World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:9, 2008

Artificial Neural Network Development by means of Genetic Programming with Graph Codification

Daniel Rivero, Julián Dorado, Juan R. Rabuñal, Alejandro Pazos, and Javier Pereira

***Abstract*—**The development of Artificial Neural Networks (ANNs) is usually a slow process in which the human expert has to test several architectures until he finds the one that achieves best results to solve a certain problem. This work presents a new technique that uses Genetic Programming (GP) for automatically generating ANNs. To do this, the GP algorithm had to be changed in order to work with graph structures, so ANNs can be developed. This technique also allows the obtaining of simplified networks that solve the problem with a small group of neurons. In order to measure the performance of the system and to compare the results with other ANN development methods by means of Evolutionary Computation (EC) techniques, several tests were performed with problems based on some of the most used test databases. The results of those comparisons show that the system achieves good results comparable with the already existing techniques and, in most of the cases, they worked better than those techniques.

***Keywords*—**Artificial Neural Networks, Evolutionary Computation, Genetic Programming.

I. Introduction

ANNs are learning systems that have solved a large amount of complex problems related to different areas (classification, clustering, regression, etc.) [1]. The interesting characteristics of this powerful technique have induced its use by researchers in different environments [2].

Nevertheless, the use of ANNs has some problems, mainly related to their development process. This process can be divided into two parts: architecture development and training and validation. As the network architecture is problem-dependant, the design process of this architecture used to be manually performed, meaning that the expert had to design different architectures and train them until he finds the one that achieves best results after the training process. The manual nature of this process determines its slow performance although the recent use of ANNs development techniques have contributed to achieve a more automatic procedure.

Manuscript received August 30, 2006. This work was supported in part by the Spanish Ministry of Education and Culture (Ref. TIC2003-07593, TIN2006-13274), the INBIOMED network (Ref PI0/52048) financed by the Carlos III Health Institute, grants from the General Directorate of Research of the Xunta de Galicia (Ref. PGIDIT03-PXIC10504PN PGIDIT04-PXIC10503PN, PGIDIT04-PXIC10504PN), and the European project Interreg (Ref. IIIA-PROLIT-SP1E194/03).

D. Rivero, J. Dorado, J. R. Rabuñal, A. Pazos and J. Pereira are with the University of A Coruña, Department of Information & Communications Technologies, Campus Elviña s/n, 15071, A Coruña, Spain, (e-mail: {drivero, julian, juanra, apazos, javierp}@udc.es).

II. State of the Art

A. Genetic Programming

Genetic Programming (GP) [3] is based on the evolution of a given population. In this population, every individual represents a solution for a problem that is intended to be solved. The evolution is achieved by means of selection of the best individuals – although the worst ones also have a little chance of being selected – and their mutual combination for creating new solutions. This process is developed using selection, crossover and mutation operators. After several generations, it is expected that the population might contain some good solutions for the problem.

The GP encoding for the solutions is tree-shaped, so the user must specify which are the terminals (leaves of the tree) and the functions (nodes with children) for being used by the evolutionary algorithm in order to build complex expressions.

The wide application of GP to various environments and its consequent success are due to its capability for being adapted to different environments. Although the main and more direct application is the generation of mathematical expressions [3], GP has been also used in others fields such as knowledge extraction [4], rule generation [5], filter design [6], etc.

B. ANN Development with EC Tools

The development of ANNs is a topic that has been extensively dealt with very diverse techniques. The world of evolutionary algorithms is not an exception, there is a great amount of works that have been published about this topic with different techniques in this area, even with GAs or GP [3] [12] [14] [28]. These techniques follow the general strategy of an evolutionary algorithm: an initial population consisting of different genotypes, each one of them codifying ANN parameters (typically, the weight of the connections and/or the architecture of the network and/or the learning rules), is randomly created. This population is evaluated in order to determine the fitness of each individual. Afterwards, this group is made evolve repeatedly by means of different genetic operators (replication, crossover, mutation, etc.) until a determined termination condition is fulfilled. This condition can be, for example, if a sufficiently good individual is

obtained, or that a predetermined maximum number of generations is reached.

As a general rule, the field of ANN generation using evolutionary algorithms is divided into three main fields: evolution of weights, architectures and learning rules.

