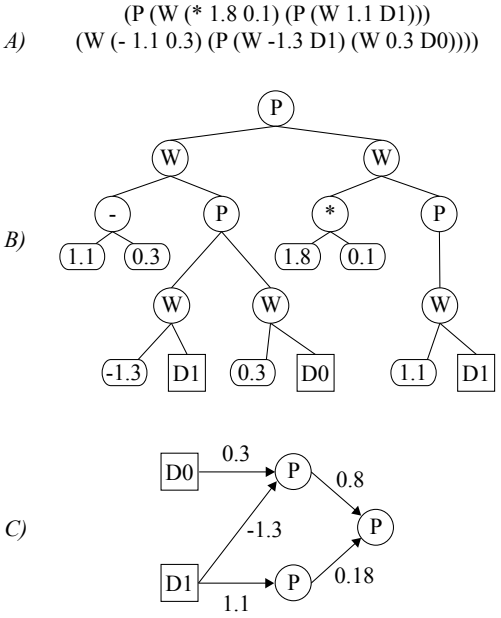

Figure 5. Example of node-based encoding by the means of genetic programming. LISP expression (A) is used as a tree (B) from which the network is constructed (C).

C. Pathway-based encoding

This approach can be used to evolve flexible and recurrent networks. It looks at the network as a set of paths from inputs to outputs. Every one of these paths begins in one of the inputs, continues through variable set of labeled nodes and ends in one of the output nodes. Of course, for one pair of a certain input and output, many possible paths could exist.

There is a context-free grammar proposed in [10], which describes the correct form of the paths in the genotype.

The process of the network construction begins with the input and the output nodes. Then it continues at the input node specified by the beginning of the current path. After that, for every node label in the path, a node with the same label in the network is found. If the node does not exist, it is created. This node is connected to the previous one in the path. Then another node label from the path is taken and the process repeats until the end of the path is reached (output). An example of this encoding can be seen in Figure 6.

The genetic operator of mutation has four possibilities to change the genotype: creating a new path, deleting an existing path, adding a neuron or removing a neuron from an already existing path.

The crossover operator is responsible for exchanging the paths between individuals and it cuts the chromosomes at two points between the path boundaries. Then, as usually, the paths between the cut points are exchanged.

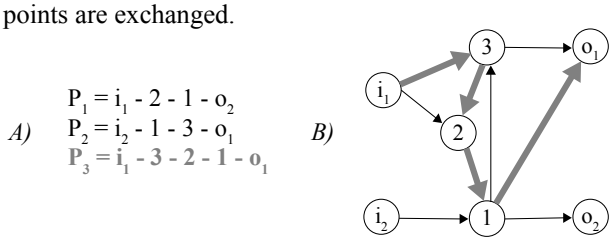

Figure 6. Example of path-based encoding. Paths (A) merged together create the final network (B). Path P_3 is displayed with wide gray lines in the network.

D. Neuroevolution of augmenting topologies (NEAT)

NEAT (described in [11]) is an evolutionary method on its own, not only a method of encoding. However, the encoding used by NEAT seems to be very useful and flexible. But due to some of its properties mentioned below, it is limited to be used with the NEAT evolutionary algorithm.

The genetic encoding applied by NEAT uses two chromosomes – one of them holds the enumeration of all available nodes in the network; the other one holds the enumeration of edges between the nodes (Figure 7). Every gene in the "node chromosome" contains a unique identifier of the node and the type of the node – a node can be an input node (sensor, receptor), an output node (actuator) or a hidden node. Genes in "edge chromosome" contain information about *begin* and *end nodes* of the edge, *weight* of the edge, information about *activation* of the gene and a *historical marker*. Activation of a gene simply tells if the edge described by a gene should be created or not and its meaning will be mentioned later on. Historical marker is a global counter which tells which mutation in all history of evolution caused creation of that gene.

