On Applying an Evolutionary Engineering Method to Evolve a Neural Net XOR System

A. Lehireche, A. Rahmoun *
Evolutionary Engineering and Distributed Information Systems Laboratory, EEDIS
Computer Science Department, University Djilali Liabès of Sidi Bel-Abbès, Algeria
* College of Planning & Management, King Faisal University
Al_Hassa Kingdom of Saudi Arabia

Abstract

Evolutionary Engineering (EE) challenge is to prove that it is possible to build systems without going through any design process.

Evolutionary Engineering is defined to be "the art of using evolutionary algorithms approach such as genetic algorithms to build complex systems" .

In this paper, we attempt to solve the neural net XOR problem through using a method that relies on evolving neural structures and based on genetic techniques. A formal EE method had been proposed in the past .

Our main purpose is to show that the EE-Method steps are highly relevant, and that the evolving principle is effective. We had implemented software using the EE concepts to build/evolve a neural net that solve the XOR problem.

Results are prominent and show clearly that the proposed EE method can be easily extended to any type of neural network.

Further works may emphasize on how this method would be effective when size and complexity of the system to be designed (evolved) increase.

Key words: Evolutionary Engineering, Genetic Algorithms, EE-Method, XOR problem, Neural Net, Evolve.

1. Introduction

Computer science theory is based on two major concepts: computability and complexity. The computability concept deals with the fact that problem solving induces the existence of algorithms. Furthermore, computer scientists may analyse problems and need strong capabilities to design the desired algorithms. It is evident that algorithm design (the solution) is the result of the problem analysis. A highly relevant question might be: *is it possible to solve a problem without going through any design process ?*

Evolutionary Engineering (EE) is a discipline of engineering soft computing. EE aims to solve the problem of building complex systems without going through any design process (N.J.Macias, 1999),(H.De Garis, 1993),(W.B.Langton et al, 1996).
EE is defined to be "the art of using evolutionary algorithms approach such as genetic algorithms (D.E.Goldberg,1989) to build complex systems"(H.De Garis , 1993).

Essentially, by imitating nature, the Evolutionary Engineering scientists describe an elementary structure of the system and then evolve this structure toward the desired system.. The Genetic Algorithm is a key-tool for evolving such huge systems.

However, one may keep in mind that EE design might start from scratch, only prior knowledge, learning techniques, and a powerful physical (or logical) machine support and devices are necessary to make grow a particular application.

The EE-Method (A.Lehireche et al, 2001) intends to guide EE designer along the design process to achieve and implement a particular application. We carried out several experiments in the EEDIS laboratory applying the EE-Method (A.Lehireche et al, 2001) on several different applications so to test its efficiency.

In this paper, we intend to apply and test the EE-Method on the well-known XOR problem. EE-Method is carried out to design or build a Neural Network XOR relying on an evolving algorithm.
Simulation results show that the proposed EE-Method is highly relevant in evolving small systems.

In section 2 we report the EE-Method as specified in (A.Lehireche et al, 2001), sections 3,4,5,6,7,8 are the application of the related method to the neural net XOR problem, section 9 yield the implementation results and section 10 concludes this paper.

2. Evolutionary Engineering Method (EE-Method)

The EE-Method (A.Lehireche et al, 2001) aspires to explicit and simplifies the EE approach. It also tries to enlarge the scope of Ggenetic Programming. Six steps make up the EE-Method. Each step is a vital phase. The step order

ensures the coherence of the approach but it is not unique (A.Lehireche et al, 2001), (A.Lehireche et al, 2000).

Given a Complex system:

Step 1: ***Ensure the Availability Of***

- The Inputs (data, parameters, or values),
- The outputs, and
- The Input/Output relationships: this means that we are able to express any output in terms of input of the desired system.

Step 2: ***Choose a Model***

Choose a model with which the desired system should be implemented, such as:

- Neural Networks,
- Automata,
- Petri Nets,
- Electronic Circuits,
- Graphs,
- Programs,

etc…

Step 3: ***Choose a System Genotype***

Use the Genetic Programming (GP) techniques to encode the model. This step produces the structure of a Chromosome. A chromosome must encode all the system. The chromosome is the genotype form of the system.