First, the weight evolution starts from an ANN with an already determined topology. In this case, the problem to be solved is the training of the connection weights, attempting to minimize the network failure. Most of the training algorithms, such as backpropagation (BP) algorithm, are based on gradient descent, which presents several inconveniences [7], mainly the possibility of getting stuck into a local minimum of the fitness function. With the use of an evolutionary algorithm, the weights can be represented either as the concatenation of binary values [8] or real numbers [9]. The main disadvantage of this type of encoding is the permutation problem. This problem means that the order in which weights are taken at the array can make equivalent networks correspond to completely different chromosomes, making the crossover operator inefficient.

Second, the evolution of architectures involves the generation of the topological structure. This means establishing the connectivity and the transfer function of each neuron. The network architecture is highly important for the successful application of the ANN, since the architecture has a very significant impact on the processing capability of the network. Therefore, the network design, traditionally performed by a human expert using trial and error techniques on a group of different architectures, is crucial. The automatic architecture design has been possible thanks to the use of evolutionary algorithms. In order to use them to develop ANN architectures, it is needed to choose how to encode the genotype of a given network for it to be used by the genetic operators.

At the first option, direct encoding, there is a one-to-one correspondence between every one of the genes and their subsequent phenotypes. The most typical encoding method consists of a matrix that represents an architecture where every element reveals the presence or absence of connection between two nodes [10]. These types of encoding are generally quite simple and easy to implement. However, they also have a large amount of inconveniences as scalability [11], the incapability of encoding repeated structures, or permutation [12].

Apart from direct encoding, there are some indirect encoding methods. In these methods, only some characteristics of the architecture are encoded in the chromosome. These methods have several types of representation. Firstly, the parametric representations represent the network as a group of parameters such as number of hidden layers, number of nodes for each layer, number of connections between two layers, etc [13]. Although the parametric representation can reduce the length of the chromosome, the evolutionary algorithm performs the search within a restricted area in the search space containing all the possible architectures. Another non direct representation type is based on grammatical rules [11]. In this system, the network is represented by a group of rules, with the shape of production rules which develop a matrix that represents the network, which has several restrictions.

The growing methods represent another type of encoding. In this case, the genotype does not encode a network directly, but it contains a set of instructions for building up the phenotype. The genotype decoding consists on the execution of those instructions [14].

With regards to the evolution of the learning rule, there are several approaches [15], although most of them are only based on how learning can modify or guide the evolution and also on the relationship among the architecture and the connection weights.

C. Graph-Based Genetic Programming

As was described, the codification type of the GP algorithm has the shape of trees. This allows the solving of a great size of different problems. It also allows the finding of solutions that other codifications, e.g. bit or real strings as used on GAs, can not find. Graph codification also allows the solving of problems that could not be solved with tree-shape codification. Few after the appearance of GP, some researchers have studied the possibility of using graphs inside GP to represent and solve these problems.

As first approximations of graph-based GP, some solutions appeared that used trees with special operators with the objective of solving very specific problems, e.g. to develop stack automata [16] or electrical circuits. In this field there are some works in which classic GP is used to develop different types of electrical circuits and analogical filters [17] [18]. To do this, some operators had to be created to allow GP represent so complex structures. Although the results have been very satisfactory, this kind of codification, using these operators, is very limited, and has only allowed the solving of problems in this field.

Other approximations to the codification of graphs by using trees are Gruau's and Luke's [19] [20], mainly used to develop ANNs with GP. These works use the operators on the GP tree to build graphs as this tree and its operators are being executed. These operators can be used, e.g, to create new nodes or to create links between nodes. These codifications are called cellular encoding or edge encoding, and have some drawbacks. First, the represented phenotype (i.e., the obtained graph) is too dependent of the execution order of the operators of the tree. A subtree inside an individual can turn into a completely different subtree after being crossed with another individual. Therefore, it is desirable to use a crossover algorithm that preserves better the phenotype on this operation. Also, this type of codification produces a high number of interconnected nodes, which can not be very desirable on many domains.

Teller describes a system called PADO which uses a stack and lineal discriminators in order to obtain parallel programs used for signal and image classification [21]. All of these methods use special types of parallelisms in the use of graphs,

and can not be considered as a natural generalization of GP to the use of graphs.

