A Sequential Approximation Method Using Neural Networks for Nonlinear Discrete-Variable Optimization with Implicit Constraints*

Yeh-Liang HSU**, Yu-Hsin DONG**
and Ming-Sho HSU**

This paper presents a sequential approximation method that combines a back-propagation neural network with a search algorithm for nonlinear discrete-variable engineering optimization problems with implicit constraints. This is an iteration process. A back-propagation neural network is trained to simulate the feasible domain formed by the implicit constraints. A search algorithm then searches for the “optimal point” in the feasible domain simulated by the neural network. This new design point is checked against the true constraints to see whether it is feasible, and is added to the training set. Then the neural network is trained again. With more design points in the training set, information about the whole search domain is accumulated to progressively form a better approximation for the feasible domain. This iteration process continues until the approximate model insists the same “optimal” point in consecutive iterations. In each iteration, only one evaluation of the implicit constraints is needed to see whether the current design point is feasible. No precise function value or sensitivity calculation is required. Several engineering design examples are used to demonstrate the practicality of this approach.

Key Words: Discrete-Variable Optimization, Implicit Constraints, Neural Network

1. Introduction

Engineering design optimization problems have two characteristics that differ themselves from general mathematical programming problems: (1) their design variables are often discrete physical quantities, (2) their constraints often cannot be expressed analytically in terms of design variables. Such constraints are called implicit constraints. An engineering design optimization problem can usually be formulated as follows:

$$\begin{aligned} \min. \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}_e(\mathbf{x}) \leq \mathbf{0} \\ & \mathbf{g}_i(\mathbf{x}) \leq \mathbf{0} \\ & x_i \in X_i = \{d_{i1}, d_{i2}, \dots, d_{in}\}, i \in I_d \end{aligned} \quad (1)$$

where $\mathbf{x}$ is the vector of design variables, $f(\mathbf{x})$ is the objective function, $\mathbf{g}_e(\mathbf{x}) \leq \mathbf{0}$ and $\mathbf{g}_i(\mathbf{x}) \leq \mathbf{0}$ are con-

straints; I_d is the set of design variables that can take only discrete values, and X_i is the set of allowable discrete values for variable x_i .

There are two types of constraints in the formulation: $\mathbf{g}_e(\mathbf{x}) \leq \mathbf{0}$ are the explicit constraints which can be expressed explicitly in terms of design variables (e.g., the upper and lower bounds of the design variables); $\mathbf{g}_i(\mathbf{x}) \leq \mathbf{0}$ are the implicit constraints. In an engineering design problem, typical examples of implicit constraints are those on structural responses such as stress, displacement, and natural frequency, and constraints involving manufacturing parameters, measurement data, etc.

The implicit constraints are usually evaluated by computer simulation such as finite element analysis, or by experiments. Thus the evaluation of the implicit constraints is usually the major cost of the optimization process. Moreover, the analytical forms of the gradients of the implicit constraints with respect to the design variables are usually not available. Sensitivity of such constraints often has to be calculated through finite difference schemes, which requires

* Received 18th June, 1999

** Department of Mechanical Engineering, Yuan Ze University, 135 Yuan-Tung Rd., Nei-Li, Chung-Li, 320 Taiwan, R.O.C. E-mail: mehsu@saturn.yzu.edu.tw

many function evaluations, and often have problems with accuracy.

One important category of numerical optimization algorithms is the sequential approximation methods. The basic idea of sequential approximation methods is to use a “simple” sub-problem to approximate the hard, exact problem. By “simple” sub-problem, we mean the type of problems that can be readily solved by existing numerical algorithms. For example, linear programming sub-problems are widely used in sequential approximation methods. The solution point of the simple sub-problem is then used to form a better approximation to the hard, exact problem for the next iteration. In an iterative manner, it is expected that the solution point of the simple approximate problem will get closer to the optimum point of the hard exact problem. For continuous nonlinear optimization problems, the well-known sequential linear programming which utilizes linear programming sub-problems, and sequential quadratic programming which utilizes quadratic programming sub-problems, both fall into this category.

