Automatic Construction of General Non-Recurrent Neural Network Using Genetic Programming

Noora A. Al-Saidi *

* computer Science Dept. , AL-Rafidain University College.

Abstract

An Artificial Neural Network (ANNs) is a model that has been studied for many years in the hope of achieving human like performance. It was used in many applications like (pattern recognition, signal processing, vision, speech recognition, and decisions making aids and robotics).

There are many types of ANNs, most of these types are suffering from some problems such as Convergence, Design and some of these ANNs cannot solve the non-linear problems.

Since Genetic Programming (GP) is a machine learning technique used in the automatic induction of computer programs, therefore we used it as automatic system for designing and implementing ANNs trying to overcome the previous problems.

In this paper we proposed an approach to build ANNs based on evolutionary computation, which uses the GP to evolve both the architecture and weights of General non-recurrent NN simultaneously. New GP modification operations were proposed which are (structure-preserving crossover, structure-preserving mutation, and structure-preserving permutation), these modification operations have clear effects in improving the results of the proposed system.

The proposed system has been used in solving five simple problems which are (AND, OR, NOT, Exclusive-OR, and Half-Adder) problems. In addition, we compare the results of the proposed system with Koza work and with three major types of supervised ANNs, which are (Perceptron, Back-Propagation, and Adaline).

1. Introduction

Genetic Programming (GP) is a domain-independent problem-solving approach in which computer programs are evolved to solve, or approximately solve, problems. Thus, it addresses one of the central goals of computer science, namely automatic programming. The goal of automatic programming is to create, in an automated way, a computer program that enables a computer to solve a problem [1].

GP is a stochastic search algorithm based on the Darwinian principles of reproduction, sexual recombination and survival of the fittest. It continues the trend of dealing with the problem of representation in Genetic Algorithms (GA) by increasing the complexity of the structure undergoing adaptation [1,2,3], this paradigm has several properties that make it more suitable than other paradigms (best-first search, hill-climbing, etc.), these properties are [1,3]:

1- GP produces a solution to a problem as a computer program.
2- The structures undergoing adaptation in GP are general hierarchical computer programs of dynamically varying size and shape.
3- It is a probabilistic search algorithm.
4- It differs from most other search techniques in that they simultaneously involve a parallel search involving hundreds or thousand of points in the search space.

Therefore, GP has been used in wide area of applications. Koza [3] gives a good illustration of these applications. The construction of ANNs is one of the problems, which was addressed by Koza as a good example of the application of GP.

ANNs are a parallel, distributed information processing structure consisting of processing elements interconnected via unidirectional signal channels called connections. This structure has many characteristics that are not presented in modern parallel computers. Some of these characteristics are massive parallelism, fault tolerance and robustness, adaptation or learning ability, and self-organization [4].

These features enable ANNs to be used in many applications like pattern classification, speech recognition, printed circuit board inspection, image processing, weather forecasting, and cipher systems analysis and attacking [5,6,7]. ANNs can be classified into many types, this classification is done according to (architecture, learning algorithm, type of input, etc.) [8,9].

Most of the ANNs types are suffering from some problems such as Convergence, and Design. By convergence we mean that learning algorithm requires a long time for a neural network to learn the training patterns. The design problem includes the determination of the required number of layers and the number of hidden units in each layer. In addition, some of ANN types can not solve non-linear problem [10].

Therefore, Koza tried to overcomes the previous ANN problems using GP, Koza illustrate S-expressions with a syntactic structure and structure-preserving crossover with problem having an even more complex set of rules of construction

[3]. This method can be used to construct non-recurrent ANNs, then he shows how can be implement ANNs for Half-Adder problem using his method [3].

In this work the proposed system is used to construct supervised ANNs (it means that the construction of ANNs required specifying the correct output pattern for each input pattern) using GP, which is the main concept. We see that the problem of the construction of ANNs required special rules and syntax. This syntax is appropriate to build non-recurrent ANNs (without feedback) with or without Bias.

The GP in the proposed system uses new additional modification operations with crossover to adjust the weights and build the structure of required ANNs. These additional modification operations are (structure-preserving mutation, and structure-preserving permutation).

These operations have useful effects on the results of the proposed systems when we compare the results of the proposed system with Koza work.

To analyze this system, we used it and three major supervised ANNs (Perceptron, Back-Propagation, and Adaline) to solve some of simple problems likes OR, NOT, AND, Exclusive-OR, and Half-Adder logic circuits problem, and compare the results. We choose these simple problems because these problems easily can be implemented in all the types of ANNs to implement the comparison.

2. Automatic Construction of Artificial Neural Networks Using GP

In supervised learning of ANNs, the network must be providing with a correct answer (output) for every input pattern [11]. By using GP, designing ANNs required to specifying the input pattern and what is the relative output pattern of this ANNs.

In this section we illustrate the design of ANN using GP in terms of:

1-The structure undergoing adaptation,
2-The initial structure,
3-Fitness measure,
4-Genetic operators, and
5-Stopping criteria.

2.1 The Structure Undergoing Adaptation:

The structure that undergoes adaptation in this work is a population of neural networks represented as S-expressions (prefix expression) with a syntactic structure. These neural networks have different shapes and weights. GP applies its modification operators on this population during successive generations.

