[6] D. G. Luenberger, *Linear and Nonlinear Programming*, 1989, New York.
[7] G.V. Reklaitis, A. Ravindran, K. M. Ragsdell, *Engineering Optimization Methods and Applications*, John Wiley & Sons, N.Y., 1983.
[8] R. Fletcher and C.M. Reeves, "Function Minimization by Conjugate Gradients," Comput. J. 7, pp. 149-154, 1964.
[9] G. E. P. Box, "Evolutionary Operation: A Method for Increasing Industrial Productivity," Journal of the Royal Statistical Society, C, 6(2), pp. 81-101, 1957.
[10] L. Davis and M. Steenstrup, "Genetic Algorithms & Simulated Annealing," In GAs & SA, L. Davis, Ed., Pitman, London, UK, pp. 1-11,1987.
[11] S. Kirkpatrick, C.D. Gellat Jo., and M.P. Vecchi, "Optimization by Simulated Annealing," Science, Vol. 220, No. 4598, pp. 671-680, 1983.
[12] J. H. Holland, "Genetic Algorithms and the Optimal Allocations of Trails," SIAM Journal of Computing, 2(2), 88-105, 1973a.
[13] J. H. Holland, *Adaptation In Natural And Artificial Systems*, Ann Arbor, MI: The University of Michigan Press, 1975.
[14] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.
[15] H. Miura, G. N. Vanderplaats, and S. Kodiyalam, " Experiences in Large Scale Structural Design Optimization," Applications of Supper Computer in Engineering: Fluid Flow and Stress Analysis Applications, Elsevier, Amsterdam, 1989.

Therefore, the CGA cannot make one signed fitness function. However the RPLNN solves this minimization problem (see Fig. 12-b). Remember that the minimum point for this problem, as stated above, will occur at the lower limit of x_1, which is dictated by the desired precision (i.e. individual size). This value in the simulation becomes x_1=-31.75 & x_2=-0, because we used individual size of 8 bits, a bit for sign and 5 bits for integer part and 2 bits for decimal part, for each parameter x_1 and x_2. Therefore the minimum point will be (-31.75, 0) as shown in Figure 12-b. To show how fast the RPLNN reaches the minimum point, we performed 40 different runs and the average time of convergent becomes 242 seconds.

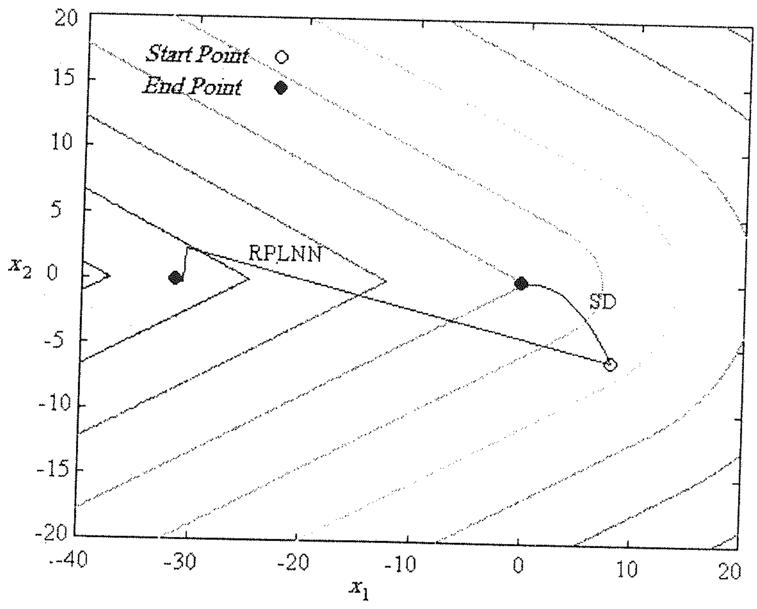


Figure (12) The RPLNN Performance and the SD performance for example 2-iii.

4-Conclusion

In this paper, a new technique, Ring Probabilistic Logic Neural Networks, was proposed for optimization. Its performances have been fully discussed and compared with those of gradient-based and conventional genetic algorithms through several illustrative examples. In all cases the RPLNN converged to the global point with less amount of storage memory and computations time. In contrast, the gradient-based techniques and the CGA fail to reach the optimum point. We should further note that the CGA in some cases (i.e., example 2-*iii*) could not be applied and for the other cases it might converge to the global point with some modifications, but it needs high amount of computational times and storage memory.

References

[1] W. K. Kan and I. Aleksander," A Probabilistic Logic Neural Networks for Associative Learning," Proc. IEEE 1st Int. Conf. On Neural Networks, San Diego, June 1987, Vol. II, pp. 541-548.
[2] I. Aleksander, "The logic connection of system," In I. Aleksander (Ed.), Neural Computing Artuitecturs , pp. 131-155, Cambridge, MA: MIT Press, 1989.
[3] W. K. Kan, K. H. Wong, and H. M. Law, "Non-Overlapped Trees of Probabilistic Logic Neurons," IEEE Region 10 Conf. On Computer and Communication Systems, September 1990, Hong Kong.
[4] B. Zhang, L. Zhang, and H. Zhang, "A Quantitative Analysis of the Behaviors of the PLN Networks," Neural Networks, Vol. 5, pp. 639-644, 1992.
[5] B. Zhang, L. Zhang, and H. Zhang, "The Complexity of Learning in PLN Networks," Neural Networks, Vol. 8, No. 2, pp. 221-228, 1995.

