Journal of Global Optimization **16:** 229–243, 2000.
© 2000 *Kluwer Academic Publishers. Printed in the Netherlands.*

Test Problem Generator by Neural Network for Algorithms that Try Solving Nonlinear Programming Problems Globally*

DEGANG LIU and XIANG-SUN ZHANG
Academy of Mathematics and System sciences, Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing 100080, China
(e-mail: xszhang@amath2.amt.ac.cn; dliu@math2.amt.ac.cn)

(Received 8 April 1997; accepted 26 November 1999)

Abstract. A test problem generator, by means of neural network's nonlinear function approximation capability, is given in this paper which provides test problems, with many predetermined local minima and a global minimum, to evaluate nonlinear programming algorithms that are designed to solve the problem globally.

Key words: Feedforward network; Global solution; Nonlinear programming; Test problem generation

1. Introduction

A general nonlinear programming problem may have many local optimal solutions. For the case of searching minimal (or maximal) solution, a local minimum (maximum) with the smallest (largest) objective function value among the all local solutions is the global solution. It is natural that one prefers to solve a given problem by ending at a global solution. But it is well known that most of the traditional nonlinear programming algorithms converge to a local solution. Many authors devoted to design algorithms that have global behavior, that is, the designed algorithm is expected to end at a global solution. We denote such algorithms as *G*-algorithms to distinguish them from the traditional algorithms.

There are two classes of *G*-algorithms. One is in the deterministic nature and the other is in the stochastic nature in which some statistical techniques are involved. Surveys on the developments of deterministic *G*-algorithms can e found in papers and books [8, 9, 13, 14, 18, 20]. The representative works and surveys for the stochastic nature *G*-algorithms are in [2, 12, 17]. Recently Artificial Neural Networks are used as efficient tools of *G*-algorithm design, Boltzmann Machine and Simulated Annealing are two of the popular methods, which are basically in the class of stochastic nature (see [10, 1]).

* This research was supported by the National Science Foundation of China (No. 39830070).

To evaluate and compare the existing/new G-algorithms, one needs a system of representative test problems. Despite some contributions for constructing and collecting test problems for G-algorithms can be found (see [11, 15, 16]), there still exists a lack of representative test problems for G-algorithms. There are two sources of test problems collections:

(2) Collection of problem instances that have been used in published papers and reports to evaluate G-algorithms, and real world problems that were raised in practical applications with significant characteristics. Test problems in this kind of collection may differ from each other greatly and no relationship between them. The algorithm designer will be satisfied if he/she can solve these test problems one by one by using his/her new algorithm, but may neglect a fact that such a solution is probably made in a well prepared situation, such as the initial points were sophisticatedly chosen for the specified test problem. When the test problem changes, adjusting the proper initial points could be time-consuming. This makes difficulty in comparing G-algorithms based on computational results with this collection. [4] is the first book that systematically collects test problems in this category.

(2) Collection of randomly generated test problems with known global solution. A package of software that produces test problems with the following expected properties is called a test problem generator: (i) Randomly choosing parameters in the software, it provides virtually limitless supply of test problems that cover a wide range of problems that mimic practical applications; (ii) it produces test problems that have predetermined features which are specified by the research objectives, such as test problems with predetermined local solutions, predetermined global solution or ill-conditioned, degenerate, indefinite structures, etc. A generator gives user the ability to conduct computer simulation of parameter variation and the option to generate rather than just store the test problems.

The first systematic research on test problem generator for nonlinear programming algorithms is due to K. Schittkowski in 1980 [15]. His test problems have the following general form:

$$\begin{aligned} \min \quad & f(x) \\ \text{s.t.} \quad & g_j(x) = 0\,, \quad j = 1, \dots, m_e \\ & g_j(x) \geqslant 0\,, \quad j = m_e + 1, \dots, m \\ & x_l \leqslant x \leqslant x_u \end{aligned} \tag{1}$$

where the objective function $f(x)$ is defined by

$$f(x) \equiv s_0(x) + \frac{1}{2} x^T H x + q^T x + \alpha \tag{2}$$

with an $n \times n$ matrix H, $q \in R^n$ and $\alpha \in R$. Let x^* be randomly chosen with $x_l < x^* < x_u$ as a local minimizer of the problem and with exactly m_a active constraints. He therefore defines the restrictions by

$$\begin{aligned} g_j(x) &\equiv s_j(x) - s_j(x^*) + d_j^T(x^* - x)\,, \quad j = 1, \dots, m_e + m_a \\ g_j(x) &\equiv s_j(x) - s_j(x^*) + \mu_j\,, \quad j = m_e + m_a + 1, \dots, m\,, \end{aligned} \tag{3}$$

where $d_j \in R^n$, $j = 1, \dots, m_e + m_a$, and the real numbers μ_j, $j = m_e + m_a + 1, \dots, m$, are randomly chosen within the interval $(0, m)$. And the functions $s_0, s_1, \dots, s_m$ are defined by signomials, i.e., generalized polynomial functions:

$$s_j(x) = \sum_{p=1}^{P} c_p^{(j)} \prod_{i=1}^{n} x_i^{a_{ip}^{(j)}}\,, \quad x > 0\,, \quad j = 0, \dots, m\,, \tag{4}$$

where the coefficients $c_p^{(j)}$ and the exponents $a_{ip}^{(j)}$ are randomly chosen real numbers with predetermined bounds. The rest things are to properly choose the matrix H, q and α in $f(x)$ so that x^* satisfies the Kuhn-Tucker conditions, the second-order conditions and $f(x^*) = 0$, i.e., x^* is a local solution with objective value zero. The test problem generator discussed above was effectively employed by many authors in their comparative research of existing nonlinear programming codes and new suggested algorithms because of its simple structure and flexibility to fit different purpose of research. But as one noticed, when we need to test a G-algorithm, what we can draw from the running of the test problem is:

$$\begin{aligned} &\text{The algorithm finds the global solution} && \text{if } f(\bar{x}) < 0 \\ &\text{The algorithm failed to find the global solution} && \text{if } f(\bar{x}) \geqslant 0 \end{aligned} \tag{5}$$

where $\bar{x}$ is the solution ended by the tested algorithm. But in fact, $\bar{x}$ is not necessarily a global solution even if $f(\bar{x}) < 0$ and we can not have a reliable answer for the algorithm if it finds the global solution.

For quadratic programming algorithms, there is a test problem generator in [11]. Both the generators in [15] and [11] are for the constrained optimization algorithms and also not specifically for the G-algorithms. On the other hand, G-algorithms for unconstrained optimization play a very important role in global optimization research, because they are the algorithms without taking advantages of the problem's structure and then have much wider applications. In fact any constrained problem can be transformed into an unconstrained problem by introducing various penalty terms.

Many test functions of various formats for unconstrained G-algorithms have been provided. To name a few, Wingo [19] gave a test problem with one variable and the known global solution:

$$\begin{aligned} \min \quad & x^6 - \frac{52}{25}x^5 + \frac{39}{80}x^4 + \frac{71}{10}x^3 - \frac{79}{20}x^2 - x + \frac{1}{10} \\ & -2 \leqslant x \leqslant 11 \end{aligned} \tag{6}$$

with the global solution $x^* = 10$ and $f(x^*) = -2763.233$. Goldstein and Price [6] provided a test problem with two variables:

$$\begin{aligned} \min \quad & [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)] \\ & \cdot [30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)] \end{aligned} \tag{7}$$

with the global solution $x^* = 0$, $y^* = -1$ and the optimal objective value 3.0.

Unfortunately there is a lack of generators that can produce test problems with general nonlinear objective function and known global solution for unconstrained optimization research in the literature.

In most cases a test problem for unconstrained optimization takes the form of a combination of elementary functions whose local minima are known. For given test problems, it is easy to evaluate the similarity of these objective functions in one-dimensional case, just by comparing their graphs from which one can identify the locations of their local minima through looking at valleys of the function curves in some regions. When we look at an objective function graph, global minimum locates in the lowest valley among all others, and the depth and rudeness of the valley curves represent toughness of the problem to a *G*-algorithm. However, the choice of test problems in the multidimensional case is more complicated than in the one-dimensional case. It is difficult to use the graphs of the function ($n \geqslant 3$) for evaluating if they are suitable problems one would like the *G*-algorithms to solve. Features of a test function are hard to control by users. A well-designed test problem generator can be superior in overcoming this disadvantage.

Reviewing the published works, one way of designing a test problem generator is to formulate a template of analytical functions, usually using signomials as its building blocks. At minimum point a quadratic function is used to help obtain the convexity as optimality conditions requested. However, when many local points are to be known, the final polynomial function could be very complicated because it had to interpolate these quadratic functions and the computation of the each quadratic function coefficient could be very cumbersome.

Function approximation is recognized as one of the many successful applications of feedforward type Artificial Neural Networks. From mathematical point of view, a feedforward network performs a nonlinear mapping between an input and an output space. The learning of the network can be regarded as synthesizing an approximation of a multidimensional function which is performing a simple fitting operation or a hypersurface reconstruction in a multidimensional space to a finite set of data points (the training examples). From this point of view the generalization is nothing more than interpolating the test set on the fitting (or reconstructed) hypersurface.