2.2 The Initial Structure:

At first GP requires explaining the function set (F) and terminal set (T) that can be used to construct the required ANNs before initializing the structures in GP. The terminal set contains the input signals and a random floating point constant called (n), for example ranging between (-2.000) and (+2.000). If there exists two input signals (A,B), and a random floating point constant (n), then the terminal set is :

T={A,B,n}

The function set contains the arithmetic operations (+, -, *, /) and four special functions (#, R, P, W). Thus the function set is

F={P, W, R, #, +, -, *, /}

Where:-

P: Is a processing element (or also called neuron), it has a specific activation function, in this work the threshold function was used. The threshold value is (1.0), (P) works by getting the weighted sum of all its inputs, then compare it with the threshold value, if the sum is (greater than or equal) to the threshold value then (P) gives "1" otherwise it gives "0". (P) can be have any number of inputs and give its result to any number of neurons. Each input in (P) must be came from (W) function as we see in ANN rule construction.

W: Is the weight function, which is used to give a weight to a signal going into a (P) function. The (W) function takes only two arguments and the result of this function is the products of its two arguments.

R: The function R is used for the neural network that has multi output. Therefore each input of (R) must be an output of a neuron (P). The function (R) copies its input to a vector this is because always the output of a neural network is a vector.

#: Is the bias function, which has only one argument, the output of (#) function is the same input argument. It gives its output to the neuron, if that neuron has a bias function.

The problem of designing ANNs illustrates the need to create the initial random population so that all individuals comply with certain specified syntactic rules of construction.

Not all-possible occurrences correspond to what we would reasonably call a neural network. Thus, the problem of designing ANNs requires rules of construction that specify what structures are allowable for this particular problem.

Each ANNs in GP is represented as S-expression. There are six types of points in the S-expressions:

- points with an (R) function,
- points with a (P) function,
- points with a (W) function,
- points with a (#) function,
- points with arithmetic functions or floating-point random constants, and
- points with input data signals (such as A and B).

2.2.1 Artificial Neural Networks Construction Rules [3]:

In particular, the rules for constructing neural networks are as follows:

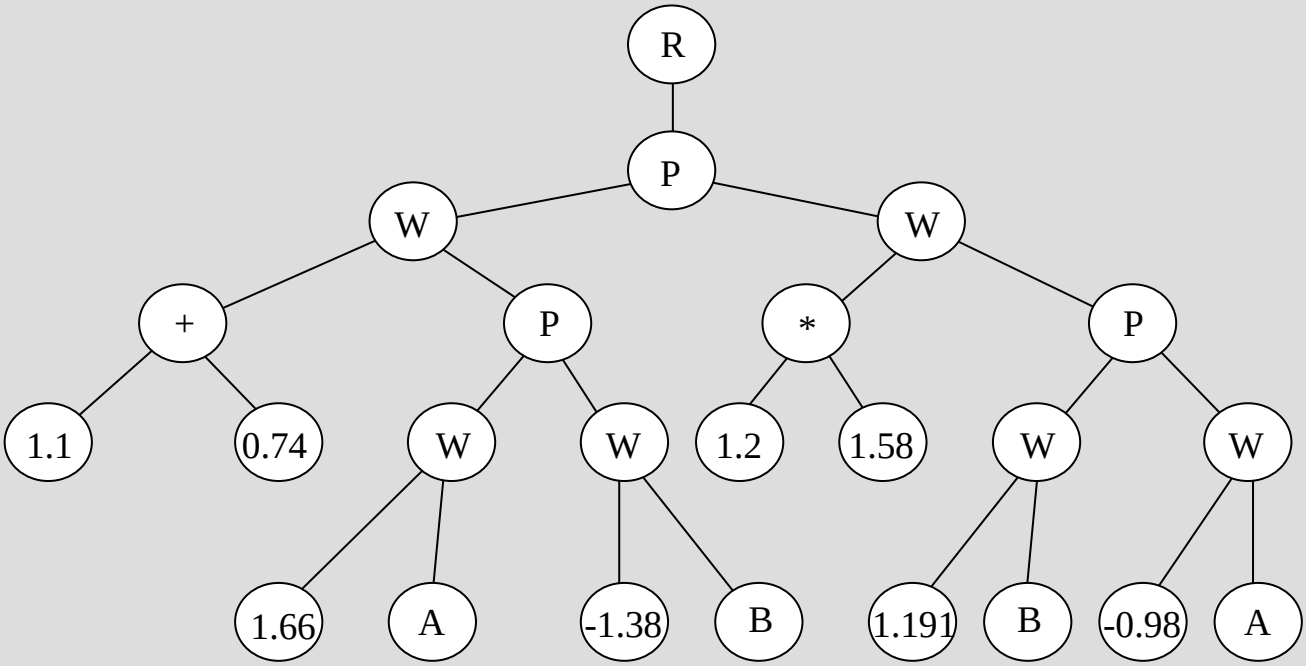
1- The root of the tree must be an (R) function.
2- The only thing allowed at the level immediately below (R) is a (P) function. The number of inputs to (R) depends on the number of outputs from the neural network.
3- The only thing allowed at the level immediately below any (P) function is having one (#) function, and a number of (W) functions depending on the number of inputs to that neuron (P).
4- The only thing allowed below a weight function (W) on the left (i.e., the first line) is either a floating-point random constant or an arithmetic function.
5- The only thing allowed below a weight function (W) on the right (i.e., the second line) is either an input data signal (such as A or B) or the output of a function (P).
6- The only thing allowed below a (#) function is a floating-point random constant.
7- The only thing allowed below an arithmetic function is either a floating-point random constant or an arithmetic function (+,-,*,/).

