Integrating Evolutionary Computation with Neural Networks

E. Vonk *, L.C. Jain **, L.P.J. Veelenturf *, and R. Hibbs *

*Control, Systems and Computer Engineering Group (BSC),
Laboratory for Network Theory,
Department of Electrical Engineering,
University of Twente, Postbus 217,
7500 AE Enschede,
The Netherlands.
Tel. : +61 8 302 3984
Fax : +61 8 302 3384
Email: 940021a@lux.levels.unisa.edu.au

**Knowledge-based Engineering Systems Group,
School of Electronic Engineering,
University of South Australia,
Adelaide, The Levels, 5095,
Australia.
Tel. : +61 8 302 3315
Fax : +61 8 302 3384
Email: etlcj@lv.levels.unisa.edu.au

Keywords - evolutionary computation, genetic algorithms, evolutionary algorithms, evolutionary strategies, evolutionary programming, genetic programming, neural networks

Abstract

There is a tremendous interest in the development of the evolutionary computation techniques as they are well suited to deal with optimization of functions containing a large number of variables. This paper presents a brief review of evolutionary computing techniques. It also discusses briefly the hybridization of evolutionary computation and neural networks and presents a solution of a classical problem using neural computing and evolutionary computing techniques.

1. Introduction

This paper presents a brief review of evolutionary computation techniques, hybridization of these techniques and neural computing techniques and the application of evolutionary computing techniques to a classical exclusive-OR problem. Section 2 introduces briefly the evolutionary computation techniques incorporating genetic algorithms (GAs), genetic programming (GP) and evolutionary algorithms (EA). Section 3 presents a brief summary of the possible hybridizations of evolutionary computation and neural networks. Section 4 presents some experiments on the application of an evolutionary computing technique to a classical exclusive-OR problem. Section 5 presents the conclusion and the future direction.

2. Evolutionary Computation

Evolutionary computation is the name given to a collection of algorithms based on the evolution of a population towards a solution to a certain problem. These techniques are successfully used in many applications including the automatic generation of a neural network architecture. The population of possible solutions evolves from one generation to the next, ultimately arriving at a satisfactory solution to the problem. The algorithms differ in the way a new population is generated from the present one and in the way the members are represented within the algorithm.

There is much confusion about the grouping and naming of the various kinds of evolutionary computations. In this paper we distinguish between three kinds of evolutionary computations: Genetic Algorithms (GAs), Genetic Programming (GP) and Evolutionary Algorithms (EAs). The latter can be divided into Evolutionary Strategies (ES) and Evolutionary Programming (EP).

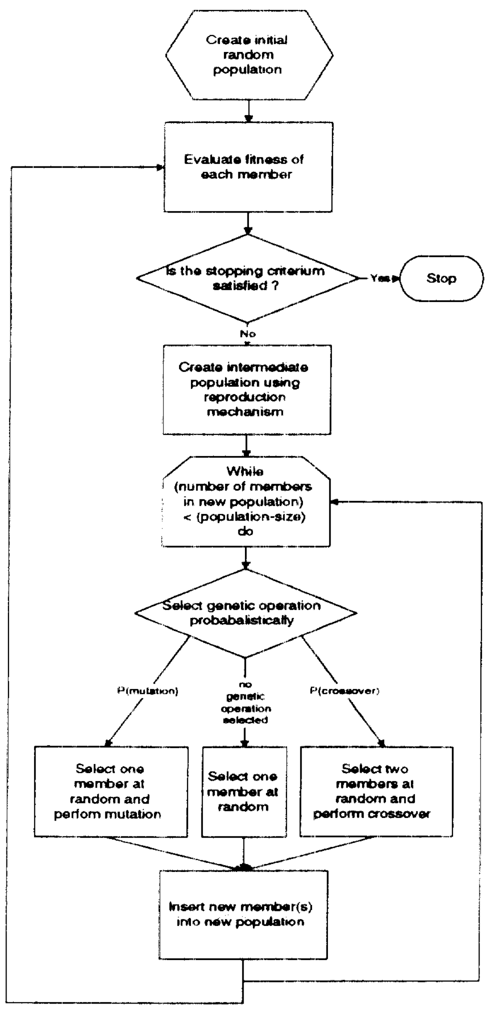
2.1. Genetic Algorithms (GAs)

Genetic algorithms were developed by John Holland in the 1970's and they rely on a linear representation of the genetic material. In GAs the members of the population are called chromosomes and are often coded as fixed-length binary strings although variable length strings have been used as well. Chromosomes are made up of a set of genes. In the case of binary strings they are just bits.

