NEURAL NETWORK PARALLEL COMPUTING

by

Yoshiyasu Takefuji
Case Western Reserve University, USA
Keio University, Japan

kluwer

Chapter 1

NEURAL NETWORK MODELS AND N-QUEEN PROBLEMS

Brief history of neural network parallel computing and four mathematical neural network models including a McCulloch-Pitts neuron, a sigmoid neuron, a hysteresis McCulloch-Pitts neuron, and a maximum neuron are introduced. "What is an N-queen problem?" and a neural network model of the N-queen problem are first presented to show how to use the neural network (sequential/parallel) model for solving general optimization problems. Two important issues are highlighted in Chapter 1: I) how to construct a neural representation from a given problem, and II) how to build the motion equation considering the necessary and sufficient constraints and/or the cost function from the problem. Three Pascal programs for N-queen problems are given for the reader to readily observe the neural network system which can run on virtually any computer.

1.1 INTRODUCTION

In 1943 mathematical models based on biological computation were proposed by W. S. McCulloch and W. H. Pitts (McCulloch and Pitts 1943). They attempted to take advantage of the elegant natural biological computation in the brain of animals and human beings. D. O. Hebb presented the learning theory for realizing the associative memory where the strengths of the existing synaptic connections between neurons are modified by the input patterns (Hebb 1949). B. Widrow at Stanford University demonstrated adaptive switching circuits in 1960 (Widrow and Hoff 1960). In 1961

F. Rosenblatt at Cornell University presented Perceptrons and the theory of brain mechanisms in his book (Rosenblatt 1962). In 1969 M. Minsky and S. Papert at MIT showed the limitation of Perceptrons in their book (Minsky and Papert 1969). The negative results against the artificial neural network computing had caused less support and interest from governments/industries and consequently shrank the scale of neural network study and the number of investigators. However a small number of researchers such as S. Amari, L. Cooper, K. Fukushima, and S. Grossberg studied the neural network computing during the 1960's and 1970's. In the 1970's J. A. Anderson and T. Kohonen developed mathematical models of associative memory.

The new discovery in neurobiology and the explosive interest in parallel computation along with the inexpensive VLSI (very-large-scale-integrated) circuit technology have caused a dramatic resurgence. In 1985 J. J. Hopfield and D. Tank proposed an artificial neural network for optimization problems which has attracted many new investigators to get involved in neural computing (Hopfield and Tank 1985). However in 1988 G. V. Wilson and G. S. Pawley strongly criticized the neural network for optimization problems (Wilson and Pawley 1988) and R. Paielli reported the simulation test of the Hopfield neural network in the same problem (Paielli 1988) both of which discouraged US federal agencies to support the neural network research for optimization problems. Since 1988 it has been widely believed that the artificial neural network is not suitable for optimization problems.

This book is intended to demonstrate the capability of the artificial neural network for solving optimization problems over the best known algorithms or the best methods if they exist. The book covers a variety of professional fields including game theory, computer science, graph theory, molecular biology, VLSI computer aided design, reliability, management science, and communications and computer networks. It contains ten applications including N-queen problems, crossbar switch scheduling problems, four-coloring and k-colorability problems, graph planarization problems, channel routing problems, RNA secondary structure prediction problems, knight's tour problems, spare allocation problems, sorting and searching problems, and tiling problems. In Chapter 1 N-queen problems are first introduced for the reader to understand a basic neural network approach including how to represent a problem with artificial neurons called neural representation, how to construct the motion equation from the necessary and the sufficient constraints and/or the cost function in a given problem, and how to develop a software simulator on a Unix workstation, a

PC machine, or a Macintosh machine. Three Turbo Pascal programs are given to simulate the artificial neural network in sequential/parallel for N-queen problems in Chapter 1. The reader is encouraged to solve some of exercises in the end of every Chapter. After reading Chapter 1, Chapter 2 through 10 can be read independently except Chapter 5 and Chapter 6. In order to understand RNA secondary structure prediction problems in Chapter 6 and channel routing problems in Chapter 5, reading Chapter 4 is recommended. The important mathematical background involved in neural computing for optimization problems is summarized in Chapter 12. Chapter 13 shows on-going research applications including the module orientation problem, the maximum clique problem, the max cut problem, and other crossbar switch scheduling applications. Chapter 13 also depicts the future research of neural computing for optimization problems. Finally Chapter 14 introduces Conjunctoids and artificial learning.

1.2 MATHEMATICAL NEURAL NETWORK MODELS

The mathematical model of the artificial neural network consists of two components; neurons and synaptic links. The output signal transmitted from a neuron propagates to other neurons through the synaptic links. The state of the input signal of a neuron is determined by the linear sum of weighted input signals from the other neurons where the respective weight is the strength of the synaptic links. Every artificial neuron has the input U and the output V. The output of the *i*th neuron is given by $V_i=f(U_i)$ where f is called the neuron's input/output function. In this book we introduce a sigmoid function, a McCulloch-Pitts function, a hysteresis McCulloch-Pitts function, a modified McCulloch-Pitts function, and a maximum function as the neuron's input/output function.