These rules are applied recursively. Note that the external points of the tree are either input signals or floating-point random constants. These rules of construction produce a tree (S-expression) that one can reasonably call a neural network.

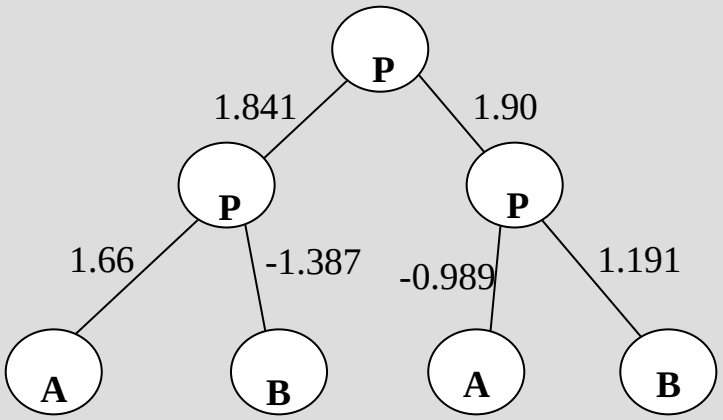
To illustrate this structure, consider the Exclusive-OR (odd-2-parity) task on the inputs A and B. This function has one output. The following S-expression represents a neural network that performs this task:

R(P(W(+(1.1 0.741) P(W(1.66 A) W(-1.37 B)))
W(*(1.2 1.584) P(W(1.191 B) W(-0.989 A)))))

Figure (1-a) graphically depicts this S-expression as a rooted point-labeled tree with ordered branches. This S-expression represents the same neural network as shown in figure (1-b).



- a -



- b -

Fig. (1):
(a) – Tree for the S-expression that performs the Exclusive-OR (odd-2-parity) task.
(b) – Neural network for Exclusive-or (odd-2-parity) function.

2.3 Fitness measure:-

The fitness measure of the neural network depends on the input data and the desired output, for example the fitness cases for the exclusive-or consist of four cases representing the four combinations of binary input signals (i.e., 00,01,10,and 11) that could appear on A and B. The correct output signal O (namely, 0,1,1, and 0, respectively) are then associated with each of these four fitness cases.

The raw fitness measure is the sum of the binary differences, taken over the four fitness cases, between the output signal from the ANN and the correct output signal. For example if we have a truth table and the output obtained from the ANN as shown below.

Input		Actual Output	Obtained Output	Error
A	B			
0	0	0	0	Success 0
0	1	1	1	Success 0
1	0	1	0	Fail 1
1	1	0	0	Success 0
				Sum 1

Table (1): Compute deference between real output and obtained output.

This means that the raw fitness is equal to the sum of error (1). While the adjusted fitness is equal to (1 / (1 + raw fitness)). The selection operation depends on the fitness to select the individuals. The selection algorithm used in our work is q-tournament selection.

2.4 Genetic Programming Operators:-

In our work, the GP operations must be performed so as to preserve the required structure of the individuals for the problem. That is, the result must be an allowable ANNs in every case.

Structure Preserving Crossover (SPC):-

Structure-preserving crossover is performed as follows:-

1. In the first parent, any point may be selected as a cross point without restriction.
2. However, the selection of the cross point in the second parent is then restricted to a point of the same type as the point chosen in the first parent.

There are five types of cross points:

a) A processing element function (P),
b) A weighting function (W),
c) A bias function (#),
d) An input data signal (such as D0 or D1), and
e) An arithmetic function or random constant.

Structure Preserving Mutation (SPM):-

The mutation operation is an asexual operation, which mean that it needs only one parent and generate one child. In mutation operation, a point is selected randomly in the parent without any restriction. Then remove the sub tree that the selected point is the root of it. Then generate a random tree (Using the same construction rule) where the root is the same type of selected point. Mutation point can be any point except (R) function.

Structure Preserving Permutation (SPP):-

Permutation operation is another asexual operation that has been used in our work. This operation is performed only on an arithmetic operation point to save the syntactic structure of the chromosome (neural network).

2.5 Stopping Criterions:

GP is an iterative procedure that needs some condition to stop iterating. In the problem of designing ANNs the stopping condition is to find the desired ANN. This means that the obtained output must be equal to the actual output.

3. The proposed system:

The main program of the proposed system is implemented by using C++ to construct a ANNs for a specific problem, this system is illustrated in fig. (2) in the next page which represents the flowchart of this system.

