Structural Engineering and Mechanics, Vol. 11, No. 5 (2001) 485-502
DOI: http://dx.doi.org/10.12989/sem.2001.11.5.485

Biologically inspired soft computing methods in structural mechanics and engineering

Jamshid Ghaboussi†

Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, U.S.A.

Abstract. Modern soft computing methods, such as neural networks, evolutionary models and fuzzy logic, are mainly inspired by the problem solving strategies the biological systems use in nature. As such, the soft computing methods are fundamentally different from the conventional engineering problem solving methods, which are based on mathematics. In the author's opinion, these fundamental differences are the key to the full understanding of the soft computing methods and in the realization of their full potential in engineering applications. The main theme of this paper is to discuss the fundamental differences between the soft computing methods and the mathematically based conventional methods in engineering problems, and to explore the potential of soft computing methods in new ways of formulating and solving the otherwise intractable engineering problems. Inverse problems are identified as a class of particularly difficult engineering problems, and the special capabilities of the soft computing methods in inverse problems are discussed. Soft computing methods are especially suited for engineering design, which can be considered as a special class of inverse problems. Several examples from the research work of the author and his co-workers are presented and discussed to illustrate the main points raised in this paper.

Key words: soft computing; newral networks; genetic algorithm; structural mechanics; inverse problems.

1. Introduction

Soft computing methods are the class of methods which have been inspired by the biological computational methods and nature's problem solving strategies. Currently, these methods include a variety of neural networks, evolutionary computational models such as genetic algorithm, and linguistic based methods such as fuzzy logic. These methods are also collectively referred to as Computational Intelligence Methods. These classes of methods share some basic characteristics which are fundamentally different from the basic characteristics of the conventional mathematically based methods in engineering analysis and computation. Understanding of the basic characteristics and the fundamental differences between soft computing methods and the mathematically based methods is essential in realizing the full potential of the soft computing methods.

In the majority of the applications of neural networks and genetic algorithm, researchers have used these methods in limited ways in simple problems. In the early stages of introduction of radically new methods, such as the soft computing methods, they are initially used similar to the

† Professor

way we use the conventional mathematically based methods. Such applications seldom exploit the full potential of the soft computing methods. For example, neural networks are often used as simple function approximators or to perform tasks where simple regression analysis will suffice. Genetic algorithm is often used in highly restricted optimization problems. The learning capabilities of neural networks and the powerful search capabilities of genetic algorithm can accomplish far more. They can be used in innovative ways to solve problems which are currently intractable and are beyond the capability of the mathematically based conventional methods. Problem solving strategies of the biological systems in nature often provide good examples of how these methods can be used very effectively. Natural systems and animals routinely solve extremely difficult inverse problems by using soft computing methods. Many of these problem solving strategies can be applied to engineering problems. We do not even consider solving a vast majority of these difficult inverse problems because they are beyond the capabilities of the conventional mathematically based methods.

After a general discussion of the soft computing methods, we will consider the application of soft computing methods to two broad classes of problems: engineering design and inverse problems in engineering. In fact, engineering design can also be considered an inverse problem. We have considered it as a separate class since engineering design differs in some significant aspects from the other inverse problems.

A vast majority of engineering problems are in fact inverse problems, but they are rarely formulated as such. This may be due to the fact that the conventional mathematically based engineering methodology are not well suited for solving inverse problems. All mathematically based methods are inherently forward or direct methods. On the other hand, soft computing methods have capabilities which are suitable for solving the inverse problems in engineering. The solution of certain inverse problems, such as cognition, is essential for the survival of species. Nature has evolved highly effective strategies for solving such inverse problems, and soft computing methods can be used to mimic those strategies, which can result in highly effective ways of dealing with very difficult engineering problems. We will describe the ongoing research of the author and his co-workers in dealing with these broad classes of currently intractable problems.

2. Soft computing methods

Almost all the computing, including computational mechanics and finite element analysis, have to be considered as hard computing. They are based on mathematical approaches to problem solving, and they inherit their basic characteristics from mathematics. On the other hand, soft computing methods are based on natural biological approaches to problem solving, where mathematics does not play as central a role as it does in our engineering problem solving methods.

Precision requirement is the most significant difference between the soft computing and hard computing methods. Mathematically based hard computing methods require a high degree of precision. This is carried over from the absolute precision or exactness of the mathematical concepts and forms. Precision requirements may not be obvious to the users of engineering software. For example, the input to a typical finite element analysis program may define the geometry of the structure and specify the material parameter and loads. These quantities, especially the material parameters, cannot be determined with a high degree of precision. However, the computer software requires precise values. Users of the computer software often provide the best estimates of the input

parameters and they do not view them as precise. However, the mathematically based methods and hard computing software consider the input parameters to be precise to within round off. Similarly, the output of the engineering computer software are also precise to within round off. However, they are not often considered so by the users of the software. In fact, there may not be much use for a high degree of precision in most engineering problems.

Universality and functional uniqueness are two other characteristics that the hard computing methods have inherited from mathematics. Universality is such a basic property of all mathematical functions that we usually do not think about it. Universality means that the mathematical functions are defined and realized universally for all the possible values of their variables. Mathematical functions are also unique in the sense that each function provides a unique mapping different from all the other functions.

The unwanted precision, universality, and functional uniqueness in the mathematically based engineering problem solving methods are very restrictive. These restrictions are the primary reason that many important classes of engineering problems, such as inverse problems, cannot be solved with the existing methodology. Mathematically based methods are inherently direct and forward methods, and as such, they are not suitable for inverse problems. In fact, there are no inverse mathematically based methods. The vast majority of inverse problems are not solved at all, and the few inverse problems that are tackled with the conventional mathematically based methods are formulated as forward problems.

Biological systems have evolved highly robust and effective methods for dealing with the many difficult inverse problems whose solutions are imperative for the survival of the species. Probably the two most important inverse problems for the biological systems are vision and cognition. Evolution has developed highly effective methodology for vision and cognition. A closer examination of cognition will reveal that for animals, precision and universality are of no significance. On the other hand, real time response and robustness, including methods for dealing with uncertainty, are essential.

Soft computing methods have inherited imprecision tolerance and non-universality from biological systems. For example, a multi-layer, feed-forward neural network representing a functional association is only expected to learn that association approximately over the range of the parameters present in the training data set. Although different levels of approximation can be attained, the approximate nature of the association is inherent.

Another inherent characteristic of the soft computing methods is their non-universality. Soft computing methods are inherently designed and intended to be non-universal. For instance, neural networks learn associations between their input and output vectors from the training data set, and the training data sets only cover the range of input variables which are of practical interest. Neural networks will also function outside that range. However, the results become less and less meaningful as we move away from the the range of the input variables covered in the training set.

