A Parallel Genetic/Neural Network Learning Algorithm for MIMD Shared Memory Machines

S. L. Hung and H. Adeli

Abstract—**A new algorithm is presented for training of multilayer feedforward neural networks by integrating a genetic algorithm with an adaptive conjugate gradient neural network learning algorithm. The parallel hybrid learning algorithm has been implemented in C on an MIMD shared memory machine (Cray Y-MP8/864 supercomputer). It has been applied to two different domains, engineering design and image recognition. The performance of the algorithm has been evaluated by applying it to three examples. The superior convergence property of the parallel hybrid neural network learning algorithm presented in this paper is demonstrated.**

I. Introduction

SUPERVISED learning algorithms have been investigated and explored in several domains. The convergence speed of these algorithms is often slow. Several hours or even days of computer time are often required to train neural networks using the conventional serial workstations. In addition, the total number of iterations for learning an example in neural networks is often in the order of thousands [1], [10]. Thus, how to improve the learning performance of neural networks is currently an important research problem. One approach, inspired by the human brain neurons performing many operations simultaneously, is the development of learning algorithms on general-purpose parallel computers with the objective of reducing the overall computing time [11]. Hung and Adeli [12] present parallel backpropagation neural networks learning algorithms employing the vectorization and microtasking capabilities of vector MIMD machines. They report a maximum speedup of 6.7 using eight processors for a large network with 5950 links.

Another approach is the development of more effective neural network learning algorithms with the objective of reducing the learning time. For example, we have developed an adaptive conjugate gradient neural networks learning algorithm and applied it to the domains of engineering design and image recognition. The problem of arbitrary trial-and-error selection of the learning and momentum ratios encountered in the momentum backpropagation algorithm is circumvented in the new adaptive algorithm. Instead of constant learning and momentum ratios, the step length in the inexact line search is adapted during the learning process through a mathematical approach. Also, it is shown that the adaptive neural networks algorithm has superior convergence property compared with the momentum backpropagation algorithm.

A third approach is the development of hybrid learning algorithms by integrating genetic algorithms with neural network learning algorithms [5], [11], [13].

In this research, we have developed a parallel hybrid learning algorithm by integrating genetic algorithm with the adaptive conjugate gradient neural network learning algorithm and implemented it in C on an MIMD machine (Cray Y-MP8/864 supercomputer). The parallel hybrid learning algorithm has been applied to two different domains, engineering design and image recognition. Three examples have been used to test the performance of the new parallel learning algorithm. The first example is design of steel beams used in multistory steel structures. A small neural network with 52 links is used for this example. The other two examples are from the domain of image recognition. Large neural networks with 4160 and 5950 links are used for these examples, respectively.

II. Genetic Algorithms

A. GA Abstraction

For solution of optimization problems, genetic algorithms have been investigated recently and shown to be effective at exploring a large and complex space in an adaptive way guided by the equivalent biological evolution mechanisms of reproduction, crossover, and mutation [3], [5], [7].

There are five basic components in a genetic algorithm: a method for encoding of chromosomes, a fitness (or objective) function, an initial population, a set of operators to perform evolution between two consecutive chromosome populations, and working parameters [11], [13]. Hoffmeister and Bäck [8] presented genetic algorithm as an eight-tuple entity. In this work, we extend the previous five components of genetic algorithm and abstract them as a nine-tuple entity:

$$\mathrm{GA} = (p^0, I, \lambda, L, f, s, c, m, T)$$

where

Manuscript received September 21, 1992; revised July 26, 1993. This work has been supported in part by the National Science Foundation under Grant MSS-9222114.

S. L. Hung is with the Department of Civil Engineering, National Chiao Tung University, HsinChu, Taiwan 30050, R.O.C.

H. Adeli is with the College of Engineering, The Ohio State University, Columbus, OH 43210-1275 USA.

IEEE Log Number 9212202.

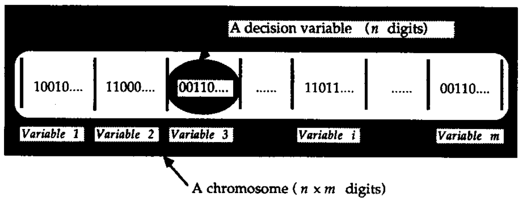


Fig. 1. Encoding decision variables as a chromosome.

$p^0 = (\mathbf{a}_1^0, \cdots, \mathbf{a}_\lambda^0) = \left(\begin{bmatrix} a_{1,1} \\ \vdots \\ a_{1,L} \end{bmatrix} \cdots \begin{bmatrix} a_{\lambda,1} \\ \vdots \\ a_{\lambda,L} \end{bmatrix} \right) \in I^\lambda$	Initial population
$I = \{0,1\}^L$	Encoding of chromosomes
$\lambda \in N$	Population size
$L \in N$	Length of Chromosome
$f : I \to R$	Fitness function
$s : I^\lambda \to I$	Parent-selection operation
$c : I^2 \to I^2$	Crossover operation
$m : I \to I$	Mutation operation
$T : I^\lambda \to \{0,1\}$	Termination criterion

There are λ chromosomes in each population. The initial population of chromosomes, p^0, is generated randomly. The entity $\mathbf{a}_k^t$ denotes the kth chromosome in the tth generation of population, p^t. A chromosome, I, is encoded as a string of binary digits, 1's and 0's. If there are v decision variables in an optimization problem and each decision variable is encoded as an n-digit binary number, then a chromosome is a string of $L = v \times n$ binary digits (Fig. 1) and represented as a column vector $[a_{k,1}, \cdots, a_{k,L}]^T$. The term $g : X \to Y$ denotes a function g maps x to y where $x \in X$ and $y \in Y$. Variables N and R are sets of integer and real numbers, respectively. The evolution process of genetic algorithm is continued ($T = 0$) until one of the termination criteria is met ($T = 1$).