2.1.1. The algorithm

Fig. 1 shows the flowchart of the standard genetic algorithm. The reproduction operator that is most commonly used is the fitness proportionate or roulette wheel method, where members of a population are extracted using a probabilistic Monte Carlo procedure based on their average fitness. For example, a chromosome with a fitness of 20% of the total fitness will on average make up 20% of the intermediate generation.

The heuristics of GAs are mainly based on this reproduction and on the crossover operator, and only on a very small scale on the mutation operator. The crossover operator exchanges parts of the chromosomes (strings) of two randomly chosen members in the intermediate population and the newly created chromosomes are placed into the new population. Sometimes instead of two, only one newly created chromosome is put into the new population; the other one is discarded. The mutation operator works only on a single chromosome and randomly alters some part of the representation string. Both operators (and sometimes more) are applied with a certain probability.



Create initial random population
Evaluate fitness of each member
Is the stopping criterium satisfied ?
Yes
Stop
No
Create intermediate population using reproduction mechanism
While (number of members in new population) < (population-size) do
Select genetic operation probabalistically
P(mutation)
no genetic operation selected
P(crossover)
Select one member at random and perfom mutation
Select one member at random
Select two members at random and perform crossover
Insert new member(s) into new population

Fig. 1: Flowchart for the standard genetic algorithm

The stopping criteria is when there is a chromosome in the current population that gives an adequate solution or when a set number of generations have been completed. Many variations on the above algorithm are possible. For example, the mutation is very often performed after a new population has been made; i.e. sequentially. Also, instead of selecting a genetic operator probabalistically, often a certain exact percentage of the new population is made using this operator.

2.1.2. Genotypes and phenotypes

GAs rely on two separate representational spaces. One is the recombination space, where the actual genetic operations are performed on the (binary) strings or genotypes. The other space is the evaluation space where the actual problem-structures or phenotypes are evaluated on their ability to perform the task and where their fitness is calculated. An interpretation or mapping function is necessary between the two. The coding of the genetic material plays an important role in the performance of the GAs. The genetic operators perform their task on the genotypes without any knowledge of their interpretation in the evaluation space. This works fine as long as the interpretation function is such that the application of the genetic operators in the recombination space leads to good points in the evolution space. Problems occur when a structure (or several very similar structures) in the evaluation space can be represented by very different genes in the recombination space. Schaffer et al. [9] calls this "competing conventions", but it is also referred to as the phenomenon of different structural mappings (genotypes) coding the same or very similar functional mappings (phenotypes) [17]. Basically it means that a unimodal error landscape becomes multimodal where each peak represents a representation (convention) of the structure. It is very unlikely that crossover between two different chromosomes having the same convention will result in a useful offspring.

2.1.3. The Steady State Genetic Algorithm (SSGA)

Sometimes instead of first making an intermediate population and then applying the genetic operators another approach is used where the operators are applied directly to members of the current population. These members are chosen based on their fitness. The newly made chromosome is then merged into the current population taking the place of a chromosome that was chosen based on its inversed fitness. For a single generation step, this process is repeated until the number of removed chromosomes equals the number of members in the population. This approach is called a Steady State Genetic

Algorithm (SSGA) as opposed to the standard Batch Genetic Algorithm. It requires much less memory storage as only one population instead of two needs to be stored. A certain notion of age can be built into the system where for a certain number of iterations these newly made members can not be reselected to create a new offspring.

Apart from the roulette wheel method another reproduction mechanism called tournament selection is often used. Here a certain number of chromosomes (the tournament size) are selected randomly from the population and the best member of this group replaces the worst

2.2. Genetic Programming (GP)

Genetic programming is a technique derived from genetic algorithms and was developed by John Koza [1]. De Garis [13] uses the same name for his work, but there is no relation between the two except that both are based on genetic algorithms. Genetic programming can be seen as a special kind of genetic algorithms but differs in that it uses hierarchical genetic material that is not limited in size. The members of a population or chromosomes are tree structured programs and the genetic operators work on the branches of these trees. Originally genetic programming was implemented in the LISP programming language, because of its build-in tree like data structures (S-expressions), but it has been implemented in various languages since. The main advantage of GP over GA is that the size and shape of the final solution does not need to be known in advance. Of course GP is only advantageous over GA if the chromosomes can be represented adequately by hierarchical tree structures. Research has shown that GP can be successfully applied to many problems in the fields of artificial intelligence, machine learning and symbolic processing [1].