Step 4: ***Determine The Adaptation Function of the System***

The adaptation function is a measure that points out to the system evolution degree during the evolving phase. In general, this represents the measure of the input-output association conformity (the fitness or objective function in Genetic Algorithms).

Step 5: ***Choose a System Phenotype***

During the evolving phase, to be able to evaluate the adaptation function of an occurrence of the system; we must proceed as follow:

1. Extract the real characteristics of the occurrence of the system by decoding the chromosome.

2. Implement the system model according to its characteristics.
3. Simulate the behaviour of the system.

The real characteristics of an occurrence of the system are called system phenotype.

Step 6: ***Use A Genetic Algorithm As Follow***

A) Generate randomly a population of chromosomes (each chromosome is a system genotype).
B) For each chromosome do: genotype → phenotype → system implementation,
Start up the system,
Inject inputs, retrieve outputs,
Evaluate the system adaptation function.
C) Select genotypes of the most adapted occurrences of the system.
D) Produce a new generation by applying the crossover and the mutation operators to the selected genotypes.
E) Repeat B), C) and D) until the desired system is reached (with an acceptable fitness).

3. Evolving an Neural Network XOR using the EE-Mehod

3.1 The XOR problem:

The exclusive-OR (XOR) problem is a standard problem and is often used as a test for neural network. It is well known that the XOR is not linearly separable (Yan Le Cun, 1987) and for this reason we consider it as a good test (A.J.F.Van Rooij et al, 1995). The following table depicts the XOR function.

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

3.2 EE-Method Step 1: Problem definition

The following relation expresses the input/output association:

$E = \{0 , 1\}$

$R: E \times E \longrightarrow E$

$R = \{ ((0,0),0) , ((0,1),1) , ((1,0),1) , ((1,1),0) \}$

3.3 EE-Method Step 2: Choosing the model

The model with which we desire to implement our example is the "Neural Networks" (NN). A NN is characterised by its topology and the functionality of the artificial neuron.

- Topology of the NN.

A recurrent NN is selected for this purpose. A recurrent NN is a self fully connected NN. This topology has the advantage that does not deal with any specific details of the NN, such as number of layers, number of neurons in each layer, number of hidden layers, how neurons are connected and so on ...Only the number of neurones has to be chosen. For our example the NN contains 5 neurones. This choice is motivated by our knowledge on the XOR problem. Fig. 1 shows such topology.

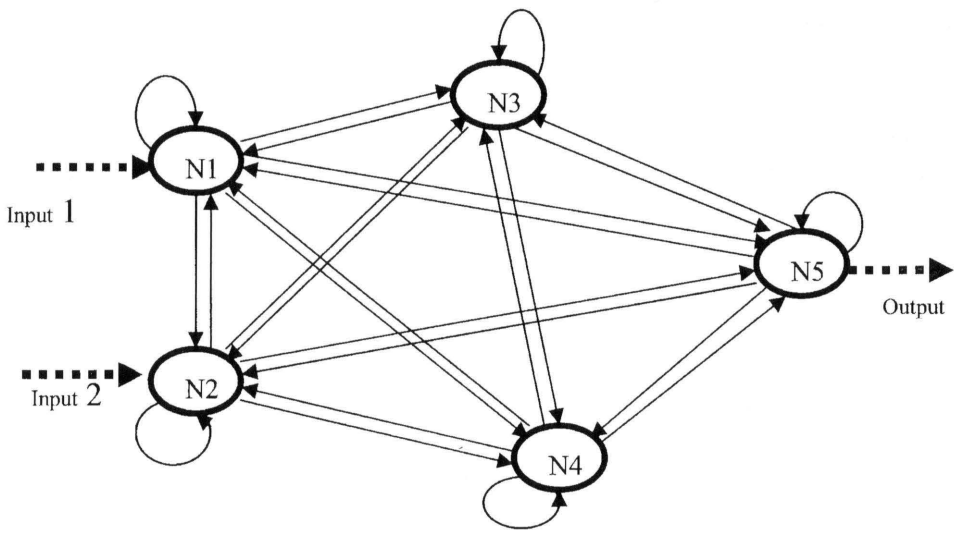


Fig. 1: Recurrent NN (5 neurons)

- Artificial neuron functionality.