Consider an unconstrained nonlinear optimization problem:

$$\begin{aligned} \min \quad & f(x) \\ & x \in R^n \end{aligned} \tag{8}$$

where $f(x)$ is a general nonlinear objective function. As a test problem for *G*-algorithms, some information at the local and global solutions, we call it *Solution Information*, of the problem (8) are expected, including some of its local and global optimal points, function values at these points, and/or Hessian properties. Traditionally, this set of solution information is predetermined by designers or stochastically selected as inputs to a test problem generator, then a system of parameters can be calculated within the analytical form of $f(x)$, so that the optimal conditions at these points are satisfied by the generated objective function $f(x)$. Therefore, a general test problem of unconstrained nonlinear optimization problem can be expressed in the form:

$$\begin{aligned} &\text{min.} \quad f(w, x) \\ &\text{s.t.} \quad v \leqslant x \leqslant u \end{aligned} \tag{9}$$

where $x \in R^n$, w is a set of parameters to be determined before (9) is used to test a G-algorithm, and $v, u \in R^n$ are lower and upper bounds telling the range of the solutions and limiting the search area. Once w is chosen $f(w, x)$ is a function of x and (9) can e used as a specific test problem.

In this paper we propose that a neural network can work as a test problem generator with features just discussed in the above paragraph. The test problem we suggested for G-algorithms takes the form:

$$\begin{aligned} &\text{min} \quad F(W, x) \\ &\text{s.t.} \quad v \leqslant x \leqslant u \end{aligned} \tag{10}$$

where $x, v, u \in R^n$, F is a feedforward neural network, i.e., a multi-layer perceptron (MLP) of two hidden layers and an output layer with only one output node with *synaptic weights* denoted as $W \in R^m$. For any given $W \in R^m$, $F(W, x)$ works as a nonlinear mapping: $F : R^n \rightarrow R^1$, or a smooth nonlinear function defined in R^n. Once solution information is properly presented, the neural network $F(W, x)$ has the ability to learn the weights W from the information to behave as designer expected at the solution points.

Theoretical proofs concerning the power of the neural networks in approximating nonlinear functions have been given by many authors. Here we cite a theorem by Girosi and Poggio [5]:

THEOREM 1. Networks with two layers of hidden units with sigmoidal nonlinearity can uniformly approximate (to within any $\epsilon > 0$) any real continuous function of multiple real inputs.

In Section 2 of this paper we give a procedure of test problem generator construction in a way that we use a two-hidden-layer perceptron network architecture with sigmoidal nonlinearity for its hidden layers and one linear neuron for its output layer. Back-propagation is employed to learn nonlinear objective functions based on the information given at user defined local/global points. The each trained network then can be used as a test problem in the form of (10) for unconstrained global minimization. The generated test problems have nonlinear objective functions, many known local minima and one global solution. Network training for several specific test problem examples are presented in Section 3.

2. Test problem generator construction

Choose $x^l \in R^n$, $v \leqslant x^l \leqslant u$, $l = 1, \ldots, L$ to be the expected isolated local minima points of the objective function of (8) and specify their local minima values as z^l, $l = 1, \ldots, L$ respectively. Suppose $x^* = \text{argmin}\{x^l \mid z^* \leqslant z^l,\ l = 1, \ldots, L\}$ be the global minimum point and corresponding z^* be the optimal value of $f(x^*)$.

Decide a δ^l for each x^l such that $\Omega_l = \{x \mid \|x - x^l\|_2 \leqslant \delta^l,\ l = 1, \ldots, L\}$ are not overlapped each other. Construct quadratic functions:

$$Q^l(x) \equiv \left(\frac{1}{2} x^T H^l x + q^{l^T} x + \alpha_l\right), \qquad \|x - x^l\|_2 \leqslant \delta\ , \quad i = 1, \ldots, L \tag{11}$$

with

$$\begin{aligned} H^l &= U^{l^T} U^l \\ q^l &= -H^l x^l \\ \alpha_l &= z^l - \frac{1}{2} x^{l^T} H^l x^l - q^{l^T} x^l\ . \end{aligned} \tag{12}$$

where U is a randomly generated $n \times n$ up-triangle matrix with its coefficients chosen from a uniform or normal distribution.

Our purpose is to fabricate such an objective function in form of (10) that its behavior at each of the δ^l-neighbourhood of x^l is just like the positive semidefinite quadratic function $Q^l(x)$ of (11) with the property that

$$\begin{aligned} &\nabla_x Q^L(x^l) = 0 \\ &\nabla_x^2 Q^l(x^l) = U^{l^T} U^l \\ &Q^l(x^l) = z^l\ . \end{aligned} \tag{13}$$

This quadratic function guarantees x^l to be the optimal point locally and can be used to take data samples within the hypersphere Ω_l for training the network (10).

A learning data set P and T are taken in the following way:

$$\begin{aligned} &P = \bigcup_{t=1}^{L} P^l \\ &P^l = \{p^l(k) \mid \|p^l(k) - x^l\|_2 \leqslant \delta^l,\ k = 1, \ldots, K\}\ , \quad l = 1, \ldots, L \end{aligned} \tag{14}$$

where $p^l(k)$ distribute uniformly in Ω_l. K is a predefined constant of local learning set sample size at Ω_l. The total learning size of P is $L \times K$. Let

$$T^l = \{t^l(k) \mid t^l(k) = Q^l(p^l(k)),\ k = 1, \dots, K\}, \quad l = 1, \dots, L$$
$$T = \bigcup_{l=1}^{L} T^l \tag{15}$$

Now the examples of input-outut pairs (P, T) are ready to be used in learning the network (10).