One major disadvantage of applying numerical sequential approximation methods to engineering design optimization problems is that, these are usually derivative-based approximation methods which require at least the first derivatives of the implicit constraints with respect to the design variables. Moreover, only the local numerical information from the solution point of the previous iteration is used to form the next approximation. Knowledge about the whole search domain is not accumulated for formulating better approximation, or for better understanding of the optimization model. Precious function evaluations and sensitivity calculations of the implicit constraints are often not most efficiently utilized.

Non-linear, discrete variable optimization is a well studied field. Lu and Siddall⁽¹⁾ pointed out that two general approaches can be identified in research work on discrete programming problems. The first attempts to find methods that treat the variables as discrete from the beginning. The second approach is to first approximate an optimum by treating all variables as continuous and then using some rounding technique to find the best feasible discrete point in the region of the continuous variable optimum. Both approaches tend to be adaptations of continuous variable methods. Thanedar and Vanderplaats⁽²⁾ further classified the discrete variable optimization methods into three categories: branch and bound, approximation using branch and bound, and ad-hoc methods, including simulated annealing and genetic algorithms.

This paper presents a “Sequential Neural-Network Approximation Method (the SNA method),”

which attempts to handle discrete variable optimization problems with nonlinear implicit constraints. It is also a sequential approximation method that combines back-propagation neural networks with a search algorithm. In this method, first a back-propagation neural network is trained to simulate a rough map of the feasible domain formed by the implicit constraints using just a few training data. A training data consists of a design point and whether this design point is feasible or infeasible. A search algorithm then searches for the “optimal point” in the feasible domain simulated by the neural network. This new design point is checked against the true constraints to see whether it is feasible, and is then added to the training set. The neural network is trained again with this added information, in the hope that the network will better simulate the boundary of the feasible domain of the true optimization problem. Then we search for the “optimal point” in this new approximated feasible domain again. This process continues in an iterative manner until the approximate model insists the same “optimal point” in consecutive iterations.

Hopfield and Tank⁽³⁾ first extended neural network application to optimization problems. In the Hopfield model, the optimization problem is converted into a problem of minimizing an energy function formulated by the design restraints. Lee and Chen⁽⁴⁾ further explored the idea of applying neural network technology in design optimization. In their approach, neural networks were used to simulate the constraint functions. Oda and Mizukami⁽⁵⁾ developed an application of a “mutual combination-type” neural network to the optimum design problems of truss structure. In their work, the Hopfield model and the quadratic energy function are used to solve the design problems, and the method is able to obtain feasible solutions for several optimum design problems.

The SNA method presented in this paper is attractive in two ways over the numerical sequential approximation methods. First, in each iteration, only one evaluation of the implicit constraints is needed to see whether the current design point is feasible. No precise function value or sensitivity calculation is required. Therefore it is especially suitable for engineering optimization problems with expensive implicit constraints. Secondly, in the SNA method, precious function evaluations of the implicit constraints are accumulated and utilized repeatedly to form better approximation to the feasible domain.

One key point of the SNA method is that, the computational cost of the training of the neural network has to be relatively small comparing with that of the function value and sensitivity calculations of

the implicit constraints. Otherwise using neural network in optimization will not have much advantage computationally over the traditional numerical optimization algorithms, in the cases which traditional algorithms are still applicable. As will be illustrated in this paper, in the SNA method, all training data in the input, output pairs have values of either 0 or 1. This clear 0-1 pattern makes the training process relatively fast.

Arora et al.⁽⁶⁾ also reviewed the methods of nonlinear problems with discrete variables. Their conclusion was, none of the methods are guaranteed to produce the global minimizer for nonlinear problems, but “good and practical” solutions can be obtained. The optimality of the “optimal” design point obtained by the SNA method cannot be proved. The quality of the “optimal” design point really depends on the quality of the feasible domain simulated by the neural network. However, several engineering design examples are used in this paper to demonstrate the practicality of this approach. The following sections will further elaborate on the basic ideas of the SNA method.