A third characteristic of the soft computing methods is that they are non unique. While mathematical functions are unique, neural network representations are not unique. There is no unique neural network architecture for any given task. Since neural networks are only intended to learn associations approximately, many neural networks with different numbers of hidden layers and different numbers of nodes in each layer can represent the same association to within a satisfactory level of approximation. It can be clearly seen that the functional non-uniqueness of the neural networks is the direct consequence of their imprecision tolerance.

An added attraction of soft computing models in computational mechanics is as a result of the

imprecision tolerance and random initial state of the soft computing tools. This introduces a random variability in the model of the mechanical systems, very similar to the random variability which exists in the real systems. The random variability plays an important role in most practical problems, including nonlinear quasi static and dynamic behavior of solids and fluids where bifurcation and turbulence may occur. Finite element models are often idealized models of actual systems and do not contain any geometric or material variability. An artificial random variation of the input parameters is sometimes introduced into the idealized finite element models to account for the random scatter of those parameters in the real systems. On the other hand, the soft computing methods inherently contain random variability.

2.1 Neural networks as soft computing tools

The discussion in this section will be focused on multi-layer, feed-forward neural networks, which are currently the most commonly used neural networks in engineering applications (Ghaboussi and Wu 1998). These neural networks consist of several layers of artificial neurons of processing units. The input and the output layers are the first and the last layers. The layers between the input and output layers are referred to as the hidden layers. Each neuron is connected to all the neurons in the next layer, and the signals propagate from the input layer to the output layer. The number of neurons in the input and output layers are determined by formulation of the problem. The number of neurons in the hidden layers define the capacity of the neural network which is related to the complexity of the underlying process in the training data set. The relationship between the number of neurons in the hidden layers, the capacity of neural networks, and the process being modelled is not very well understood in qualitative terms at the present. Input vector is provided as the activation of the neurons at the input layer. Signals propagate from the input layer, through the hidden layer, to the output layer. The activations of the output nodes constitute the output of the neural network.

In earlier publications, Ghaboussi and his co-workers have introduced a new notation to represent the feed-forward neural networks. This notation is intended to facilitate the discussion of the neural networks as well as to facilitate their use in computational mechanics. The general form of the notation is as follows.

$$\boldsymbol{F}=\boldsymbol{F}_{NN}\ (\{\text{input variabless}\} : \{NN \text{ architecture}\}) \tag{1}$$

Note that the second argument field is the network architecture which contains the number of nodes in each layer and some information about the training process. This information is an integral part of the neural network and therefore is included in the function description.

The multi-layer, feed-forward neural networks are used to establish associations (mappings) between the input vector of variables, $\boldsymbol{x}$, and the output vector function, $\boldsymbol{y}$, within the domain of the training data set $\boldsymbol{D}=\{(\boldsymbol{x}_j, \boldsymbol{y}_j),\ j=1,\ ...,\ k\}$. The mathematical expression of the problem is a function relating $\boldsymbol{x}$ to $\boldsymbol{y}$ and coinciding with the k points of the training data set in the $(\boldsymbol{x}, \boldsymbol{y})$ space. The function satisfying these conditions is the interpolation function $\boldsymbol{f}$ (.).

$$\boldsymbol{y}=\boldsymbol{f}(\boldsymbol{x}) \tag{2}$$

The parameters of the interpolation function $\boldsymbol{f}$ can be determined to satisfy the following condition.

$$\boldsymbol{y}_j=\boldsymbol{f}(\boldsymbol{x}_j);\ (\boldsymbol{x}_j, \boldsymbol{y}_j) \in \boldsymbol{D} \tag{3}$$

An example of an approximate approach to this problem is regression analysis, which uses a best fit, often posed as a minimization problem and expressed as follows. A function $\hat{\boldsymbol{f}}$ is selected,

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{f}}(\boldsymbol{x}) \tag{4}$$

and the parameters of the function are determined by minimizing the error for the training set.

$$\text{minimize } \left\| \boldsymbol{y}_j - \hat{\boldsymbol{f}}(\boldsymbol{x}_j) \right\| \text{ for all } (\boldsymbol{x}_j, \boldsymbol{y}_j) \in \boldsymbol{D} \tag{5}$$

For the function in Eq. (2) the neural network representation is given by the following equation, where the network architecture field is left blank, signifying a generic neural network.

$$\boldsymbol{y} = \boldsymbol{y}_{NN}(\boldsymbol{x}:) \tag{6}$$

Such a neural network is trained with the training set, such that within the domain of the training data it approximately represents the training data set. The approximation in neural networks is represented by the error vectors $\boldsymbol{e}_j$ within the domain of the training data. Training of the neural network is the process of reducing the norm of the error vector to a level below a tolerance.

$$\left\| \boldsymbol{e}_j \right\| = \left\| \boldsymbol{y}_j - \boldsymbol{y}_{NN}(\boldsymbol{x}_j:) \right\| \le \varepsilon; \quad (\boldsymbol{x}_j, \boldsymbol{y}_j) \in \boldsymbol{D} \tag{7}$$

A neural network learns to satisfy its training data set approximately. It differs fundamentally from a mathematical interpolation function, which matches the training data set exactly. Neural networks are also different from a regression analysis, which requires a specified function whose parameters are determined.

In general, the output errors for the training data depend on a number of factors relating to the complexity of the underlying process represented in the training data set, as well as the network architecture and the training process. It is important to note that it is not desirable to reduce the error too much. In the limit, if the output error is reduced to zero, then the neural network would be functioning similar to an interpolation function and it will lose its generalization capability between the points in the training data set. The generalization capability of neural networks is the direct consequence of the imprecision tolerance.

2.2 Genetic algorithm as a soft computing tool

Genetic algorithm is a computational model based on natural evolution (Holland 1975). In its simplest form, it is used as a powerful optimization tool, although we will see later that it has capabilities well beyond simple optimization. A system to be optimized is represented by a binary string which encodes the parameters of the system. A population of strings with initial random parameters are used. A number of generations are simulated with operators representing the major elements of evolution such as competition, fitness based selection, recombination, and mutation. The whole process is highly random. However, the evolutionary pressures lead to fitter individuals in the population and move the fitter individuals in the population closer to satisfying the objective functions of the optimization problem.

Genetic algorithms have all the characteristics of soft computing approaches to optimization. The methodology is highly robust and imprecision tolerant. The parameters of the problem, including the definition of the fitness, are not required to be exact and the results have a degree of approximation. If a unique optimum solution exists, the method approaches it through gradual improvement of the fitness. If the optimum solution is not unique, the method will approach one of the optimum

solutions. More general formulations of the genetic algorithm applied to design problems can operate in infinite dimensional solution space where optimum solutions do not exist. In such cases, the process of continually improving the solutions leads to designs which are also locally optimized.