As shown in figure (2), the flowchart consists of (9) stages (each block represents a stage). Each stage has its own task as shown in next page:

1- In stage one we specify the properties of the required ANN (number of neurons in ANN, number of input, number of output, has bias or not). Then give the input patterns and actual output patterns.

2- In stage two we specify the GP parameters (max number of generation, population size, probability of crossover, probability of mutation, probability of permutation). These parameters have default values as shown in table (2) below, or can be entered by user.

GP parameters	Default Value
Population size	500
Max number of generation	100
Probability of crossover	0.9
Probability of mutation	0.1
Probability of permutation	0.1

3- In stage three GP will start to build the initial population of the ANNs according to the previous properties and rule of construction. The population is saved in a file because it has unlimited size.

4- In stage four GP evaluate each neural network and compute the fitness by comparing the output of ANN with the actual output, then compute no. of differences and number of hits. For example:

Input		Output	Actual Output	Differences
1	1	1	1	0
1	0	1	0	1

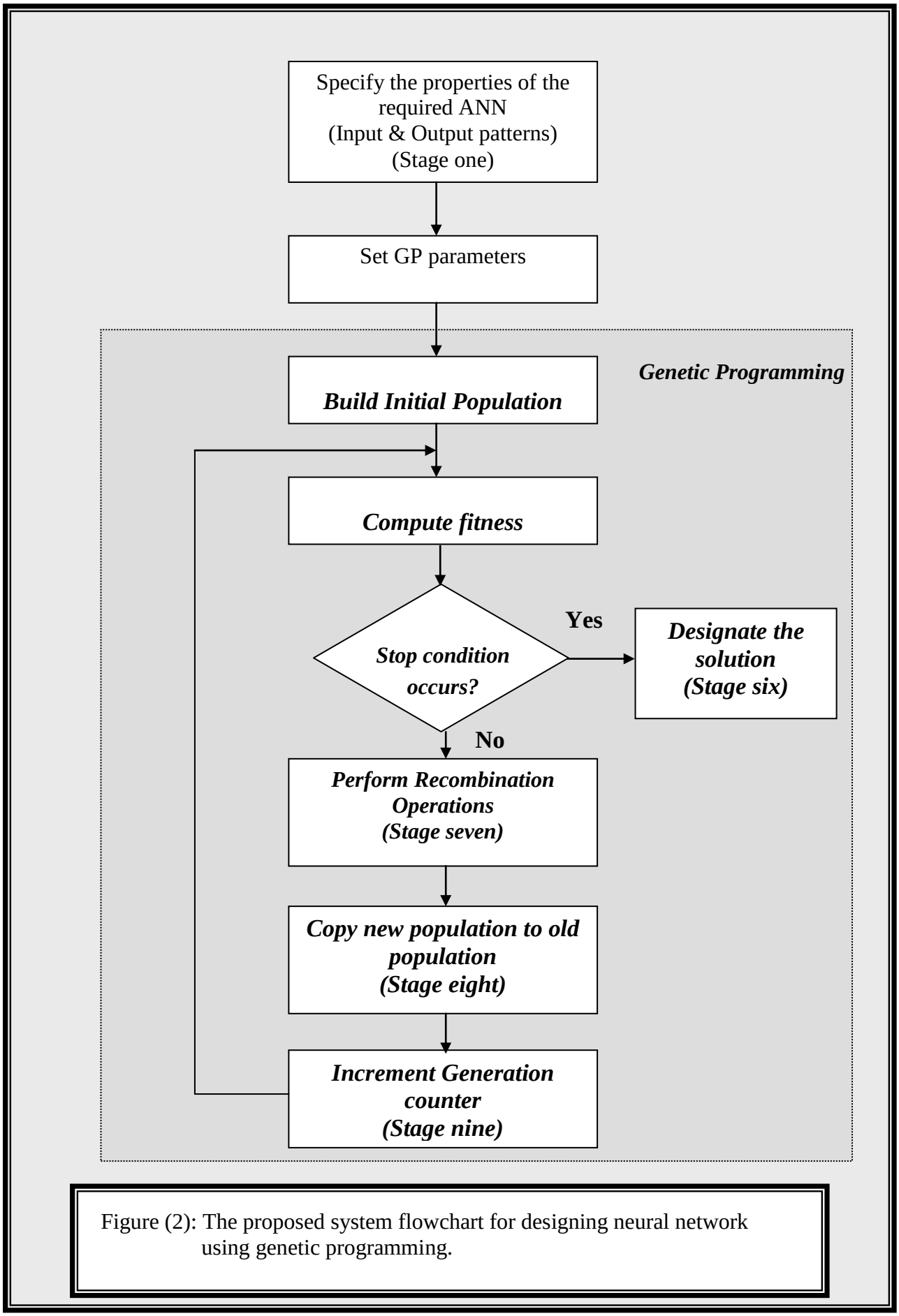


Figure (2): The proposed system flowchart for designing neural network using genetic programming.

From the previous table, the sum of difference equals to (1) and the number of hits (No. of success raw) is (1), and the fitness value equal to (0.5).