2. Sequential Approximation Using Neural Networks

The basic concept of the SNA method presented here is exactly the same as that of the numerical sequential approximation techniques discussed in the previous section. Equation (1) in the previous section can be thought of as a hard, exact problem because of the existence of the implicit constraints. The idea is to use a neural network to simulate the boundary of the feasible domain formed by the implicit constraints. The hard, exact problem in Eq. (1) is approximated by a “simple” approximate problem below

$$\begin{aligned} \min. \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & \mathbf{g}_e(\mathbf{x}) \leq 0 \\ & NN(\mathbf{x}) = 0 \end{aligned} \quad (2)$$

where the binary constraint $NN(\mathbf{x})=0$ approximates the feasible domain of the hard, exact implicit constraint $\mathbf{g}_i(\mathbf{x}) \leq 0$. If $NN(\mathbf{x})=0$, the design point $\mathbf{x}$ is feasible; if $NN(\mathbf{x})=1$, the design point $\mathbf{x}$ is infeasible. The discrete variable constraints are embedded in the format of the input nodes of the neural network, as will be described in the next section. For convenience, the optimization model in Eq. (1) will be denoted M_{real} throughout the paper, and the approximate model in Eq. (2) will be denoted M_{NN} .

Figure 1 is a flow chart of the SNA method. A back-propagation neural network is trained to simulate a rough map of the feasible domain formed by the implicit constraints using just a few training data (the

Fig. 1 Flow chart of the algorithm

set of initial training data X_0), to form the initial optimization model M_{NN} . M_{NN} is a “simple” model because once the training is completed, the computational cost of evaluating $NN(\mathbf{x})$ is much less than that of evaluating the implicit constraints. A search algorithm is used to search for the solution point $\mathbf{x}^j$ of M_{NN} . This solution point is then evaluated to see whether it is feasible in M_{real} , and whether M_{NN} generates the same solution point as in the previous iteration. If the feasible solution point $\mathbf{x}^j$ is the same as $\mathbf{x}^{j-1}$, no new training point is generated and this procedure has to stop. If not, a new solution point $\mathbf{x}^j$ is added to the set of training points $\mathbf{X}$, and the neural network is trained again. With the increasing number of training points, it is expected that the feasible domain simulated by the neural network in M_{NN} will get closer to that of the exact model M_{real} , and the solution point of M_{NN} will get closer to the true optimum point.

Figure 2 shows a simple three-bar-truss problem which is commonly used in structural optimization literatures to demonstrate various optimization techniques⁽⁷⁾. It is used here as an example to illustrate the detailed procedure of the SNA method. The objective of this problem is to find the optimal values of the cross-sectional areas of the bars A_1 , A_2 , A_3 , which minimize the weight of the structure, subject to maximum stress constraints on the three bars. The horizontal force P can either act to the right or to the left, thus we have a symmetric structure in which A_1 must equal A_3 .

Substituting values of the parameters r , E , P , L , and σ_{max} assumed for the example, this optimization problem can be formulated as below:

$$\min. \quad W = 7.85(4A_1 + A_2)$$

$$\begin{aligned} \text{s.t. } g_1: & \frac{200}{(A_1+4A_2)} + \frac{25\sqrt{3}}{3A_1} \leq 1 \\ g_2: & \frac{800}{(A_1+4A_2)} \leq 1 \end{aligned} \quad (3)$$

where constraint g_1 is the maximum stress constraint for bar 1 and 3, constraint g_2 is the maximum stress constraint for bar 2. Also assume that all three bars have square cross sections, and the sizes of the raw material available from manufacturers range from $1 \times 1 \text{ mm}^2$, $2 \times 2 \text{ mm}^2$, $\dots$ to $17 \times 17 \text{ mm}^2$. Therefore we have the discrete-value constraints below:

$$A_i \in \{1, 4, 9, \dots, 289\}, i=1, 2. \quad (4)$$

Equations (3) and (4) form M_{real} of this problem. For illustrative purpose, g_1, g_2 are viewed as implicit constraints. In M_{NN} of this problem, a back-propagation neural network is trained to simulate the boundary of the feasible domain.

Figure 3 shows the architecture of the three-layer network. The size of the input layer depends on the number of variables and the number of discrete values of each variable. In this example, there are 2 design variables, with 17 discrete values each. Thus a total of 2×17 neurons are used in the input layer. Each neuron in the input layer has value 0 or 1 to represent a discrete value the corresponding variable takes, as

Fig. 2 Three bar truss

Fig. 3 The architecture of the neural network

will be explained later in this section. There is only a single neuron in the output layer to represent whether this design point is feasible (the output neuron has value 0) or infeasible (the output neuron has value 1).