3. Structural design

Engineering design involves an ad hoc combination of experience-based preliminary design, iterative improvements, and some form of optimization. The whole design process uses analysis and simulation to aid in the decision making in the intermediate and final steps of the design process. The design process can be divided into three steps: preliminary design, final design, and optimization. Optimization is a key component of engineering design, and its role in the design process requires a careful examination. In the current practice, the optimization step is often uncoupled from the design process and is performed after each stage of the design has been reached.

3.1 Soft computing approaches to optimization

The conventional mathematically based optimization methods, such as dynamic programing and nonlinear programing, seek to find the exact optimum solution, and often the intermediate steps do not contain useful information. This is the fundamental difference between the mathematically based hard computing optimization methods and the biologically inspired soft computing approaches to optimization, which rely on incremental improvement as an optimization strategy, without seeking the exact optimum. Nature has evolved two effective and robust methods for optimization. One method is the evolution itself. The second method uses reduction of disorder as an optimization strategy. Computational models have been developed using both of nature's optimization strategies. They work with incremental improvement and produce locally near-optimal solutions. In nature, near-optimal solutions are sufficient, a property which is also shared by most engineering problems. Often the difference between the exact optimum and the near optimum solution is small enough to be of no practical value.

When the problem is posed as a simple optimization, then the number of variables are fixed and the search is in a finite dimensional vector space. In such cases one or more optimal solutions do exist. When the problem is posed as a design problem, then the search takes place in an infinite dimensional vector space. As will be discussed later, optimal states may not exist in the infinite dimensional design solution space.

In the overwhelming majority of the applications, evolutionary computational models, such as genetic algorithm, have been used as optimization methods. However, these methods are not strictly optimization tools. They have potential for far more. Evolutionary methods use improvement as the basic strategy and they seek to improve the system's adaptation to their virtual environment, defined through the fitness function. As will be discussed later, the current formulations of the evolutionary computational methods are more suited for the limited applications in optimization. They lack the basic tools for dealing with the much broader design problem. We will show some examples of what is needed to develop the full potential of these methods in design.

Another strategy nature uses in optimization is based on the reduction of disorder or increase of the entropy of the system. It is proposed, without proof, that optimal or near-optimal solutions correspond to the states of minimum disorder or maximum entropy. Strategies which seek to

incrementally reduce the disorder in the system, gradually approach a near-optimal solution. In addition to genetic algorithm, this strategy is present in the methods which use Hopfield nets and Kohonen's self-organizing maps for optimization (Kohonen 1990, Hertz, Krogh, and Palmer 1991, Ritter, Martinetz, and Schulten 1992).

3.2 Genetic algorithm in design

As was mentioned earlier, design is an inverse problem in the sense that we are seeking a solution from among infinite numbers of possible solutions to satisfy the design specification and at the same time to minimize the cost. Genetic algorithm can be considered a tool for solving this inverse problem. However, to use genetic algorithm for design, we must move beyond the current formulations in which the variable field is fixed and predefined. Since the evolution of the design will cause changes in the number and meaning of the variables, they cannot be defined and fixed a-priori. Genetic algorithm must be formulated with self-organizing capability, so that the variable field itself can evolve. In these formulations of genetic algorithm the redundancy will play an important role. The strings must have enough redundant capacity to allow emergence of new variables.

Self-organizing formulations of genetic algorithms with redundant capacity are also biologically plausible. Geneticists believe that the useful information is contained in less than 5 per cent of the genes. The rest are redundant information. Also, the useful information in the genes has not been pre-defined. It has evolved during the process of the evolution of the organism. Redundant capacity is both a product of the evolution of the useful information in the gene and a necessity for further evolution of the new information in the gene.

Implicit Redundant Representation Genetic Algorithm (IRRGA) is the first major step in developing self-organizing genetic algorithms suitable for design problems (Raich and Ghaboussi 1997). The variable field in IRRGA is not specified, it evolves. A typical string in IRRGA is shown in Fig. 1. The gene instances, which encode the variables of the problem, emerge from a background of redundant segments. The emergence of each gene instance is indicated by the appearance of prescribed sequence of bits, referred to as the Gene Locator Pattern. To determine the parameter values, the string is parsed from left. When a prescribed Gene Locator pattern is encountered, then the next several bits are taken as the gene and it encodes a specific sequence of variables. The Gene Locator pattern is a specified sequence of bits, such as [1, 1, 1]. The remaining portions of the string, outside the genes, are treated as redundant. The IRRGA has been remarkably successful in structural design problems (Raich 1998, Raich and Ghaboussi 1999) and it has performed better than simple genetic algorithm in other applications (Chou and Ghaboussi 1998).

The results of the application of the IRRGA to the design of a three story steel moment resisting plane frame are shown in Fig. 2. The frames are required to evolve within a prescribed physical

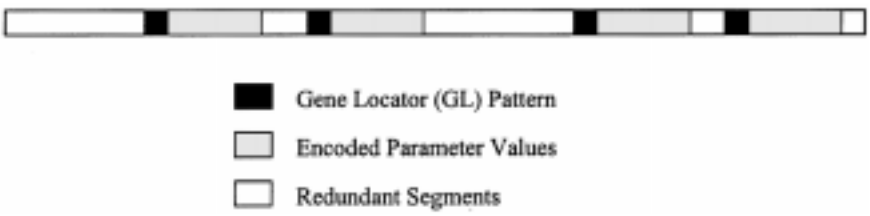

Fig. 1 A typical string in implicit redundant representation genetic algorithm

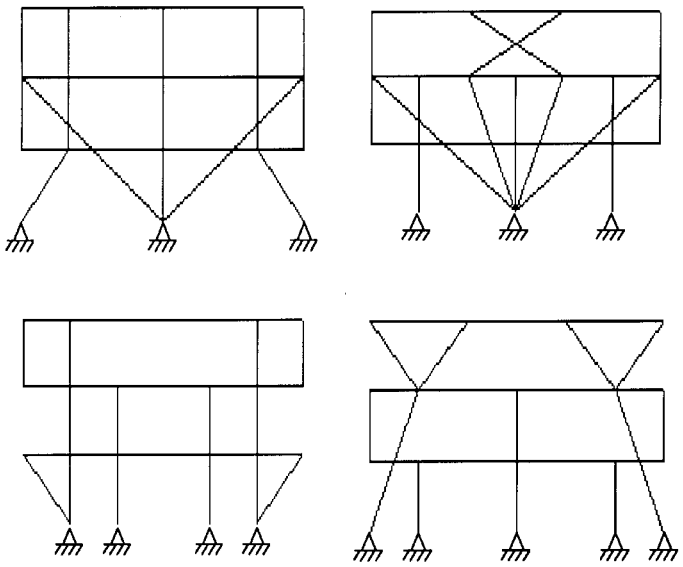

Fig. 2 Four frame designs which evolved by using the IRR genetic algorithm (Raich and Ghaboussi 1999)

region, the floor levels are specified, and the loads (dead load, live load, and wind load) are given. All the design code restrictions are assumed to apply. Each of the four frames is the result of the application of the IRRGA, starting from a random initial state. Conventional plane frames usually consist of vertical columns and horizontal girders, with occasional inclined members used as lateral bracing. Very little innovation would seem possible in such a simple problem as a three story plane frame. However, the four frames shown in Fig. 2 indicate that the IRRGA was able to evolve highly innovative designs with creative features. Some of the designs combine truss and frame actions, while others place an arch within the frame. In both cases, members are employed to carry axial force as much as possible. It can also be seen that all the frames exploit the cantilever sections to reduce bending moments at the center spans.