B. Parent Selection

The parent selection operation, s, produces an intermediate population $p'^t = (\mathbf{a}_1'^t, \cdots, \mathbf{a}_\lambda'^t)$ from the population $p^t = (\mathbf{a}_1^t, \cdots, \mathbf{a}_\lambda^t)$ in the tth generation. Any $\mathbf{a}_i'^t = \mathbf{a}_q^t = s(p^t)$ in p'^t is selected by a given random real number α_i satisfying the following condition:

$$0 \leq \alpha_i \leq \sum_{j=1}^{\lambda} f(\mathbf{a}_j^t)$$

The index q is obtained from

$$q = \min\left\{ k \mid \forall k \in \{1, \cdots, \lambda\}, \text{s.t. } \alpha_i \leq \sum_{k=1}^{\lambda} f(\mathbf{a}_k^t) \right\}.$$

C. Crossover Operation

For any pair of selected chromosomes in a population p^t, an associated real value, $0 \leq \rho \leq 1$, is generated randomly. If ρ is greater than the predefined crossover threshold, ρ_c, the crossover operator is applied to this pair of chromosomes. Three different crossover strategies have been applied in this work. The first one is two-point crossover, $c_{\text{tp}'}$, that produces an intermediate population p'^t from the population p^t and is defined below in (1), where $1 \leq \rho_1 < \rho_2 \leq L$. In this crossover strategy, two positions in a pair of chromosomes are selected. The pair of chromosomes are divided into three sub-chromosomes by these two points and crossed over to each other by swapping the first and third sub-chromosomes.

The second crossover strategy is multi-point crossover, c_{mp}, that produces an intermediate population $p\prime^t$ from the population p^t and is defined as (see (2) below) where $0 \leq \rho_k, \rho_{\text{mp}} \leq 1$. In this crossover strategy, more than one crossover points are selected in a pair of chromosomes. The crossover operator is performed in bit level (allele in a chromosome). That is, the process of crossover is performed bit by bit. The numbers of crossover points and crossover positions in each pair of chromosomes are selected randomly, distinctly from each other.

The third crossover strategy is uniform crossover, c_{un}, that produces an intermediate population p'^t from the population p^t. First, a mask, a binary array with length L, is generated. L real values, $0 \leq r_j \leq 1$ ($j = 1, 2, \cdots, L$), are generated randomly. If the jth random number, r_j, is greater than or equal to the predefined threshold value, ρ_{ma}, the value of the jth element in the binary array is set as 1. Otherwise, it is set

$$\begin{aligned} \left\{ \begin{matrix} \mathbf{a}_i'^t \\ \mathbf{a}_{(i+1)}'^t \end{matrix} \right\} &= c_{tp}\left(\left\{ \begin{matrix} \mathbf{a}_i^t \\ \mathbf{a}_{(i+1)}^t \end{matrix} \right\} \right) \qquad \forall\, i \in \{1, 3, \cdots, 2k+1, \cdots, \lambda - 1\} \\ &= \left\{ \begin{matrix} \left[a_{(i+1),1}, a_{(i+1),1}, \cdots, a_{(i+1),\rho_1}, a_{i,(\rho_i+1)}, \cdots, a_{i,\rho_2}, a_{(i+1),(\rho_2+1)}, \cdots, a_{(i+1),L}\right]^T \\ \left[a_{i,1}, a_{i,2}, \cdots, a_{i,\rho_1}, a_{(i+1),(\rho_1+1)}, \cdots, a_{(i+1),\rho_2}, a_{i,(\rho_2+1)}, \cdots, a_{i,L}\right]^T \end{matrix} \right\} \end{aligned} \tag{1}$$

$$\begin{aligned} \left\{ \begin{matrix} \mathbf{a}_i'^t \\ \mathbf{a}_{(i+1)}'^t \end{matrix} \right\} &= c_{\text{mp}}\left(\left\{ \begin{matrix} \mathbf{a}_i^t \\ \mathbf{a}_{(i+1)}^t \end{matrix} \right\} \right) \qquad \forall i \in \{1, 3, \cdots, 2k+1, \cdots, \lambda - 1\} \\ &= \bigcup_{k=1}^{L} \left\{ \left(\begin{bmatrix} a_{i,k} \\ a_{(i+1),k} \end{bmatrix}, \text{if } \rho_k \geq \rho_{\text{mp}} \right) \vee \left(\begin{bmatrix} a_{(i+1),k} \\ a_{i,k} \end{bmatrix}, \text{if } \rho_k < \rho_{\text{mp}} \right) \right\} \end{aligned} \tag{2}$$

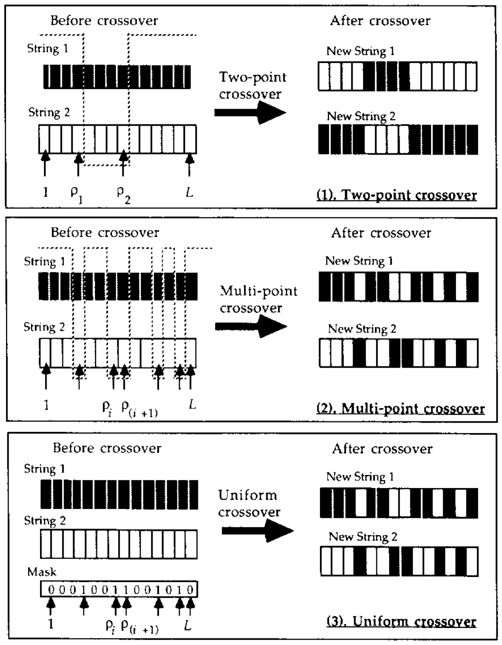


Fig. 2. Two-point, multi-point, and uniform crossovers.