In the three-bar-truss example, the center point of the search domain $(A_1, A_2) = (81, 81)$ is used as one of the initial training points. This training point is represented as shown in Fig. 4, where a blank circle represents a "0" in the node, a solid circle represents a "1" in the node. For illustrative purpose, the 34 input nodes are arranged into two rows in Fig. 4 to represent two design variables. The first 9 nodes of the first row have value 1 to represent that the first variable A_1 has the discrete value 81. The first 9 nodes of the second row also have value 1 to represent that the second variable A_2 also has the discrete value 81. Checking with M_{real} , we can determine that $(A_1, A_2) = (81, 81)$ is an infeasible design point. So the single node in the output layer has the value of 1 to represent that this point is infeasible. Similarly, Fig. 5 shows another initial training point $(A_1, A_2) = (289, 289)$, which is a feasible design point. There has to be at least one feasible design point in the set of initial training points, as the search algorithm described later has to start from a feasible design point.

The number of neurons in the hidden layer depends on the number of neurons in the input layer. There are 16 neurons in the hidden layer in this example. The transfer functions used in the hidden and output layer of the network are both log-sigmoid functions. The neuron in the output layer has a value range $[0, 1]$. After the training is completed, a threshold value is applied to the output layer when simulating the boundary of the feasible domain. In other words, given a discrete design point in the search domain, the network always output 0 (if output neuron's value is less than the given threshold) or 1

(otherwise) to indicate whether this discrete design point is feasible or infeasible.

All training data are in this clear 0 - 1 pattern, which makes the training process relatively fast. A conjugate gradient algorithm (Powell-Beale Restarts) is used for the training. In this example, the

Fig. 4 Representation of an infeasible training point $(A_1, A_2) = (81, 81)$

Fig. 5 Representation of a feasible training point $(A_1, A_2) = (289, 289)$

error goal of $1e-5$ is usually met within 100 epochs, even for cases with many training points.

3. Searching in the Feasible Domain

As described in the previous section, two design points are used as the initial training points. We can also use several evenly distributed design points as initial training points to learn a rough map of the feasible domain. After this initial training is completed, the mathematical optimization model M_{real} is transformed into M_{NN} , in which the feasible domain of the optimization model is simulated by $NN(x)$.

A search for the optimum point of M_{NN} is then started. Figure 6 is a simple flow chart of this discrete search algorithm. The search algorithm has to start from a feasible design point. If the new design point generated from the previous iteration is a feasible design point in M_{real} , it is used as the starting point in the current search. If the new design point is infeasible in M_{real} , then use the same starting point of the previous iteration in the current search. As stated in the previous section, there has to be at least one feasible design point in the set of initial training points, so that the search algorithm can always start from a feasible design point.

From the feasible starting point, the search algorithm then moves one variable at a time to the neighboring discrete value. This next design point should be feasible and have a lower objective function value. Moving one variable at a time, the search terminates

Fig. 6 Flow chart of the search algorithm

when all neighboring points of the current design point are infeasible or have higher objective function values.

As shown in Fig. 6, at the beginning of the search all design variables form a “usable set”, which is denoted U . The gradient of the objective function with respect to the design variables is calculated at the starting point. Note that the assumption of the optimization model (Eq.(1) or Eq.(2)) is that its objective function is an explicit function, whose gradient has an analytical form and can be easily calculated.

Then we have to pick one variable to move to the neighboring discrete value. From all variables $x_i \in U$, the variable x_k that has the largest $\left| \frac{\partial f}{\partial x_k} \right|$ is picked.

Since we are minimizing the objective function, if $-\frac{\partial f}{\partial x_k} < 0$, then x_k is moved to the higher neighboring discrete value; on the contrary, if $\frac{\partial f}{\partial x_k} > 0$, then x_k is moved to the lower neighboring discrete value.