Genetic algorithm has also been applied to the design of trusses. In these applications the topology and the shape of the truss is not specified; they evolve (Shrestha and Ghaboussi 1998). An example is shown in Fig. 3. The span of the truss and the location of the loads along the bottom cord are specified and the truss is restricted to be within the rectangular region shown. Number of nodes, member connectivity and member sizes evolve. The initial configurations are generated randomly. Fig. 3 shows the fittest member at several generations. A Mitchell truss emerges after 5400 generations. This application required very long strings (more than 80,000 bits), and a large portion of the strings contained redundant segments.

The examples shown in Figs. 2 and 3 are fundamentally different from most of the standard applications of genetic algorithm. We have used the term *unstructured design* to distinguish them from the applications of genetic algorithm to optimization of structures with fixed topology and fixed configuration. As can be seen from these two examples, in the unstructured design problem only the very basic requirements such as the physical region, the loads, and the floor levels are specified and the structure itself is allowed to evolve without any outside interference.

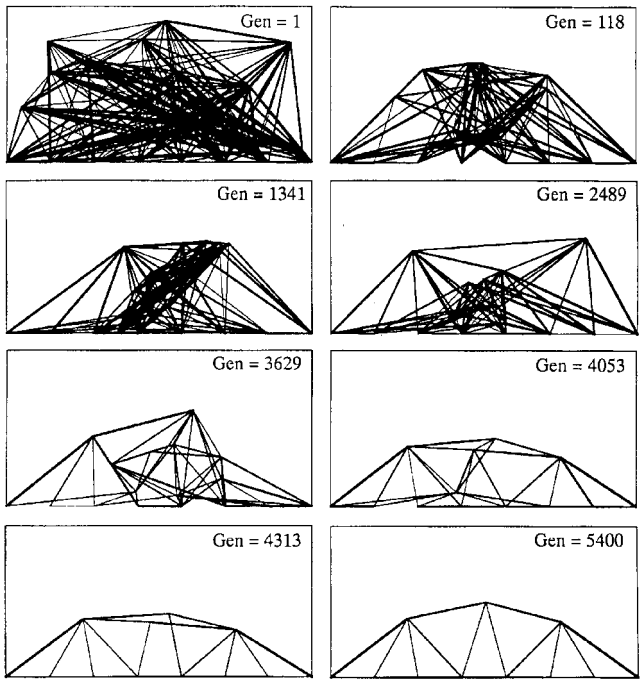

Fig. 3 Evolution of a truss design in an application of genetic algorithm to unstructured design problem (Shrestha and Ghaboussi 1998)

4. Inverse problems in engineering

Most engineering problems are inherently inverse problems. However, they are seldom solved as inverse problems. They are only treated as inverse problems on the rare occasions when they can be formulated as direct problems in highly idealized and simplified forms. Nearly all the computer simulations with hard computing methods, including finite element analyses, are direct or forward problems, where a physical phenomenon is modeled and the response of the system is computed. In the most common inverse problems the model of the system and its output are known. The inverse problem is then formulated to compute the input which produced the known output. In other cases, the input and the output of the system are known, and the inverse problem is formulated to determine the characteristics of the system. These classes of problems are called the system identification. We will also include design in the category of inverse problems. Design is a special type of inverse problem, in that it seeks a solution which satisfies the design specifications, including esthetics. In the design problem, we seek the system itself, and the design specifications impose some constraints on the output of the system (response of the structure) for some known input patterns (loads).

An important characteristic of the inverse problems is that they often do not have mathematically unique solutions. The measured output (response) of the system may not contain sufficient information to uniquely determine the input to the system. Similarly, the measured input and response of the system may not contain enough information to uniquely identify the system, and there may be many solutions which satisfy the problem. This is especially the case in the

unstructured design problems, where there are often many near optimal designs which satisfy the design specifications and other constraints.

Lack of unique solutions is one of the reasons why the mathematically based methods are not suitable for inverse problems. On the other hand, biological systems have evolved highly robust methods for solving the very difficult inverse problems encountered in nature. In fact, most of the computational problems in biological systems are in the form of inverse problems. The survival of the higher level organisms in nature depends on their ability to solve these inverse problems. The examples of these inverse problems are recognition of their food, recognition of threats, and paths for their movements. Nature's basic strategy for solving the inverse problems is to use approximate and imprecision tolerant learning within a domain of interest, thus forgoing precision and the universality of mathematically based methods.

Researchers in neural networks know that a set of training data, which has been used to train a neural network, can also be used to train an inverse neural network. In this case, we are justified to call the first neural network a direct neural network. Then the input and output of the inverse neural network are the same as the output and input, respectively, of the direct neural network. These neural networks can be shown in the following equations.

$$\boldsymbol{y} = \boldsymbol{y}_{NN}\,(\boldsymbol{x}\colon) \tag{8}$$

$$\boldsymbol{x} = \boldsymbol{x}_{NN}\,(\boldsymbol{y}\colon) \tag{9}$$

However, our experience in applying neural networks in material modeling (Ghaboussi, Garrett, and Wu 1991) showed that the learning process for these two neural networks is not the same. If the data has been generated by measuring the response (output) of a physical process to a given stimulus (input), then the inverse neural network, which tries to determine the stimulus from the response, may be far more difficult to train. In material tests the stimulus and response are often the stresses and strains, respectively. In most material tests stresses σ are applied to the specimen and strains ε are measured. The direct neural network material model is given in the following equation.

$$\boldsymbol{\varepsilon} = \boldsymbol{\varepsilon}_{NN}\,(\boldsymbol{\sigma}\colon) \tag{10}$$

Although the material test is performed by measuring the strains at each applied stress level, the material data would have enough information to train an inverse neural network, which determines the stresses from strains.

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}_{NN}\,(\boldsymbol{\varepsilon}\colon) \tag{11}$$

However, the inverse neural network needed more nodes in the hidden layers than the direct neural network. This was clear evidence that the inverse problem was far more difficult for the neural network to learn than the direct problem.