The interconnections between the *i*th neuron and other neurons are determined by the motion equation. The change of the input state of the *i*th neuron is given by the partial derivatives of the computational energy function E with respect to the output of the *i*th neuron where E follows an n-variable function: $E(V_1, V_2,...,V_n)$. The motion equation of the *i*th neuron is given by:

$$\frac{dU_i}{dt} = - \frac{\partial E(V_1, V_2,..., V_n)}{\partial V_i} \qquad (1.1)$$

The goal of the artificial neural network for solving optimization problems is to minimize the fabricated computational energy function E in Eq. (1.1). The energy function not only determines how many neurons should be used in the system but also the strength of the synaptic links between neurons. It is constructed by considering the necessary and sufficient constraints and/or the cost function from the given problem. It is usually easier to build the motion equation than the energy function. From Eq. (1.1) the energy function E can be obtained:

$$E = \int dE = - \int \frac{dU_i}{dt} dV_i \qquad (1.2)$$

The artificial neural network provides a parallel gradient descent method to minimize the fabricated energy function E. Six convergence theorems/proofs of the artificial neural network are given in Chapter 12 to provide a mathematical background: 1) the proof of the harmfulness of using the decay term in the motion equation although it is still widely believed that the decay term is absolutely necessary, 2) the convergence theorem of the analog neural network without the decay term, 3) the convergence theorem of the McCulloch-Pitts neural network, 4) the convergence theorem of the discrete neural network, 5) the convergence theorem of the hysteresis McCulloch-Pitts neural network, and 6) the convergence theorem of the maximum neural network. It is not required for the novice reader to fully understand all of the mathematical background in Chapter 12 but it may be needed for the advanced reader.

The input/output function of the McCulloch-Pitts binary neuron model is shown in Fig. 1-1, that of the hysteresis McCulloch-Pitts neuron model in Fig. 1-2, and that of the sigmoid neuron model in Fig. 1-3 respectively.

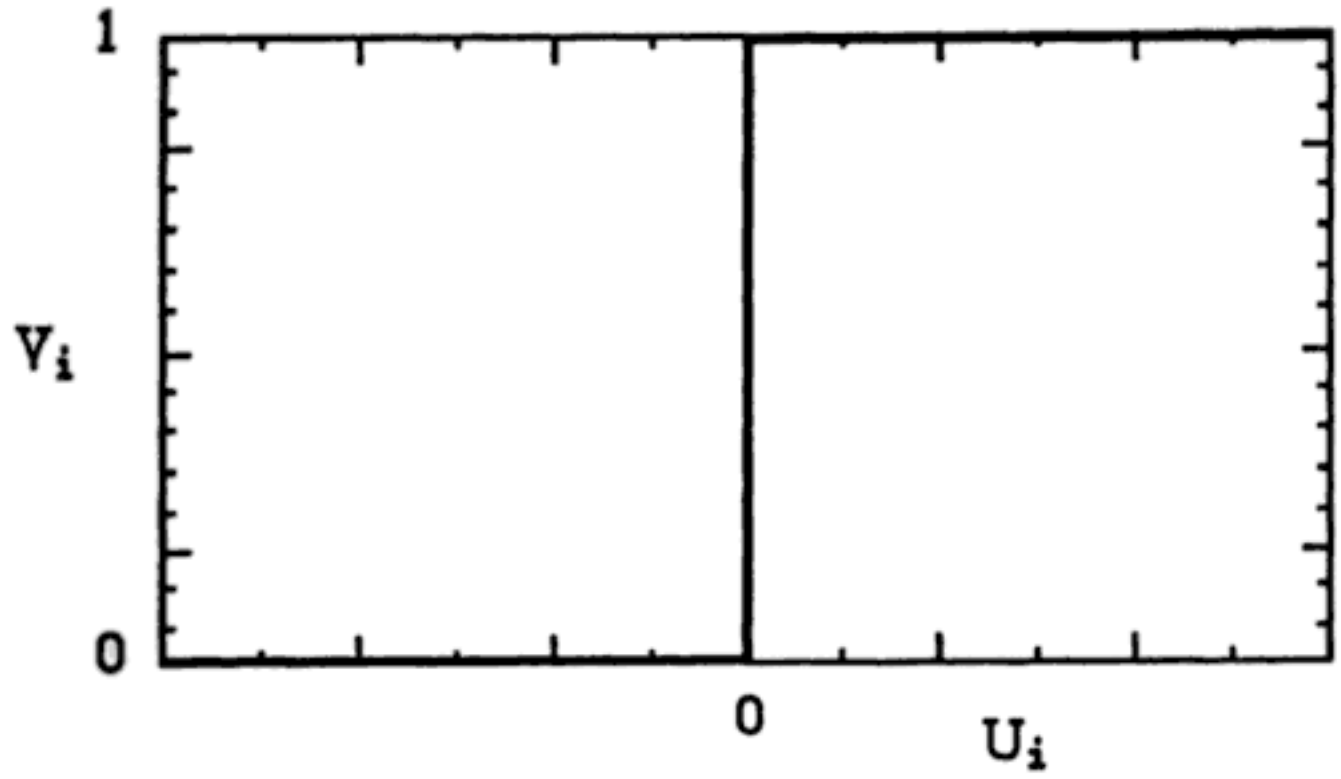


Fig. 1-1 McCulloch-Pitts Input/Output function

The McCulloch-Pitts input/output function is given by:

$V_i=f(U_i)=1$ if $U_i>0$
0 otherwise (1.3)

where V_i and U_i are the output and the input of the *i*th neuron respectively. If the energy function follows the quadratic form, U_i will be given by: $U_i=\sum_k W_{ki}V_k$ where W_{ki} is the strength of the synaptic link from the *k*th neuron to the *i*th neuron. The state of the McCulloch-Pitts neural network is allowed to converge to the local minimum where the convergence speed is relatively faster than that of the sigmoid neural network. However the McCulloch-Pitts neural network sometimes generates undesirable oscillatory behaviors. In order to suppress oscillatory behaviors, the hysteresis McCulloch-Pitts neuron model is introduced where the input/output function of the *i*th hysteresis neuron is given by:

V_i=1 if U_i>UTP (Upper Trip Point)
=0 if U_i<LTP (Lower Trip Point)
unchanged otherwise (1.4)

where UTP is always larger than LTP. Because of suppressing the oscillatory

behavior it shortens the convergence time consequently.

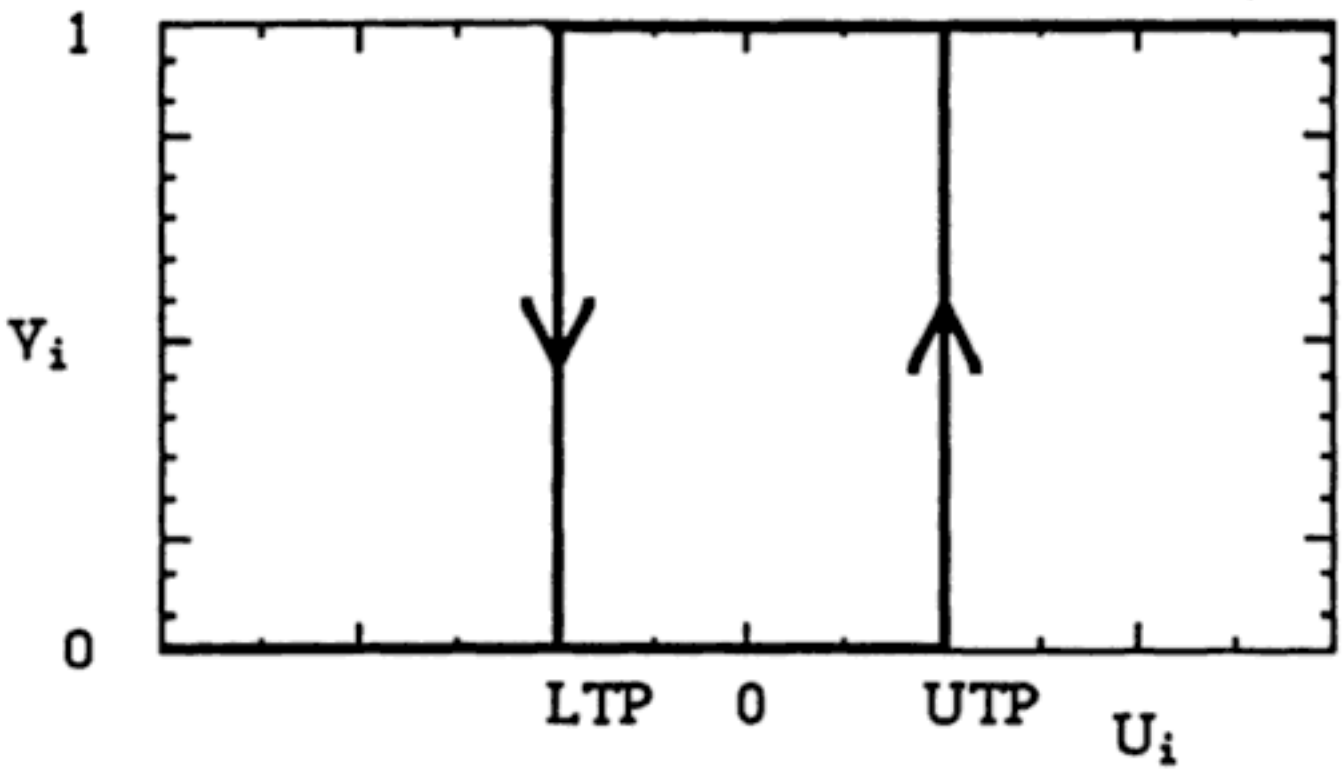


Fig.1-2 Hysteresis McCulloch-Pitts input/output function

The neural network for combinatorial optimization problems was first introduced by Hopfield and Tank in 1985 (Hopfield and Tank 1985). They use the predefined energy function E which follows the quadratic form:

$$E=\sum_{i=1}^{N}\sum_{j=1}^{N} W_{ij}V_iV_j+\sum_{i=1}^{N} V_iI_i \qquad (1.5)$$

where W_{ij} is the strength of a synaptic link between the *i*th and the *j*th neuron where the condition of $W_{ij}=W_{ji}$ must be always satisfied. Note that I_i is the constant bias of the *i*th neuron.