Structure of our test problem generator is a three-layer perceptron. Mathematically

$$F(W, x) = \Psi^{[3]}(W^{[3]}\Psi^{[2]}(W^{[2]}\Psi^{[1]}(W^{[1]}x))) . \tag{16}$$

For convenience and simplicity the bias term in (16) is handled here (as usual) in a manner of uniform with synaptic weights by considering it as a weight connecting a neuron whose activation is always equal to unity. A functional block diagram representation of above network is shown in Figure 1.

Designing the structure of network (16) is to decide:

1. Sufficient number of neurons for each hidden layer;
2. Activation function $\Psi^{[s]}$, $s = 1, 2, 3$.

Typically for the purpose of function approximation,

$$\Psi^{[1]}(u) = \Psi^{[2]}(u) = \tanh(u) = \frac{1 - e^{-u}}{1 + e^{-u}}; \qquad \Psi^{[3]}(u) = u . \tag{17}$$

The number of neurons required for hidden layers depends on the complexity of a specific test problem. The more complex F is, the more hidden neurons are needed to realize the nonlinearity of the mapping. Hence the requirement of hidden neurons

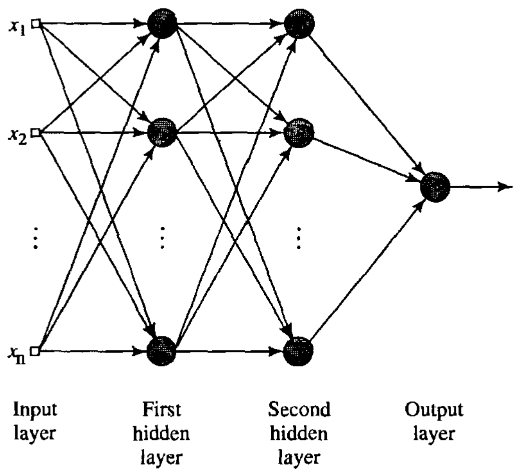

Figure 1. Architecture of the three-layer feedforward network.

is related with dimensionality and L, the number of local minimum points. Not enough hidden neurons will result in too big learning errors because of lack of nonlinearity while too many neurons may bring the problem of overfitting. Optimal strategies in designing a MLP was summarized by Simon Haykin [7].

Given the structure of $F(W, x)$ and the learning set (P, T), the back-propagation algorithm with any other learning strategy such as presented in [3] can be used here, following a process that starts with random initial weights and continuously adjusting the weights until the total errors between $F(W, x)$ and the target T reach to a predetermined goal ϵ, or the epochs (complete presentations of the entire training set during the learning process) is larger than a pre-set sufficiently large constant. After learning, the final weight matrix $W^* = \{W^{[1]}, W^{[2]}, W^{[3]}\}$ are recorded. The $F(W^*, x)$ then can be used as a test problem objective function.

Now we conclude the whole process that produce a test problem:

Algorithm for Test Problem Construction

Step 1. Choose $v \leqslant x^l \leqslant u$ and z^l, $l = 1, \ldots, L$.
Find $x^* = \mathrm{argmin}\{x^l \mid z^* \leqslant z^l,\ l = 1, \ldots, L\}$ and compute $\delta^l = \min\{\|x^l - x^j\|_2/3,\ j = 1, \ldots, L\}$.

Step 2. For $l = 1, \ldots, L$, construct quadratic function $Q^l(x)$ defined by (12) and (13).

Step 3. Randomly take data samples from each Ω_l. Collect all samples into P according (14). Compute target data set T according to (15).

Step 4. Select number of neurons for each hidden layer. Start learning process, after specifying the error goal, learning rate, and the maximum epoch as argument, by using back-propagation algorithm program, standard, or combined with various learning strategies such as momentum and learning rate adaptation to accelerate convergence.

Step 5. Record the learned weights W^* when converged.

REMARK 1 (Stopping Criteria). The back-propagation algorithm in step 4 is considered to have converged at W^* when summed squared error or the absolute rate of changes in the average squared error per epoch is sufficiently small, or when iteration of epochs is greater than a pre-set threshold.