Fig. 2 and 3 shows in details the behaviour of the artificial neuron. The neuron transfer function is a sigmoid function. External inputs are used to control the NN (H.De Garis , 1993). We use them to direct input data into the NN.

Sj ≡ input signal "j".
Wji ≡ weight associated to Sj for the neuron "i".
Ei ≡ external input of the neuron "i"; when used its weight is set to 1.0 if not to 0.0.

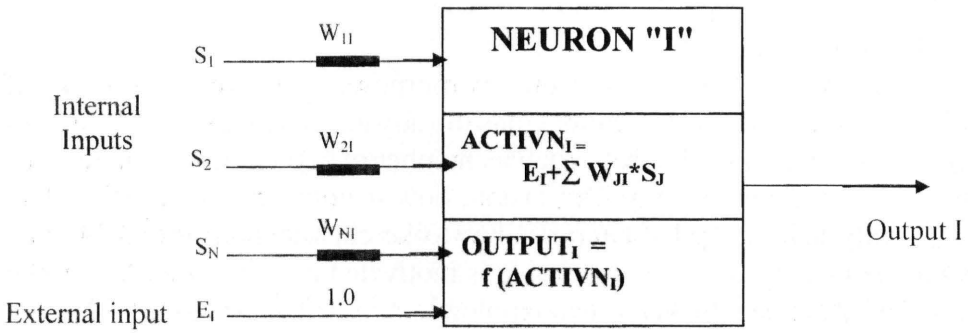


Fig. 2: Artificial Neuron as a single node

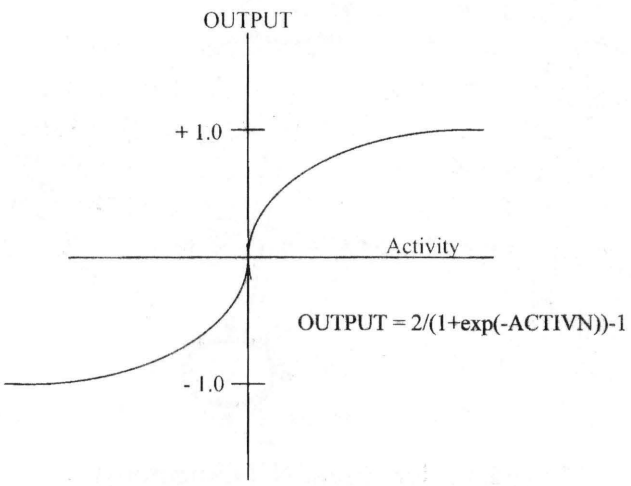


Fig. 3: Neuron's output function

3.4 EE-Method Step 3: System genotype

The set of weights determines fully the behaviour of the recurrent NN; then the chromosome witch represents the system genotype is simply: the set of coded weights.

Fig. 4 describes the chromosome structure.

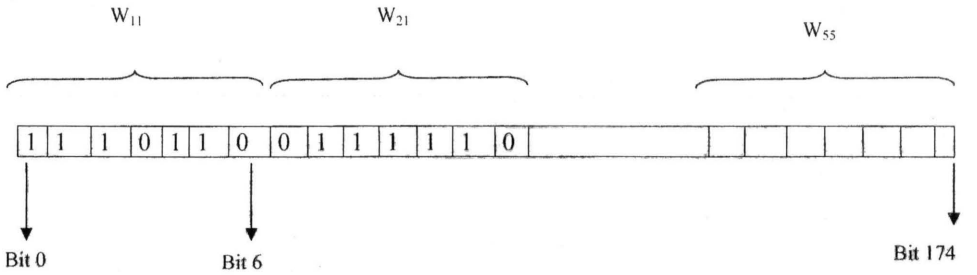


Fig.4: Chromosome structure

Wji denotes the weight associated to the signal Sj comming from neuron j into neuron i.