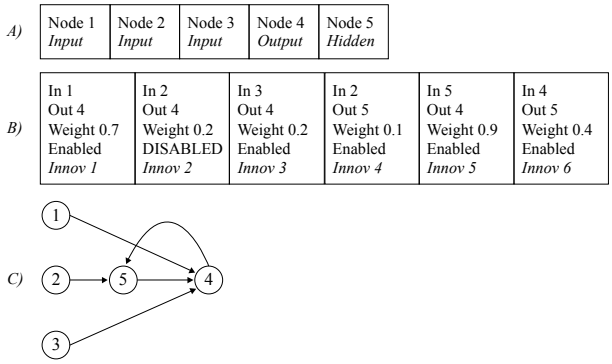

Figure 7. Example of NEAT network encoding. Genome of the individual contains node enumeration (A) and connections enumeration (B). Final network constructed from this genome can be seen in the bottom (C).

The genetic operator of mutation can influence the genotype in many ways. One of the possibilities is adding of a new node. In that case, an existing edge is

"split" and the new node is inserted in the middle. That means deactivating the gene describing an existing edge (turn to *DISABLED*) and inserting two new edges connecting two existing nodes with a new one. Another possibility is to add a new edge, what means creating a new connection gene from one existing node to another. To fine-tune the weights of a created network, the algorithm can also mutate the weights in the "edge chromosome".

Crossover in NEAT uses the aforementioned historical markers to align genes. It improves the validity of the offspring, since only compatible modifications are crossed-over (Figure 8).

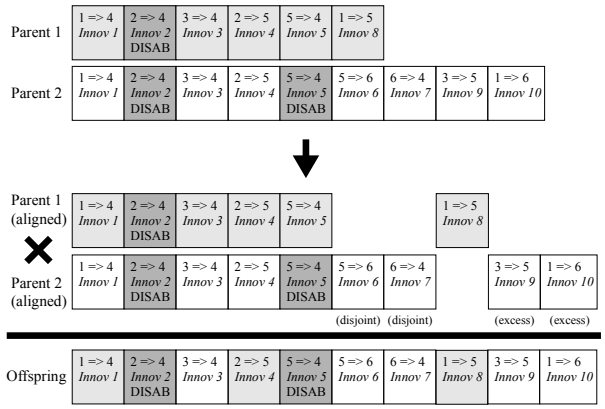

Figure 8. Aligned crossover of networks encoded by NEAT encoding.

III. PARAMETRIC ENCODING

The following approaches describe networks as genes with a set of parameters, from which the network is created by given rules. In this case, the topology of the network can be assumed from the problem domain, but the evolutionary algorithm is used to fine-tune the setting of the network.

A. Simple feedforward network encoding

Typical example of parametric encoding is a simple encoding of a feed-forward network with one hidden layer. When designing this kind of network, the back-propagation learning algorithm is typically used. Inputs and outputs that define the problem are usually given, too. Then the search space for an ideal network solving the given problem consists of finding an acceptable number of neurons in the hidden layer and finding the learning algorithm parameters, which would not get stuck in local optima in the learning phase.

Figure 9 shows an example of feed-forward neural network with one hidden layer, encoded simply in one gene (part A – number 5). Other two genes in the chromosome contain parameters of the back-propagation learning algorithm used to train this network, so together with the size of the hidden layer, also the ideal type of learning algorithm could be evolved.

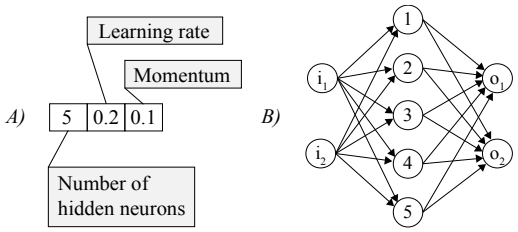

Figure 9. Simple encoding of feed-forward network with one hidden layer and the back-propagation learning algorithm. Chromosome (A) contains the number of hidden neurons and parameters of the learning algorithm. Final network (B) for the example genome has 5 neurons in the hidden layer.

B. Layer-based encoding

For some neural network applications it can be assumed that the optimal solution will be found as a multi-layer feed-forward network. In that case, it might be useful to apply the layer-based encoding. This encoding supposes a multi-layer feed-forward architecture and the back-propagation learning algorithm. The genotype of this encoding contains back-propagation learning parameters (learning rate and momentum) and parameters of a variable number of individual layers (Figure 10). Layer parameters contain information about the number of neurons in the layer and information about the output connections (to the following layer) and the input connections (from previous layers).