REMARK 2. In (12), U^l is a $n \times n$ up-triangle nonsingular matrix generated randomly. This means that $H^l = {U^l}^T U^l$ is positive definite and x^l is expected to be the unique optimal solution locally in Ω_l. If H^l is generated using normally distributed variates, it is very likely that we will obtain a well-conditioned $Q^l(x)$ in Ω_l. When constructing the QP test problem, Lenard [11] adopted a method that the condition number of the Hessian matrix can be controlled. In his method, the condition number of H is determined by the element of a $n \times n$ diagonal matrix D, which are chosen as follows:

$$
\begin{aligned}
&D_{11} = \frac{1}{t'} \\
&D_{ii} = (t')^{u_i}\,, \quad i = 2, \ldots, n-1 \\
&D_{nn} = t'
\end{aligned}
\tag{18}
$$

where t' is the square root of the desired condition number and each u_i is a uniform variate on the interval $(-1, +1)$. As a consequence, the eigenvalues of H are distributed on $(1/t, t)$. To generate a positive semidefinite quadratic function $Q^l(x)$ one or more of diagonal elements of D should be set to zero. If we use Lenard's method in (12), H^l can be constructed by

$$H^l = B^{l^T} D^l B^l \tag{19}$$

where B is a randomly generated orthogonal matrix which can be obtained through QR decomposition technique from a square matrix N, each of whose elements is chosen from the standard normal distribution.

REMARK 3. By this algorithm, because there exist errors between data samples and actual output of $F(W^*, x)$, x^l may not be expected as the accurate local/global solutions. However, as a test problem, it is enough to know that x^l's are very close to the exact local minima of $F(W^*, x)$. Actually, it is not necessary to set the error goal very small, because the most important is to let $F(W^*, x)$ learn the trends and solution structures near each x^l.

3. Network training

In this section, we are going to demonstrate the algorithm proposed in the last section by several different experiments of one- and two-dimensional cases in order to show that the algorithm is well defined. Network complexity and generalization will be briefly discussed.

In our computer experiments, the core step in generating a test problem is to train the network (10) with the following sets of input for each run to the procedures programmed in Matlab:

- lower and upper bounds v, u of the free variable space R^n.
- a matrix of user selected local/global solution points $[x^1, x^2, \ldots, x^L]^T$ and a vector of corresponding solution values $[z^1, z^2, \ldots, z^L]^T$.
- local training set size, K, which defines how many example pairs will be taken from the neighborhood of a local solution.
- numbers of neurons for the first and second layers, and
- iteration parameters of maximum learning cycles (epochs), error goal, learning rate and the display frequency.

The first two types of arguments are the most fundamental and ar the only test problem specific information needed, while the last two sets of parameters are

Table 1. Parameters of experiment problems 1–3

Problem number	Locals	Objective values	#1st hidden	#2nd hidden	K	max epochs
1	(2.7, 5.0, 7.31)	(6, 0, 4)	30	30	10	1000
2	(1.9, 3.0, 5.25, 7.0, 8.93)	(−2, −5, 3, 7, 0)	50	50	10	1000
3	(−7.36, −2.5, 1.0, 3.0, 5.14, 7.2, 8.9)	(−3, 10, 5, 3, 8, 0, 4)	70	70	10	1000

related with the neural network toolbox sub-procedures of back-propagation algorithm *initff* for designing network structure, *trainbpx* for training the network with momentum and learning rate adaptation mechanism, and *simuff* for network generalization. Although there exist some hints of making a back-propagation algorithm perform better, it is often considered that the design of a neural network using the back-propagation algorithm is more of an art than a science that many of numerous factors involved in the design are indeed the result of one's own personal experience. In the following test problem training experiments we define that the first and the second hidden layer have the same number of neurons N, and we choose $N \geqslant KL^2$, where K is the local training set size which increases with dimensionality of test problem, and L is the number of known local/global solutions.

For the case of single variable ($n = 1$), three test problems have been generated via network training. We randomly choose 3 and 5 points in the interval $[0, 10]$ for problem 1 and problem 2 and 7 points in a wider interval $[-10, 10]$ for problem 3,

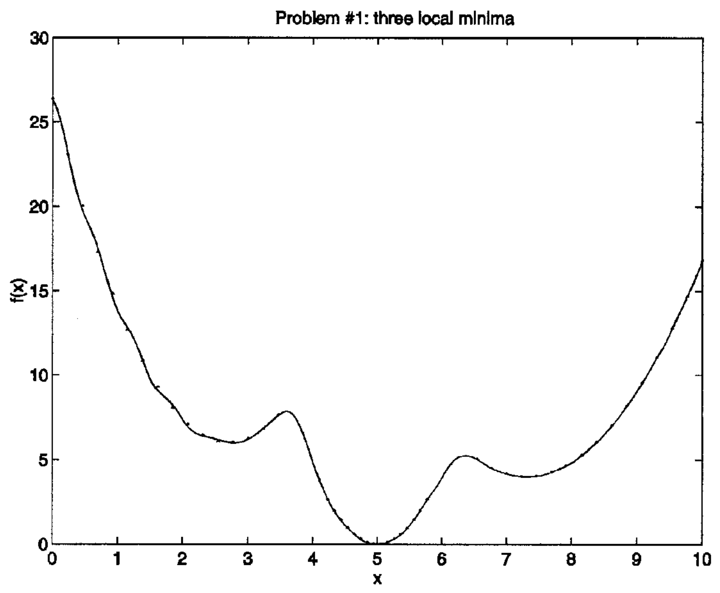

Figure 2. Test problem 1.