5- In stage five GP tests if the stopping condition is true or false, this is done by testing if the fitness value equal to (1) and the number of hits equal to the number of cases in the truth table of the ANN. If the stop condition is true then exit and designate the result; otherwise perform the genetic recombination operations.
6- In stage seven GP performs its recombination operation according to the probability value. It generates a random number, then if this number is less than or equal to the probability of the recombination operation then do this operation, and otherwise do another recombination operation.
The selection algorithm used in this work is the Q-tournament selection, because this selection method is the best method [1], which works as follow:
- Select q individuals randomly from the population.
-Select the best individual from the q selected individuals according to its fitness values. Where q is an integer number in the range $1 \leq q \leq$ population size.
7- Stages eight and nine copy the new population from a file called (new) to the old population in a file called (old), and increment the generation counter.
8- Stage six designates the solution as s-expression and saves it in an array of characters. And print it in prefix expression form.

To illustrate how this system works, see the example of designing the Exclusive-OR in a ANN using GP, which is shown below:

Example: This example explains how to use the proposed system to construct ANN that performs the exclusive-OR function.
The truth table of the exclusive-OR function is shown below:-

Input Output

The cases

A	B	Result
0	0	0
0	1	1
1	0	1
1	1	0

The proposed system requires some input parameter, we entered these parameters experimentally as show below:-

The parameters	Value
Population size	500
Number of neuron	3
Threshold	1.0
Number of input	2
Number of output	1
Number of cases	4

From these parameter values the system gets the solution (required ANN) in generation (17). The obtained result is:

R(P(W(-(0.08 /(-1.05 1.08))P(W(1.42 A)W(-1.44 B)))W
(-(+(0.23 0.83)-0.5)P(W(-0.5 A)W(-(1.41 0.36)B)))))

4. GP parameters and modification operations effects on the proposed system:

Now, we test the effect of the main parameters of GP on the results of the proposed system, for implementing this test we used the proposed system to solve three problems, which are (AND, XOR, Half-Adder). The main parameters of GP which are used for this test are as follow:

1. Effect of different generative methods.
2. Effect of the mutation operation (SPM).
3. Effect of the permutation operation (SPP).
4. Effect of the crossover operation (SPC).

Since GP is a probabilistic algorithm, not all runs are successful at yielding a solution to the problem by generation (i). When a particular run of GP is not successful after running the pre-specified maximum number of generations, there is no way to know whether or not the run would ever be successful. Thus, there is no knowable value for the number of generations that will yield a solution. We can compute the probability of success that the particular run with a population of size (M) yields the solution by (i). Each curved is based on 100 independent runs.

4.1 Effect of different generative methods:

In this section, we compare the effects of using two different methods of creating the initial random population, namely "**full**", and "**grow**" methods. The next figure (3) show the probability of success according to generations range value. As we see in this figure, it contains three graphs for AND.

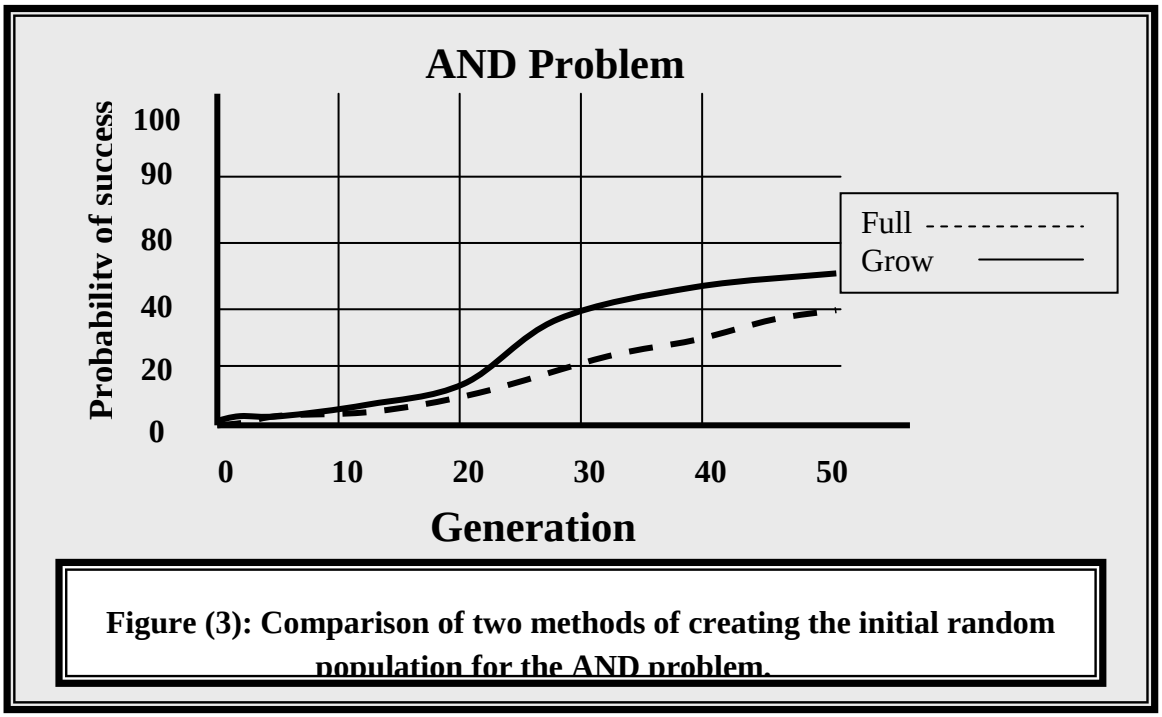


Figure (3): Comparison of two methods of creating the initial random population for the AND problem.