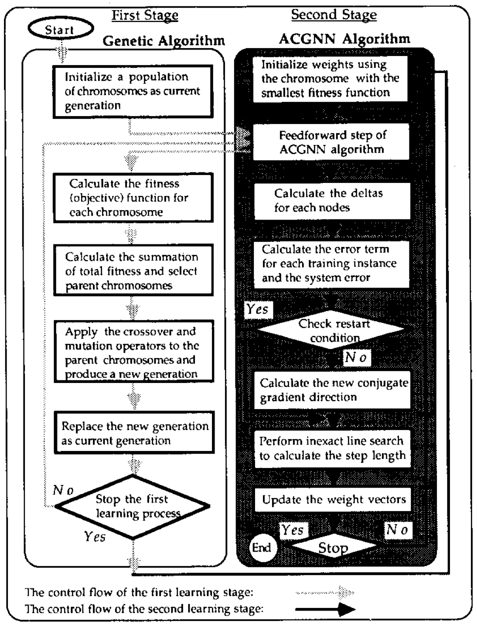

Start
First Stage
Genetic Algorithm
Second Stage
ACGNN Algorithm
Initialize a population of chromosomes as current generation
Calculate the fitness (objective) function for each chromosome
Calculate the summation of total fitness and select parent chromosomes
Apply the crossover and mutation operators to the parent chromosomes and produce a new generation
Replace the new generation as current generation
Stop the first learning process
No
Yes
Initialize weights using the chromosome with the smallest fitness function
Feedforward step of ACGNN algorithm
Calculate the deltas for each nodes
Calculate the error term for each training instance and the system error
Check restart condition
Yes
No
Calculate the new conjugate gradient direction
Perform inexact line search to calculate the step length
Update the weight vectors
Stop
Yes
No
End
The control flow of the first learning stage:
The control flow of the second learning stage:

Fig. 3. A hybrid learning algorithm using genetic algorithm with adaptive conjugate gradient neural network learning algorithm.

as 0. The mask, ma, is defined as

$$\mathrm{ma} = \cup_{j=1}^{L}\{([1], \text{if } r_j \geq \rho_{\mathrm{ma}}) \vee ([0], \text{if } r_j < \rho_{\mathrm{ma}})\}$$

The uniform crossover is defined as shown in (3) below. Similar to the multi-point crossover strategy, the process of uniform crossover is performed bit by bit in a pair of chromosomes. In the uniform crossover strategy, the crossover positions are predefined in a mask. All chromosomes in a population are crossed over in the same positions. On the other hand, in the multi-point crossover strategy, each pair of chromosomes are crossed over at different points because no predefined mask is used. Three aforementioned crossover operators are shown schematically in Fig. 2.

D. Mutation Operation

For any chromosome in a population p^t, an associated real value, $0 \leq \rho \leq 1$, is generated randomly. If ρ is less than the predefined mutation threshold, ρ_m, the mutation operator is applied to this chromosome. The mutation operator simply alters one bit from 0 to 1 (or 1 to 0) in a chromosome. As the mutation operator is not guided by the fitness (objective) function, the result of mutation operator can make an instant change between two successive generations. The operator of mutation, m, produces an intermediate population p'^t from the population p^t and is defined as:

$$\mathbf{a}_i'^t = m(\mathbf{a}_i^t) \quad \forall\, i \in \{1, \cdots, \lambda\}$$
$$a'_{i,k} = \left\{\begin{matrix} a_{i,k} & \text{for } k \in \{1, 2, \cdots, p-1, p+1, \cdots, L\} \\ \bar{a}_{i,k} & \text{for } k = p \end{matrix}\right\}$$

where $1 \leq p < L$.

III. A Hybrid Genetic/ Neural Network Learning Algorithm

A hybrid learning algorithm using genetic algorithm with adaptive conjugate gradient multilayer neural networks is presented in Fig. 3. It consists of two learning stages. The first learning stage is used to accelerate the whole learning process by using a genetic algorithm with the feedforward step of the adaptive conjugate gradient neural network (ACGNN) learning algorithm. The genetic algorithm performs global search and seeks a near-optimal initial point (weight vector) for the second stage. In this stage, each chromosome is used to encode the weights of neural network. The fitness (objective) function for the genetic algorithm is defined as the average squared system error of the corresponding neural network. Therefore, it becomes an unconstrained optimization problem: find a set of

$$\begin{aligned} \left\{\begin{matrix} \mathbf{a}_i'^t \\ \mathbf{a}_{(i+1)}'^t \end{matrix}\right\} &= c_{un}\left(\left\{\begin{matrix} \mathbf{a}_i^t \\ \mathbf{a}_{(i+1)}^t \end{matrix}\right\}\right) \qquad \forall\, i \in \{1, 3, \cdots, 2k+1, \cdots, \lambda - 1\} \\ &= \bigcup_{k=1}^{L}\left\{\left(\begin{bmatrix} a_{i,k} \\ a_{(i+1),k} \end{bmatrix}, \text{if } \mathrm{ma}_k = 0\right) \vee \left(\begin{bmatrix} a_{(i+1),k} \\ a_{i,k} \end{bmatrix}, \text{if } \mathrm{ma}_k = 1\right)\right\} \end{aligned} \tag{3}$$

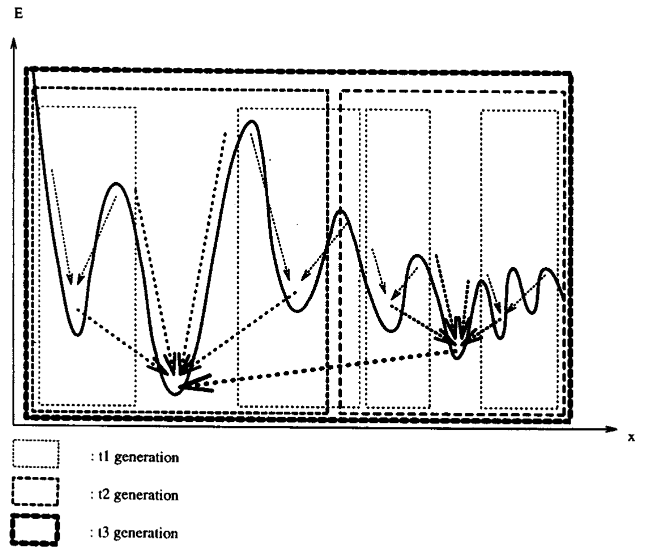


Fig. 4. Global search using genetic algorithm.

decision variables minimizing the objective function. The best fitness function value in a population is defined as the smallest value of the objective function in the current population.