Each weight is coded with 7 bits and has its value in the range [-1,+1].

The chromsome legth is 5*5*7=175 bits.

In fig.4 W_{11} is interpreted as follows:

Bit 0 = 1 $\Rightarrow$ weight is a negative value.

Bit 1 = 1 $\Rightarrow$ weight = weight + 1* 2^{-1} = weight + 1*0.5.

Bit 2 = 1 $\Rightarrow$ weight = weight + 1* 2^{-2} = weight + 1*0.25.

Bit 3 = 1 $\Rightarrow$ weight = weight + 0* 2^{-3} = weight + 0*0.125.

Bit 4 = 1 $\Rightarrow$ weight = weight + 1* 2^{-4} = weight + 1*0.0625.

Bit 5 = 1 $\Rightarrow$ weight = weight + 1* 2^{-5} = weight + 1*0.03125.

Bit 6 = 1 $\Rightarrow$ weight = weight + 0* 2^{-6} = weight + 0*0.015625.

So:

W_{11} = -(0.5 + 0.25 + 0.0625 + 0.03125) = -0.84375.

3.5 EE-Method Step 4: The Adaptation Function of the System

During the evolution phase, for a system in test, we note :

- A_i : the actual output for a set i of input data .
- D_i : the desired ouput for a set i of input data.
- Fitness : the adaptation function.

The adaptation function (i.e. the fitness) is the distance (i.e the gap) between the actual outputs and the desired outputs. The fitness is not an absolute measure but is subject to the way we compute it. The fitness formula influences strongly this measure.

For the XOR the fitness is expressed as follow:

$SSD = \sum (D_i - A_i)^2, (i=1,4)$
Fitness = if (SSD > 1) then 1/SSD else 1 – SSD.

- Example:

Input 1	Input 2	Desired output (Di)	Actual output (Ai)
0	0	0	0.43
0	1	1	0.25
1	0	1	0.12
1	1	0	0.03

Given the above data, the fitness is computed as follows:

$SSD = \sum (D_i - A_i)^2, (i=1,4) = (0-0.43)^2 + (1-0.25)^2 + (1-0.12)^2 + (0-0.23)^2$
$= 0.1849 + 0.5625 + 0.7744 + 0.0529$
$= 1.5747$

Fitness = 1/ SSD = 1/ 1.5747 = **0.635**
This result means that the gap between the system in evolution and the desired system is **.365**.

3.6 EE-Method Step 5: System Phenotype

In this step the evolutionary engineer have to make choices on how his system must be implemented. In our case we have to implement (simulate) a recurrent NN.

The fact that we are evolving systems by mean of their genotype form, we need to decode each chromosome to obtain the recurrent NN weights real values. These values are stored in a "weight table" (Fig. 5). The external input data are stored separately in vector noted "E" and for each neuron the computed output signal is stored in a vector noted S.

The weight table, the vectors E and S are the phenotype form of the system in evolution.