Grow generative methods is higher than probability of success for the full generative method for all problems. Therefore we standardized Grow generative method in this work.

4.2 Effect of the mutation operation (SPM):

The mutation operation is one of the secondary operations of genetic programming. In this test, we consider the effect on performance when mutation is added to runs involving the usual default number of occurrences of fitness-proportionate reproduction (i.e., 10%) and crossover (i.e., 90%).

From the figure (4), we see the probability of success for the solution with mutation is higher than the probability of success for the solution without mutation. The different is very clear for the all problems. Therefore in the proposed system we used structure preserved mutation to save the syntactic structure for the chromosome and to improve the work of proposed system.

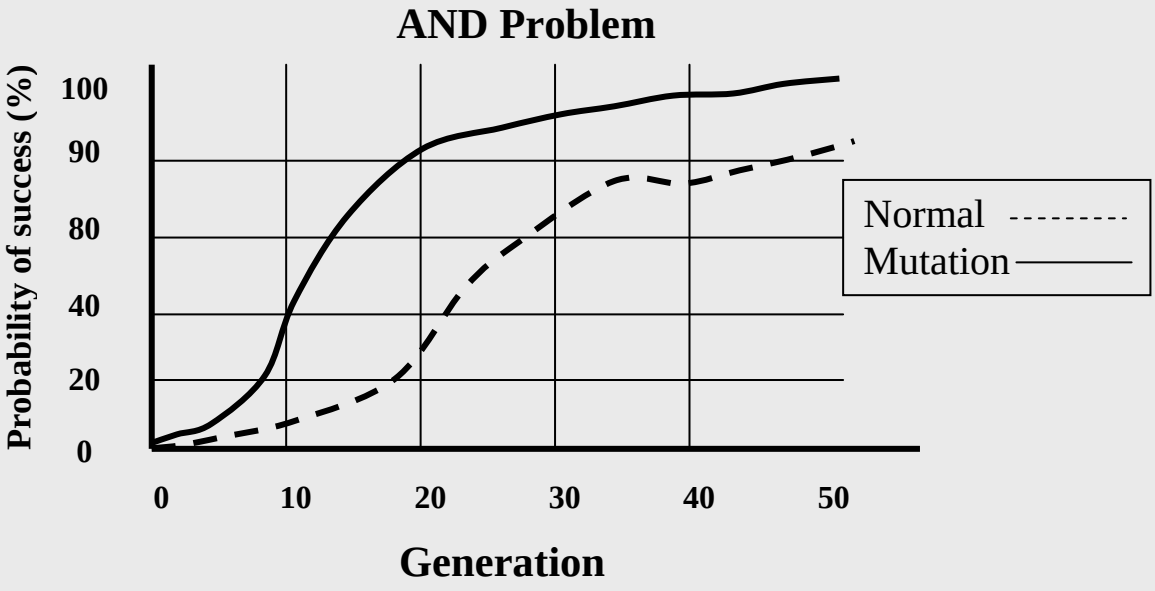


Figure (4): Effect of the mutation operation in AND problem.

4.3 Effect of permutation operation (SPP):

Also, the permutation is one of the secondary operations of genetic programming. It is a generalization of the inversion operation for the conventional genetic algorithm operating on strings.

Form the figure (5) we see that the probability of success for the solution with permutation is higher than the probability for the solution without permutation. The differences between them not clear. In general the effect of permutation becomes very clear when we solve a problem with tree of many branches.

The problems were used in this test represent a simple problems that have small trees. The proposed system designed to construct ANNs varying in shape and size, therefore a structure-preserving permutation was used.