In a similar way as this last work, Kanstchik uses graphs as codification system, having for this purpose an indexed memory, which is used to transfer data between nodes [22]. These nodes are divided in two parts: action and ramification. The action part can be a constant or a function that will be executed when the node is reached during the program execution. To do this execution, this function takes its parameters from the stack, and writes its result on the stack too. After this, the ramification part chooses, by examining the top of the stack, which node will go on the execution between the nodes to which is connected the current one.

Another GP system which uses graphs is called Parallel Distributed Genetic Programming (PDGP) [23]. In PDGP, programs are represented as graphs with nodes that represent program primitives and links that represent the execution flow and the results. Therefore, PDGP can be used to evolve parallel programs or to produce sequential programs with shared (or reutilized) subtrees. This work defines new crossover and mutation operators for their use with these graphs.

In a new approximation to the use of graphs, this time called linear-graph GP, some special nodes were used [24]. These nodes execute a set of sequential operations and end in a conditional ramification. As a result of this ramification, these nodes point to other nodes of the same type, allowing the pointing to a node referenced more than once. Although this study allows the working directly with graphs, this work was only created to develop graph-shaped sequential programs, and can not be used to solve more generic problems.

One of the most representative works in this field is called Neural Programming [25]. This work describes a system that uses graphs as codification type. Also, this work uses a credit system on the connections between nodes to choose better which ones will be used in the individual combinations. However, this system only works with mathematical graphs (i.e., graphs with nodes representing mathematical operations such as arithmetical, trigonometrical, etc.) because it was developed for image and signal processing. Anyway, this system is one of the most complete existing ones.

III. Model

This work will use a graph-based codification to represent ANNs in the genotype. These graphs will not contain any cycles. Due to this type of codification the genetic operators had to be changed in order to be able to use the GP algorithm. The operators were changed in this way:

- The creation algorithm must allow the creation of graphs. This means that, at the moment of the creation of a node's child, this algorithm must allow not only the creation of this node, but also a link to an existing one in the same graph, without making cycles inside the graph.
- The crossover algorithm must allow the crossing of graphs. This algorithm works very similar to the existing one for trees, i.e. a node is chosen on each individual to change the whole subgraph it represents to the other individual. Special care has to be taken with graphs, because before the crossover there may be links from outside this subgraph to any nodes on it. In this case, after the crossover these links are updated and changed to point to random nodes in the new subgraph.
- The mutation algorithm has been changed too, and also works very similar to the GP tree-based mutation algorithm. A node is chosen from the individual and its subgraph is deleted and replaced with a new one. Before the mutation occurs, there may be nodes in the individual pointing to other nodes in the subgraph. These links are updated and made to point to random nodes in the new subgraph.

These algorithms must also follow two restrictions in GP: typing and maximum height. The GP typing property [26] means that each node will have a type and will also provide which type will have each of its children. This property provides the ability of developing structures that follow a specific grammar. The maximum height is a restriction of the complexity of the graph, not allowing the creation of very large graphs that could lead to obtaining too big ANNs with over-fitting problems. These two restrictions are applied on the genetic operators making the resulting graphs follow these restrictions.

The nodes used to build ANNs with this system are the following:

- ***ANN***. Node that defines the network. It appears only at the root of the tree. It has the same number of descendants as the network expected outputs, each of them a neuron.
- ***n-Neuron***. Node that identifies a neuron with n inputs. This node will have 2*n descendants. The first n descendants will be other neurons, either input or hidden ones. The second n descendants will be arithmetical sub-trees. These sub-trees represent real values. These values correspond to values of the respective connection weights of the input neurons – the first descendants – of this neuron.
- ***n-Input*** neuron. Nodes that define an input neuron which receives its activation value from the input variable n. These nodes will not have any children.
- Finally, the arithmetic operator set {+,-,*,%}, where % designs the operation of protected division (returns 1 as a result if the divisor is 0). They will generate the values of connection weights (sub-trees of the n-Neuron nodes). These nodes perform operations among constants in order to obtain new values. As real values are also needed for such operations, they have to be introduced by means of the addition of random constants to the terminal set in the range [-4, 4].