Figure 12 plots this function. It can be easily shown that f is continuously differentiable for $x_1>0$, and in addition, the set of possible discontinuities of the gradient satisfies the condition $x_2=0$ and $x_1\leq 0$. To be sure about it, let us determine the gradient:

$$\frac{\partial f_1}{\partial \vec{x}} = \left[\frac{4\cdot\sqrt{13}\cdot x_1}{\sqrt{4\cdot x_1^2+9\cdot x_2^2}} \quad \frac{9\cdot\sqrt{13}\cdot x_2}{\sqrt{4\cdot x_1^2+9\cdot x_2^2}} \right], \tag{16-a}$$

$$\frac{\partial f_2}{\partial \vec{x}} = [4 \quad 9\cdot Sign(x_2)], \tag{16-b}$$

$$\frac{\partial f_1}{\partial \vec{x}} = [4 \quad 9\cdot Sign(x_2)] \quad for \quad x_1 = |x_2|. \tag{17}$$

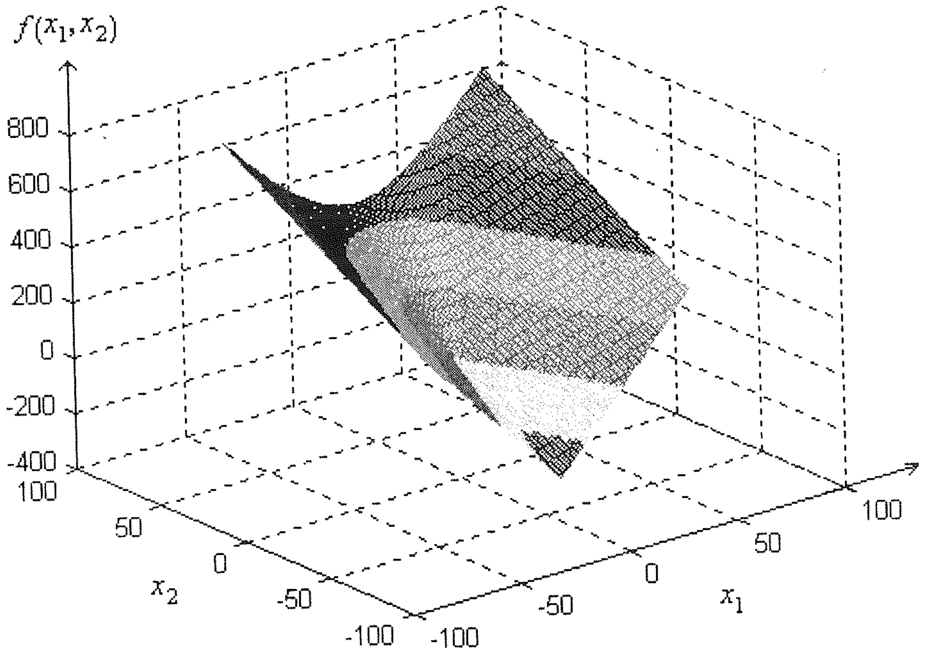
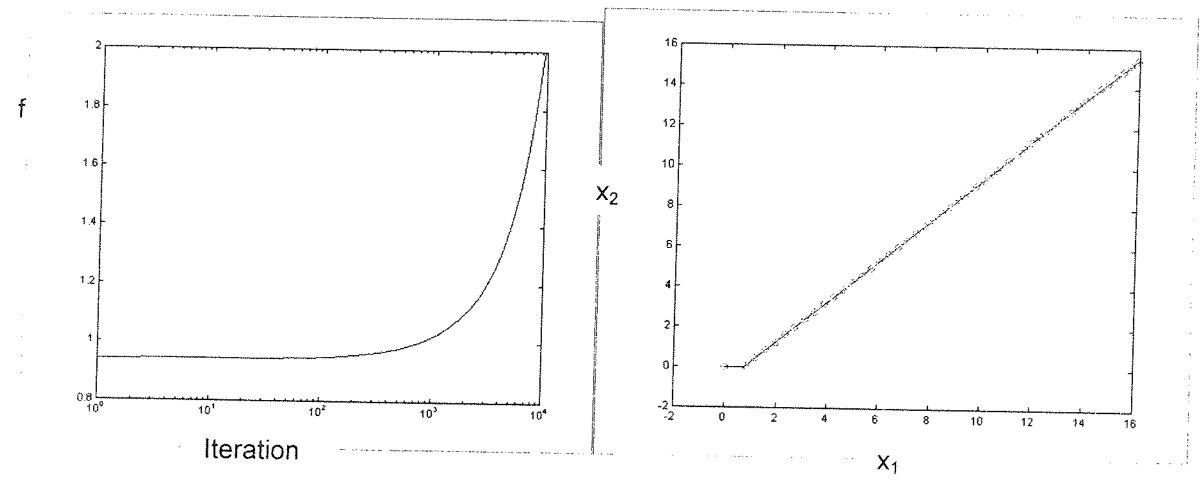


Figure (12) Function of example 2-*iii*

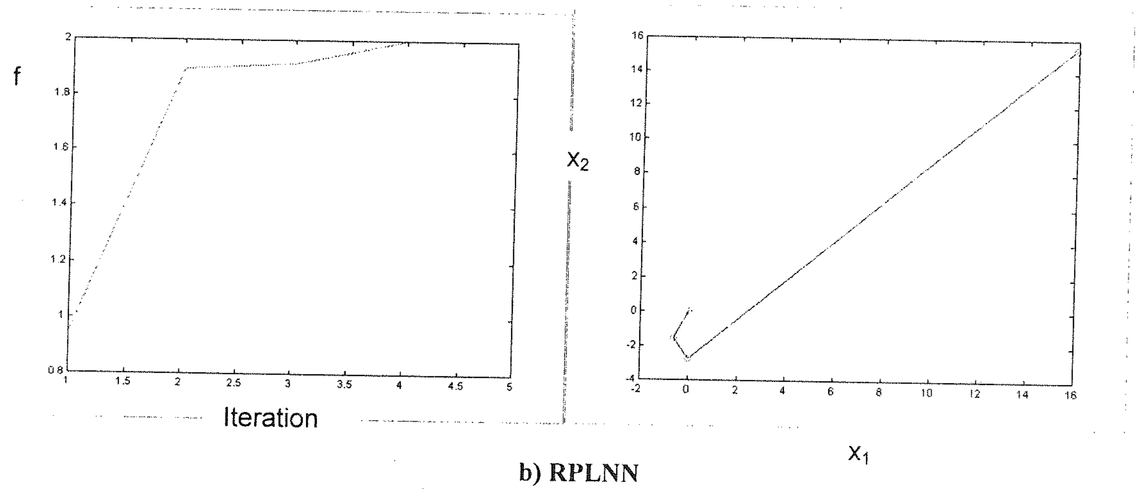
As easily observed, no discontinuity of the gradient on the line $x_1=|x_2|$ exists, therefore, f is continuously differentiable on the half-plane $x_1>0$. In contrast, the gradient is discontinuous on the ray $x_2=0$, $x_1\leq 0$. To show this, let us compute the gradient on this ray:

$$\frac{\partial f_1}{\partial \vec{x}} = [2\cdot\sqrt{13} \quad 0], \quad \frac{\partial f_2}{\partial \vec{x}} = [4 \quad 0]$$