In GP the chromosomes are made up of a set of functions and terminals connected to each other by a tree structure. Typically the set of functions include arithmetic operations, logical operations and problem specific operations. The terminal set is made up of the data inputs to the system and the numerical constants. Functions can generally have both other functions as well as terminals as their arguments and must therefore be well-defined to handle any input combination. The number of arguments a function has must be defined beforehand. GP incorporates 'variable selection' so that it is not needed to set a priori which data-inputs are going to be used. These are selected on the run. This can be a useful concept when it is not known in advance exactly which data-inputs are needed in order to solve the problem.

As in the standard genetic algorithm paradigm, genetic programming relies mainly on the reproduction mechanism and the crossover operator. The flowchart for the standard genetic algorithm (fig. 1) also applies for genetic programming and the same reproduction mechanisms apply. Crossover is performed on branches of trees, which means that entire branches or subtrees are swapped between two chromosomes.

As in the genetic algorithm paradigm, there exists a steady state approach to genetic programming. Steady State Genetic Programming (SSGP) has proven to be advantageous over the standard, or batch GP paradigm in certain applications [2].

An interesting feature within the GP paradigm which accounts for modularity is the possibility to include the so called Automatically Defined Functions (ADFs) [1]. These ADFs perform a subtask of the problem and can be called upon more than once. An ADF does not have particular fixed terminals as its inputs, but instead is parameterized by dummy variables. When an ADF is called upon from within the main program (Koza calls this the result producing branch), the dummy variables are instantiated with specific values or terminals. The ADFs are defined in the so called function-defining branch. The complete genetic tree that represents a certain solution therefore consists of a result-producing and a function-defining branch. The genetic operators work on both branches. The idea is that GP will dynamically evolve functions that are useful to the problem (ADFs) as well as a main program that calls upon these functions. A parallel can be drawn here to the field of neural networks where a certain part of the network performs a function that can be seen as a subtask for the complete problem. The difference is that its position within the neural network is fixed and that it is of no use to the network if its needs this same function somewhere else but with other actual inputs.

2.3. Evolutionary Algorithms (EAs)

Evolutionary algorithms [22] are another form of evolutionary computation but instead of GAs (and GP), they focus on phenotypes and not on genotypes. There is no need for a separation between a recombination and an evaluation space. The genetic operators work directly on the actual structure or phenotype. The structures used in evolutionary algorithms are representations that are problem dependent and more natural for the task than the general representations used for GA. Originally evolutionary algorithms focused on a single parent only, but extensions have been made for a population consisting of more members. Evolutionary algorithms can be divided into Evolutionary Strategies (ES) which focus on the behaviour of individuals, while Evolutionary Programming (EP) focus on the behaviour of entire species.

In EP, the only genetic operator that is used is the representation-dependent mutation operator, although several different mutation operators can be used in the same algorithm. A commonly used mutation operator just adds a Gaussian random variable to each component of a chromosome. Because ES deal with individuals instead of entire species sexual operators (crossover) are possible as well and extensions have been made to include these.

3. *Hybridization of Evolutionary Computation and Neural Networks*

Evolutionary computation can be used in neural network design in several ways. For example, it can be used :

- on a fixed neural network structure to train the network; i.e. to determine the weights
- to generate the architecture of the neural network to be trained by a separate learning algorithm (usually back propagation)
- to analyse a neural network
- to generate both the neural network architecture and the weights

3.1. Evolutionary computation as a learning algorithm for a NN

In the first application above, the genetic algorithm provides a good alternative to a learning algorithm such as back propagation which often gets stuck in local minima. The genetic algorithm performs a global search of the weight space and therefore is unlikely to get stuck in a local minima. Evolutionary computation does not use error-gradient information. Therefore, unlike algorithms such as BP, they can be used where this information is not available or computationally expensive. It also means that the activation function of the neurons does not have to be differentiable or even continuous. Genetic algorithms can be used to train any type of neural network including fully recurrent networks. A problem with genetic algorithms is that they are very slow in fine tuning once they are close to a good solution. Therefore the hybridization of GA and BP, where BP is used to fine-tune a near-optimal solution found by GA, has proved to be successful [3].