When a unique inverse relationship does exist, then both neural networks and mathematically based methods can be used to model the inverse mapping. However, when the inverse mapping is not unique, then modeling with the mathematically based methods becomes increasingly difficult, if not impossible. On the other hand, neural network based methods can deal with the non-unique inverse problems by using the learning capabilities of the neural networks. In fact, this is how the biological systems solve inverse problems, through learning. Even if mathematically unique inverse mappings do not exist, approximate (imprecision tolerant) inverse mappings may exist over regions of domain and range. Neural networks use learning to establish an admissible inverse mapping.

Modern biologically inspired soft computing methods seem to display characteristics suitable for operating in the domains where mathematical certainty is not necessary. These methods have robust capabilities to formulate the problem in a way that reasonable and admissible solutions can be obtained even when we can show that mathematically unique relationships do not exist.

The key to the ability of biological systems to find reasonable solutions to the inverse problems lie in learning and being able to generalize what has been learned. The soft computing methods applied to inverse problems produce solution which are only valid over limited ranges of these variables. This is in contrast to the mathematical based methods which rely on the universal mathematical proofs and strict uniqueness requirements in producing solutions which must remain valid over all the possible ranges of variables. Soft computing methods address a particular need which applies equally well to the decision making in engineering design, as it does apply to decision making in nature by biological systems. In both cases, mathematically exact and unique solutions are not required. The difference between the mathematically exact and unique solutions and a good admissible solution is often of little consequence for the outcome of the decision making process. Soft computing techniques seem to display the same imprecision tolerant characteristics which makes them useful tools for finding good admissible solutions within a limited range of the variables and parameters. The restriction in the range of variables and parameters is also based on a practical imperative. The universality of the mathematically precise methods are of little value in most practical problem solving.

4.1 Autoprogressive neural networks in material modeling from structural tests

Neural networks material models were first introduced by the author and his co-workers in 1991 (Ghaboussi, Garrett, and Wu 1991). Typically, the neural network material models relate the state of stresses to the state of strains. The input may also include some past history of stresses and strains to account for the path dependence of the material behavior.

$$\boldsymbol{\varepsilon}_j = \boldsymbol{\varepsilon}_{jNN} \left(\boldsymbol{\sigma}_j; \boldsymbol{\varepsilon}_{j-1}; \boldsymbol{\sigma}_{j-1}; ...; \boldsymbol{\varepsilon}_{j-k}; \boldsymbol{\sigma}_{j-k}; \right) \tag{12}$$

The subscript in this equation represents the loading steps. In the material model shown in this equation the past history of loading is represented by the past k states of stresses and strains to represent the path dependence of the material behavior.

The latest research in this area has led to the development of Nested Adaptive Neural Networks (Ghaboussi, Zhang, Wu, and Pecknold 1997, Ghaboussi and Sidarta 1998), which takes advantage of the nested structure of the material data. Two types of nested structure are identified in the material data. The nested structure based on dimensionality arises from the observation that a lower dimensional material behavior is a subset of the higher dimensional material behavior. For example, a one dimensional material behavior is the subset of the two dimensional material behavior, which in turn is a subset of the three dimensional material behavior. The same type of nest structure is also observed in the path dependence of the material behavior. In material models represented by Eq. (12), the model with lower value of k is a subset of the material model with higher value of k. The nested structure of the material data has important implications in the internal structure of the neural network material models, which has been discussed in detail in the references cited above.

Constitutive modeling can be viewed as the inverse problem of determining a model which matches the input and output of a material test. A more complex inverse problem is the determination of the constitutive model from the force and displacement measurements on an structural test.

Unlike material tests, which ideally induce a uniform state of stress within the sample, structural tests induce non-uniform states of stresses and strains within the sample. It is obvious that the response of any structural test contains information on the constitutive behavior of the material. Moreover, since points in the specimen follow different stress paths, a structural test potentially has far more information on the material behavior than a material test in which all the points follow the same stress path.

Determination of the material properties from structural tests is an extremely difficult problem with the conventional mathematically based methods. This is probably the reason why it has not been attempted in the past. On the other hand, soft computing methods are ideally suited for this very difficult inverse problem. The learning capabilities of neural network offer a solution to this problem.

The problem of determining the constitutive properties of material from structural tests is becoming more relevant with the wider use of modern composite material. It is often not feasible to perform comprehensive sets of material tests on composite materials. However, structural tests on composite materials are far more common. The author and his co-workers have developed the Autoprogressive method for training the neural network constitutive models from the results of structural tests (Ghaboussi, Pecknold, Zhang, and HajAli 1998). Initially a neural network material model is pretrained with an idealized material behavior (usually linearly elastic material properties). This neural network is then used to represent the material behavior in a finite element model of the specimen in the structural test. The Autoprogressive method simulates the structural test through a series of load steps. Two analyses are performed in each load step. The first analysis is a standard finite element analysis, where the load increment is applied. In the second analysis a displacement correction is imposed, which is the difference between the computed displacements from the first analysis and the measured displacement. The results from these two analyses are then used to further train the neural network material model. A number of iterations may be required in each load step. When the incremental and iterative process of autoprogressive train is completed, the neural network material model learns the material behavior, so that the finite element model will satisfy both the displacement and force boundary conditions.

An application of the Autoprogressive methodology from (Ghaboussi, Pecknold, Zhang, and HajAli 1998) is shown in Fig. 4. The specimen shown is a composite plate with a central whole, which is clearly a structural test with non-uniform state of stresses within the sample. Applied axial force and the elongation of the sample are the only measured quantities. A finite element model of the specimen is constructed in which the material behavior at the lamina level is modeled by a neural network. It is obvious that a material test cannot be performed on a lamina of the composite material. In the Autoprogressive method the measured forces and displacements are imposed in a series of dual finite element analyses and the neural network is trained so that the computed values match the measured quantities. The force-displacement curves from the finite element analysis using the autoprogressively trained neural network material at the end of the first two passes are shown in Fig. 5, and they are compared with the measured force-displacement curve. The results clearly indicate that the neural network material model has learned the material behavior from the result of the structural test. Additional results on the Autoprogressive training of neural network material models are given in the reference cited above.