Hopfield gives the motion equation of the *i*th neuron (Hopfield and Tank 1985):

$$\frac{dU_i}{dt}=-\frac{U_i}{\tau}-\frac{\partial E}{\partial V_i} \qquad (1.6)$$

where the output follows the continuous, nondecreasing, and differentiable function called sigmoid function:

$$V_i=f(U_i)=\frac{1}{2}(\tanh(\lambda_0 U_i)+1) \qquad (1.7)$$

where λ_0 is a constant which is called gain and it determines the slope of the sigmoid function.

Wilson and Pawley strongly criticized the Hopfield and Tank neural network through the travelling salesman problem (Wilson and Pawley 1988). Unfortunately Wilson and Pawley did not know what causes the problem. The use of the decay term ($-U_i/\tau$) in Eq. (1.6) increases the computational energy function E under some conditions instead of decreasing it. The detail of the conditions and the proof are given in Chapter 12.

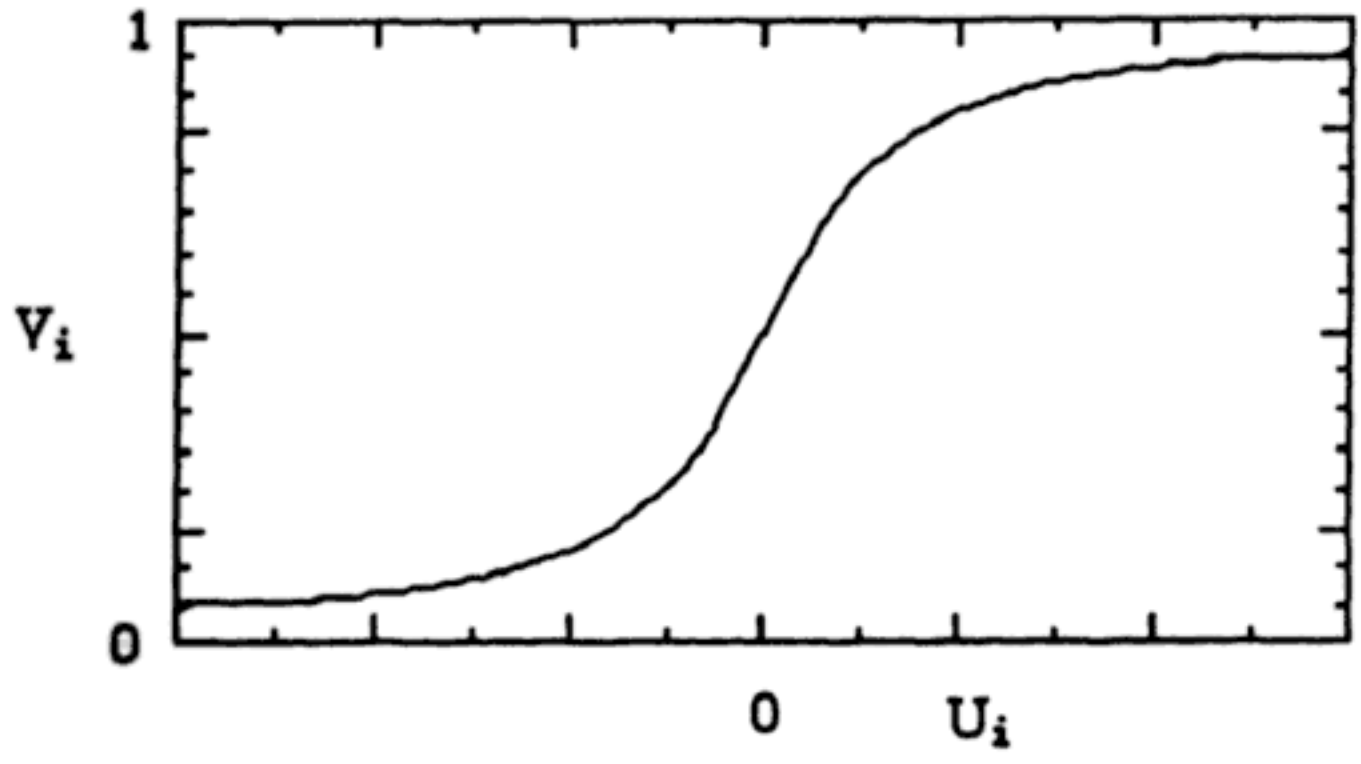


Fig. 1-3 Sigmoid input/output function

In the McCulloch-Pitts neural network with/without hysteresis or in the sigmoid neural network it is not always guaranteed that the converged state is equivalent to an acceptable solution. The acceptable solution means that it satisfies the required constraints, however it may or may not be the best solution. However in the maximum neural network it is always guaranteed to generate acccptable solutions. The maximum neural network is composed of M clusters where each cluster consists of n neurons. One and only one neuron among n neurons with the maximum input per cluster is encouraged to fire in the maximum neural network. The input/output function of the *i*th maximum neuron in the *m*th cluster is given by: $V_{m\,i}=1$ if $U_{m\,i}=\max\{U_{m\,1},..,U_{m\,n}\}$ and $U_{m\,i}\geq U_{m\,j}$ for $i>j$, and 0 otherwise. In the

maximum neuron model it is always guaranteed to keep one and only one neuron to fire per cluster. The maximum neuron model has the following advantages: 1) every converged state is equivalent to a feasible solution; 2) tuning the coefficients parameters is not needed; and 3) the termination condition of the equilibrium state is given by the simple mathematical formula, while the other existing neural network models must suffer from the unacceptable solution from the converged state (where the definition of the convergence condition is mathematically unclear) and must tune the parameters in the motion equation very carefully.