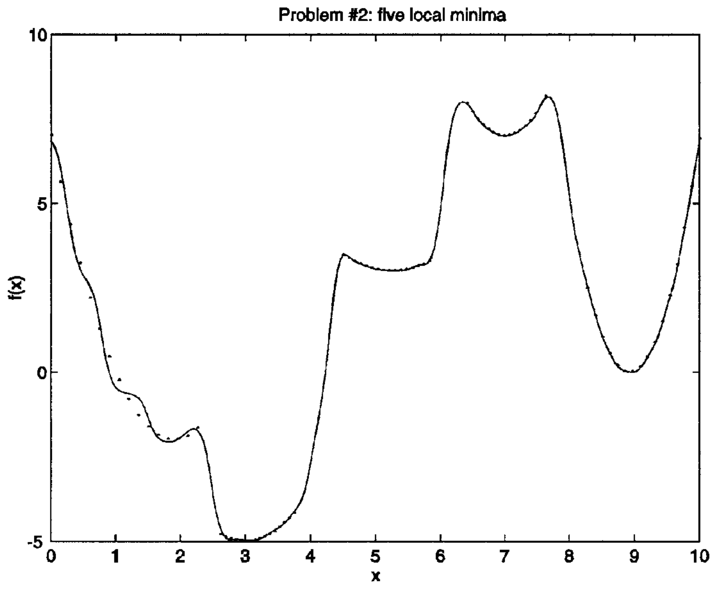

Figure 3. Test problem 2.

then according to the algorithm three networks with different hidden neurons have been trained to achieve the test problems. Parameters input for problem 1–3 to the procedure are listed in Table 1. The graphs of generated objective functions are

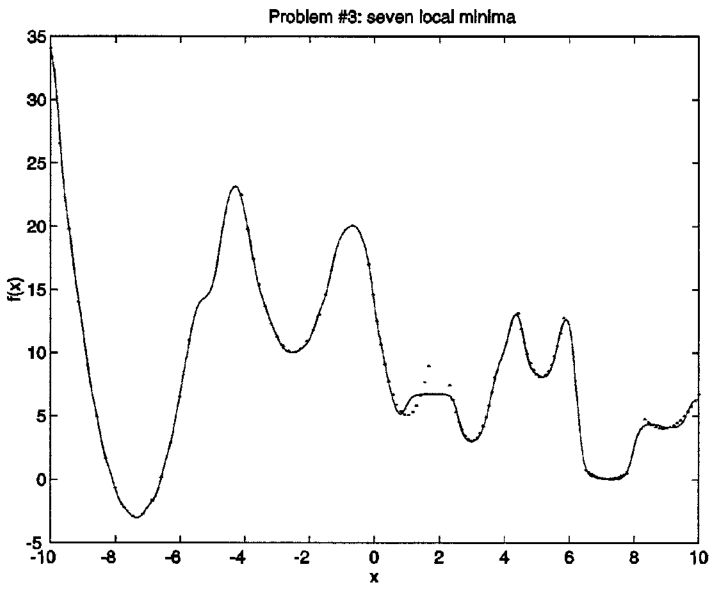

Figure 4. Test problem 3.

Table 2. Bounds and local solution information of experiment problems 4–6

Problem #	v	u	Locals	Objective values
4	$(-3, -3)$	$(16, 16)$	$(0, 1)\ (2, 5)\ (10, 13)$	$(6, 0, 4)$
5	$(-3, -3)$	$(16, 16)$	$(0, 0)\ (1, 3)\ (2, 5)\ (10, 12.5)\ (14, 15)$	$(0.2, 0, 2, 4, 1.5)$
6	$(-20, -20)$	$(20, 20)$	$(-15, -14)\ (0, 0)\ (5, 8)\ (2, 5)\ (14, 13)\ (18, 20)$	$(-5, 0.2, 0, 2, 4, 1.5)$

Table 3. Parameters of experiment problems 4–6

Problem #	#1st hidden	#2nd hidden	K	maximum epochs	Sum-squared error
4	90	90	30	1000	2.08
5	150	150	30	500	10.70
6	180	180	30	500	23.42

shown in Figures 2–4, in which dotted and solid curves represent the sample data and the learned function value generalizations.

For the case $n = 2$, networks are designed for generating another 3 problems denoted as problem 4–6, with 3, 5, and 6 local/global points stochastically chosen within their boundaries listed in Table 2.

Similarly, parameters input to the network design are shown in Table 3. The mesh plots of these two-variable test problems trained are shown in Figures 5–8. Figure 8 is the contour of problem 5 with the '+' signs showing the expected local/global valleys of the problem.