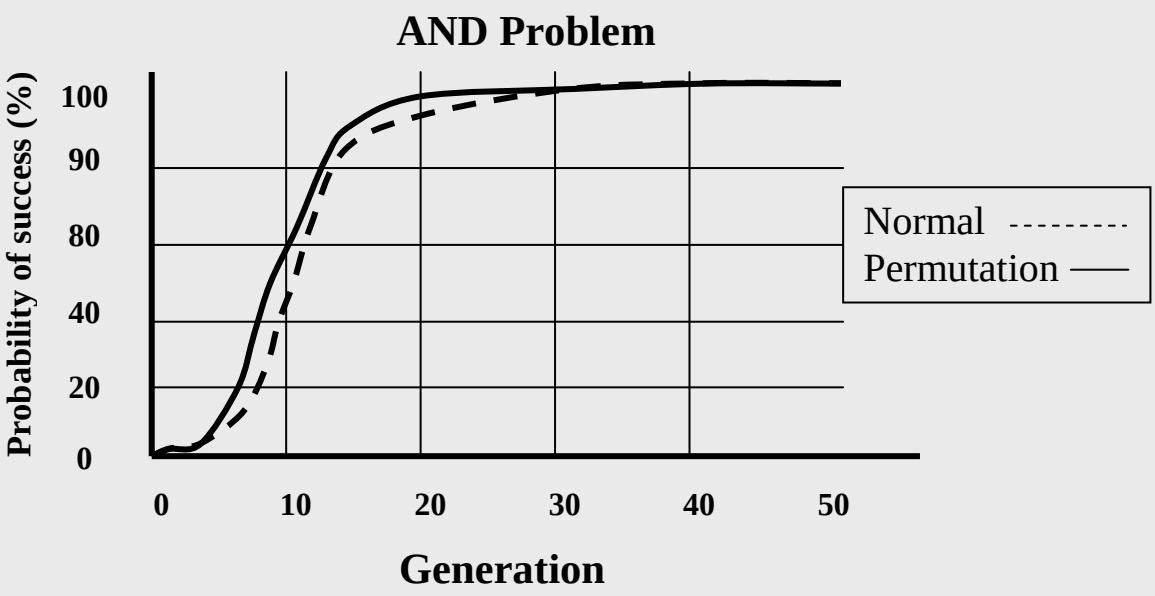


Figure (5): Effect of the permutation operation in AND problem.

4.4 Effect of crossover operation (SPC):

Crossover is one of the primary modification operations of the GP. In this section we test the effect of the crossover operation by selecting different values of probability of crossover.

The tables below show the effect of changing the probability of the crossover on the work of the proposed system to find the solution for the specified problem.

From the table (3) we see that in all problems the proposed system with high probability of crossover gets the solution faster than with low value of probability of crossover (see the average number of generation related with each value of probability of crossover). Therefore a (0.9) is the value that selected for the probability of crossover in the proposed system.

Probability of crossover	Average number of generation
0.5	13
0.6	6
0.7	3
0.8	1
0.9	1

Table (3): Effect of the crossover operation on AND problem

4.5 Compares the result of the proposed system with ANN (Perceptron, Back-Propagation, and Adaline).

In this section we solved five simple problems which are (NOT, OR, AND, XOR, Half-Adder) using the Proposed System, Single Layer Perceptron, Back-Propagation, and Adaline then we compared the results in terms of number of generations and time of execution as shown in the tables in the next page.

Where:

H = hours,

M = Minutes,

S = Second,

PS = (1/100) second,

A = number of input neuron,

B = number of neuron in hidden layer,

C = number of neuron in output layer.

Note that the symbol (X) in execution time and iteration fields in tables means that these ANN can not solve this problem.

All the test of this section was implemented on IBM PC with Pentium 166 MHz MMX processor.

Application	No. of neurons	Execution time H:M:S:PS	Average No. of generations
NOT	**1**	**0:0:0:3**	**1**
OR	**1**	**0:0:0:5**	**1**
AND	**1**	**0:0:0:5**	**1**
XOR	**3**	**0:0:7:6**	**10**
Half-Adder	**3**	**0:0:9:8**	**20**

Table (4): Results of the proposed system for solving specified problems.

Application	A	C	Execution time H:M:S:PS	Average No. of iteration
NOT	**1**	**1**	**0:0:0:5**	**2**
OR	**2**	**1**	**0:0:0:5**	**2**
AND	**2**	**1**	**0:0:0:5**	**3**
XOR	**2**	**1**	**X**	**X**
Half-Adder	**2**	**2**	**X**	**X**

Table (5): Results of the Perceptron for solving specified problems.

Application	A	B	C	Execution time H:M:S:PS	Average No. of iteration
NOT	**2**	**2**	**1**	**0:0:2:3**	**7**
OR	**2**	**3**	**1**	**0:0:6:5**	**27**
AND	**2**	**3**	**1**	**0:0:8:22**	**34**
XOR	**2**	**3**	**1**	**0:0:11:3**	**42**
Half-Adder	**2**	**3**	**2**	**0:0:13:7**	**57**

Table (6): Results of the Back-Propagation for solving specified problems.

Application	A	C	Execution time H:M:S:PS	Average No. of iteration
NOT	1	1	0:0:0:95	2
OR	2	1	0:0:1:2	3
AND	2	1	0:0:1:10	4
XOR	2	1	X	X
Half-Adder	2	2	X	X

Table (7): Results of the Adaline for solving specified problems.

From the above tables we obtained that the proposed system and Back-Propagation were solved all problems, while Perceptron and Adaline were failed to solve (XOR and Half-Adder).

The proposed system solved the all the problems faster than each of the other ANNs. As we saw in previous tables (in execution time and iteration).

4.6 Comparison between Koza's work and the proposed system:

Finally, we compare the result of the proposed system with Koza's work for solving Half-Adder problem. As we see in table (8) the proposed system constructs the ANN for Half-Adder in generation 20 as average of results obtained from 50 runs, while Koza find the result in generation 31 [3]. See table (8). This because proposed system used two additional modifications which are (Mutation, and permutation) these operations makes proposed system faster than Koza work. These operations have clear effects on the work of the proposed system as we saw in the previous tests.