1.3 N-QUEEN NEURAL NETWORK

The 8-queen problem was proposed in 1848 and it was investigated by several famous mathematicians including C. F. Gauss in 1850 where the goal of the problem is to place 8 queens on an 8 x 8 chessboard in mutually nonattacking positions. The 8-queen problem has been used as a benchmark problem to demonstrate divide-and-conquer methods (Abramson and Yung 1989), trial-and-error methods including back-tracking algorithms (Bitner and Reingold 1975) (Stone and Stone 1987), and other methods (Yaglom and Yaglom 1964) (Kale 1990). Although few parallel algorithms have been proposed (Filman and Friedman 1984) (Finkel and Manber 1987) (Abramson and Yung 1989), there exists no satisfactory parallel algorithm. Page et al. presented the first deterministic parallel algorithm based on the Hopfield neural network (Page and Tagliarini 1987). Akiyama et al. proposed an $O(N^2)$ parallel algorithm based on the stochastic neural network model (Akiyama et al. 1989). However the state of the system based on any one of the existing neural network algorithms often converges to the unacceptable local minimum. In other words, their solution quality dramatically degrades with the problem size.

The goal of a general N-queen problem is to locate N queens on an N x N chessboard where any pair of queens does not command each other. Note that a queen commands vertically, horizontally, and diagonally as shown in Fig. 1-4.

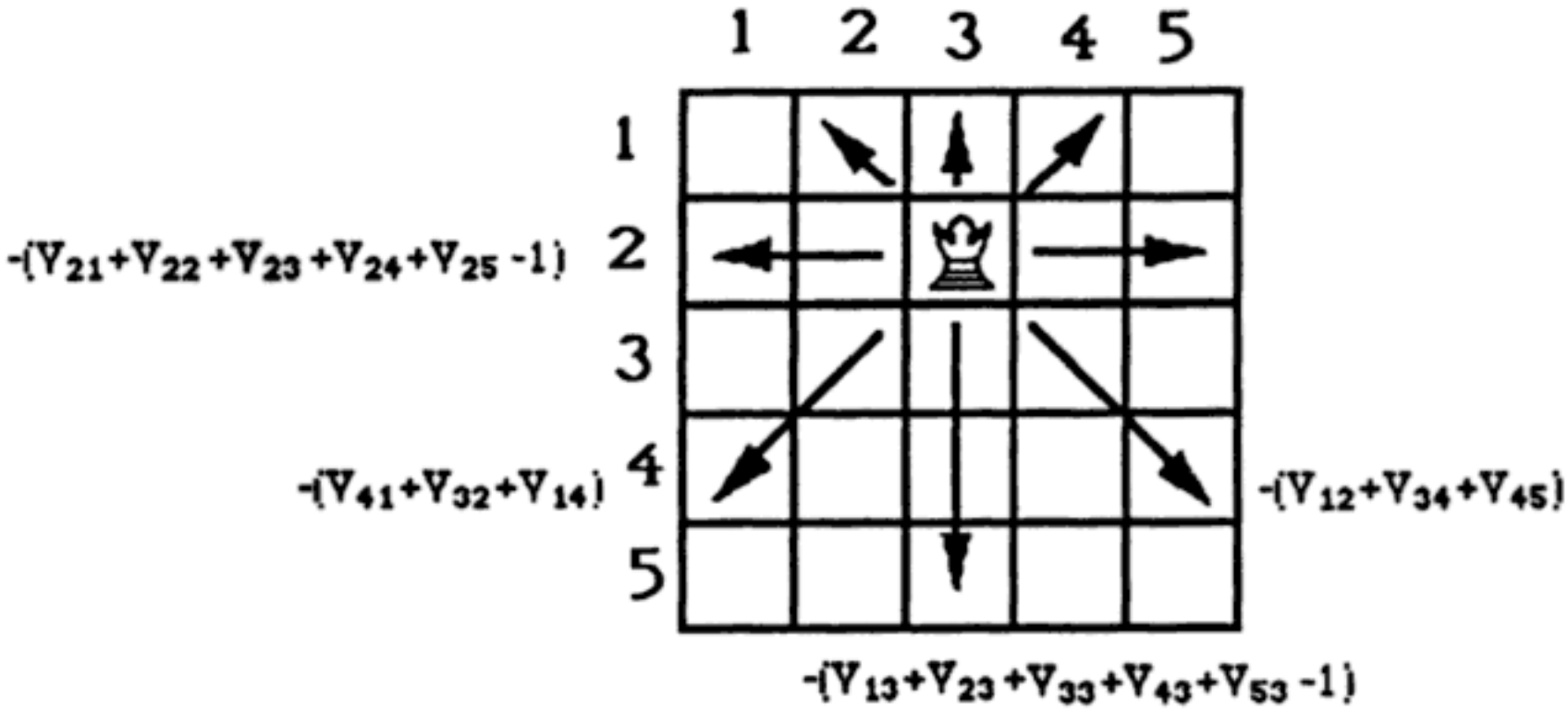


Fig. 1-4 How a queen commands

It is well understood that one and only one queen must be located per row and per column so as to locate N queens on an N x N chessboard. In other words more than one queen should not be located per row and per column. The condition of the constraints or the violations gives the interconnections of the artificial neural network. An N x N neural array is prepared for the N-queen problem where each element of the neural array represents a square of the N x N chessboard. The output state of the *ij*th neuron gives the location of a queen in the *i*th row and the *j*th column. In other words, $V_{ij}=1$ means that a queen is located in the *i*th row and the *j*th column. $V_{ij}=0$ means that no queen is located in the *i*th row and the *j*th column. For example, to locate a queen in the second row and the third column as shown in Fig. 1-4, V_{23} must be $V_{23}=1$. The motion equation of the *ij*th neuron for the N-queen problem is given by:

$$\frac{dU_{ij}}{dt} = -A\left(\sum_{k=1}^{N} V_{ik}-1\right) -A\left(\sum_{k=1}^{N} V_{kj}-1\right) -B \sum_{\substack{1\le i-k,j-k\le N \\ k\neq 0}} V_{i-k,j-k} -B \sum_{\substack{1\le i-k,j+k\le N \\ k\neq 0}} V_{i-k,j+k} \qquad \text{for } i,j=1,..,5 \qquad (1.8)$$

In Eq. (1.8) the first term gives the row constraint such that one and only one queen/neuron should be located/fired in the *i*th row. The second term describes the

column constraint such that one and only one queen/neuron should be located/fired in the *j*th column. The third and the fourth term represent the diagonal constraints such that no pair of queens should not diagonally command each other. The following two terms must be added to Eq. (1.8) to enable the state of the system to escape from the local minimum and to converge to the global minimum or the acceptable solution:

$$+Ch\left(\sum_{k=1}^{N} V_{ik}\right) +Ch\left(\sum_{k=1}^{N} V_{kj}\right) \qquad (1.9)$$

where h(x) is 1 if x=0, and 0 otherwise. The function of h(x) is called hill-climbing term. In the local minimum no neurons/queens are fired/located on the *i*th row or the *j*th column. For example in the 5-queen problem with A=B=C=1, the motion equation of the neuron in the second row and the third column as shown in Fig. 1-4 is given by:

$$\Delta U_{23} = \frac{dU_{23}}{dt} = -(V_{21}+V_{22}+V_{23}+V_{24}+V_{25}-1) -(V_{13}+V_{23}+V_{33}+V_{43}+V_{53}-1)$$
$$-(V_{12}+V_{34}+V_{45}) -(V_{14}+V_{32}+V_{41})$$
$$+h(V_{21}+V_{22}+V_{23}+V_{24}+V_{25})+h(V_{13}+V_{23}+V_{33}+V_{43}+V_{53}) \qquad (1.10)$$

Fig. 1-5 and Fig. 1-6 show one of the solutions for the 5-queen problem and one of the local minima for the 6-queen problem respectively. Remember that in the local minima less than N queens are placed on an N x N chessboard but any additional queen cannot be placed in any row or column. For example, as shown in Fig. 1-6 we cannot locate any queen in the second column or in the fourth row so that we say, the state of the system is in the local minimum.