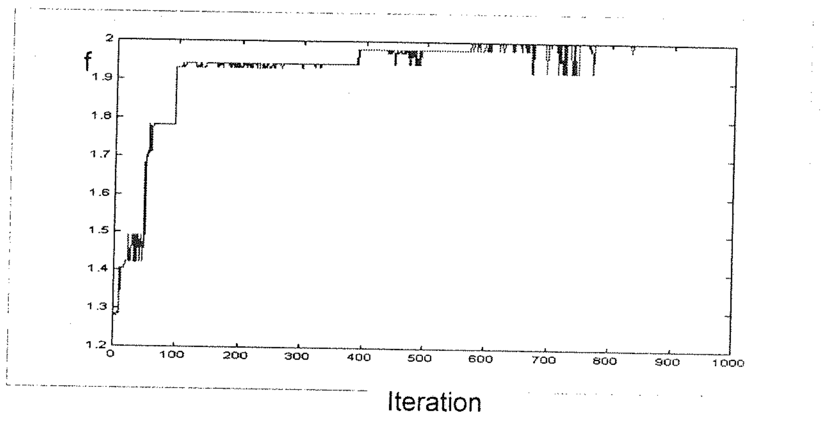
So, we faced with a non-differentiable convex function. We may further see that the limit of f on this ray becomes $-\infty$ as x_1 goes to $-\infty$. But, if we choose a starting point $(x_1, x_2)=(8, -6)$ then, as depicted in Figure 12-a, the method of SD converges to the point $(x_1, x_2)=(0, 0)$. Note that the origin is a non-stationary point for this example; this contradicts the fact that for continuously differentiable functions the set of accumulation points of a segment generated by the method of SD is not well suited for minimizing non-differentiable convex functions. The contrast of this example is that the CGA cannot be applied for minimizing this sort of functions, because f is defined in the above and below of horizontal coordinate surface.



a) SD Performance



b) RPLNN

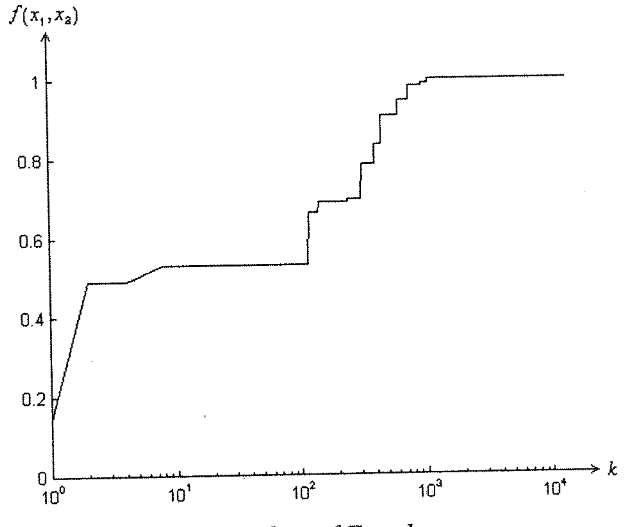


c) CGA Performance
Figure (11) SD, CGA and RPLNN for example 2-*ii*.

iii- A Function with Discontinuity in its Gradient

Consider the third example that is a function of two variables

$$f(x_1,x_2)=\begin{cases} f_1(x_1,x_2)=\sqrt{13}\cdot\sqrt{(2\cdot x_1)^2+(3\cdot x_2)^2} & x_1>|x_2| \\ f_2(x_1,x_2)=4\cdot x_1+9\cdot|x_2| & x_1\le|x_2| \end{cases} \qquad (15)$$



Number of Epochs

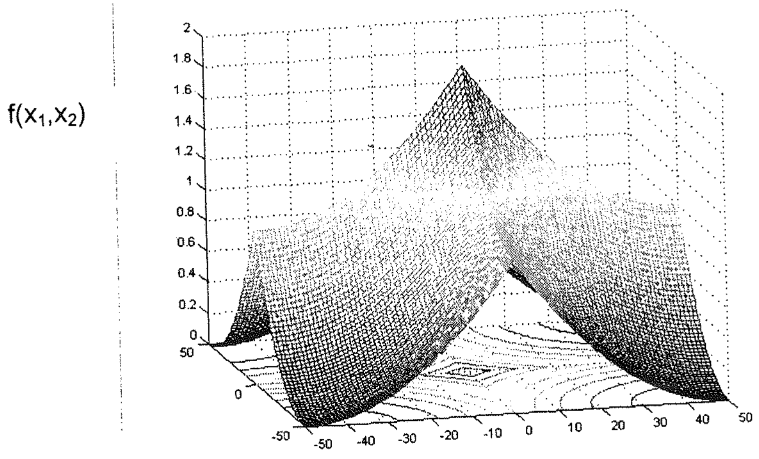
Figure (9) Conventional Genetic Algorithms Performance (Function Values at each *k*).

ii- A non-smooth function

The second function optimization problem is the maximization of the following function, which is non-smooth along the coordinate axis.

$$f(x_1,x_2)=\left(0.02|x_1|-1\right)^2+\left(0.02|x_2|-1\right)^2 \tag{14}$$

Figure 10 shows this function. As seen this function is not differentiable at origin, because its directional derivative is different in each direction. The SD, CGA and RPLNN are applied to find the values of x_1 and x_2 that maximize the function given in Eq. (14). Again to compare the performance of these methods, we used the initial condition (15.94,15.44). Figure 11 summarizes the results. SD reaches the point (.001,.001) which is very close to the optimum point, the origin with total number of flops 249590 and the RPLNN reaches the optimum point (0,0) with number of flops 2 89750. We have also implemented the CGA on this problem. The result is shown in Fig. 11c. After 1000 iterations with population size 60, Pc=.97 and Pm=.01, the final point (0,.01), which is very close to the optimum point, has been reached with the total number of flops 15097968.