The Autoprogressive method has also been applied in geotechnical engineering to model the behavior of geo-materials (Sidarta and Ghaboussi 1998). The Autoprogressive method has wider applications than material modelling from structural tests. The measured input and output of any

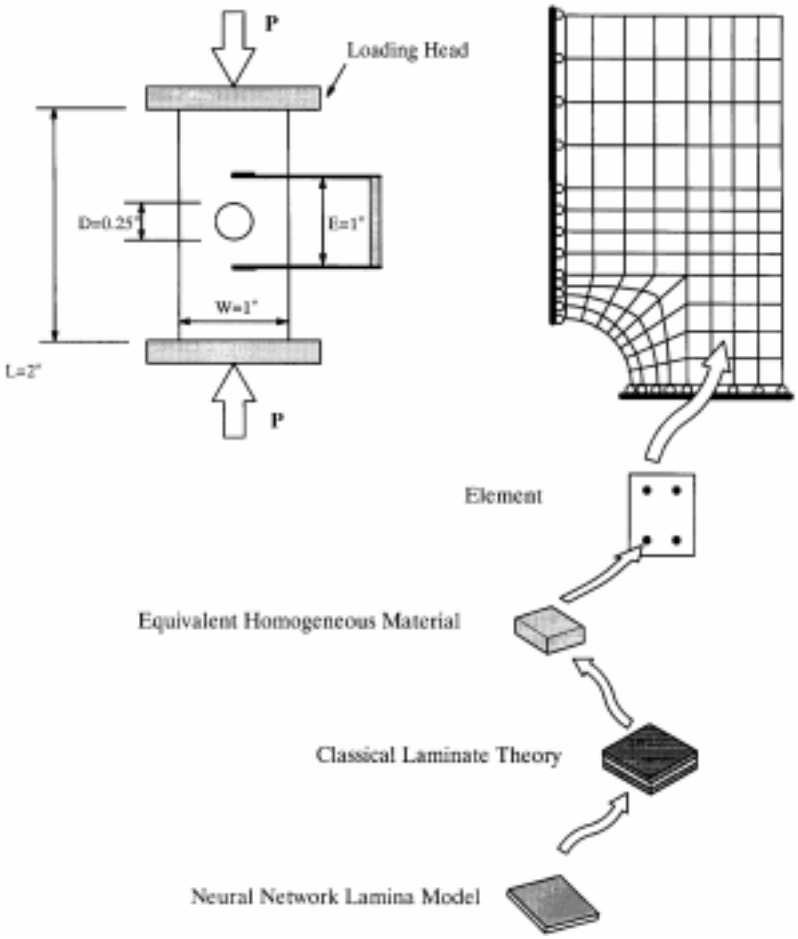

Fig. 4 Test setup (Lessard and Chang 1991), FE model, and results of the Autoprogressive training of material model for a fiber reinforced composite plate (Ghaboussi, Pecknold, Zhang, and HajAli 1998)

system can be used to develop a neural network model of the behavior of its components. The potential application of the Autoprogressive method in characterization of the components of complex systems is described in (Ghaboussi, Pecknold, and Zhang 1998).

4.2 Inverse problem of generating artificial spectrum compatible accelerograms

This is very similar to the inverse problems which the organisms solve in nature. As such, the soft computing methods should be suitable for dealing with this problem. The author and his co-workers have developed a neural network based method for generating artificial spectrum compatible accelerograms (Ghaboussi and Lin 1998, Lin and Ghaboussi 2000). The objective is to train a neural network with an ensemble of recorded earthquake accelerograms. The input to the neural network will be a vector of the discretised ordinates of the response spectrum. The output of the neural network is the vector of discretised ordinates of the real and imaginary parts of the Fourier spectrum. As mentioned earlier, the accelerogram can then be uniquely computed from the Fourier

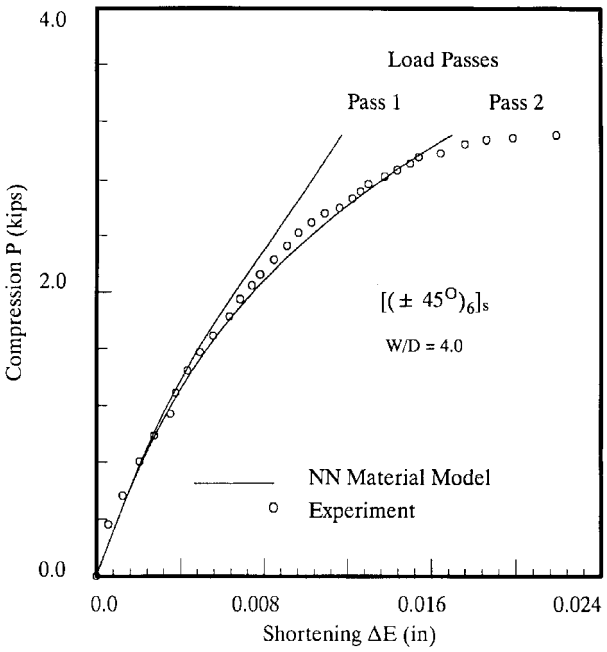

Fig. 5 Load vs Shortening from forward analysis, using the neural network lamina material model trained on $[(L45)_6]_s$. Experimental Data (Ghaboussi, Pecknold, Zhang, and HajAli 1998)

spectrum.

The neural networks are trained in two stages. First, a Replicator Neural Network (RNN) is trained to compress the information content of the Fourier spectra of the accelerograms in the training data sets. As shown in Fig. 6, the input and output of the RNN are the same. The middle hidden layer of RNN has very few nodes. The activations of the middle hidden layer represent the compressed data. It is obvious that the RNN is performing two distinct functions of encoding and decoding in such a way that it can be separated into two neural networks. The lower part of RNN (input layer to the middle hidden layer) can be considered as a encoder neural network. Similarly the upper part of the RNN (from the middle hidden layer to the output layer) can be considered as a decoder neural network. The upper part of the trained RNN is used in the accelerogram Generator Neural Network (AGNN) shown in Fig. 7. The lower part of the AGNN is then trained with a number of recorded earthquake accelerograms. Thirty recorded accelerograms were used to train the AGNN. The results of a test of the trained neural network is shown in Fig. 8. A smoothed design spectrum is used as the input to the AGNN. The generated accelerogram and comparison of its response spectrum with the input response spectrum are also shown in Fig. 8. It is important again to note that this problem does not have a unique solution. The generated accelerogram is one of many possible accelerograms. It is interesting to note that the neural network learns to generate a very realistic looking artificial accelerogram.

The methodology has been extended to develop and train stochastic neural networks which are capable of generating multiple accelerograms from a single input design response spectrum (Lin and Ghaboussi 2000).