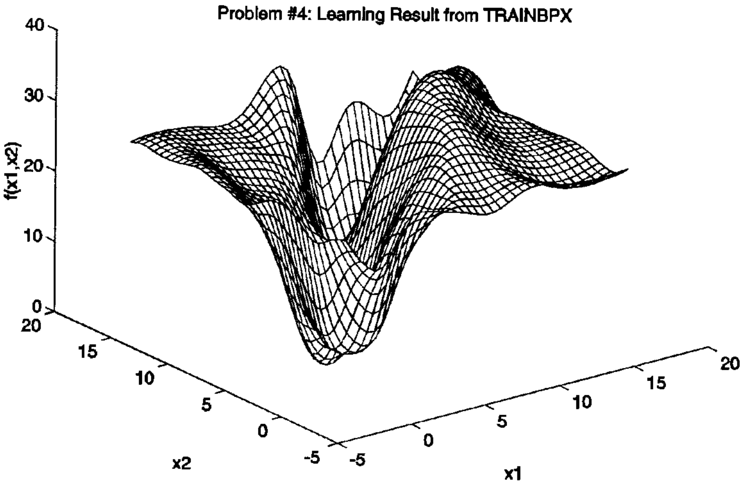

Figure 5. Test problem 5.

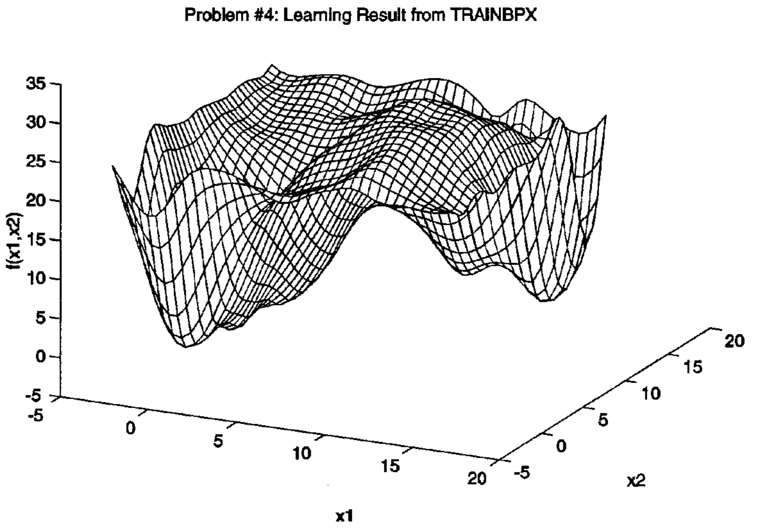

Figure 6. Test problem 5.

Noted by (16), the above generated test problems are smooth within their sample boundaries. Although there is no analytical form of the generated objective functions available from the networks, orders of derivatives of the objective functions can easily be computed by using the difference approximation technique at any point of x within the boundaries. One should be aware, however, that if we want to generalize the network outside the sample boundary some unexpected behavior of the network may occur due to the limitation of the feedforward network generalization capabilities. Figure 9 shows he result for generalizing the problem 2 outside its boundary [0, 10]. This may be a shortcoming of this approach that the test problem objective function can not be defined universally and it is the reason why we restrict our test problem in the form of (10).

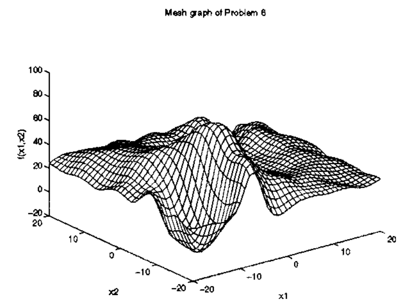

Figure 7. Test problem 6.

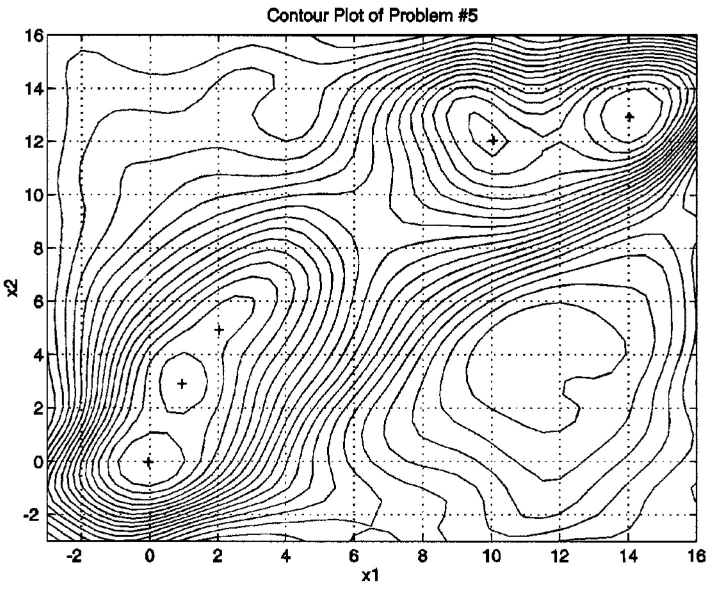

Figure 8. Contour of problem 5 outside its boundary.