The execution of the graph will make the creation of the ANN: each n-Neuron node will make the creation on one neuron and links to the neurons which are connected to that, each n-Input node will connect an input neuron to another neuron, and an arithmetical subgraph will set the value of a weight. An example of this can be seen on Fig. 1.

Note that, during the neuron creation, a given neuron -

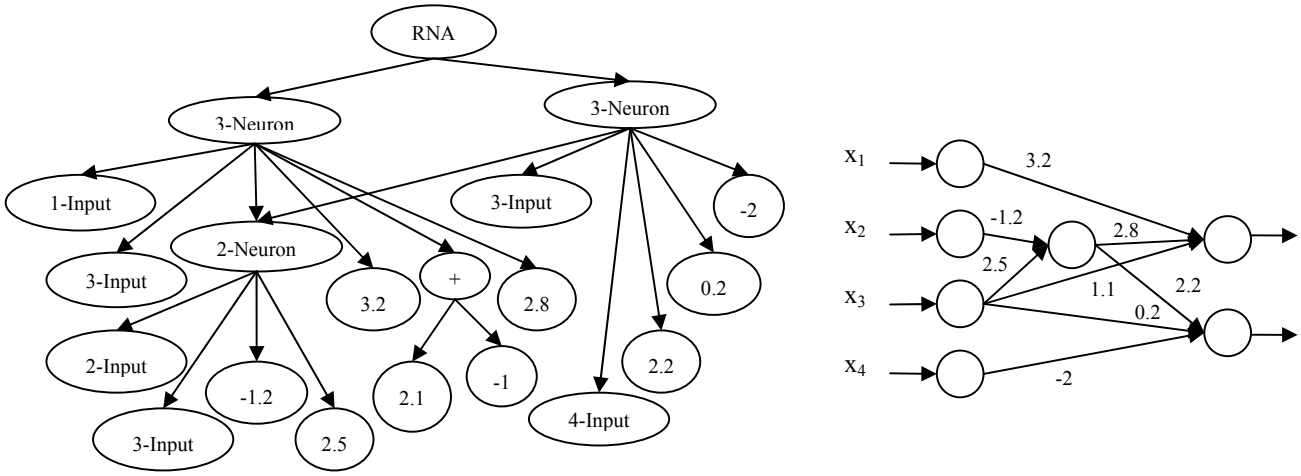

Fig. 1 GP graph and its resulting network

either an input one or a referenced one - can be repeated several times as predecessor. In such case, there is no new input connection from that processing element, but the weight of the already existing connection will be added with the value of the new connection.

Once the tree has been evaluated, the genotype turns into phenotype. In other words, it is converted into an ANN with its weights already set (thus it does not need to be trained) and therefore it can be evaluated. The evolutionary process demands the assignation of a fitness value to every genotype. Such value is the result of the evaluation of the network with the pattern set representing the problem. This result is the mean square error (MSE) of this evaluation.

Nevertheless, this error value considered as fitness value has been modified in order to induce the system to generate simple networks. The modification has been made by adding a penalization value multiplied by the number of neurons of the network. In such way, and given that the evolutionary system has been designed in order to minimize the error value, when adding a fitness value, a larger network would have a worse fitness value. Therefore, the existence of simple networks is preferred as the penalization value that is added is proportional to the number of neurons of the ANN. The calculus of the final fitness will be as follows:

$$fitness = MSE + N * P \quad (1)$$

where MSE is the mean square error of the ANN with the training pattern set, N is the number of neurons of the network and P is the penalization value for such number.

IV. Problems to be Solved

This technique has been used for solving problems of different complexity taken from the UCI database [27]. All these problems are knowledge-extraction problems from databases, where, taking certain features as a basis, it is intended to perform a prediction about another attribute of the database. The value that is intended to be predicted might be a diagnosis value (when using medical databases), a classification value or a prediction one. A small summary of the problems to be solved can be seen on Table I.

TABLE I
Summary of the Problems to be Solved

Problem	Number of inputs	Number of data points	Number of outputs
Breast Cancer	9	699	1
Iris Flower	4	150	3
Heart Disease	13	303	1
Ionosphere	34	351	1

All these databases values have been normalized between 0 and 1 and the pattern sets divided into two parts for each problem, taking the 70% of the database for training and using the remaining 30% for performing tests.