Figr :(10) Function of example 2-*ii*.

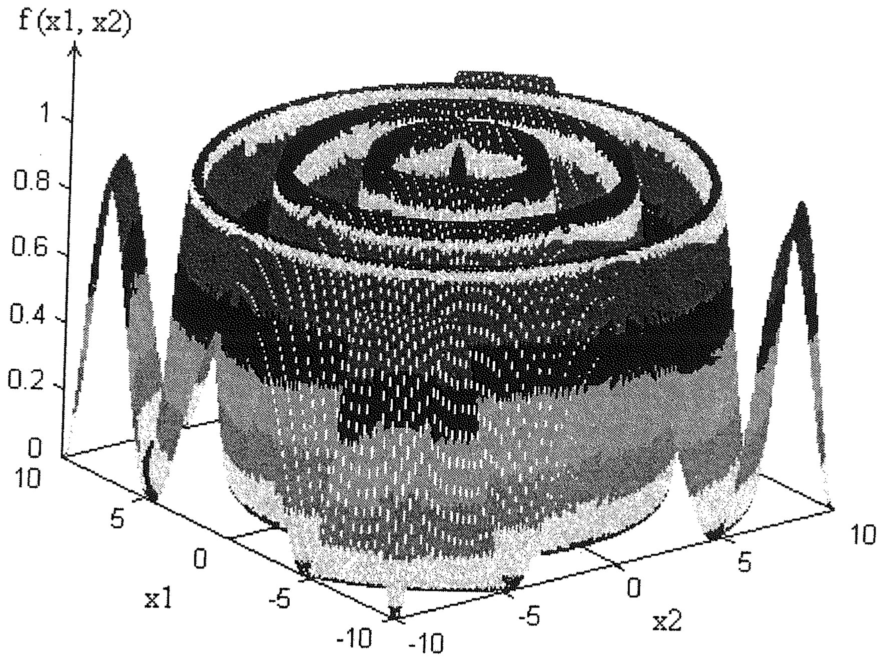


Figure (8) Function of example 2-*I*.

Table (4) Simulation Results for example 2-*I*

	SD	**CG**	**RPLNN**
Initial Point A:(x_1, x_2)	(31.9844 , 31.9844)	(31.9844 , 31.9844)	(31.9844 , 31.9844)
Final Point B:(x_1, x_2)	(31. 0896 , 31.0896)	(31.0976 , 31.0976)	(0 , 0)
Mean Time (Sec.)	Not Converged	Not Converged	10.6340
Initial Point A:(x_1, x_2)	(106562 , 106562)	(10.6562 , 10.6562)	(106562 , 106562)
Final Point B:(x_1, x_2)	(11.0983 , 11.0983)	(11.1060 , 11.1060)	(0 , 0)
Mean Time (Sec.)	Not Converged	Not Converged	12.5755
Initial Point A:(x_1, x_2)	(31.9844 , 10.6562)	(31.9844 , 10.6562)	(31.9844 , 10.6562)
Final Point B:(x_1, x_2)	(32.7708 , 10.9182)	(32.7828 , 10.9222)	(0 , 0)
Mean Time (Sec.)	Not Converged	Not Converged	8.3407
Initial Point A:(x_1, x_2)	(-3.5312 , -3.5312)	(-3.5312 , -3.5312)	(-3.5312 , -3.5312)
Final Point B:(x_1, x_2)	(-4.4386 , -4.4386)	(-4.4424 , -4.4424)	(0 , 0)
Mean Time (Sec.)	Not Converged	Not Converged	9.7083

As we could expect the hill climbing methods starting from an initially point in the two dimensional parameter space, and following the steepest descent gradient directions, get trapped in one of the local optima. In contrast, the CGA and the RPLNN show robust performance. Figure 9 illustrates the performance of the CGA for the function given in Eq.(13). The population size of the CGA is chosen 100 and new population is generated in 10000 epochs with run time of 51754 seconds (about 14 hours). The CGA could reach to point (1.57,2.71) with the maximum value of 0.9902 instead of 1 that is the global maximum. In contrast, the RPLNN reached to the global response and the other methods were trapped in one of the local solutions.

The proposed modified CGA is accomplished in such a way that at first all individuals of the initial population are randomly selected from the allowable solution space. Then, like the CGA discussed in section II, all permissible individuals are evaluated and the process continues until crossover operator is done. When the crossover operator combines two permissible individuals, they may produce an offspring outside the allowable space. In this case, the offspring is omitted and substituted by a random permissible individual from the allowable space. Then, by this injection of random individuals in new population, it is not necessary to use the mutation operator. If all produced offspring by crossover recombination is valid, the mutation operator executes on population as explained in section II.

By this procedure we solved the cantilevered beam problem. The results are listed in Table3. The sub-optimum performance of $V^*=47811$ has been obtained after 3000 revisions on population (3000 epochs) for the optimum design variables as shown in Table 3.

Table 3 also lists the results of the cantilevered beam problem, obtained from the RPLNN. After 3000 iterations, the sub-optimum performance became $V^*=29745$. By comparing these two methods, we can obviously see that the RPLNN converges to a better solution with fewer amounts of computation time and memory locations.

Table (3) Design variable values.