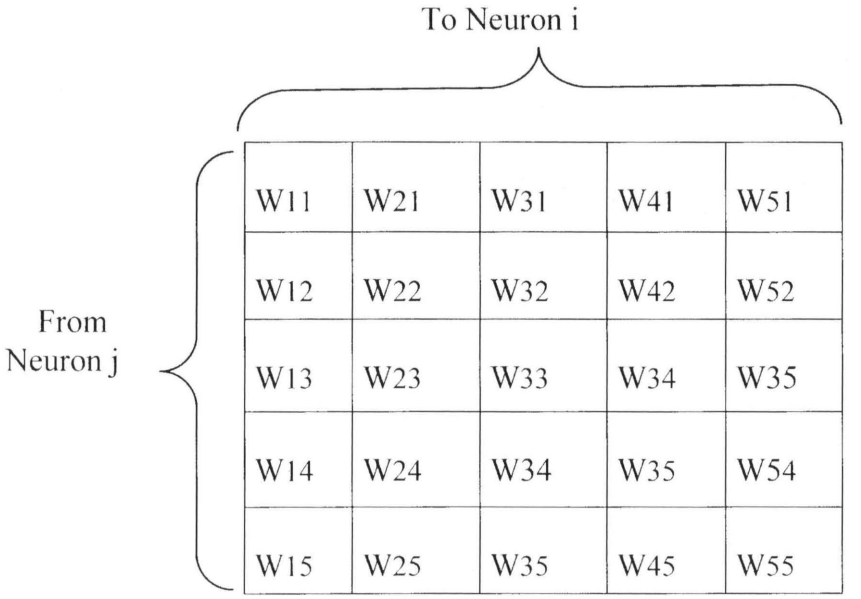


Fig. 5: The Weight Table

- The recurrent NN simulation algorithm:

Given the weight table W, the external input data vector E and the output signal S the recurrent NN simulation program is as follows:

```
Loop
 {
   For i= 1 to neuron number          // compute the output signal for each neuron.
    {
      For j=1 to neuron number
      {ACTIVI = ACTIVI + Wji * Sj}   // compute the activity of neuron i (Fig. 2).
      ACTIVI=ACTIVI + Ei;             // inject input data to neurone i (Fig. 2).
      Si = 2 / (1 + exp (-ACTIVI))-1  // apply the output function f  (Fig. 3).
    }
 } Until Stability                   //the recurrent NN is activated many times to reach
                                     //the stability (equilibrium state).
```

3.7 EE-Method Step 6: Evolution phase

The evolution phase is an operational phase; the evolutionary engineer must implement the overall software, taking into account all the decision made in steps 1 to 5.
Fig. 6 describes the architecture of such software.

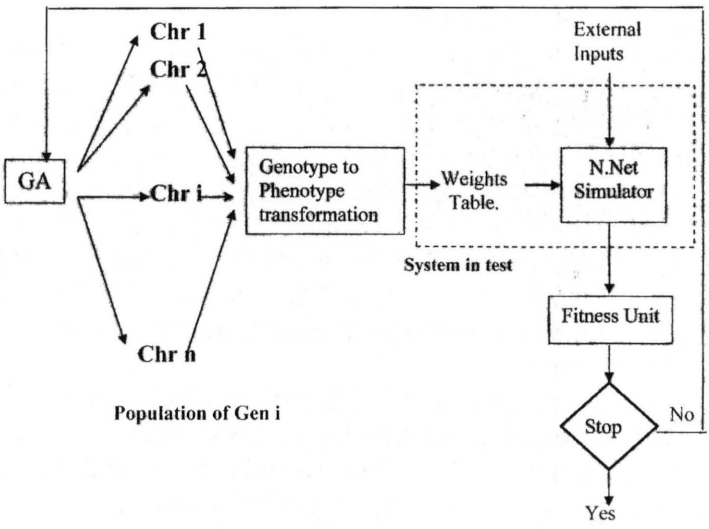


Fig. 6: Evolution scheme

- Evolution Software Parameters for the XOR :

Number of Neurons	5,
Number of input neurons	2,
Number of output neurones	1,
Weight code length	7 bits,
Chromosome length	175 bits,
Numbers of cycle to reach the NN stability	100,
Type of crossover	Uniform crossover,
Crossover probability	0.6,
Mutation probability	0.001,
Selection strategy	Roulette wheel,
Evolution strategy	Elitism,
Scaling constant	2.0,
Population size	100,
Number of generation	until fitness > 0.99