V. Results and Comparison with Other Methods

Several experiments have been performed in order to evaluate the system performance. The values taken for the parameters at these experiments were the following:

- Crossover rate: 95%.
- Mutation probability: 4%.
- Selection algorithm: 2-individual tournament.
- Graph maximum height: 5.
- Maximum inputs for each neuron: 9.
- Population size: 1000 individuals.
- Penalization value: 0.00001.

To achieve these values, several experiments had to be done in order to obtain values for these parameters that would return good results to all of the problems. These problems are very different in complexity, so it is expected that these parameters give good results to many different problems.

This last parameter, the penalization to the number of neurons, is important. The values range from very high (0.1) to very small (0.00001 or 0). High values only enables the creation of very small networks with a subsequent high error, and low values lead to overfitting problem. Experiments

showed that is preferable for this parameter to take low values instead of higher ones, allowing the creation of networks large enough for solving the problem, avoiding overfitting.

In order to evaluate its performance, the system presented here has been compared with other ANN generation and training methods.

The method 5x2cv [29] is used in Cantú-Paz and Kamath's work [28] for the comparison of different ANN generation and training techniques based on EC tools. This work presents as results the average precisions obtained in the 10 test results generated by this method. Such values are the basis for the comparison of the technique described here with other well known ones. These algorithms used on the comparison are widely explained with detail in Cantú-Paz and Kamath's work [28]. Such work shows the average times needed to achieve the results. Not having the same processor that was used, the computational effort needed for achieving the results can be estimated. This effort represents the number of times that the pattern file was evaluated. The computational effort for every technique can be measured using the population size, the number of generations, the number of times that the BP algorithm was applied, etc. This calculation varies for every algorithm used. All the techniques that are compared with the work are related to the use of evolutionary algorithms for ANN design. Five iterations of a 5-fold crossed validation test [29] were performed in all these techniques in order to evaluate the precision of the networks. These techniques are connectivity matrix, pruning, parameter search and graph-rewriting grammar.

Table II shows a summary of the number of neurons used in Cantú-Paz and Kamath's work [28] in order to solve the problems that were used with connectivity matrix and pruning techniques. The epoch number of the BP algorithm, when used, is also indicated here.

TABLE II
ARCHITECTURES USED

	Inputs	Hidden	Outputs	BP Epochs
Breast Cancer	9	5	1	20
Iris Flower	4	5	3	80
Heart Cleveland	26	5	1	40
Ionosphere	34	10	1	40

Table III shows the parameter configuration used by these techniques. The execution was stopped after 5 generations with no improvement or after 50 total generations.

TABLE III
PARAMETERS OF THE TECHNIQUES USED DURING THE COMPARISON

	Matrix	Pruning	Parameters	Grammar
Chromosome length (L)	N	N	36	256
Population size	$\lfloor 3\sqrt{L} \rfloor$	$\lfloor 3\sqrt{L} \rfloor$	25	64
Crossover points	L/10	L/10	2	L/10
Mutation rate	1/L	1/L	0.04	0.004

N = (hidden+output)*input + output*hidden

The results obtained with these 4 methods are shown in Table IV. Every box of the table indicates 3 different values: precision value obtained in Cantú-Paz and Kamath's work [28] (left), computational effort needed for obtaining such value with that technique (below) and precision value obtained with the technique described here and related to the previously mentioned computational effort value (right).

Watching this table, it is obvious that the results obtained with the method proposed here are, not only similar to the ones presented in Cantú-Paz and Kamath's work [28], but better in many cases. The reason of this lies in the fact that these methods need a high computational load since training is necessary for every case of network (individual) evaluation, which therefore turns to be time-consuming. During the work described here, the procedures for design and training are performed simultaneously, and therefore, the times needed for designing as well as for evaluating the network are combined.

Most of the techniques used for the ANN development are quite costly, due in some cases to the combination of training with architecture evolution. The technique described here is able to achieve good results with a low computational cost and besides, the added advantage is that, not only the architecture and the connectivity of the network are evolved, but also the network itself undergoes an optimization process.