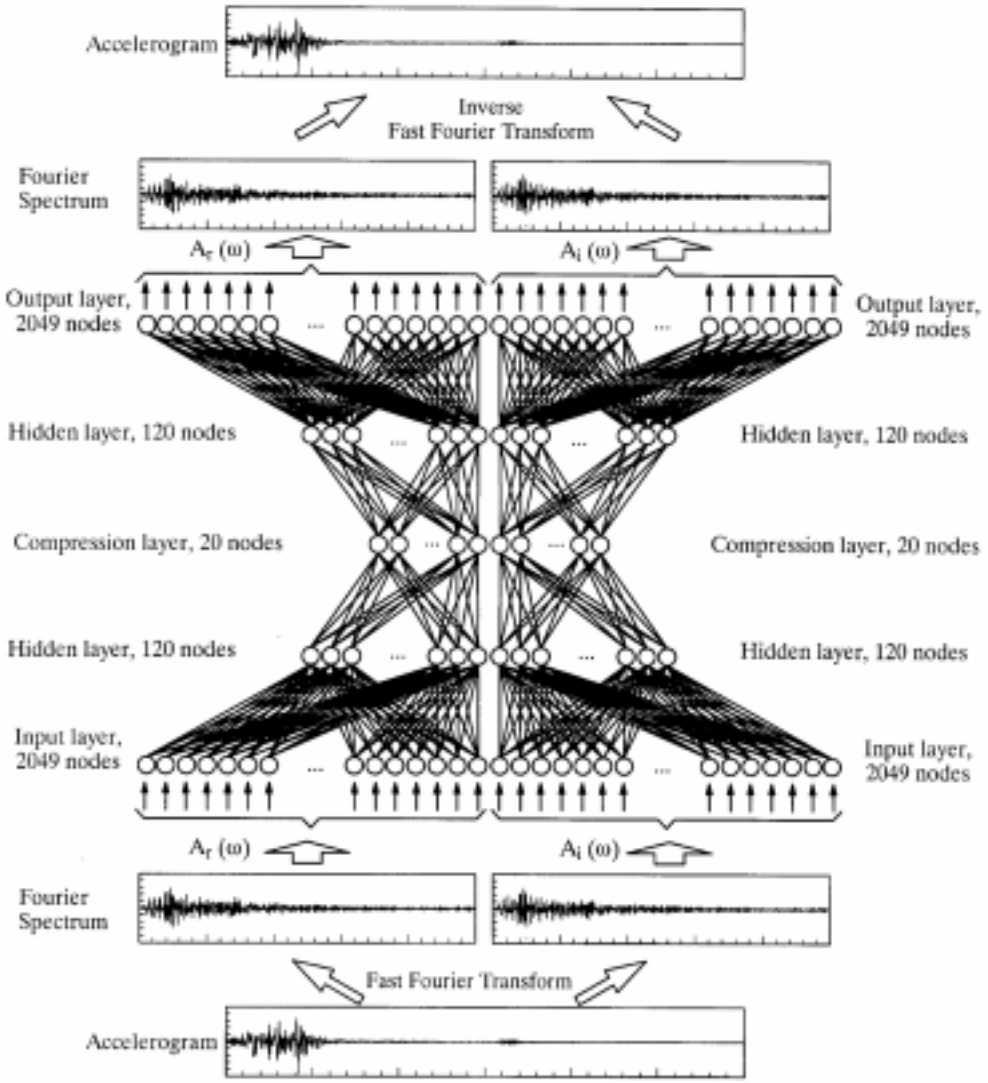

Fig. 6 Replicator neural network used for data compression of the fourier transforms of earthquake accelerograms (Ghaboussi and Lin 1998)

4.3 Condition monitoring and damage detection in bridges

Another obvious class of inverse problems is the condition monitoring and damage detection from the measured response of the structure. Soft computing methods again offer unique opportunities for dealing with this class of difficult problems. The author and his co-workers have applied genetic algorithm in bridge damage detection and condition monitoring (Chou and Ghaboussi 1998) and have applied neural networks in damage detection in railway engineering (Ghaboussi, Banan, and Florom 1994, Chou, Ghaboussi, and Clark 1998). The bridge condition monitoring is formulated as an optimization problem. The error between the measured response and the response computed from a model of the bridge are minimized and in the process the condition of the bridge is

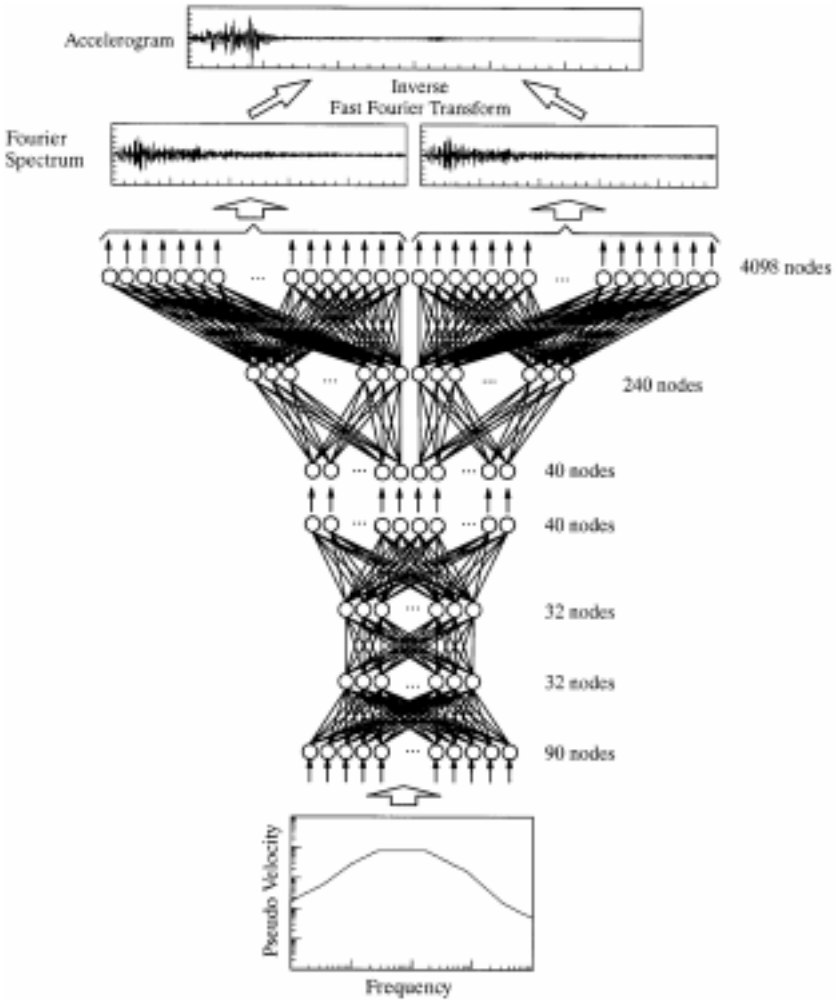

Fig. 7 The accelerogram generator neural network (Ghaboussi and Lin 1998)

determined. A general solution of this optimization problem with traditional optimization methods is extremely difficult. Genetic algorithm is ideally suited for this problem, because of its high degree of flexibility in formulating the problem and encoding its parameters.