4. Experimental results

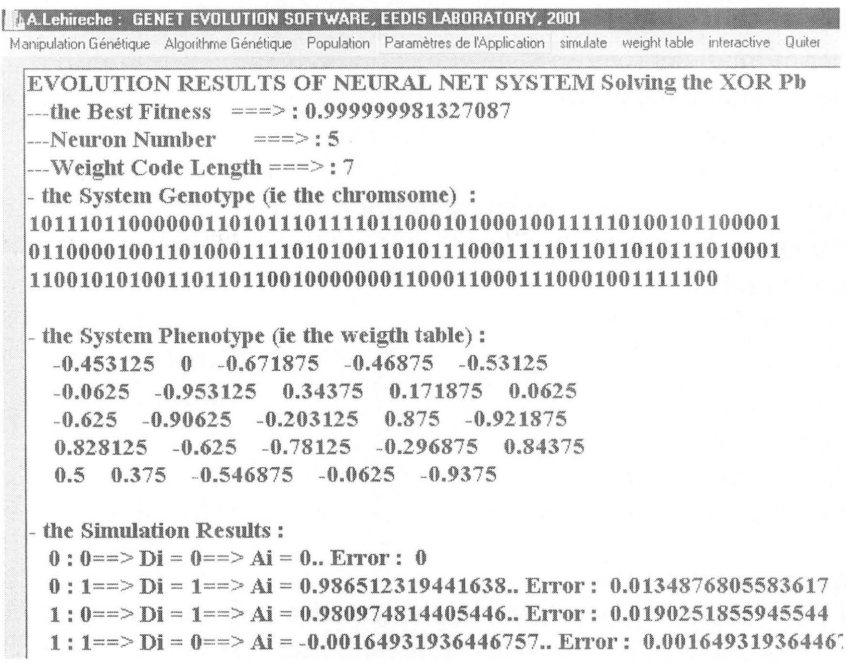
In EEDIS laboratory, we have implemented a software; namely GES or Genet Evolution Software. This software uses evolutionary engineering concepts as specified by the EE-Method (A.Lehireche et al, 2001).

Notice that a similar work has been presented by G.A.Jayalakshmi et al. They performed almost the same experiment (The XOR) with a totally different approach in a sense that the chromosome structure is different, and the evaluation scheme is different also.

In their work, the selection procedure and the algorithm termination criteria lead to a premature convergence (within four generations!) . In such case, the evolved architecture does not converge to a global optimum (best topology) as pretended in the paper.

Our paper, however, proposes a different method that evolves the system topology (as mentioned in "CAM Brain project" (H.De Garis)), and takes into consideration the best-fit GA parameters, and therefore led to fast GA convergence. Such GA parameters are difficult to establish at first glance. These have been set according to several simulation tests so to avoid premature convergence.

Notice also that the same method (EE method) had been used to evolve a real-time neural network controller for a robot using the Genet Evolution Software (A.Lehireche et al, 2003). Results show that the evolved NN controller performs as good as a classical PID controller.



5 . Conclusion

Evolutionary Engineering creates an elementary structure of the system and then evolves this structure toward the desired system. Evolutionary Algorithms are powerful tools for evolving such huge systems. Such process relies on observing and imitating natural systems. EE-Method intends to guide EE designer along the design process to achieve and implement a particular application. This paper shows, step by step, how to apply the EE-method to a specific example: the neural net XOR problem. The results yield in section 8 are significant. We had repeated the experiment on several different Neural Networks configurations to evolve an NN XOR system.

Evolutionary Engineering techniques are proven in the recent past to be efficient methods to evolve systems. They have also capabilities of generating several different solutions on several runs for the same problem.

Furthermore, further works and investigations must emphasize on evolving more complex systems to test to what extend the EE method would be reliable and exhaustive.