Nevertheless, from these experiments we can see that the neural network as a test problem generator for global optimization algorithms has its great power of providing almost limitless source of test problems effectively.

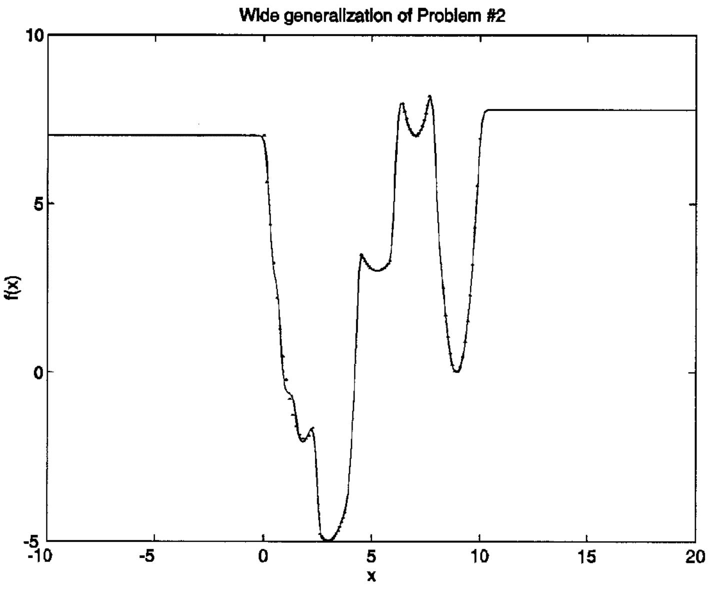

Figure 9. Generalization of problem 2.

References

[1] Aarts, E.H.L. and Korst, J.H.M. (1989), Boltzmann machines for travelling salesman problems, *European J. of Operational Research* 39: 79–95.

[2] Chew, S.H. and Zheng, Q. (1988), *Integral Global Optimization*, volume 298 of *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag.

[3] Cichocki, A. and Unbehauen, A. (1993), *Neural Networks for Optimizations and Signal Processing*, John Wiley & Sons.

[4] Floudas, C.A. and Pardalos, P.M. (1987), *A Collection of Test Problems for Constrained Global Optimization Algorithms*, volume 455 of *Lecture Notes in Computer Science*, Springer-Verlag.

[5] Girosi, F. and Poggio, T. (1991), Networks for Learning – A View from the Theory of Approximation of Functions, *Neural Networks Concepts*, *Applications and Implementations*, *Vol. 1*, Prentice Hall Inc.

[6] Goldstein, A.A. and Price, J.F. (1971), On descent from local minima, *Mathematics of Computation* 25: 569–574.

[7] *Haykin*, *S.* (1994), *Neural Networks – A Comprehensive Foundation*, Macmillan College Publishing Company.

[8] Horst, R. and Tuy, H. (1990), *Global Optimization: Deterministic Approaches*, Springer-Verlag.

[9] Horst, R., Pardalos, P.M., and Thoai, N.V. (1995), *Introduction to Global Optimization*, Kluwer Academic Publishers.

[10] Kirkpatrick, S. (1984), Optimization by simulated annealing: quantitative studies, *J. Statist. Physics* 34: 974.

[11] Lenard, M.L. and Minkoff, M. (1984), Randomly generated test problems for positive definite quadratic programming, *ACM Trans. Math. Soft.* 10(1): 86–96.

[12] Mockus, J. (1989), *Bayesian Approach to Global Optimization*, Kluwer Academic Publishers.

[13] Pardalos, P.M. and Rosen, J.B. (1986), Methods for global concave minimization: A bibliographic survey, *SIAM Rev.* 28(3): 367–379.

[14] Ratschek, H. and Rokne, J. (1988), *New Computer Methods for Global Optimization*, Halsted Press.

[15] Schittkowski, K. 91980), *Nonlinear Programming Codes*, volume 183 of *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag.

[16] Schittkowski, K. 91987), *More Text Examples for Nonlinear Programming Codes*, volume 282 of *Lecture Notes in Economics and Mathematical Systems*, Springer-Verlag.

[17] Schoen, F. (1991), Stochastic techniques for global optimization: a survey of recent advances, *J. of Global Optimization* 1(2): 207–228.

[18] Torn, A. and Zilinskas, A. (1989), *Global Optimization*, volume 350 of *Lecture Notes in Computer Science*, Springer-Verlag.

[19] Wingo, D.R. (1985), Globally minimizing polynomials without evaluating derivatives, *Intern. J. Computer Math.* 17: 287–294.

[20] Zhang, X.-S. (1984), A survey on deterministic methods for searching global optimum, *Chinese J. of Operations Research* 3(2): 1–3 (in Chinese).