Variables	b_1	b_2	b_3	b_4	b_5	h_1	h_2	h_3	h_4	h_5
CGA	1.125	3.000	1.125	2.250	5.125	59.000	53.500	37.125	44.125	21.500
RPLNN	1.375	8.500	2.250	1.250	2.250	30.875	20.875	10.625	13.000	16.625

Example 2: Some Function Optimization

This section considers the problem of finding the optimum values of three different functions used as benchmark in order to test the performance of the proposed RPLNN and compare it with CGA, and two gradient-based optimization methods: steepest descent (SD) and conjugate-gradient techniques (CG) [7-8].

i- A function with a unique global maximum surrounded by local maximum

The first function is defined as

$$f(x_1,x_2)=\frac{Cos^2\sqrt{x_1^2+x_2^2}}{1+0.001\cdot\left(x_1^2+x_2^2\right)} \tag{13}$$

As one observes from Figure 8, this function possesses a unique global maximum of 1 at the origin surrounded by an army of local optima. Because of these oscillations many different optimization techniques had hard time to find the maximum of this function.

To show this, we apply two conventional hill-climbing methods (S.D. & CG), the CGA and RPLNN to find the optimum of this function. The results and run time of them are summarized in Table 4.

Because all the three methods, steepest descent, conjugate gradient and RPLNN act on individuals, for the purpose of comparison, we set the initial conditions to four different value as,

(x_{01},x_{02}) = (31.9844,31.9844), (x_{01},x_{02}) = (10.6562,10.6562), (x_{01},x_{02}) = (31.9844,10.6562) and (x_{01},x_{02}) = (-3.5312,-3.5312). For each initial condition, we performed the RPLNN method 40 runs (times) and the average value of running times is listed in Table 4.

moment of inertia for segment i, I_i, bending moment at the left end of segment i, and M_i, are defined by:

$$I_i = \frac{b_i \cdot h_i^3}{12}, \tag{4}$$

$$M_i = P\left(L + l_i - \sum_{j=1}^{i} l_j\right). \tag{5}$$

And finally the corresponding maximum bending stress becomes:

$$\sigma_i = \frac{M_i \cdot h_i}{2 \cdot I_i} \tag{6}$$

Now, we may state the design problem as:

$$\textit{Minimize:} \quad V = \sum_{i=1}^{N} b_i \cdot h_i \cdot l_i \tag{7}$$

Subject to:

$$\frac{\sigma_i}{\bar{\sigma}} - 1 \leq 0 \qquad i = 1, \cdots, N, \tag{8}$$

$$\frac{y_N}{\bar{y}} - 1 \geq 0, \tag{9}$$

$$h_i - 20 \cdot b_i \leq 0 \qquad i = 1, \cdots, N, \tag{10}$$

$$b_i \geq 0.1 \qquad i = 1, \cdots, N, \tag{11}$$

$$h_i \geq 0.5 \qquad i = 1, \cdots, N, \tag{12}$$

In the above, $\bar{\sigma}$ and $\bar{y}$ are the allowable bending stress and displacement, respectively. This is an optimization, indeed a constraint, problem in $n = 2N$ parameters with $N + 1$ nonlinear constraints defined by Eqs. (8) and (9), N linear constraints defined by Eq. (10), and $2N$ side constraints, which are treated here as general inequality constraints, on the design variables defined by Eqs. (11) and (12). It is further assumed that lower bounds of 0.1 are imposed explicitly on b_i and h_i, $i = 1, \ldots, N$ within the optimization procedure to guarantee that the design remains physically meaningful.

The CGA and the RPLNN have been applied to this problem with five segments (10 design variables). Because of the above several constraints, the ordinary CGA could poorly solve the problem, we propose a modified CGA in which any individual is not acceptable, and only those that rest in the constraints (8)-(12) are valid in order to show the high performance of the proposed network.

3-Illustrative Examples

In this section the performance of RPLNN is investigated and compared with that of the CGA. Two different examples are considered as case studies. In the first example, a problem of minimization is proposed, and in the second example, some function optimization problems are performed.

Example 1: Design of a Cantilevered Beam

Consider the cantilevered beam shown in Figure 7 [15]. This beam is to be designed for minimum material volume. The set of adjustable design variables includes the width b and the height h at each of N segments, with N=5. The objective is to design the beam subject to limits on stress, which are calculated at the left end of each segment, deflection under the load, and the geometric requirement that the height of any segment does not exceed twenty times the width.

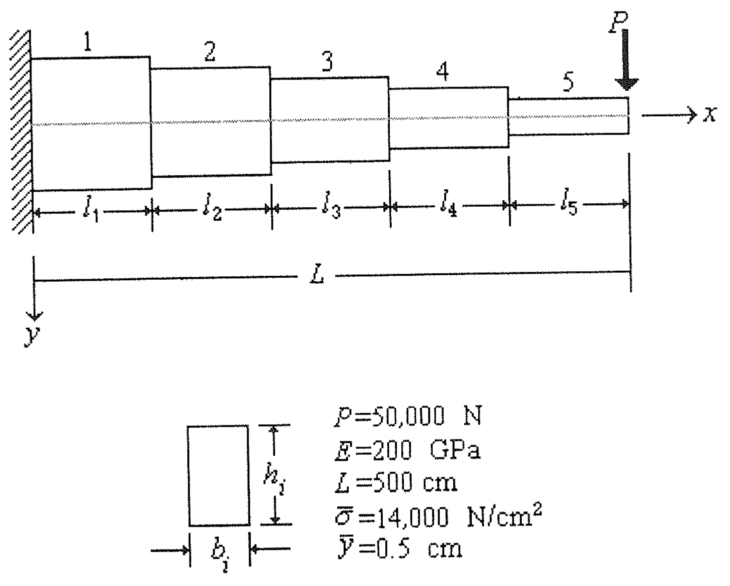


Figure (7) Cantilevered beam

The following recursion formula calculate the deflection y_i at the right end of segment i:

$$y_o = y'_o = 0, \tag{1}$$

$$y'_i = \frac{P \cdot l_i}{E \cdot I_i} \cdot \left(L + \frac{l_i}{2} - \sum_{j=1}^{i} l_j \right) + y'_{i-1}, \tag{2}$$

$$y_i = \frac{P \cdot l_i^2}{2 \cdot E \cdot I_i} \cdot \left(L + \frac{2 \cdot l_i}{3} - \sum_{j=1}^{i} l_j \right) + y'_{i-1} \cdot l_i + y_{i-1}. \tag{3}$$

Where l_i is the length of segment i, the deflection y is defined as positive downward, y' represents the slope, E denotes Young's module which is the same for all segments, the

Now by changing truth-values of the RPLNN, we may obtain some other state classes as given in Figure 5-b. For example, if the RPLNN reaches class {5}, it may remain in this class or it can move to another class state if the inputs change. Therefore, by changing the inputs or the truth table, the RPLNN can move to all classes of the state space and reach all states of the space including the global state. On the other hand, the RPLNN has the potential ability to reach the global point in the state space.