An one-point crossover operator is used to exchange layers between individuals. A two-point crossover operator is used to exchange bigger parts of genotypes between individuals. To maintain the consistency of the chromosomes, they are cut at the layer level.

This encoding uses relative mutation operator, which slowly changes genes of randomly chosen individuals. Besides the minimum and the maximum value, all genes contain also the maximum amount of change.

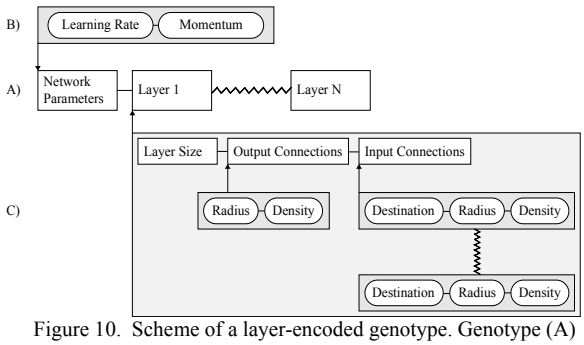

Figure 10. Scheme of a layer-encoded genotype. Genotype (A) includes common network learning parameters (B) and layer parameters (C).

IV. INDIRECT ENCODING

When trying to evolve networks able to solve complex problems, the complexity of the network is usually not big enough with the use of direct and parametric encoding of networks. Every method mentioned above (except connection matrix) supports adding new nodes and links to the network; however, there is only a little chance that any kind of regularity or modularity could evolve. Searching the space of all

possibilities with direct encoding becomes very slow for very large networks.

A. Lindenmayer systems

Lindenmayer systems (L-systems) are used for describing many biological processes in computer environments. Their most common use is in the simulation of plants growth [3].

L-systems are based on formal grammars, that means, they use *productions* (rewrite rules) that are iteratively applied on the starting string (*axiom*). The main difference is that L-systems use parallel rewriting of the string, i.e., all occurrences of the left sides of the production rules are applied at once.

In [2], context-sensitive L-systems are used to produce modular ANNs. The growth of the network starts with an axiom, on which the rewrite rules from chromosome are applied until the string contains only terminals. Context sensitivity of the system means that one symbol can be rewritten in different ways, according to its neighboring symbols. However, neighbors in this method are not considered as string neighbors, but final network neighbors. That means that in every stage of the rewriting there have to exist also network interpretation of the current string. Modules that are connected to the current module are considered to be the left-side neighbors, while the modules to whom the current module is connected are considered to be the right-side neighbors.

Each node is represented by an alphabet letter in the string. Modules are defined as groups of nodes. As this encoding is designed for evolving feed-forward networks, all modules are connected from left to right. Nodes in the module are automatically connected, until they are separated by a comma in the genotype.

An example of the network derivation from axiom through production rules can be seen on Figure 11. Part (A) displays the production rules; part (B) shows the iterations of the string rewriting process and on (C) the final network can be seen. In part (B.2) and (B.3), the brackets denote modules that are connected from left to right. Number "1" denotes a feed-forward connection skipping 1 module – so the node from first module is connected to the third module (which consists only of one node).

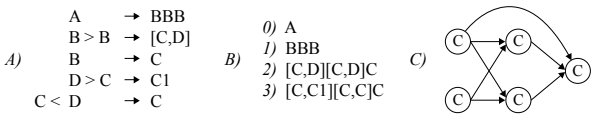

Figure 11. Example of L-system-based encoding. Rewrite rules (A) are applied to starting symbol (B.1) until the generated string contains only terminals (B.5). Then the network is constructed (C).

The authors of this approach use another encoding to transform production rules to bit strings, on which the genetic operators are applied as in classical genetic algorithms ([4]). This seems to be an unnecessary overhead, since all production rules could be stored in dynamic data types, making the recognition of their meaning more clear. However, new genetic operators would have to be designed.

B. Matrix rewriting

According to [9], L-systems can be generalized and applied to matrices. That can be used to grow a connection matrix of a network, dynamically changing its size according to the problem. Figure 12 show the derivation of a connection matrix (B.4) from genotype (A). The derivation process starts with single symbol S (B.1) and the rewrite rules are iteratively applied until only terminals (1's and 0's) are left in the generated matrix.