The process of global search using genetic algorithm is schematically presented in Fig. 4. Consider three consecutive generations, $t1$, $t2$, and $t3$. The chromosomes in $t1$ generation perform local search in some discrete domain. After applying the crossover and mutation operators to the $t1$ generation, the chromosomes with lower fitness function are selected as the parents, the chromosomes with higher fitness function are discarded, some new chromosomes are generated via the selected parent chromosomes, and the $t1$ generation is replaced by the new population of chromosomes called $t2$ generation. In the $t2$ generation, the chromosomes perform local search in a larger domain than $t1$ generation and approach to some local minimums with lower fitness function values. In the $t3$ generation, the chromosomes with lower fitness function in the $t2$ generation are selected and new chromosomes are generated that cover the whole bounded domain. In this last generation, the chromosomes perform global search in the whole bounded domain and approach to the global minimum in the domain.

After performing several iterations and meeting one of the stopping criteria, the first learning stage is terminated and the chromosome returning the minimum objective function (the best seed) is considered as the initial weights of the neural network in the second stage. Next, the adaptive conjugate gradient learning algorithm performs the second learning process until the terminal condition is satisfied.

In order to reduce the memory storage requirement and increase the computational efficiency, the allele (binary digit) of each chromosomes in a population is encoded as a bit rather than an integer. In this case, the length of each chromosome is equal to the length of an integer, such as 16 bits on a SUN SPARC station. Hence, the memory storage used for each chromosome is an integer rather than a sixteen-element integer array. Since we encode the chromosome as an integer, the operations of crossover and mutation can be operated using bitwise operators that are directly performed via computer hardware.

Consider a multilayer neural network with $N[i]$ nodes in layer i. The learning problem is mapped from $N[1]$ input nodes to $N[m]$ output nodes and a number of N_s instances are given as training examples. The total number of weights and nodes are denoted by N_w and N_n, respectively. For the genetic algorithm, we assume N_p chromosomes are generated and operated on in each iteration. The operators and other features of the genetic algorithm are the same as those defined previously.

The first learning stage is a combination of the genetic algorithm with the feedforward process of the adaptive conjugate gradient learning algorithm. In each iteration of this learning stage, the chromosome with the smallest value of objective function is saved as *sub_best_chromosome* and compared with the one saved in the previous iteration, called *best_chromosome*. After the first learning stage is terminated, the *best_chromosome* is used as the initial weight vector for the second learning stage. In order to reduce redundant computations in this learning stage, three different stopping criteria are employed to terminate this learning process. If one of these three stopping criteria is met, the first learning stage is terminated.

- — The smallest value of the objective functions in a population is less than the acceptable predefined value.
- — The fitness ratio, defined as the value of the *best_chromosome*'s objective function to the average value of objective functions, is greater than 0.95.
- — If the objective function of the *best_chromosome* does not change in a predefined number of consecutive iterations (in this work, a value of 10 is used for this number).

TABLE I
THE FIRST STAGE OF THE PARALLEL HYBRID LEARNING ALGORITHM

```
/* First Learning Stage */
1) Initialize the chromosomes randomly as current generation, initialize working parameters, and set the first chromosome
   as the best_ chromosome.
   Parallel region 1—entry
2) For $i = 1$ to $N_p$ , do concurrently
   a) Initialize sub_total_fitness to zero and sub_best_ chromosomes as null.
   b) For $j = 1$ to chunksize, do sequentially
      a) For $k = 1$ to $N_s$, do sequentially
         a1) Perform feedforward procedure of the adaptive conjugate gradient learning algorithm.
         a2) Calculate the objective function (system error for the neural network).
      Next $k$
      b) Calculate the sub_total_ fitness by accumulating objective function of each chromosome.
      c) Store the best_ chromosome as sub_best_chromosome.
   Next $j$
   Guarded section 1—entry
   c) Calculate the total_ fitness by accumulating sub_total_ fitness.
   d) Compare the sub_best_chromosome with each other and set the best sub_best_chromosome as the best_ chromosome.
   Guarded section 1—end
   Next $i$
   Parallel region 1—end
3) DO
   Parallel region 2—entry
   e) For $i = 1$ to $(N_p/2)$, do concurrently
      d) Initialize sub_total_ fitness to zero and sub_best_ chromosome as null.
      e) For $j = 1$ to chunksize, do sequentially
         e1) Select parents using roulette wheel parent selection.
         e2) Apply two-point, multi-point, or uniform crossovers and mutation to the parents.
         e3) For $k = 1$ to $N_s$, do sequentially
            e3.1) Perform feedforward procedure of the adaptive conjugate gradient learning algorithm to parent chro-
                  mosomes.
            e3.2) Calculate the objective function (system error for the neural network) for parent chromosomes.
                  Next $k$
         e4) Calculate the sub_total_ fitness by accumulating objective function of each chromosome.
         e5) Store the best_ chromosome as sub_best_ chromosome.
   Next $j$
   Guarded section 2—entry
      f) Calculate the total_ fitness by accumulating sub_total_ fitness.
      g) Compare the sub_best_ chromosome to each other and set the best sub_best_ chromosome as the best_ chromosome.
   Guarded section 2—end
   Next $i$
   Parallel region 2—end
   f) Replace the old generation by the new generation.
      WHILE (~stopping criteria).
```

The first stage of the parallel hybrid learning algorithm is presented in Table I and shown schematically in Fig. 5.

The second learning stage is a stand-alone adaptive conjugate gradient neural network learning algorithm. A number of N_s tasks are created and executed concurrently. The second stage of the parallel hybrid neural network learning algorithm is presented in Table II and shown schematically in Fig. 6.

IV. APPLICATIONS

We apply the parallel hybrid genetic/neural network learning algorithm developed in this research to two different domains:

TABLE II
THE SECOND STAGE OF THE PARALLEL HYBRID LEARNING ALGORITHM

```
/* Second Learning Stage */
1) Set the best_chromosome as the initial weight vector, set up the topological structure of neural network, and set the
   counter cnt to zero.
2) DO
   Parallel region—entry
   a) For $i = 1$ to $N_s$, do concurrently
      a) Initialize sub_system_error and sub_delta_weights to zero.
      b) For $j = 1$ to chunksize, do sequentially
         b1) Perform feedforward procedure of the adaptive conjugate gradient learning algorithm.
         b2) Calculate sub_system_error.
         b3) Calculate the deltas in output layer.
         b4) Calculate the deltas in hidden layers (from layer $m - 1$ to layer 1).
         b5) Calculate the sub_delta_weights in hidden layers (from layer $m - 1$ to layer 1).
   Next $j$.
   Guarded section—entry
      c) Calculate the system error, E, by accumulating sub_system_error.
      d) Calculate deltas of weights by accumulating the sub_delta_weights.
   Guarded section—end
   Next $i$.
   Parallel region—end
b) If cnt $\geq$ 1, calculate the new conjugate gradient direction.
c) Perform inexact line search to calculate the step length.
d) Update the weight vector.
   WHILE (~stop criteria).}
```

engineering design and image recognition. Three examples are presented, one in the domain of engineering design and two in the domain of image recognition.