In short, noting that 1- the RPLNN has the ablility to divide the search space into a set of class-states, 2- by imposing the RPLNN external inputs or changing truth values of its neurons, one can either change the state communicating class or make the network experience some other states in the current class, this network will get the opportunity to reach any point in the search space including the global point.

Now the question is how to do the search in the state space. This is summarized below.

A-Search Algorithm:

1- Select the length of the RPLNN (i.e., number of neurons) for a given state space resolution,
2- Set the outputs of the RPLNN to some numbers selected at random in the state space,
3- Set all truth values of neurons to '*u*',
4- Perform the RPLNN ,
5- If the next state leads to a better performance depending on the objective function, update the truth table; set the corresponding truth-values to this state.
6- If the next state does not lead to a satisfactory result, let the network move to another state.
7- Do steps 5 and 6 for *P* times, (*P* is chosen by users), and then go to step 3.
8- Do the above steps Q times, (Q is set by user); note that *Q*>>*P*.

B-Convergence Analysis:

Although the convergence proof of the RPLNN is independently under investigation by the authors using the theory of statistical convergence and analysis (i.e., probability in the limit), the RPLNN convergence analysis can be easily done by adopting the procedure established by Zhang et al. [4-5] for convergence analysis of a multi-layer PLN network whose general structure is shown in Figure 6. By a deep look at the RPLNN shown in Figure 3 and comparing it with the structure given in Figure 6, one may observe that the RPLNN is indeed a one layer PLN network with a special feed-back/feed-forward ordering.

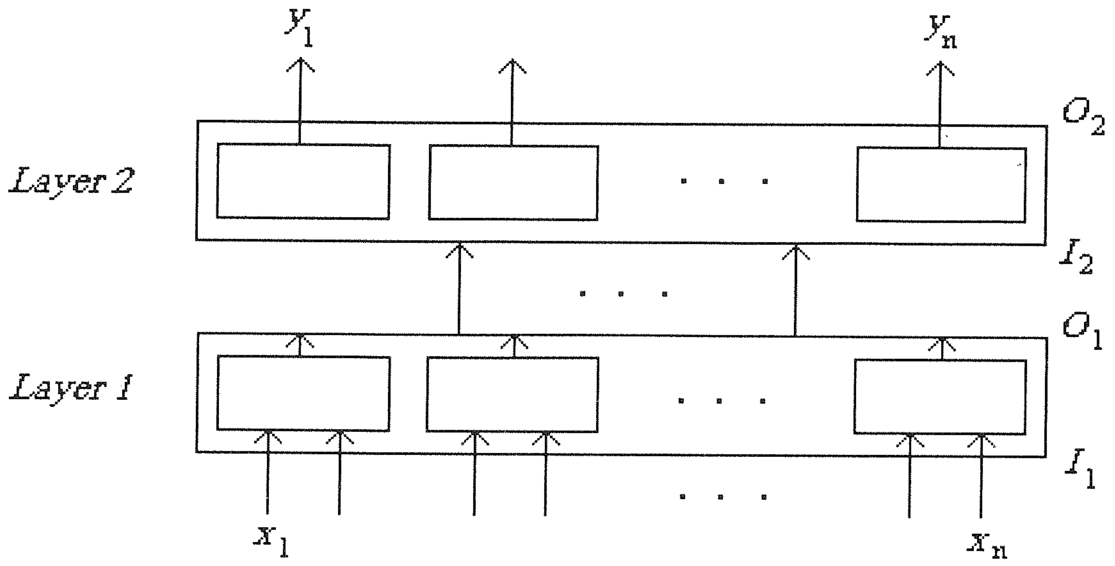


Figure (6) Multi-Layer PLN Network.

Table (2) Truth table of neurons of the RPLNN, *steady state case.*

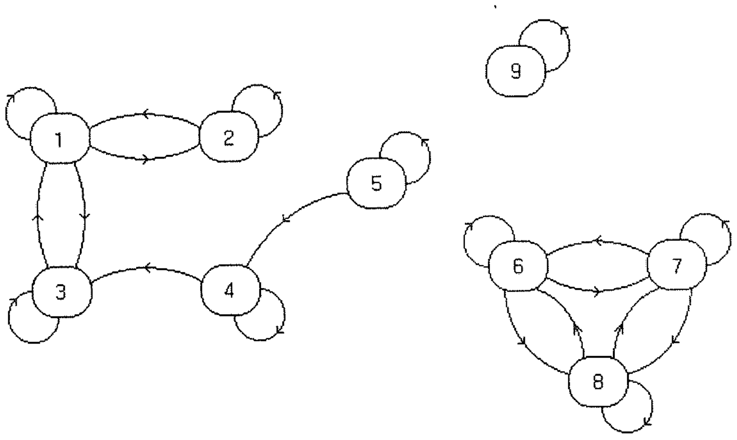
I_1^i	I_2^i	O^1	O^2	O^3	O^4	O^5	O^6	O^7
0	0	u	0	1	u	0	u	u
0	1	u	u	1	0	u	1	1
1	0	0	0	u	0	u	1	u
1	1	0	u	u	u	0	u	1

It is evident that the outputs of the RPLNN are obtained by looking at its truth table. For example, assume that the output of each PLN is *u, i.e.* if the inputs of unit 5 are $I_1^5 = 1$, $I_2^5 = 1$, the output of this unit becomes 0 or 1 with equal probability.