Mutation and crossover operators are not exactly specified in the literature, however looking at the rewrite rules, their design should be intuitive and straightforward.

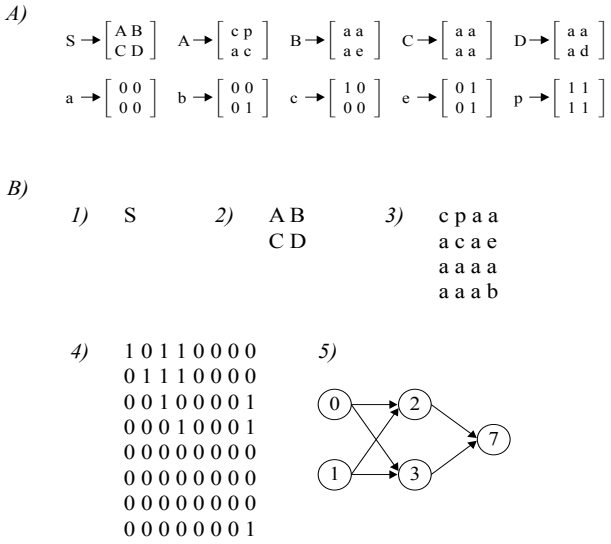

Figure 12. Example of a matrix-rewriting encoding method. A set of rewrite rules (A) is repeatedly applied to the starting symbol (B.1) until the generated matrix contains only terminals (B.4). This matrix is used as connectivity matrix and the network (B.5) is created according to it.

C. Cellular encoding

Cellular encoding is inspired by the cell splitting in the process of a living organism growth and is proposed in [12]. This encoding is based on a simple graph grammar, which is represented by a grammar tree. This graph grammar tree encodes the growth process of whole network from one initial cell. This tree can also contain control commands to influence the growth of the network. Basic commands and instructions are:

- Sequential division (SEQ) – splits the current cell in two, connected in series.
- Parallel division (PAR) – splits the current cell in two, connected in parallel.
- End program (END) – makes a neuron from the current cell and stops rewriting.
- Recursive derivation (REC) – starts applying the rewrite rules from the root of the grammar tree, until a given recursion level is reached.
- Increment/decrement the neuron threshold value (INCBIAS / DECBIAS).

- Create recursive link (CYC) – creates a link from the current cell's outputs to its inputs. This instruction fulfills the need for a recursive network topology.
- Increment/decrement link register (INCLR / DECLR). The link register stores current link from/to current cell, on which one of the following operation could be applied.
- Set positive/negative weight (VAL+ / VAL-) – sets the link in the link register to +1 or -1.
- Delete link (CUT) – deletes the link stored in the link register.

Figure 13 shows the derivation of a network from a simple grammar tree. Only four cellular instructions are applied in this case, what is enough to create a simple feed-forward network.

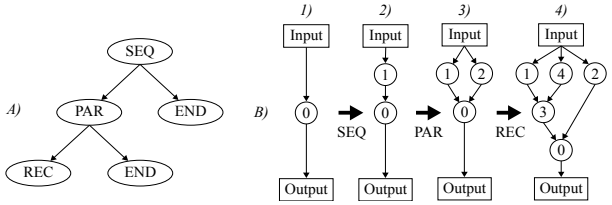

Figure 13. Example of a cellular encoded network. Instructions of the genetic tree (A) are sequentially applied to the nodes of the growing network (B). Basic instructions include SEQ – serial splitting of node (B.2), PAR – parallel splitting of node (B.3), REC – repeated application of genetic tree instructions (B.4) and END – replacing node by terminal and finishing its growth (e.g. node 0 in B.2).

Crossover and mutation operation are applied according to the common GP paradigm. That means that when mutating a chromosome tree, a random node of the tree is chosen and it is replaced by a different instruction of the same arity, or the whole sub-tree under it is replaced by a random sub-tree. Crossover is done by exchanging random sub-trees between chromosomes of two individuals.