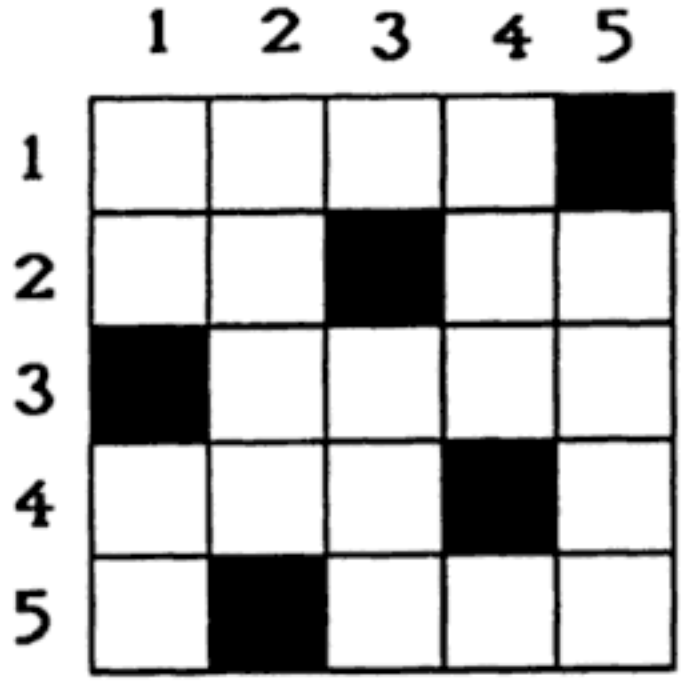


Fig. 1-5 One of the solutions

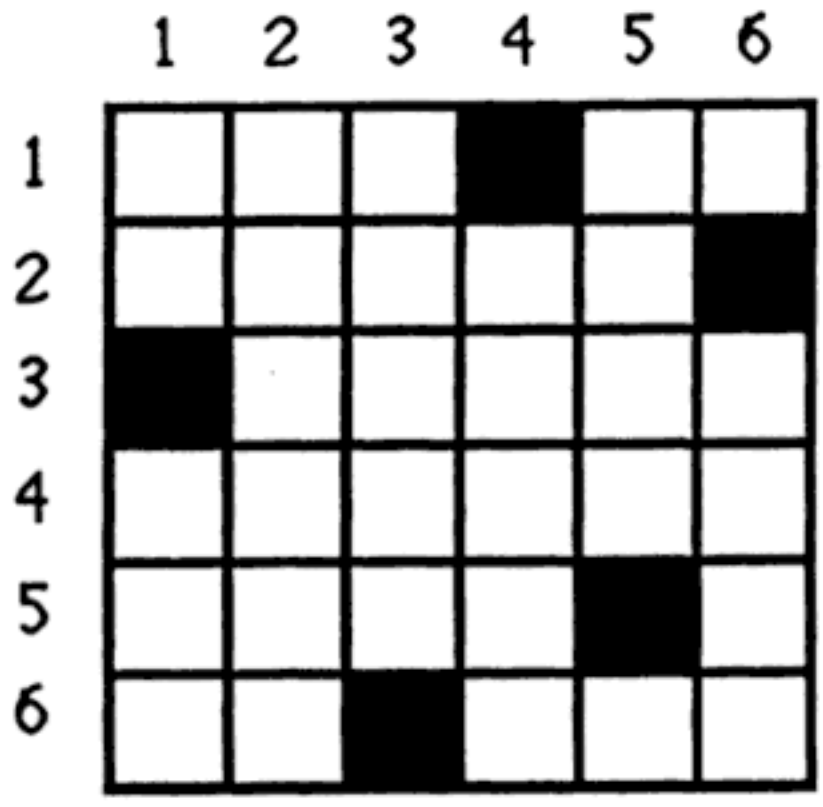


Fig. 1-6 One of the local minima

1.4 GENERAL OPTIMIZATION PROGRAMS

Since a parallel machine is not widely available to many readers for parallel simulation, two programs are provided to simulate the behavior of the parallel machine using a sequential machine as if the program is running on the parallel machine. In order to distinguish a sequential model and two parallel models, three general programs for solving optimization problems are given: Program1 to simulate a sequential neural network model, Program2 for a synchronous parallel neural network model using a sequential machine, and Program 3 for an asynchronous parallel neural network model using a sequential machine.

In order to numerically solve the partial differential equation or the differential equation, the first order Euler method is widely used where it is the simplest among the existing numerical methods. Based on the first order Euler method, the value of U(t+1) is determined by U(t) and ΔU(t):

$$U(t+1)=U(t)+\Delta U(t)\cdot \Delta t \qquad (1.11)$$

where ΔU(t) is given by Eq. (1.1). With Δt=1, Eq. (1.11) for N-queen problems (where a two dimensional neural array is used) is given by:

$$U_{ij}(t+1)=U_{ij}(t)+\Delta U_{ij}(t) \quad \text{for } i,j=1,...,N \qquad (1.12)$$

In order to solve N-queen problems, the initial values of $U_{ij}(0)$ for i,j=1,...,N are randomly given and Eq. (1.12) is iteratively used until the state of the system reaches the equilibrium state. However the equilibrium state is not always equivalent to the acceptable solution. In the N-queen problem, the condition of the program termination is given by:

$$\Delta U_{ij}(t)=0 \quad \text{for } i,j=1,...,N \qquad (1.13)$$

The condition of Eq. (1.13) implies that the constraints are all satisfied. In other words, no violations can be found on the condition. Eq. (1.12) is either sequentially updated or parallelly updated. In parallel computing there are a synchronous parallel model and an asynchronous parallel model. In the synchronous parallel model, clock signals or handshaking signals are used for synchronous communications between processing elements (neurons). In the asynchronous parallel model the state of every processing element (neuron) is updated asynchronously without using any clock/handshaking signals. In Fig. 1-7 Program1 shows a general program for simulating the sequential neural network model where a two dimensional neural representation and the McCulloch-Pitts neuron model are used. Of course, it is possible to use a single dimensional neural representation or a higher dimensional neural representation in Program1. As soon as updating the value of U_{ij}, the value of V_{ij} is evaluated based on the input/output function of the neuron model in Program1.

It is quite straightforward to simulate a parallel computing model using a sequential machine. Of course if the reader is able to use a parallel machine the motion equations can be solved parallelly on it. There are two parallel computing models: a synchronous parallel model and an asynchronous parallel model. Program2 in Fig. 1-8 illustrates a program to simulate the synchronous parallel model which runs on a sequential machine as if it runs on a parallel machine. In the first loop all input values of U_{ij} for i,j=1,...,N are sequentially updated while all output values of V_{ij} for i,j=1,...,N are fixed. Then in the second loop all output values of V_{ij} for i,j=1,...,N are sequentially updated while all input values of U_{ij} for i,j=1,...,N are fixed. It is equivalent to simultaneously updating the values of all inputs and outputs. In Program2 the McCulloch-Pitts neuron model is used:

```
Program1 sequential-simulator
begin
initialization of Uij and Vij for i,j:=1 to N;
/*** Main Program ***/
 while (a set of conflicts is not empty) do
 for i:=1 to  N
 for j:=1 to  N
       begin
             Uij:=Uij+ ΔUij;
             If Uij>0 then Vij:=1 else Vij:=0;
       end;

end;
/*** Main Program end ***/
```