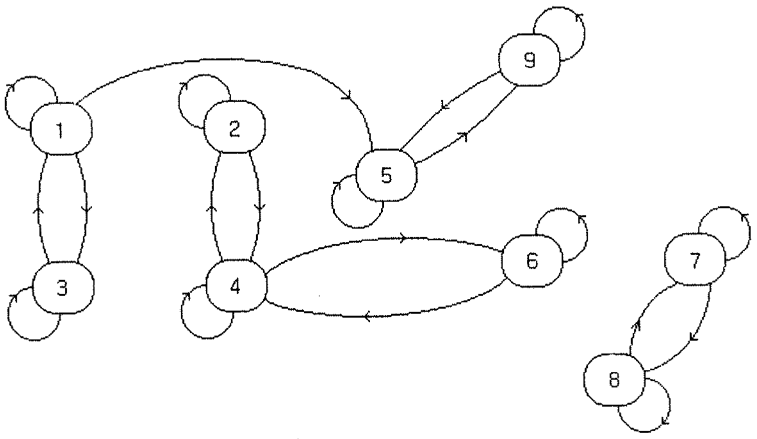
Now assume that the RPLNN has 9 states and its truth table has been partitioned in the state space communicating classes: $\{1,2,3\}$, $\{4\}$, $\{5\}$, $\{6,7,8\}$ or $\{9\}$ as shown in Fig. 5. If the RPLNN is in state 2, see Figure 5-a, then it can jump to the state class $\{1,2,3\}$; consequently the RPLNN can see all states in this class. We should note that by varying the values of the inputs, the network may go to the other state classes, i.e. state class $\{4\},\{5\},\{6,7,8\}$ or $\{9\}$, or remains in

the present state class $\{1,2,3\}$.

State communication classes the RPLNN



(i) State communication classes the RPLNN



(b) Moving around State Communicating Classes

Figure (5) State Communicating Classes of the RPLNN based on its current truth table.

Now we propose the structure given in Figure 3. It obviously has a ring structured format. This is the reason why the acronym of Ring Probabilistic Logic Neural Networks (RPLNN),.has been taken.

2–2- The Optimization Property

Our objective in this paper is to use the RPLNN structure to optimize an object function.

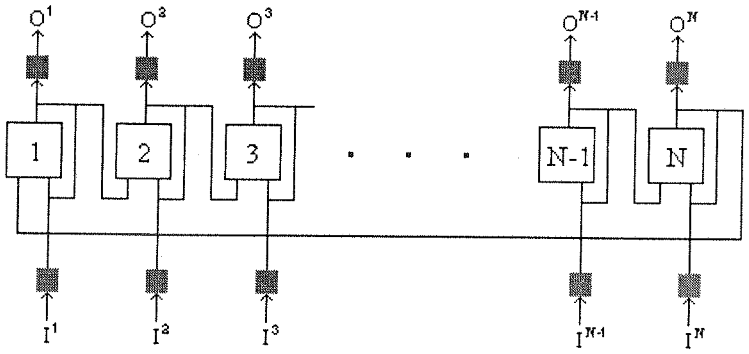


Figure (3) The Ring Probabilistic Logic Neuron Networks, RPLNN.

For this reason, this subsection is devoted to expose this property of the proposed network. Assume that the current inputs of the RPLNN are I_1=1100110 and I_2=1001101 and the output of the net is desired to be O=0010011 as illustrated in Fig. 4..

Present State: 1001101
Next Desired State: 0010011

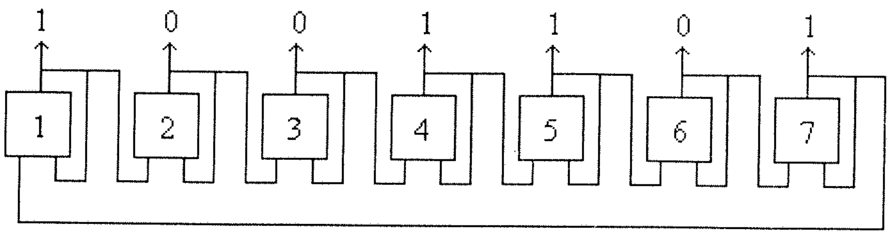


Fig (4) Transient of the RPLNN from the present State to the next desired State.

If we further assume that Table 1 represents the truth table of neurons of the RPLNN given in Fig. 4, then the next state of the RPLNN will be 0010011 and if we also wish to keep the RPLNN in this state, the truth values of neurons of the RPLNN must be as those given in Table 2.

Table (1) Truth table of neurons of the RPLNN, *transition case.*

I_1^i	I_2^i	O^1	O^2	O^3	O^4	O^5	O^6	O^7
0	0	u	u	1	u	u	u	u
0	1	u	u	u	0	u	u	1
1	0	u	0	u	u	u	1	u
1	1	0	u	u	u	0	u	u

neurons, we can employ the network to recall the prototype patterns and to recognize the patterns from noisy environment as well.

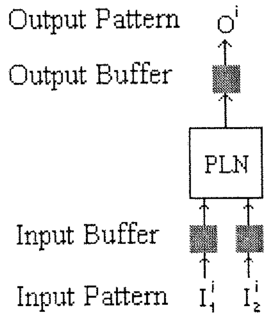


Figure (1) Probabilistic Logic Neuron (PLN).
(a) Non-Overlapped Trees of PLN Networks (taken from [3])
(b) The Structure of PLN Networks (taken from [4])

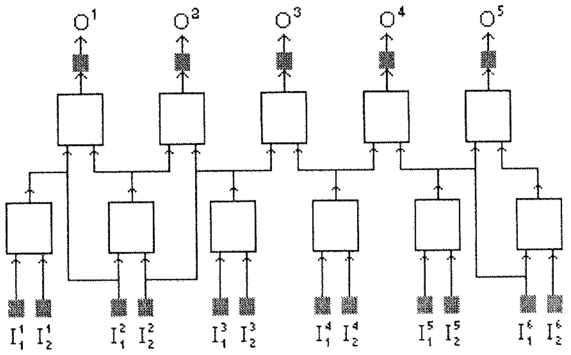


Figure (2) Two Structures of PLN Networks.