Before accepting the move, this new design point is checked against $NN(\mathbf{x})$ to see if it is feasible in M_{NN} . If the new design point is feasible, this move is accepted, its gradient is calculated, and we go back to the outer iteration loop as shown in Fig. 6. If the new design point is infeasible in M_{NN} , this move is discarded, the variable x_k is not usable, so we update the usable set, $U = U - x_k$, then go back to the inner iteration loop as shown in Fig. 6. Another variable with the largest gradient value is picked from the usable set, and we proceed with the same procedure. This search process terminates when $U = \phi$, since all moves from the current design point for lower objective function values are infeasible in M_{NN} . Therefore

Fig. 7 The feasible domain simulated by M_{NN} with two initial training points

the current design point is the optimum design point (at least locally) in M_{NN} .

Back to our three-bar-truss example, Fig. 7 shows the feasible domain simulated by $NN(\mathbf{x})$ of M_{NN} using two initial training points $\{(81, 81), \text{infeasible}\}$ and $\{(289, 289), \text{feasible}\}$. The search algorithm described above is then started to search for the optimum point in the feasible domain in Fig. 7. The feasible design point $(A_1, A_2) = (289, 289)$ is used as the starting point, and the search algorithm terminates at a new design point $(A_1, A_2) = (81, 289)$.

This new design point is then evaluated using the true constraints of M_{real} in Eq. (3), and is determined feasible. Thus a new training point $\{(81, 289), \text{feasible}\}$ is added to the set of training points, and the

(a) M_{NN}

(b) M_{real}

Fig. 8 The feasible domains of M_{NN} and M_{real}

neural-network is trained again. After the training is completed, $NN(\mathbf{x})$ is updated, the search algorithm is started again to search for the optimum point in the feasible domain simulated by M_{NN} with three training points.

The iteration continues, with one more training point added into the set of training points after each iteration. The whole process terminates when we get the same design point repeatedly and no new training point is generated. In our three-bar-truss example, the SNA method terminates after 13 iterations. The final design point obtained by the SNA method is $(A_1, A_2) = (25, 196)$, which is the same as the true global minimum.

Figure 8 compares the feasible domain simulated by $NN(\mathbf{x})$ at the end of the process and the true feasible domain of M_{real} in Eq.(3). The feasible domain of M_{NN} in the neighborhood of the optimum point is exactly the same as that of the M_{real} , because most training points (nodes enclosed by square brackets in the figures) fall in this region. Figure 9 shows the iteration history of this example; the points enclosed by circles are the feasible design points. Note that only 13 training points are used, i.e., only 13 evaluations, out of the $17 \times 17 = 289$ possible

combinatorial combinations, of the implicit constraints are needed. No gradient calculation is required.

4. Design Examples

Two design examples are presented to demonstrate the practicality of the SNA method. The first example is a pressure vessel design example. It is widely used in literatures to demonstrate various algorithms for discrete variable optimization. This example does not have implicit constraints. It is used here to compare the result obtained from the SNA method, treating all constraints as implicit constraints, with those obtained from other discrete programming algorithms in the literatures. The second example is on the optimal design of a cross member of a vehicle. This example has an implicit constraint that has to be evaluated by finite element analysis. The result obtained by the SNA method is also compared with that obtained by sequential linear programming.

4.1 Pressure vessel design

A cylindrical pressure vessel is capped at both ends by hemispherical heads as shown in Fig. 10. The total cost, including the cost of material, cost of

Fig. 9 Iteration history of the three-bar-truss example

Fig. 10 The pressure vessel design

Table 1 Possible discrete values of the design variables

	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.
x_1	0.625	0.6875	0.75	0.8125	0.875	0.9375	1	1.0625	1.125	1.1875	1.25
x_2	0.0	0.0625	0.125	0.1875	0.25	0.3125	0.375	0.4375	0.5	0.5625	0.625
x_3	45	45.5	46	46.5	47	47.5	48	48.5	49	49.5	50
x_4	100	102	104	106	108	110	112	114	116	118	120

forming and welding, is to be minimized. The design variables are the thickness of the shell and head, and the inner radius and length of the cylindrical section. These variables are denoted x_1 , x_2 , x_3 , and x_4 , respectively.