	Proposed system	Koza Work
Population Size (M)	500	500
Max Generation	50	50
Crossover rate	0.9	0.9
Mutation rate	0.1	-
Permutation rate	0.1	-
Random constant (n) range	-2000 and +2000	-2000 and +2000
Generation number that system found the solution on it.	20	31

Table (8): Result of proposed system and Koza Work to solve Half-Adder problem.

5. Conclusions

1-The construction of ANN using this proposed system requires entering all input patterns and output patterns at once.

2-The S-expression is saved in the individual as a string or array of characters, but this string has limited size.

3-We used a file data structure to save the population, as it has unlimited size, but the operations of loading and saving the population in the file makes GP work slower.

4-The proposed system has syntax structure for representing non-recurrent supervised ANN only.

5-From solving the five simple problems (NOT, OR, AND, Exclusive-OR, Full-Adder), we see that the proposed system get the solution faster than ANNs (Perceptron, Back-Propagation, and Adaline).
6-Some of the problems cannot be solved by the proposed system without using the Bias function like (NOT problem).

البناء الألي للشبكات العصبية العامة ذات التغذية الأمامية

بأستخدام البرمجة الجينية

*نورا احمد مولى الساعدي

* قسم علوم الحاسبات كلية الرافدين الجامعة

الخلاصة

الشبكات العصبية الاصطناعية هي عبارة عن نموذج تم دراسته خلال سنين طويلة أملاً للوصول به لكفاءة الإنسان ، و تم استخدام هذا النموذج في الكثير من التطبيقات مثلاً (تمييز الأنماط ، معالجة الإشارات ، الرؤيا في الحاسبات ، تمييز الأصوات ، أنظمة اتخاذ القرارات ، و في الإنسان الآلي).

يوجد هناك أنواع كثيرة من الشبكات العصبية الاصطناعية و لكن أكثر أنواعها يعاني من بعض المشاكل مثل مشكلة التقرب للحل و مشكلة التصميم و بعض هذه الشبكات لا يمكن استخدامها لحل المشاكل اللاخطية.

لكون البرمجة الجينية تمثل ماكنة تعلم تستخدم في البناء الآلي لبرامج الحاسوب لذلك قمنا باستخدامها كنظام آلي لتصميم و بناء الشبكات العصبية الاصطناعية لحل المشاكل السابقة و التي ذكرت اعلاه.

في هذا البحث تم اقتراح وسيلة جديدة لبناء الشبكات العصبية الاصطناعية معتمدةً على استخدام البرمجة الجينية لتعديل كل من معمارية و أوزان الشبكات العصبية الاصطناعية في نفس الوقت.

تم اقتراح عمليات جديدة تخص البرمجة الجينية في هذا البحث و هي (Structure-preserving Crossover) و (Structure-preserving Mutation) و (Structure-preserving Permutation) . و قد اتضح تأثيرها الإيجابي في عملية تحسين نتائج النظام المقترح.

تم استخدام النظام المقترح لحل خمسة مشاكل و هي (AND, OR, NOT, XOR, HALF-ADDER) بالإضافة إلى ذلك تمت مقارنة النتائج المستحصلة من النظام مع نتائج عمل (Koza) و مع ثلاث أنواع رئيسية من الشبكات العصبية الاصطناعية و هي (Perceptron, Back-Propagation, and Adaline).

References

[1] – Abdul – Wahab R. S., "**CGADS (Constrained GADS) AND ADGADS (Automatically Defined Function with GADS). New Development In GADS (Genetic Algorithm for Developing Software**)", MSc. Thesis , Department of Computer Science , Univ. of Technology , 2000 .

[2] – Awad W.S.,"**Cipher Text Only Attack Using Genetic Programming**", Ph.D. Thesis , Department of Computer Science , Univ. of Technology , 1998.

[3] – Koza J. R., "**Genetic Programming on The Programming of Computers By Means of Natural Selection**", A Bradford Book, The MIT Press Cambridge, Massachusetts London, England, 1992.

[4]– Fausett L., "**Fundamentals of Neural Networks, Architectures, Algorithms, And Applications**", Prentice Hall International, Inc.,1994.

[5] – Caudill M., "**Neural Networks Primer**", AI Expert, P:61 , August 1988.

[6] – Chitra S. P., "**Use Neural Networks for Problem Solving**", Chemical Engineering Progress, April 1993.

[7] - Jones W. P., and Hoskins J., "**Back Propagation – A Generalized Delta Learning Rule.**", BYTE Magazine, P:135, October 1987.

[8] – Al - Moli A.F. , "**Image Compression Using Backpropagation Neural Network**", M.Sc. Thesis , Department of Computer Science , Univ. of Technology , 1995.

[9] – Davalo E., and Naim P., "**Neural Networks**", Machllan, 1991.

[10] – Hadi A. S., "**Hybrid Method (GA – BP) for Designing And Learning The Error Back-Propagation Network with Principle of Weight Decay**", M. Sc. Thesis, Department of Computer Science, Science College, University of Babylon 2000.

[11] – Jain A. K., & Mao J., "**Artificial Neural Networks: A Tutorial**", IEEE on Computer, P:31 – 44, March, 1996.