Table IV also shows a small overfitting problem. This is due to the fact that the system has been left to training up to a certain number of fitness function evaluations. This usually leads to overfitting the training set when it keeps training for a long time.

VI. CONCLUSIONS

This paper presents a technique for ANN generation with GP based on graphs. To develop this system, the evolutionary operators had to be modified in order to be able to work with graphs instead of trees. This system has been compared to a

TABLE IV
COMPARISON OF THE RESULTS OBTAINED WITH OTHER METHODS AND WITH THE PRESENT ONE

	Matrix		Pruning		Parameters		Grammar	
Breast Cancer	96.77	96.27	96.31	95.79	96.69	96.27	96.71	96.31
	92000		4620		100000		300000	
Iris Flower	92.40	95.49	92.40	81.58	91.73	95.52	92.93	95.66
	320000		4080		400000		1200000	
Heart Cleveland	76.78	81.11	89.50	78.28	65.89	81.05	72.8	80.97
	304000		7640		200000		600000	
Ionosphere	87.06	88.34	83.66	82.37	85.58	87.81	88.03	88.36
	464000		11640		200000		600000	
Average	88.25	90.30	90.46	84.50	84.97	90.16	87.61	90.32

set of different techniques that use evolutionary algorithms for ANN design and training. The conclusion of such comparison is that the results of 5x2cv tests using this method are not only comparable to those obtained with other methods, but also better than them in most of the cases. It should be borne in mind that if the parameters of the system had been adapted to every problem to be solved the results would have been better. However, the parameters used have been the same for all the problems because it is intended to find a parameter set useful for any problem, and therefore there is no need of human participation. In such way, it can be stated that even without human participation, this method can improve the results of other algorithms.

This system has another advantage over other methods of ANN generation since -after a short analysis by the system- it is possible to differentiate the variables that are not relevant for problem solving, as they would be not present at the ANN. This is an important feature, since it gives new knowledge about the problem being solved.

VII. Future Work

Once the system has been proved, the work continues towards several directions. One interesting research line would be the possible integration of a GA into the system in order to train the networks being generated.

As was explained earlier, this system has an overfitting problem. Another research line could be to use any technique to avoid overfitting, such as early stop.

Acknowledgments

The development of the experiments described in this work, has been performed with equipments belonging to the Super Computation Center of Galicia (CESGA).

The Cleveland heart disease database was available thanks to Robert Detrano, M.D., Ph.D., V.A. Medical Center, Long Beach and Cleveland Clinic Foundation.

References

[1] S. Haykin, *Neural Networks* (2nd ed.), Englewood Cliffs, NJ: Prentice Hall, 1999.

[2] J. R. Rabuñal and J. Dorado, (eds.) *Artificial Neural Networks in Real-Life Applications*, Idea Group Inc, 2005.

[3] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, Cambridge, MA, MIT Press, 1992.

[4] J.R. Rabuñal, J. Dorado, A. Pazos, J. Pereira and D. Rivero, "A New Approach to the Extraction of ANN Rules and to Their Generalization Capacity Through GP". Neural Computation, vol. 16, n. 7. 2004. pp. 1483-1523.

[5] M. Bot, "Application of Genetic Programming to Induction of Linear Classification Trees", Final Term Project Report, Vrije Universiteit, Amsterdam, 1999.

[6] J. R. Rabuñal, J. Dorado, J. Puertas, A. Pazos, A. Santos and D. Rivero, "Prediction and Modelling of the Rainfall-Runoff Transformation of a Typical Urban Basin using ANN and GP", *Applied Artificial Intelligence*, 2003.

[7] R. S. Sutton, "Two problems with backpropagation and other steepest-descent learning procedure for networks", *Proc. 8th Annual Conf. Cognitive Science Society*, Hillsdale, NJ: Erlbaum, 1986, pp. 823-831.

[8] D. J. Janson and J. F. Frenzel, "Training product unit neural networks with genetic algorithms", *IEEE Expert*, vol. 8, 1993, pp. 26-33.

[9] G. W. Greenwood, "Training partially recurrent neural networks using evolutionary strategies", *IEEE Trans. Speech Audio Processing*, vol. 5, 1997, pp. 192-194.