D. Cellular graph grammars

The cellular graph grammar approach to evolve network topology proposed in [13] is also based on the similarity of the network growing with the biological processes of growth. However, instead of a fixed set of rules (like in the cellular encoding), also the grammar generating the networks evolves through generations. Another notable difference is the use of hyper-edge replacement instead of node replacement. That means that in the beginning of the growth process, there exist only one hyper-edge connecting inputs with outputs, instead of a cell connecting them. Then all of the rewrite rules are applied on hyper-edges and the final network nodes (neurons) are also created by replacing a hyper-edge with a grammar terminal.

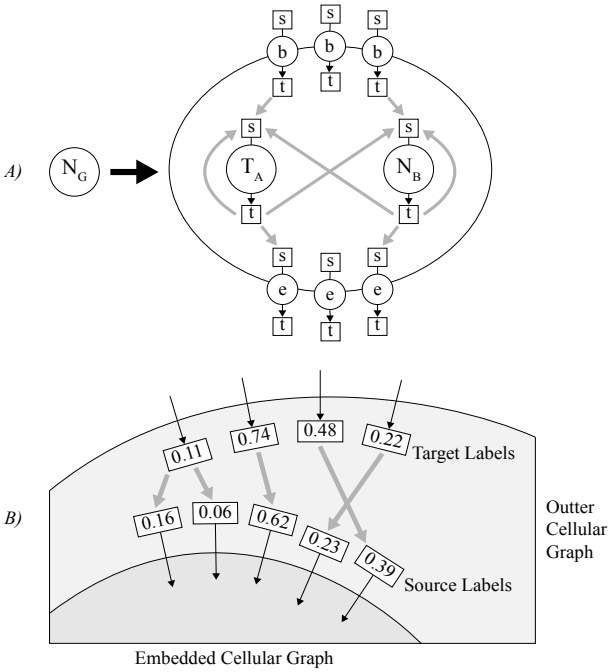

Figure 14. Basic elements of a cellular graph grammar encoding. (A) displays a general rewrite rule with non-terminal on the left side and a cellular graph on the right side with possible connections for source and target labels. When N_B in the rewrite rule is replaced by another cellular graph, the embedded cellular graph is connected to the outer by similar source and target labes (B).

To make the encoding flexible enough to handle the most possible situations during evolution, all rewrite rules evolved by grammar have added sets of labels. Then, when embedding a new sub-graph into actually growing network, all the connections are created by the similarity of the labels on appropriate positions. This embedding principle is shown on Figure 14. In the top part (A), there is a single cellular grammar production rule. N_G on the left side is a label a hyper-edge to rewrite. The right side of the rule is a *cellular graph*, by which the hyper-edge will be replaced. In this graph, *b* denotes *begin nodes*, *e* denotes *end nodes*, T_A is a *terminal* symbol and N_B is a label of another non-terminal hyper-edge (of course, the cellular graph can contain a different set of terminals and non-terminals). As mentioned above, after the replacement of a hyper-edge, the embedded cellular graph is connected to the outer graph through its begin and end nodes by the similar *source* (*s*) and *target* (*t*) labels. Direction and available levels of connections are displayed by gray arrows in the cellular graph. Part (B) of the image shows the way of label matching in detail. Two labels are matched (and then connected), when their Euclidian distance is smaller than a given threshold.

In this approach, only the genetic operator of mutation has been left. That is caused by using only a single grammar for the whole population and the individuals defined only by the label of the starting hyper-edge. Then the movement of genetic material caused by the crossover operator is also handled by the operator of mutation, because mutating one production rule modifies all individuals using that rule in their growth.

The operator of mutation operates on a single production rule, where it modifies one of the following lists: list of non-terminals, list of terminals, list of begin nodes and list of end nodes. It randomly removes an existing item from the list or adds a new item to it.

V. CONCLUSIONS

As can be seen along this paper , the differences in the three types of encoding methods are quite significant. Each of them is predetermined to solve different kind of problems. Among them, the most flexible one seems to be cellular encoding and cellular graph grammar-based encoding. However, the implementation of such sophisticated algorithms and their need for modularity might be a big overhead for real use in automation and control industry. For control purposes, well known topologies are typically used and their parameters can be found by an evolutionary algorithm using the parametric encoding.

When a network designer does not design the network directly and he decides to use evolutionary heuristics, he is facing the problem of selecting the right method of encoding; what is still some kind of "black art" and has to be done intuitively. However, we believe that this paper will make the process of choosing the right method more straightforward.