Example 1 Engineering Design: This example is the selection of a minimum weight steel beam from the AISC LRFD wide-flange (W) shape database [4] for a given loading condition [2], [9], [10]. Each instance consists of five input patterns: the member length, the unbraced length, the maximum bending moment in the member, the maximum shear force in the member, and the bending coefficient. The output pattern is the plastic modulus of the corresponding least weight member.

A four-layer feedforward neural network with two hidden layers was used to learn this problem. The numbers of nodes in the input layer, the first and second hidden layers, and the output layer are 5, 5, 3, and 1, respectively. There are 52 links in this neural network. We use ten training instances in this example. The total number of iterations for learning process is limited to 100. The working parameters for the first stage of learning (genetic algorithm) are as follows: population size: 4000, length of decision variable: 16 bits, chromosome length: 832 (52×16) bits, crossover rate: 0.8, mutation rate: 0.08, and range of decision variables: -5 to 5.

Example 2 Image Recognition (7×7 Binary Images of Numerals): This example is recognition of seven by seven (7×7) binary images of the numerals (0 to 9) (Fig. 7). A three-layer neural network with one hidden layer was used to learn this problem. The numbers of nodes in the input layer, the hidden layer, and the output layer are 49, 99, and 10, respectively. The total number of links in this three-layer

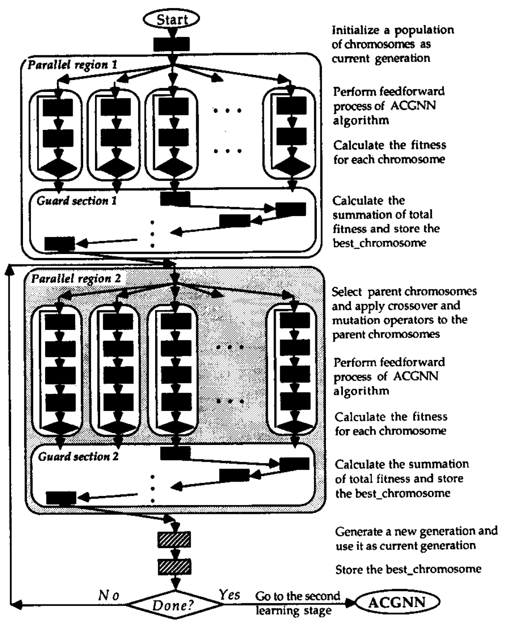

Start
Parallel region 1
Guard section 1
Parallel region 2
Guard section 2
No
Done?
Yes
Go to the second learning stage
ACGNN
Initialize a population of chromosomes as current generation
Perform feedforward process of ACGNN algorithm
Calculate the fitness for each chromosome
Calculate the summation of total fitness and store the best_chromosome
Select parent chromosomes and apply crossover and mutation operators to the parent chromosomes
Perform feedforward process of ACGNN algorithm
Calculate the fitness for each chromosome
Calculate the summation of total fitness and store the best_chromosome
Generate a new generation and use it as current generation
Store the best_chromosome

Fig. 5. The first learning stage of the parallel hybrid neural network learning algorithm.

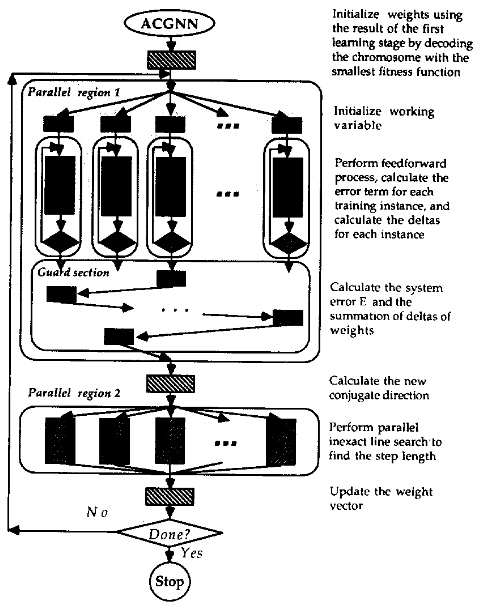

ACGNN
Initialize weights using the result of the first learning stage by decoding the chromosome with the smallest fitness function
Parallel region 1
Initialize working variable
Perform feedforward process, calculate the error term for each training instance, and calculate the deltas for each instance
Guard section
Calculate the system error E and the summation of deltas of weights
Parallel region 2
Calculate the new conjugate direction
Perform parallel inexact line search to find the step length
Update the weight vector
No
Done?
Yes
Stop

Fig. 6. The second learning stage of the parallel hybrid neural network learning algorithm.

neural networks is 5950. The total number of iterations for learning process is limited to 200. There are 30 instances in the training set: ten noiseless image instances and 20 image instances with about 10% noise (5 pixels out of 49 pixels). The working parameters for the first stage of learning (genetic algorithm) are as follows: population size: 250, length of decision variable: 16 bits, chromosome length: 95 200 (5950 × 16) bits, crossover rate: 0.8, mutation rate: 0.08, range of decision variables: -1 to 1.