The members of the population are the weights of the network which are coded as strings. When real valued weights are used, they are usually coded into a binary string using a binary or a Gray coding mechanism. The fitness measure is normally calculated as the performance error of the network on test data and the genetic algorithm can in such a case be classified as a supervised learning algorithm.

In [4] a GA is used to evolve ecological neural networks that can adapt to their changing environment. This is done by letting the fitness function, which in this case is seen as individual for every gene, co-evolve with the weights of the network. A special feature of this research is that there is no reinforcement for 'good' behaviour of the network; the network just tries to model or adapt to the world in which it lives.

De Garis [13] uses a method which is based on fully self-connected neural network modules. It is shown that using this approach a network can be taught a certain task even though the time-dependent input varies so fast that the network never settles down. The system does not use a crossover operator (it could therefore be called evolutionary programming) and is used to teach a pair of sticks to walk.

In [15] and [19], a genetic algorithm is used on a fixed three layer feedforward network to find the optimal mapping from the input to the hidden layer. In the evaluation phase the weights from the hidden to the output layer are adjusted using a simple supervised delta rule. The fitness measure again is just the performance error on the training set.

3.2. Evolutionary computation to generate an optimal NN topology

In the second application the chromosomes contain the topology of the network and sometimes the learning parameters such as the learning rate as well. Genetic algorithms are used in [5],[14],[20],[21],[25]. There are several ways to encode the network topology as a chromosome. The most commonly used methods are:

- a connectivity matrix
- a graph grammar

Graph grammar based systems are often found to perform better than methods using a connectivity matrix. This is due to the fact that when the matrix method is used, the chromosomes and accordingly the search space for the algorithm becomes very large as the network size is increased. When graph grammars are used this is not the case, as the networks produced are highly structured or modular [25].

The above methods can be classified as 'strong or low-level representations' because the complete network topology is coded in the chromosomes. When 'weak or high-level representations' are used, the chromosomes do not contain the complete network topology. Instead they consist of more abstract terms, like 'the number of hidden neurons' or 'the number of hidden layers' etc. A method that uses modules of neurons and where only the connections between these modules are coded can be seen as a weak representation as well.

During the evaluation (calculation of the fitness measure) every member of the population is translated into a neural network which is then learned using a separate learning algorithm like back propagation. As the chromosomes do not contain information concerning the weights of the network, these have to be set to an initial (random) value. After each network has learned they are tested using test data, and the fitness measure is calculated. This causes a problem as the performance of a neural network usually depends on the values of the initial weights. Therefore, in order to get a good fitness measure, the networks are to be learned several times using different initial weights each time and the results are averaged. This causes the approach to become very slow, see e.g. [14]. After evaluation, the usual genetic operators (crossover, mutation) are performed on the chromosomes representing the networks to obtain a new population.

3.3. Evolutionary computation in NN analysis

Although this combination of GA's and NN's is not commonly used, GA's can be used to analyse or explain neural networks. In [10] GA's are used as a neural network inversion tool in that they can find the input patterns that yield a certain output of the network.

3.4. Evolutionary computation to generate both a NN topology and its weights

The last option is the current focus of this study. Successfully reported applications include using a genetic algorithm where the topology and weights are encoded as variable-length binary strings [6]. Gruau [11] uses a graph grammar approach called 'cellular encoding' in a genetic programming system to implement Boolean neural networks. The mechanism is modular in that subnetworks or building blocks can be used for a subtask of the problem.

In [12] a structured GA is used that simultaneously optimizes the neural network topology and the values of the weights. A two-level genetic structure represented by a single binary string is used where one level defines the connectivity (topology) and the other the values of the weights and biases. It was found that although the algorithm worked well on small problems like XOR, it could not scale up properly to bigger real-world problems.