6. References

1. A J F Van Rooij, L C Jain, R P Johnson, "*Neural Network Training Using Genetic Algorithms*", World Scientific, Series In Machine Perception & Artificial Intelligence, Vol.26.
2. Lehireche, A. Rahmoun and A. Gafour: "Highlights the Evolutionary Engineering Approach: the EE-Method", ACS/IEEE International conference on computer systems and applications (AICCSA 2001), Beirut, 0-7695-1165-1/01, pp5-12, Copyright 2001 IEEE.
3. Lehireche, A. Rahmoun, "*Evolutionary Engineering Approach: the EE-Method*", lecture notes, EDDIS lab, Dept of computer science, UDL Sidi Bel Abbes, Algeria, 04/ 2000.
4. Lehireche, A. Rahmoun, *"Real time Evolutionary Engineering in Tracking Problems: Evolving a Real-Time "Track and Evolve" neural Network for a Robot"*, lecture notes, EDDIS lab, Dept of computer science, UDL Sidi Bel Abbes, Algeria, 02/ 2003.
5. D.E. Goldberg, "*Genetic Algorithms in Search, Optimisation, and Machine Learning*", Addison –Wesley Publishing Company, 1989.
6. Hugo de Garis ,Michael Korkin , Felix Gers , Norberto Eiji Nawa , MichaelHough , "CAM-Brain, Atr's Artificial Brain Project An Overview", Brain Builder Group, Evolutionary Systems Department, ATR Human Information Processing Research Labs, September 1998.
7. Hugo de Garis , Genetic Programming: GenNets, Artificial Nervous Systems, Artificial Embryos, PHD thesis 1993.
8. G.A.Jayalakshmi et al, An Evolutionary Programming Approach to Evolve The Architecture of Artificial Neural Networks.
9. Nicholas J.Macias, "Ring Around the PIG: A Parallel GA with only local interactions coupled with a Self-Reconfigurable Hardware Platform to implement an O(1) Evolutionary Cycle For Evolvable Hardware", Proceeding of 1999 Congress on Evolutionary Computation, Copyright 1999 IEEE.
10. William B. Langton, Adil Qureshi, "*Genetic Programming, Computers Using 'Natural Selection' To Generate Programms*", Lecture Notes of a survey, Dept Of Computer Science, University College London, 1996.
11. W.B.Langdon, Genetic Programming Bibliography, Revision Date: 2001/05/05, W.B.Langdon@cwi.nl,ftp://ftp.cs.bham.ac.uk/pub/authors/W.B.Langdon/biblio /gpsubmit.html, 2001.
12. Yan Le Cun, Modeles Connexionnistes de l'Apprentissage,These de Doctorat, Universite Paris 6,1987

تطبيقات خاصة في الهندسة التطورية: تطور ذاتي لنظم شبكات عصبية اصطناعية

أحمد الحيرش و عبد اللطيف رحمون *

مختبر نظم المعلومات الموزعة والهندسة التطويرية – قسم حاسب آلي

جامعة الجلالي اليابس – سيدي بلعباس – الجزائر

* كلية العلوم الإدارية والتخطيط – جامعة الملك فيصل

الأحساء – المملكة العربية السعودية

الملخص:

من تحديات الهندسة التطورية ابراز قدراتها على بناء نظم بدون حاجة الى تصميم. الهندسة التطورية هو علم جديد في مجال الذكاء الصناعي و يعرف أيضا بـ "فن استعمال الخوارزميات التطورية كالخوارزميات الجينية لبناء نظم كبيرة و معقدة".

في هذا البحث تم إجراء محاولة حل مسألة XOR بالشبكات الصناعية باستعمال طريقة تعتمد على تطوير هياكل عصبية و تقنيات جينية. قد اقترحنا في بحث سابق طريقة تطورية و أشرنا الى فعالية الميكانزمات التطورية. لقد تم الاعتماد على هذه الميكانزمات التطورية في هذا البحث لتطوير برامج لبناء شبكات عصبية لحل مشكلة XOR. من خلال هذه التجارب تم التركيز على قدرة الطريقة المقترحة على تصميم شبكات عصبية فعالة، كما أنه يمكن تعميمها الى نظم وأنماط عصبية أخرى.

الكلمات الأساسية:

الهندسة التطورية، الخوارزميات الجينية، طريقة تطورية، مسألة XOR ، الشبكات العصبية.