This problem was discussed by Sandgren⁽⁸⁾ and Kannan and Kramer⁽⁹⁾ to demonstrate algorithms for non-linear discrete variable optimization. The units of the variables used in the literatures are inches. Here we still use inches so that we can compare our result with those in the literatures. This problem can be formulated as follows:

$$\begin{aligned}
 \min. \quad & F = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 \\
 & + 3.1661x_1^2x_4 + 19.84x_1^2x_3 \\
 \text{s.t.} \quad & g_1 = -x_1 + 0.0193x_3 \leq 0 \\
 & g_2 = -x_2 + 0.00954x_3 \leq 0 \\
 & g_3 = -\pi x_3^2x_4 - \frac{4}{3}\pi x_3^3 + 1296000 \leq 0 \\
 & g_4 = x_4 - 240 \leq 0 \quad (5)
 \end{aligned}$$

In the literatures, variables x_1 and x_2 are integer multiples of 0.0625 inch, the available thickness of rolled steel plates; x_3 , and x_4 are continuous variables. Here we assume all four variables are discrete variables, and each variable can take 11 possible discrete values as listed below. Note the ranges of the variables are smaller than those in the literatures.

Design point $\{(1.25, 0.625, 50, 120), \text{feasible}\}$ is used as the initial training point. This is the same starting point as in the literatures. The SNA method terminates after 21 iterations. Table 2 shows the iteration history. Note that only 20 design points are evaluated, out of $11^4=14641$ possible combinatorial combinations. The final design point $(1, 0.5, 48.5, 112)$ was visited 3 times during the iterations.

Table 3 compares the results obtained here with those in the literatures. We can see that the design point obtained by the SNA method has a lower objective function value than both results in literatures.

4.2 Cross member design

Figure 11 shows the finite element model of a cross member of a vehicle. Only half model is shown because of symmetry. The cross member is bolted to the frame of the vehicle at point P and Q . The suspension of the vehicle exerts force to the cross member at point R and pressure loading on the top of the lower barrel of part A . There are four sizing variables t_1 , t_2 ,

Table 2 Iteration history on the pressure vessel design example

Iteration	x_1	x_2	x_3	x_4	Weight	Feasibility
0.	1.25	0.625	50	120	9589.9	Y
1.	0.625	0	45	100	2222.9	N
2.	0.625	0.0625	46.5	118	2881.0	N
3.	0.625	0.375	48.5	118	4316.5	N
4.	0.625	0.625	45	120	4848.2	N
5.	0.6875	0.625	49	118	5778.5	N
6.	0.9375	0.625	50	120	7485.1	N
7.	1.0625	0.625	49	118	8011.1	Y
8.	1	0.4375	49	118	6812.3	N
9.	1	0.625	49	118	7612.7	Y
10.	0.9375	0.625	49	118	7224.9	N
11.	1	0.5625	48.5	118	7250.5	Y
12.	1	0.5	46.5	118	6633.6	N
13.	1	0.5625	45	100	6035.6	N
14.	1	0.5	48.5	116	6922.4	Y
15.	1	0.375	48.5	116	6399.6	N
16.	1	0.5	48	108	6569.2	N
17.	1	0.5	48.5	112	6789.0	Y
18.	1	0.5	48.5	108	6655.6	N
19.	1	0.5	48.5	110	6772.3	N
20.	1	0.5	48.5	112	6789.0	Y
21.	1	0.5	48.5	112	6789.0	Y

Table 3 Comparing the results with those in the literatures

	Initial design point	The SNA method	Kannan and Kramer	Sandgren
x_1	1.25	1	1.125	1.125
x_2	0.625	0.5	0.625	0.625
x_3	50.0	48.5	58.291	47.7
x_4	120	112	43.69	117.701
g_1	-0.285	-0.064	0	-0.204
g_2	-0.148	-0.037	-0.069	-0.170
g_3	-170076.571	-9533.333	-21.222	54.225
g_4	-120	-128	-196.310	-122.300
F	9589.925	6788.988	7198.043	8129.104

t_3 , t_4 , which are the thickness of part A , B , C and D to be decided. The objective is to minimize the weight of the cross member, while the maximum stress of the cross member under the loading and boundary conditions is less than an allowable value. This cross member is made of steel. The thickness of the steel plates available from manufacturers is also restricted.