Presently we are working on a hybrid method of network encoding, combining the standard GP approach of the cellular encoding and the flexibility of the cellular graph grammar evolution to design large modular networks that could be described by a single compact genotype.

ACKNOWLEDGMENT

This research is supported by the Internal Grant Agency of Tomas Bata University under the project Artificial Life in Optimization No. IGA/28/FAI/10/D and by the European Regional Development Fund under the Project CEBIA-Tech No. CZ.1.05/2.1.00/03.0089.

REFERENCES

[1] Hussain T. S. and Browse R. A. "Genetic Encoding of Neural Networks using Attribute Grammars,"

[2] Boers, E. J. W. and Kuiper H. 1992. "Biological metaphors and the design of modular artificial neural networks." *Mater's thesis*, Leiden University, the Netherlands, 104p.

[3] Lindenmayer, A. and Prusinkiewicz P. "The algorithmic beauty of plants." New York: Springer-Verlag, 1990, 240 p.

[4] Holland, J. H. "Adaptation in natural and artificial systems." Ann Arbor: University of Michigan Press, 1975, 228 p.

[5] Koza, J. R. "Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems." Stanford: Stanford University, 1990, 131 p.

[6] Koza, J. R. and Rice, J. P. "Genetic generation of both the weights and architecture for a neural network," in Seattle International Joint Conference on Neural Networks, vol. 2, pp. 397-404, Jule 1991.

[7] Schiffmann, W. "Encoding Feedforward Networks for Topology Optimization by Simulated Evolution," Fourth International Conference on Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies, pp. 361-364, 2000.

[8] Mandischer, M. "Representation and Evolution of Neural Networks," Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms, pp. 643-694, 1993.

[9] Kitano, H. "Designing neural networks using genetic algorithms with graph generation systems," Complex Systems, vol. 4, issue 4, pp. 461-476, 1990.

[10] Jacob Ch. and Rehder J. "Evolution of neural net architectures by a hierarchical grammar-based genetic system," Proc. International Conference on Artificial Neural Networks and Genetic Algorithms, pp. 72-79, 1993.

[11] Stanley K. O.and Miikkulainen R. "Evolving neural networks through augmenting topologies," Evolutionary Computation, MIT Press, vol. 10, number 2, pp. 99-127, 2002.

[12] Gruau F. 1994. "Neural network synthesis using cellular encoding and the genetic algorithm." Dissertation, l'Ecole Normale Superieure de Lyon , France, 159 p.

[13] Luerssen M. "Experimental Investigations into Graph Grammar Evolution : A Novel Approach to Evolutionary Design. Saarbrucken." Saarbrucken : VDM Verlag Dr. Müller, 2009, 204 p.

AUTHOR BIOGRAPHIES

JOZEF FEKIAČ is a postgraduate student at Tomas Bata University in Zlin. Topic of his thesis is the use of artificial life methods in optimisation. His research is concerned in modular neural network evolution for inteligent agent control and synthesis of networks used in stegoanalysis. His e-mail address is: jfekiac@fai.utb.cz

IVAN ZELINKA was born in Czech Republic, and went to the Technical University of Brno, where he studied technical cybernetics and obtained his degree in 1995. He obtained his Ph.D. degree in Technical Cybernetics in 2001 at Tomas Bata University in Zlin, He is now a Professor at the Technical University in Ostrava, Czech Republic. His specialization is artificial intelligence and its interdisciplinary use and applications. His e-mail address is: ivan.zelinka@vsb.cz and his Web-site is at: http://ivanzelinka.eu

JUAN C. BURGUILLO received the M.Sc. degree in Telecommunication Engineering in 1995, and the Ph.D. degree in Telematics (cum laude) in 2001; both at the University of Vigo, Spain. He is currently an associate professor at the Department of Telematic Engineering at the same university. He has participated in several R&D projects in the areas of Telecommunications and Software Engineering, and has published more than one hundred papers in journals and conference proceedings. His research interests include game theory, optimization, telematic services, autonomous agents and multi-agent systems. His e-mail address is: jrial@det.uvigo.es and his Web-site is at: http://www.det.uvigo.es/~jrial