In [17] feedforward neural networks are generated by using GAs, incorporating a strong representation scheme where every gene in a chromosome represents a connection between two neurons. The problem of competing conventions is tackled here by introducing connection specific distance coefficients in the genetic material. For each functional mapping or phenotype, the structural mapping or genotype with the shortest amount of connection lengths is preferred. This approach is also known as 'restrictive mating' [24]. This way, some of the topology information of the phenotypes (the actual neural networks) is incorporated into the genotypes. A disadvantage of the system proposed is that a maximum size neural network topology, including the number of hidden layers used, needs to be specified in advance.

Jacob and Rehder [18] use a grammar-based genetic system, where topology creation, neuron functionality creation (e.g. activation functions) and weight creation are split up into three different modules, each using a separate GA. The modules are linked to each other by passing on a fitness measure. The grammar used is such that a neural network topology is represented as a string consisting of all the existing paths from input to output neurons. In [24] a somewhat similar modular design approach is used but here a distinction is made between structure, connectivity and weights optimization. Structure here is defined as the number of layers and the number of neurons in every layer.

Angeline et al. [7] have reported a scheme based on evolutionary programming where the networks evolve using both a parametric mutation (mutation of the weights) and a structural mutation. It is argued that EP is a better choice for this task than is GA, mainly because it is not clear that there exists an appropriate interpretation function between the recombination and evaluation space for the application of neural network design.

3.5. Modular neural networks

In sections 3.2 and 3.4 systems are described that use a modular neural network structure where groups of neurons are treated as a single module [8],[14],[23]. An advantage of using modular neural networks is that the weight space is reduced. This has a positive effect on both the generalization capability and the time needed to learn the network.

4. Experiments

Experiments were performed on genetic algorithms as a learning algorithm for the weights of a neural network. The exclusive-OR function was chosen as a test case in our preliminary investigations. Table 1 shows the results of applying gradient descent (back propagation) and a genetic algorithm for optimizing the weights of a multiple-layer perceptron network structure (2-4-1 network). The training was stopped when the network had correctly learned all of the four training facts.

Details of the two algorithms used are:

- sigmoid activation function
- training tolerance = 0.2

Back propagation:

- includes bias units
- learning rate = 0.1
- momentum term = 0.9

Genetic algorithm:

- population size = 80
- chromosome length = 192 bits
- max. range of weights = [-20,20]
- crossover probability = 0.7
- mutation probabbility = 0.02

Table 1: Optimization of the weights of a multiple-layer perceptron network

Method	Average learning time
Genetic Algorithm	12 secs
Back Propagation	14 secs

The results were averaged over several runs. In some runs the back propagation algorithm did not converge even after a very long time. These runs were left out of the results. The genetic algorithm does not suffer from being stuck in local minima and therefore is more robust. Learning times for this problem are similar for the two algorithms.

5. Conclusions

This paper has presented in brief a review of evolutionary computation techniques, hybridizations of evolutionary computation and neural computing techniques and some experiments on classical problems. The long term goal of this research work is to develop intelligent techniques for rapidly training neural networks for use in a dynamic environment.

Acknowledgments

Thanks are due to the Defence Science and Technology Organisation, Salisbury, Adelaide, South Australia, for the financial support (contract number 340479). Edgar Vonk wishes to thank the Control, Systems and Computer Engineering Group (BSC), Laboratory for Network Theory, Department of Electrical Engineering, University of Twente, for the permission to undertake this project in the Knowledge-based Engineering Systems Group, University of South Australia.

References

[1] Koza, John R., *Genetic Programming, On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, 1992.

[2] Kinnear, Kenneth E. Jr., "Evolving of a Sort: Lessons in Genetic Programming", *IEEE International Conference on Neural Networks, New York*, vol. 2, pp. 881-888, 1993.

[3] Hibbs, R. A. , "Speeding up Backpropagation: A Comparative Study", Technical Report, Knowledge-based Engineering Systems Group, University of South Australia, Australia, 1994.

[4] Lund, Henrik H. and Parisi, Domenico, "Simulations with an Evolvable Fitness Formula", Technical Report PCIA-1-94, C.N.R., Rome, 1994.

[5] Harp, S. A. and Samad T., "Genetic Synthesis of Neural Network Architecture", in: *Handbook of Genetic Algorithms*, edited by L. Davis, Van Nostrand Reinhold, pp. 202-221, 1991.

[6] Maniezzo, Vittorio, "Genetic Evolution of the Topology and Weight Distribution of Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, January 1994.