Example 3 Image Recognition (Lenna Image): This example is to recognize an 8-bit gray-scale (256 gray levels) of the Lenna image (Fig. 8). Each training instance is an eight by eight (8×8) square image Thus, the 384×384 pixel Lenna image is decomposed into 2304 training instances used to train the feedforward neural network. A flat (two-layer) neural network was used to learn this example. The number of nodes in both input and output layer is 64. The total number of links in this two-layer neural networks is 4,160. The total number of iterations for learning process is limited to 50. The working parameters for the first stage of learning (genetic algorithm) are as follows: population size: 250, length of decision variable: 16 bits, chromosome length: 66 560 (4160×16) bits, crossover rate: 0.9, mutation rate: 0.095, range of decision variables: -1 to 1.

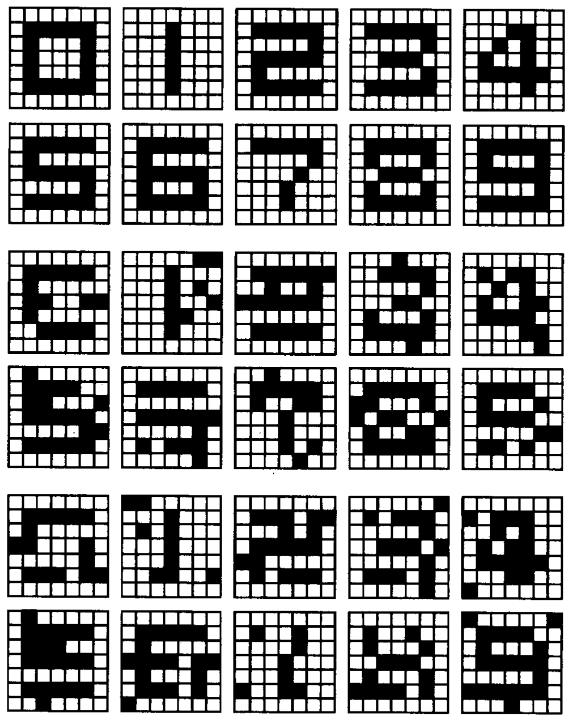


Fig. 7. Ten noiseless and 20 noisy 7×7 binary images of numerals (0 to 9).

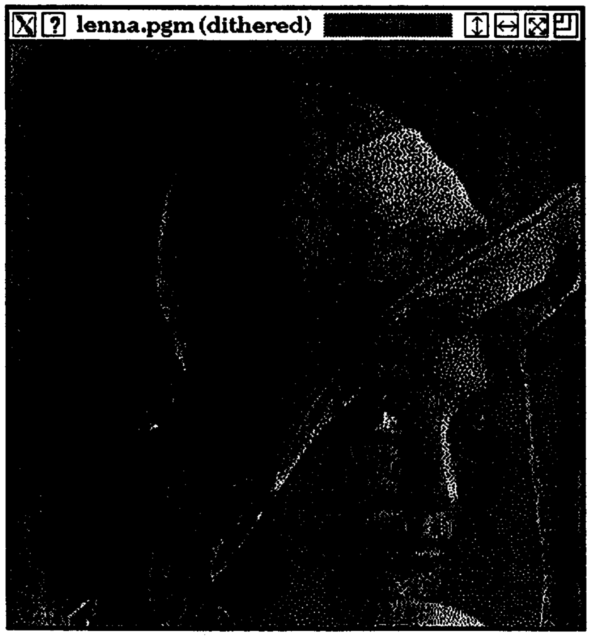


Fig. 8. The 8-bit gray-scale (256 levels) Lenna image.

V. Computation Results

A. Convergence History

Example 1: The system error for this example using the adaptive conjugate gradient neural network learning algorithm and the parallel hybrid neural network algorithm are shown in Fig. 9. After 12 iterations of learning process, the first learning stage of the parallel hybrid neural network learning algorithm met one of the stopping criteria. The best fitness function was a value of about 0.0023. The result of the first learning stage is used as an initial weight vector in the second learning stage.

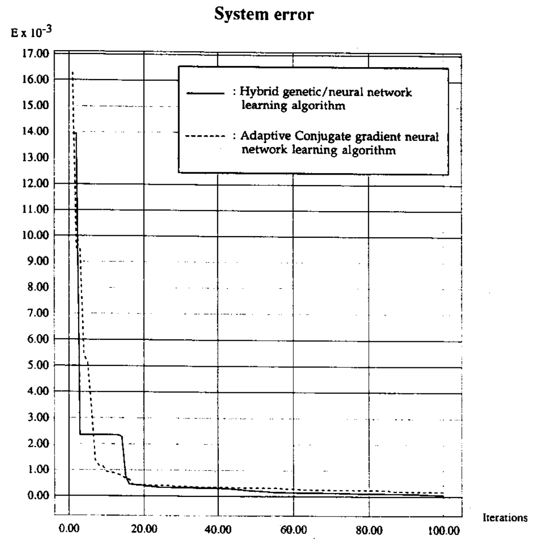


Fig. 9. System error for the minimum weight steel beam design.

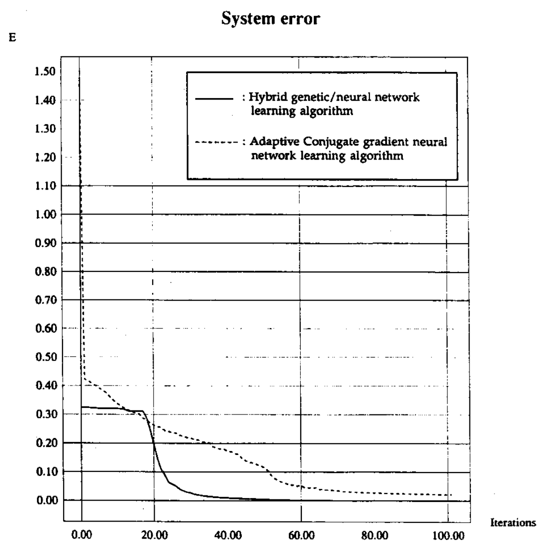


Fig. 10. System error for the 7 × 7 binary image recognition of numerals (0 to 9).

After a total of 100 iterations of the learning process, the system error in the parallel hybrid neural network learning algorithm converges to a value 6.5 × 10^{-5}. The stand-alone adaptive conjugate gradient neural network learning algorithm converges to 1.84 × 10^{-4} after 100 iterations of the learning process.