[10] E. Alba, J. F. Aldana and J. M. Troya, "Fully automatic ANN design: A genetic approach", *Proc. Int. Workshop Artificial Neural Networks (IWANN'93)*, Lecture Notes in Computer Science, vol. 686. Berlin, Germany: Springer-Verlag, 1993, pp. 399-404.

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

[12] X. Yao and Y. Liu, "Toward designing artificial neural networks by evolution", *Appl. Math. Computation*, vol. 91, no. 1, 1998, pp. 83-90.

[13] S. A. Harp, T. Samad and A. Guha, "Toward the genetic synthesis of neural networks", *Proc. 3rd Int. Conf. Genetic Algorithms and Their Applications*, J.D. Schafer, Ed. San Mateo, CA: Morgan Kaufmann, 1989, pp. 360-369.

[14] S. Nolfi and D. Parisi, "Evolution of Artificial Neural Networks", *Handbook of brain theory and neural networks, Second Edition*, Cambridge, MA: MIT Press, 2002, pp. 418-421.

[15] P. Turney, D. Whitley and R. Anderson, "Special issue on the baldwinian effect", *Evolutionary Computation*, vol. 4, no. 3, 1996, pp. 213-329.

[16] A. Zomorodian, 1995. "Context-free Language Induction by Evolution of Deterministic Push-down Automata Using Genetic Programming", in *Working Notes of the Genetic Programming Symposium, AAAI-95*, Eric Siegel and John Koza, chairs. AAAI Press. 1995.

[17] Z. Fan, K. Seo, R. C. Rosenberg, J. Hu and E. D. Goodman, "Exploring Multiple Design Topologies Using Genetic Programming And Bond Graphs". *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*. Springer-Verlag. 2002, pp. 1073-1080

[18] Z. Fan, K. Seo, J. Hu, R. C. Rosenberg and E. D. Goodman, "System-Level Synthesis of MEMS via Genetic Programming and Bond Graphs", *Genetic and Evolutionary Computation -- GECCO-2003*. Vol. 2724. 2003, pp. 2058-2071.

[19] F. Gruau, "Genetic micro programming of neural networks", in Kinnear, Jr., K. E., editor, *Advances in Genetic Programming*, chapter 24, MIT Press, 1994, pp. 495–518.

[20] S. Luke and L. Spector, "Evolving Graphs and Networks with Edge encoding: Preliminary Report". In *Late Breaking Papers at the Genetic Programming 1996 Conference (GP96)*. J. Koza, ed. Stanford: Stanford Bookstore, 1996, pp. 117-124.

[21] A. Teller, "Evolving Programmers: The Co-evolution of Intelligent Recombination Operators", in *Advances in Genetic Programming II*, P. Angeline and K. Kinnear, editors. Cambridge: MIT Press., 1996.

[22] W. Kantschik, P. Dittrich, M. Brameier and W. Banzhaf, "MetaEvolution in Graph GP", *Proceedings of EuroGP'99, LNCS*, Vol. 1598. SpringerVerlag, 1999, pp. 15-28.

[23] R. Poli "Evolution of Graph-like Programs with Parallel Distributed Genetic Programming", *Genetic Algorithms: Proceedings of the Seventh International Conference*, 1997.

[24] W. Kantschik, W. Banzhaf, "Linear-Graph GP - A new GP Structure", in *Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2002*, 2002.

[25] A. Teller A. and M. Veloso, "Internal reinforcement in a connectionist genetic programming approach", *Artificial Intelligence*. Vol. 120, N. 2, 2000, pp. 165-198.

[26] D. J. Montana, "Strongly typed genetic programming", *Evolutionary Computation*, Vol. 3, No. 2, 1995, pp. 199-200.

[27] C. J. Mertz and P. M. Murphy, UCI repository of machine learning databases. http://www-old.ics.uci.edu/pub/machine-learning-databases, 2002

[28] E. Cantú-Paz and C. Kamath, "An Empirical Comparison of Combinations of Evolutionary Algorithms and Neural Networks for Classification Problems", *IEEE Transactions on systems, Man and Cybernetics – Part B: Cybernetics*, 2005, pp. 915-927.

[29] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms", *Neural Computation*, Vol. 10, No. 7, 1998, pp. 1895-1924.