This optimization problem can be formulated as below:

$$\begin{aligned}
 \min. \quad & f(t_1, t_2, t_3, t_4) = at_1 + bt_2 + ct_3 + dt_4 \\
 \text{s.t.} \quad & \sigma(t_1, t_2, t_3, t_4) \leq \sigma_{\text{allow}} \\
 & t_1 \in T_1, t_2 \in T_2, t_3 \in T_3, t_4 \in T_4. \quad (6)
 \end{aligned}$$

where the coefficients a , b , c , d directly relate to the areas of parts A , B , C , D , and $a=0.5525$, $b=0.7606$, $c=0.4484$, $d=0.2272$. Also the allowable stress

Fig. 11 The finite element model of a cross member of a vehicle

$\sigma_{\text{allow}} = 450$ MPa, $T_1 = T_2 = \{1.4, 1.5, 1.6, 1.7, 1.8, 1.9\}$, $T_3 = T_4 = \{4.5, 5.0, 5.6, 6.0, 6.3, 7.0\}$. There are a total of 1296 (6^4) possible combinatorial combinations of the discrete values. Note that the stress constraint is an implicit constraint, since there is no analytical form of stress with respect to the design variables.

The SNA method is used to solve this discrete-variable optimization problem. $\{(1.6, 1.6, 5.6, 5.6), \text{infeasible}\}$ and $\{(1.9, 1.9, 7.0, 7.0), \text{feasible}\}$ are used as the initial training points. Note that $(t_1, t_2, t_3, t_4) = (1.6, 1.6, 5.6, 5.6)$ is the middle point of the search domain, and $(t_1, t_2, t_3, t_4) = (1.9, 1.9, 7.0, 7.0)$ was the upper bound of the search domain. The SNA method terminates after 24 iterations. The final design point is $(t_1, t_2, t_3, t_4) = (1.4, 1.9, 6.3, 5.0)$; the weight of this final design is 6.18 kg; the maximum stress of this design is 437.95 MPa. Table 4 shows the iteration history of this example.

The same problem was also solved using a sequential linear programming (SLP) algorithm. From the same starting point, SLP terminates at a continuous solution $(t_1, t_2, t_3, t_4) = (1.38, 1.82, 6.21, 4.93)$ after 20 iterations. The weight at this continuous design point is 6.058 kg, and the maximum stress is 449.26 MPa. Note that the discrete solution obtained by the SNA method is exactly the neighboring discrete point of the continuous solution.

For the SLP algorithm, each iteration requires 5 finite element analyses in this case, 1 analysis at the current design point, and 4 analyses for sensitivity calculation. Therefore SLP needs a total of more than 100 analyses to reach the continuous solution in this case. More analyses may be required to obtain the discrete solution from the continuous solution. On the other hand, the SNA method requires only one analysis in each iteration, no sensitivity analysis was required. Thus only 25 analyses were required to

Table 4 Iteration history on the cross member design example

iteration	t1	t2	t3	t4	weight(kg)	Smax(MPa)	feasibility
0	1.6	1.6	5.6	5.6	5.88	506.52	N
0	1.9	1.9	7	7	7.22	368.08	Y
1	1.7	1.7	4.5	6	5.61	613.94	N
2	1.8	1.8	7	6.3	6.93	380.69	Y
3	1.7	1.7	6	6	6.29	457.54	N
4	1.6	1.7	7	6	6.68	393.5	Y
5	1.4	1.7	6.3	6	6.26	443.15	Y
6	1.4	1.4	6	5	5.67	504.44	N
7	1.8	1.8	6	7	6.65	439.26	Y
8	1.9	1.9	4.5	6.3	5.94	597.28	N
9	1.7	1.9	6	6.3	6.51	436.02	Y
10	1.4	1.9	6	4.5	5.93	499.93	N
11	1.7	1.9	6.3	5.6	6.48	420.95	Y
12	1.7	1.9	6.3	4.5	6.23	495.57	N
13	1.9	1.8	7	4.5	6.58	492.19	N
14	1.9	1.9	7	4.5	6.66	487.13	N
15	1.9	1.7	7	5.6	6.75	391.11	Y
16	1.4	1.7	6.3	5	6.03	451.93	N
17	1.7	1.4	6.3	5.6	6.1	465.67	N
18	1.9	1.8	7	5	6.69	431.76	Y
19	1.4	1.9	5	6	5.82	528.18	N
20	1.9	1.8	6	6	6.47	450.23	N
21	1.9	1.9	6.3	5.6	6.59	416.74	Y
22	1.7	1.9	6.3	5	6.35	438.11	Y
23	1.4	1.9	6.3	5	6.18	437.95	Y
24	1.4	1.9	6.3	5	6.18	437.95	Y

reach the final discrete solution.