[7] Angeline, Peter, J., Saunders, Gregory M., and Pollack, Jordan M., "An Evolutionary Algorithm that Constructs Recurrent Neural Networks", *IEEE Transactions on Neural Networks*, Vol. 5, No. 1, January 1994.

[8] Koza, J. R. and Rice, J. P., "Genetic Generation of both the Weights and Architecture for a Neural Network", *IEEE International Joint Conference on Neural Networks*, 1991.

[9] Schaffer, J. D., Whitley D. and Eshelman, L. J., "Combinations of Genetic Algorithms and Neural Networks: A Survey of the State of the Art", *IEEE International Workshop on Combinations of Genetic Algortihms and Neural Networks (COGANN-92)*, Baltimore, pp. 1-37, 1992.

[10] Eberhart, R.C., "The Role of Genetic Algorithms in Neural Network Query-Based Learning and Explanation Facilities", *IEEE International Workshop on Combinations of Genetic Algortihms and Neural Networks (COGANN-92)*, Baltimore, pp. 169-183, 1992.

[11] Gruau, F., "Genetic Synthesis of Boolean Neural Networks with a Cell Rewiting Developmental Process", *IEEE International Workshop on Combinations of Genetic Algortihms and Neural Networks (COGANN-92)*, Baltimore, pp. 55-74, 1992.

[12] Dasgupta, D. and McGregor D. R., "Designing Application-Specific Neural Networks using the Structured Genetic Algorithm", *IEEE International Workshop on Combinations of Genetic Algortihms and Neural Networks (COGANN-92)*, Baltimore, pp. 87-96, 1992.

[13] Garis, H. de, "Genetic Programming, Building Nanobrains with Genetically Programmed Neural Network Modules", *IEEE International Joint Conference on Neural Networks, New York*, vol. 3, pp. 511-516, 1990.

[14] Boers, E.J.W and Kuiper, H., "Biological Metaphors and the Design of Modular Artificial Neural Networks", Technical Report, Departments of Computer Science and Experimental and Theoretical Psychology, Leiden University, The Netherlands, 1992.

[15] Hassoun, M.H., *Fundamentals of Artificial Neural Networks*, MIT press, 1995.

[16] Lohmann, R., "Structure Evolution in Neural Systems",in: *Dynamic, Genetic, and Chaotic Programming*, edited by B. Soucek and the IRIS Group, John Wiley & Sons, Chapter 15, pp. 395-411, 1992.

[17] Braun, H. and Weisbrod J., "Evolving Neural Feedforward Networks", *International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93)*, Innsbruck, Austria, pp. 25-32, 1993.

[18] Jacob, C. and Rehder, J., "Evolution of Neural Net Architectures by a Hierarchical Grammar-based Genetic System", *International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93)*, Innsbruck, Austria, pp. 72-79, 1993.

[19] Munro, P.W., "Genetic Search for Optimal Representations in Neural Networks", *International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93)*, Innsbruck, Austria, pp. 628-634, 1993.

[20] Mandisher, M., "Representation and Evolution of Neural Networks", *International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93)*, Innsbruck, Austria, pp. 643-649, 1993.

[21] Schiffmann, W., Joost, M., and Werner, R., "Application of Genetic Algorithms to the Construction of Topologies for Multilayer Perceptrons", *International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93)*, Innsbruck, Austria, pp. 676-682, 1993.

[22] Fogel, D.B., "An Introduction to Evolutionary Computation", *Australian Journal of Intelligent Information Processing Systems*, Vol. 1, No. 2, pp. 34-42, 1994.

[23] le Cun, Y., "Generalization and Network Design Strategies", in: *Connectionism in Perspective*, edited by: Pfeifer, P., Schreter, Z., Fogelman-Soulié F., and Steels, L., Elsevier Science Publishers B.V. (North-Holland), pp. 143-155, 1989.

[24] Alba, E., Aldana, J.F., and Troya, J.M., "Genetic Algorithms as Heuristics for Optimizing ANN Design", *International Conference on Artificial Neural Nets and Genetic Algorithms (ANNGA93)*, Innsbruck, Austria, pp. 683-689, 1993.

[25] Kitano, H., "Designing Neural Networks Using Genetic Algorithms with Graph Generation System", *Complex Systems*, vol. 4, pp. 461-476, 1990.