To fill out the truth table associated with each PLN, we have to use the input/output pairs of experimental data. The pure random search-learning algorithm for PLN networks called "A-Learning Rule" is given below.

1- Initially, u is assigned to each PLN element as its output value, where u denotes that the "don't care" state whose value is 0 or 1 with equal probability.

2- A pair of data $r^i = (x^i, y^i)$ is taken from a sequence of m training samples, where x^i and y^i respectively represent the input and output vectors.

3- Input x^i. If the output vector of the network is equal to y^i, go to step 4-1, otherwise train the network for p times, (p is a positive integer). If the output is still not equal to y^i, go to step 4.1.

4-1- Record the values in the truth table which shows partially the relationship between output and input of each element and then go to step 2.

4-2- After assigning the value "u" to the outputs of all neurons, go to step 2.

5- Repeat steps 1 to 4, until the entries of truth table do not vary with time for all training data. Here, the don't-care entries of the truth table (the entries with "u" values) are considered as float values.

Easily can be seen that with this learning algorithm, the PLN can be used as a pattern recognizer.

second category require gradient information either exactly or numerically. The common characteristic of these methods is that they all work on a point-by-point basis, i.e. they begin with an initial solution (usually supplied by the user) and a new solution is calculated according to the steps of the algorithm.

In 1957, Box introduced a new direct search technique known as "evolutionary operation (optimization)" method, [9]. Box's direct search method is different from the aforementioned methods. Box's algorithm operates with a number of solutions instead of a single one. Around the best solution of the current iteration, a new set of solutions is created for the next iteration. Because the algorithm requires no information about the non-accepted solutions in choosing new solutions in subsequent iterations, it represents a slow and not good enough efficient method. But, it is worth mentioning that, however this algorithm in general resumes with an evenly distributed set of solutions, in the case of wide spread solutions; it may find at some point the global solution.

Another important method for optimization problem is Simulated Annealing. The SA is inspired from annealing of solids and is leaned to atomic array of material depending on the material's level of energy, which becomes less for solid materials. In SA there exists a unique relation between the parameters and the objective function of the optimization problem with the atomic array and the energy level of the material, respectively. The SA is very slow because of its temperature effects and individual based property [10]. The slowness of the SA as well as its fast heating and gradually cooling process prevent SA to be used in real-time control process.

The next important alternative to solve optimization problems is Evolutionary Programming (EP). The process stages of EP and GA are almost the same and both are belong to Evolutionary Strategy (ES), but their difference is that the conventional GA represents a binary coded process whereas the EP is indeed a real coded process. In the EP, to produce new population from the present population, the strategies (μ, λ) and $(\mu + \lambda)$ have been introduced [11].

Mimicking the process of biological evolution, GA represents a type of structured random search [12-13]. In solving a given optimization task, the GA starts with a collection of solutions (i.e. parameter estimates) called by chromosomes. Each individual (chromosome) is evaluated by a fitness function. In each iteration, more fit chromosomes (parents) have higher chance to mate and bear offspring (produce new individuals). These individuals (children) or new parameters provide the basis for the next generation. The conventional GA may be completely described by the following step [14]:

1- Select a coding scheme to represent adjustable parameters.
2- Initialize and evaluate a population of chromosomes.
3- Repeat the following steps till stopping criteria are met.
    3-1- Reproduction
    3-2- Crossover
    3-3- Mutation
    3-4- Population evaluation

Below the structure of Ring Probabilistic Logic Neural Networks (RPLNN) is described.

2-Optimization by Ring Probabilistic Logic Neural Networks

1–2-The Structure

In this section we introduce the RPLNN by employing the concept of Probabilistic Logic Neuron (PLN) [1-5] that has been frequently used in pattern recognition problems. Figures 1 and 2 represent the PLN and the most two common kinds of the PLN networks, respectively. Using the structures given in Figure 2 and the truth table showing the inter-relation among the

A New Optimization Method by Ring Probabilistic Logic Neural Networks

N. Seifipour
Ph. D. Student

M. B. Menhaj
Associate Professor

S. K. Y. Nikravesh
Professor

Department of Electrical Engineering
Amirkabir University of Technology

Abstract

By adopting the concept of the probabilistic logic neuron which represents a class of RAM-based neural networks, this paper introduces a model-free optimization method called Ring Probabilistic Logic Neural Networks (RPLNN) that are more suitable for optimization problems. To highlight performance of the RPLNN, some different optimization problems are considered. The simulation results show that the Ring Probabilistic Logic Neural Networks remarkably outperform the conventional genetic algorithms and some gradient based methods as well.

Keywords
Optimization Methods, Genetic Algorithms, Neural Networks

Introduction

As an alternative to the traditional weighting-sum-threshold model of neural networks, Kan et al [1-2] originally introduced the Probabilistic Logic Neuron (PLN) model of neural networks. With the general goal of overcoming some deficiencies of traditional neural networks, the PLN is a very simple model and represents a class of RAM-based neural networks. The PLN has been used frequently in pattern recognition [3-5]. In this paper by employing the concept of probabilistic logic neuron, we introduce Ring Probabilistic Neural Networks (RPLNN) which is more suitable for optimization problems. The RPLNN represents a model free-based optimization method adopted from the PLN. The PLN categorizes the space response while the RPLNN has the extra ability to move on in each category.

The paper is organized as follows. Section II presents a brief review of optimization techniques and procedure of conventional genetic algorithms (CGA). Section III introduces the RPLNN. To show the effectiveness of the proposed genetic algorithm, in section IV, through two illustrative examples the performance RPLNN are investigated and their capabilities are listed. Finally, section V concludes the paper.

1-A brief review on optimization technique

Most traditional optimization methods employed in science and engineering can be divided into two broad categories: direct search techniques [6] and gradient search methods [7-8]. The methods in the first category need the objective function values while the techniques in the