5. Conclusions

A fundamental review of the modern computational methods, such as neural networks and genetic algorithm, has been presented within the context of biologically inspired soft computing methods. The emphasis has been placed on describing the main differences between the modern biologically inspired computational methods and the conventional mathematically based methods. The author believes that the understanding of these differences is essential in developing the full potential of the biologically inspired soft computing methods. It has been pointed out throughout the paper that

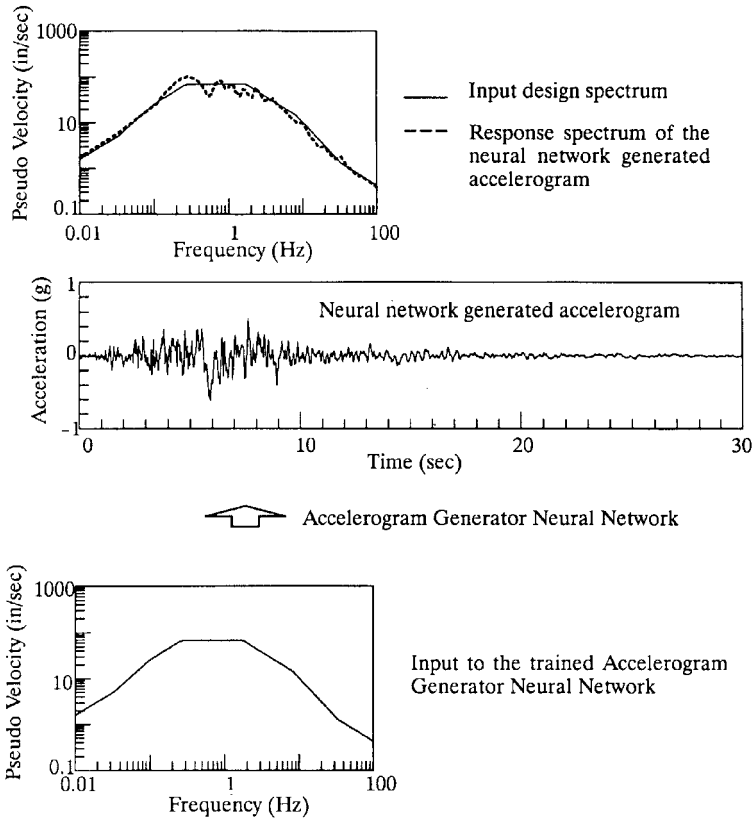

Fig. 8 Testing of the trained Accelerogram Generator Neural networks with a response spectrum (Ghaboussi and Lin 1998)

the primary difference is the imprecision tolerance of the soft computing methods. The major consequences of the imprecision tolerance have also been discussed. Nature uses the imprecision tolerance to develop very effective strategies for solving difficult inverse problems. The imprecision tolerant soft computing tools, such as neural networks and genetic algorithm, can be effectively used in problem solving strategies similar to those deployed in nature to solve difficult inverse problems in engineering, including problems in engineering design. These capabilities have been illustrated through several examples of the research work of the author and his co-workers.

Acknowledgments

This paper is based on my research on the development and application of soft computing methods over the past fifteen years. I have benefited from working with many of my current and former doctoral students on these problems. I have also benefited from the opportunity to collaborate with a number of my colleagues. I gratefully acknowledge numerous industrial grants and at least 5 NSF grants for research on soft computing methods.

References

Chang, F-K., and Lessard, L. (1991), "Damage tolerance of laminated composites containing an open hole and subjected to compressive loadings. Part I: Analysis", *J. Composite Mat.*, **25**, 2-43.

Chou, J.H., and Ghaboussi, J. (1998), "Studies in bridge damage detection using genetic algorithm", *Proc., 6th East Asia-Pacific Conf. Struct. Eng. & Constr.*, Taiwan.

Chou, J.H., Ghaboussi, J., and Clark, R. (1998), "Application of neural networks to the inspection of railroad rail", *Proc., 25th Ann. Conf. Review Progr. Quantit. NDE*, Snowbird, Utah.

Ghaboussi, J., Garrett, J.H., and Wu, X. (1991), "Knowledge-based modeling of material behavior with neural networks", *J. Eng. Mech.*, ASCE, **117**(1), 132-153.

Ghaboussi, J., Banan, M.R., and Florom, R.L. (1994), "Application of neural networks in acoustic wayside fault detection in railway engineering", *Proc., W. Cong. Railway Research*, Paris, France.

Ghaboussi, J., Zhang, M., Wu, X., and Pecknold, D.A. (1997), "Nested adaptive neural networks: A new architecture", *Proc., Int, Conf. on Artificial NN Eng.*, St. Louis, Mo.

Ghaboussi, J., and Sidarta, D.E. (1998), "New nested adaptive neural networks (NANN) for constitutive modeling", *Int. J. Comput. and Geotech.*, **22**(1), 29-52.

Ghaboussi, J., and Lin, C.-C.J. (1998), "A new method of generating earthquake accelerograms using neural networks", *Int. J. Earthquake Eng. & Struct. Dyn.*, **27**, 377-396.

Ghaboussi, J., Pecknold, D.A., and Zhang, M. (1998), "Autoprogressive training of neural networks in complex systems", *Proc., Int, Conf. on Artificial NN Eng.*, St. Louis, Mo.

Ghaboussi, J., Pecknold, D.A., Zhang, M., and HajAli, R. (1998). "Autoprogressive training of neural network constitutive models", *Int. J. Numer. Meth. Eng.*, **42**, 105-126.

Ghaboussi, J., and Wu, X. (1998), "Soft computing with neural networks for engineering applications: fundamental issues and adaptive approaches", *Struct. Eng. and Mech., An Int. J.*, **6**(8), 955 -969.

Hertz, J., Krogh, A., and Palmer, R.G. (1991), *Introduction to the Theory of Neural Computations*, Addison-Wesley, Redwood City, California.

Holland, J.H. (1975), *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor, Michigan.

Kohonen, T. (1990), "The self organizing map", *Proc. of the IEEE*, **78**(9), 1464-1480.

Lin, C.-C.J., and Ghaboussi, J. (2000), "Generating multiple spectrum compatible accelerograms using stochastic neural networks", *Int. J. Earthquake Eng. & Struct.*, to appear.

Raich, A.M. (1998), "An evolutionary based methodology for representing and evolving structural design solutions", Ph.D thesis, Dept. of Civil and Environmental Engineering, Univ. of Illinois at Urbana-Champaign, Urbana, Illinois.

Raich, A.M., and Ghaboussi, J. (1997), "Implicit representation in genetic algorithm using redundancy", *Int. J. Evolutionary Computing*, **5**(3).

Raich, A.M., and Ghaboussi, J. (1999), "Evolving structural design solutions using an implicit redundant genetic algorithm", *Proc., Genetic and Evol. Comp. Conf.*, GECCO-99, Orlando, Florida.

Ritter, H., Martinetz, T., and Schulten, K. (1992), *Neural Computation and Self-Organizing Maps*, Addison-Wesley Publishing Company, Reading, Massachusetts.

Shrestha, S.M., and Ghaboussi, J. (1998), "Evolution of optimal structural shapes using genetic algorithm", *J. Struct. Eng.*, ASCE, **124**(8), 1331 - 1338.

Sidarta, D.E., and Ghaboussi, J. (1998), "Constitutive modeling of geomaterials from non-uniform material tests", *Int. J. Comput. and Geotech.*, **22**(1), 53-71.