5. Discussion and Conclusions

This paper presents a framework of applying neural network on discrete-variable engineering optimization problems. A back-propagation neural network is used to approximate the feasible domain of the implicit constraints. A clear 0-1 pattern is used in the input-output layers of the neural network, thus the computational cost of the training of the neural network is small comparing to the evaluation of the implicit constraints. In each iteration, only one evaluation of the implicit constraints is needed to see whether the current design point is feasible. No precise function value or sensitivity calculation is required. When applied to several engineering design problems, the SNA method demonstrated the potential of being a more practical and efficient approach than the derivative-based numerical approximation algorithms.

One potential problem of the SNA method is that, when the number of design variables and the number of discrete values that a design variable can take is large, the search space will be large. This might affect the efficiency of the training of the neural network because the number of input neurons will be large. One way to overcome this potential problem is to control the "resolution" of the discrete grids of the search space. Instead of taking all discrete values all

at once, we can choose a few discrete values first, so that the size of the search space is acceptable. When the SNA method terminates at a discrete design point, we can then reduce the search space to the neighborhood of this point, but increase the resolution of the discrete grids, then start the iterations again.

Extending this concept to the extreme, the SNA method can be applied to continuous-variable optimization problems with implicit constraints. The continuous variables and functions are "digitized" into discrete points. Then the SNA method can be applied repeated until the resolution of the digitization reaches the desire accuracy. Also the discrete design point obtained from this framework can certainly be used as the starting point for the search of other continuous-variable algorithms.

Acknowledgments

This research is partially supported by National Science Council, Taiwan, ROC, grant number NSC-83-0401-E-155-023. This support is gratefully acknowledged.

References

- (1) Lu, P. and Siddall, J.N., A Boolean Local Improvement Method for General Discrete Optimization Problems, *Journal of Mechanical Design, Transactions of ASME*, Vol. 115 (1993), pp. 776-783.
- (2) Thanedar, P.B. and Vanderplaats, G.N., Survey of Discrete Variable Optimization for Structural Design, *Journal of Structural Engineering*, Vol. 121, No. 2 (1995), pp. 301-305.
- (3) Hopfield, J.J. and Tank, D.W., Neural Computation of Decisions in Optimization Problems, *Biological Cybernetics*, Vol. 52 (1985), pp. 141-152.
- (4) Lee, S.-J. and Chen, H., Design Optimization with Back-Propagation Neural Networks, *Journal of Intelligent Manufacturing*, Vol. 2 (1991), pp. 293-303.
- (5) Oda, J. and Mizukami, T., Technique for Optimum Design of Truss Structures by Neural Network, *Trans. Jpn. Soc. Mech. Eng.*, Vol. 59, No. 557, A (1993), pp. 273-278.
- (6) Arora, J.S., Huang, M.W. and Hsieh, C.C., Methods for Optimization of Nonlinear Problems with Discrete Variables: A Review, *Structural Optimization*, Vol. 8, No. 2-3 (1994), pp. 69-85.
- (7) Haftka, R.T. and Gurdal, Z., *Element of Structural Optimization*, 3rd editions, (1992), Kluwer Academic Publishers.
- (8) Sandgren, E., Nonlinear Integer and Discrete Programming in Mechanical Design, *Proceeding of the ASME Design Technology Conference*, Kissimmee, FL, (1988), pp. 95-105.
- (9) Kannan, B.K. and Kramer, S.N., An Augmented Lagrange Multiplier Based Method for Mixed Integer Discrete Continuous Optimization and Its Applications to Mechanical Design, *Journal of Mechanical Design*, Vol. 116 (1994), pp. 405-411.