Example 2: The system error for this example using the adaptive conjugate gradient neural network learning algorithm and the parallel hybrid neural network learning algorithm are shown in Fig. 10. After 18 iterations, the first learning stage of the parallel hybrid neural network learning algorithm met one of the stopping criteria. The best fitness function has a value of about 0.3. The result of the first learning stage is used as an initial weight vector for the second learning stage. After 12 more iterations, the system error in the parallel hybrid neural network learning algorithm converges to a value of less than 0.001 and the algorithm achieves 100% recognition of all the 30 training instances.

After the same number of iterations, the stand-alone adaptive conjugate gradient neural network learning algorithm converges to a value of 0.17 and recognize about 63% (19 out of 30) of the training set.

Example 3: The system error for this example using the adaptive conjugate gradient neural network learning algorithm and the parallel hybrid neural network learning algorithm are shown in Fig. 11. After eight iterations of learning process, the first learning stage of the parallel hybrid neural network learning algorithm met one of the stopping criteria. The best fitness function was a value of about 4.4. The result of the first learning stage is used as an initial weight vector for the second learning stage. After 42 more iterations, the system error in the parallel hybrid neural network learning algorithm converges to a value of 0.10. The stand-alone adaptive conjugate gradient neural network learning algorithm converges to a value of 0.15 after 50 iterations.

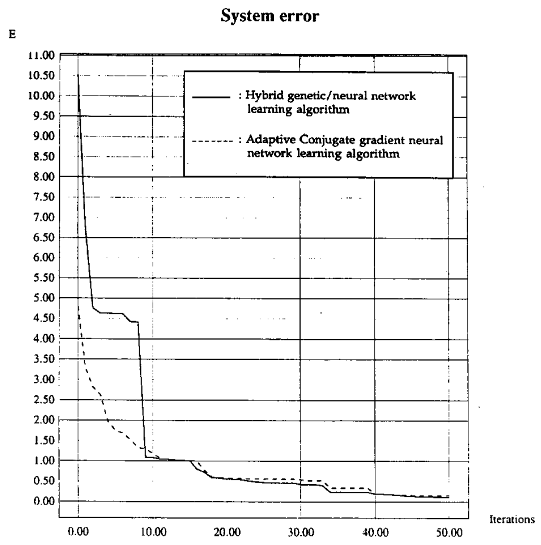


Fig. 11. System error for the Lenna image recognition problem.

B. Speedup

The speedup is measured by using an expert system tool, called *atexpert*. A neural network with 52 links is used in example 1. In the first learning stage of the parallel hybrid neural network learning algorithm, 4000 chromosomes are operated on in each learning iteration. That is, 4000 tasks are created and performed concurrently in this stage. Each concurrent task performs the computation of a stand-alone neural network with 10 training instances. The overall speedup achieved by the parallel hybrid neural network learning algorithm for example 1 is dominated by the first learning stage. As the genetic algorithm is an intrinsically parallel algorithm, the maximum

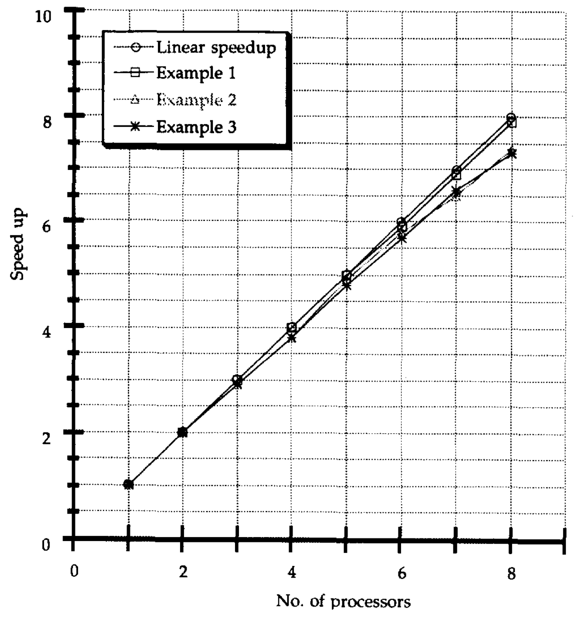


Fig. 12. Speedup for examples 1–3 using the parallel hybrid genetic/neural network learning algorithm without vectorization.

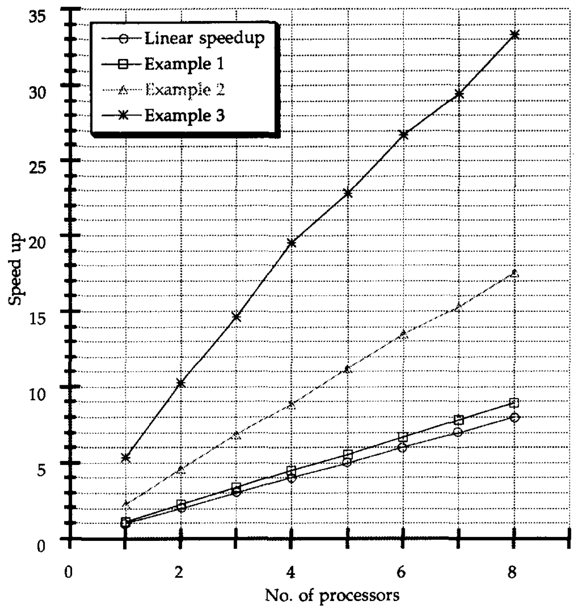


Fig. 13. Speedup for examples 1–3 using the parallel hybrid genetic/neural network learning algorithm with vectorization.

speedup of example 1 due to microtasking is about 7.9 using eight processors of the Cray Y-MP8/864 supercomputer (Fig. 12). A maximum average speedup of about 9 is achieved when microtasking is combined with vectorization using eight processors of the Cray machine (Fig. 13). In this example, the value of speedup due to a combination of microtasking with vectorization is not high because the loop performed by vector operation is short.

A very large neural network with 5950 links is used in example 2. Two hundred fifty (250) chromosomes are operated on in each learning iteration. That is, 250 tasks are created and performed concurrently in this learning stage. Each concurrent task performs the computation of a stand-alone neural network with 30 training instances. The overall speedup achieved by the parallel hybrid neural network learning algorithm for example 2 is also dominated by the first learning stage. The maximum speedup due to microtasking is about 7.4 using eight processors of the Cray Y-MP8/864 supercomputer (Fig. 12). A maximum average speedup of about 17.6 is achieved due to a combination of microtasking with vectorization (Fig. 13).

A very large neural network with 4160 links is used in example 3 with 2304 training instances. Two hundred fifty (250) chromosomes are operated on in each learning iteration. The maximum speedup due to microtasking is about 7.3 using eight processors of the Cray Y-MP8/864 supercomputer (Fig. 12). A maximum average speedup of about 33 is achieved when microtasking is combined with vectorization (Fig. 13).

VI. Final Remarks

We have presented a parallel hybrid neural network learning algorithm by integrating genetic algorithm with an adaptive conjugate gradient neural network learning algorithm. Following observations are made and conclusions drawn:

1) The results of neural network learning are sensitive to the initial value of the weight vector. In this work, a genetic algorithm is employed to perform global search and to seek a good starting weight vector for the subsequent neural network learning algorithm. The result is an improvement in the convergence speed of the algorithm.
2) The problem of entrapment in a local minimum is encountered in gradient-based neural network learning algorithms. In the hybrid learning algorithm presented in this paper, this problem is circumvented by using a genetic algorithm which is guided by the fitness function of a population rather than gradient direction. After several iterations of the global search, the first learning stage returns a near-global optimum point that is used as the initial weight vector for the second learning stage.
3) A large-scale multilayer neural network requires substantial computing processing time in order to converge to an acceptably small system error value. By developing efficient parallel learning algorithms on multiprocessor computers we can increase the computational speedup by an order of magnitude.

References

[1] H. Adeli and C. Yeh, "Perceptron learning in engineering design," *Microcomp. Civil Eng.*, vol. 4, no. 4, pp. 247–256, 1989.
[2] H. Adeli and C. Yeh, "Neural network learning in engineering design," in *Proc. Int. Neural Net. Conf.*, Paris, France, July 9–13, 1990, pp. 412–415.
[3] H. Adeli and N.-T. Cheng, "Integrated genetic algorithm for optimization of space structures," *J. Aerospace Eng., ASCE*, vol. 6, no. 4, pp. 315–328, 1993.

[4] *Manual of Steel Construction, Load and Resistance Factor Design.* Chicago, IL: American Institute of Steel Construction, 1986.

[5] R. K. Belew, J. McInerney, and N. N. Schraudolph, "Evolving networks: Using the genetic algorithm with connectionist learning," Computer Science and Engineering Tech. Rep. CS90–174, Univ. of California at San Diego, 1990.

[6] L. Davis, ed., *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley, 1989.

[8] F. Hoffmeister and T. Bäck, "Genetic algorithms and evolution strategies—Similarities and differences," in *Parallel Problem Solving from Nature* H.-P. Schwefel and R. Männer, eds. Berlin, Germany: Springer-Verlag, 1991, pp. 455–469.

[9] S. L. Hung and H. Adeli, "Multi-layer perceptron learning for design problem solving," in *Proc. Int. Neural Net. Conf.*, Espoo, Finland, July 24–28, 1991, pp. 1225–1228.

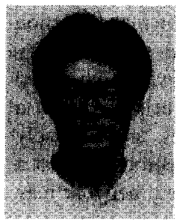
[10] S. L. Hung and H. Adeli, "A model of perceptron learning with a hidden layer for engineering design," *Neurocomputing*, vol. 3, pp. 3–14, 1991.

[11] S. L. Hung and H. Adeli, "A hybrid learning algorithm for distributed memory multicomputers," *Heuristics—The J. Knowledge Eng.*, vol. 4, no. 4, pp. 58–68, 1991.

[12] S. L. Hung and H. Adeli, "Parallel backpropagation learning algorithms on Cray Y-MP8/864 supercomputer," *Neurocomputing*, vol. 5, pp. 287–302, 1993.

[13] D. J. Montana and L. Davis, "Training feedforward networks using genetic algorithms," in *Proc. Int. Joint Conf. Artificial Intell.*, San Mateo, CA, 1989, pp. 762–767.

[14] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representation by error propagation," in *Parallel Distributed Processing*, D. E. Rumelhart, *et al.*, ed. Cambridge, MA: MIT Press, 1986, pp. 318–362.



S. L. Hung received the B.S. degree from the National Chiao Tung University, Taiwan in 1982, and the M.S. and Ph.D. degrees from The Ohio State University in 1990 and 1992, respectively.

He has published 14 papers in the areas of neural networks, genetic algorithms, parallel processing, fuzzy systems, and database management. He is currently an Associate Professor at the National Chiao Tung University.

Hojjat Adeli received the Ph.D. degree from Stanford University in 1976.

Currently, he is a Professor of Engineering and Member of the Center for Cognitive Science at The Ohio State University. A contributor to 40 research and scientific journals, he has authored 260 research and scientific publications, including four books in various fields of computer science and engineering. He has also edited 10 books, including *Knowledge Engineering, Volume 1—Fundamentals* and *Knowledge Engineering, Volume 2—Applications* (McGraw-Hill, 1990); and *Supercomputing in Engineering Analysis* and *Parallel Processing in Computational Mechanics*, (Marcel Dekker, 1992).

Dr. Adeli was the Editor-in-Chief of *Heuristics—The Journal of Knowledge Engineering* during 1991–1993. He is the Founder and Editor-in-Chief of the journal *Integrated Computer-Aided Engineering.* He has been an organizer or member of advisory boards of over 25 national and international conferences and a contributor to over 80 other conferences held in 24 different countries. He was a Keynote and Plenary Lecturer at computing conferences held in Italy (1989), Mexico (1989), Japan (1991), China (1992), Canada (1992), U.S. (1993), Germany (1993), Morocco (1994), and Singapore (1994). He has received numerous academic, research, and leadership awards, honors, and recognitions. His recent awards include The Ohio State University College of Engineering 1990 Research Award in Recognition of Outstanding Research Accomplishments, and the Lichtenstein Memorial Award for Faculty Excellence.