Fig. 1-7 Simulator for a sequential neural network model

Program3 in Fig. 1-9 shows the asynchronous parallel simulator to run on a sequential machine. Program2 is very similar to Program3 except the statement of "If random<x then." The variable "random" is an integer with a range from 0 to y where x is the constant number and x satisfies 0<x<y. If random is smaller than x then update the state of the output, otherwise keep the previous state of the output. The statement of "If random<x then" provides the pseudo asynchronous system behavior in simulation.

```
Program2 synchronous-parallel-simulator
begin
initialization of Uij and Vij for i,j:=1 to N;
/*** Main Program ***/
  while (a set of conflicts is not empty) do
  begin
/*** The first loop ***/
    for i:=1 to N
    for j:=1 to N
      Uij:=Uij+ΔUij;
/*** End of the first loop ***/
/*** The second loop ***/
    for i:=1 to N
    for j:=1 to N
      If Uij>0 then Vij:=1 else Vij:=0;
/*** End of the second loop ***/
  end;
end;
/*** Main Program end ***/
```

Fig. 1-8 Synchronous parallel simulator on a sequential machine

```
Program3 Asynchronous-parallel-simulator
begin
initialization of Uij and Vij for i,j:=1 to N;
/*** Main Program ***/
  while (a set of conflicts is not empty) do
  begin
/*** The first loop ***/
    for i:=1 to N
    for j:=1 to N
      Uij:=Uij+ΔUij;
/*** End of the first loop ***/
/*** The second loop ***/
    for i:=1 to N
    for j:=1 to N
     If random<x then
       If Uij>0 then Vij:=1 else Vij:=0;
/*** End of the second loop ***/
  end;
end;
/*** Main Program end ***/
```

Fig. 1-9 Asynchronous parallel simulator on a sequential machine

1.5 N-QUEEN SIMULATION PROGRAMS

Fig. 1-10 shows a Turbo Pascal program which is based on Program1 as shown in Fig. 1-7. It is ready to run on any Macintosh machine. The program is to simulate the sequential N-queen neural network model. In the program the first term and the second term of the motion equation in Eq. (1.8) are realized by:

```
          sum_column:=0;sum_row:=0;
          for k:=1 to max do begin sum_row:=sum_row+V[i,k];
sum_column:=sum_column+V[k,j]; end;
```

where max is the N-queen problem size. The third term and the fourth term in Eq. (1.8) or two diagonal constraints are given by:

```
          diagonal1:=0; k:=1;
         while((j+k)<=max) and ((i-k)>=1) do begin diagonal1:=diagonal1+V[i-
k,j+k]; k:=k+1; end;
          k:=1;
```

```
        while((j-k)>=1) and ((i+k)<=max) do begin diagonal1:=diagonal1+V[i+k,j-
k]; k:=k+1; end;
         diagonal2:=0; k:=1;
        while((j+k)<=max) and ((i+k)<=max) do begin
diagonal2:=diagonal2+V[i+k,j+k]; k:=k+1; end;
         k:=1;
        while((j-k)>=1) and ((i-k)>=1) do begin diagonal2:=diagonal2+V[i-k,j-k];
k:=k+1; end;
```

The hill-climbing term in Eq. (1.9) is given by:

```
      h:=0;
      if (sum_column=0) then h:=1; if (sum_row=0) then h:=h+1;
```

Therefore the new input state of the *ij*th neuron in Eq. (1.12) is finally computed by:

```
U[i,j]:=U[i,j]-A*(sum_row+sum_column-2)-B*(diagonal1+diagonal2)+C*h;
```

In order to shorten the convergence time, the values of the inputs are always kept in the certain range. The value of the input U_{ij}, the output value of the *ij*th neuron is evaluated by:

```
if (U[i,j]>15) then U[i,j]:=15; if (U[i,j]<-5) then U[i,j]:=-5;
        if U[i,j]>0 then V[i,j]:=1 else V[i,j]:=0;
```

where the maximum and the minimum value of the input are given by +15 and -5 respectively. The termination condition in Eq. (1.13) is realized by:

```
diag:=1;
  while ((not keypressed) and (diag>0) and (t<2000)) do
   begin
    diag:=0;
    for i:=1 to max do
     for j:=1 to max do
      begin
       conf:=1;
        if((sum_column+sum_row=2) and (diagonal1<2) and (diagonal2<2)) then conf:=0;
       diag:=diag+conf;
      end;{******** The end of i and j loop ********}
end; {******** The end of while loop ********}
```

If diag=0 then the program will be automatically terminated. The following routine displays the simulation result of the N-queen problem on the Macintosh screen.

```
clearscreen;
 for i:=1 to max do
  for j:=1 to max do
   if (V[i,j]=1) then
     begin
      SetRect(tempRect,0,0,length,length); OffsetRect(tempRect,(i-1)*length,(j-
1)*length); PaintRect(tempRect);
     end;
   MoveTo(0,0); LineTo(0,max*length); MoveTo(0,max*length);
LineTo(max*length,max*length);
   MoveTo(max*length,max*length); LineTo(max*length,0);
MoveTo(max*length,0); LineTo(0,0);
 for i:=1 to max do
   begin
    MoveTo(i*length,0); LineTo(i*length,max*length); MoveTo(0,i*length);
LineTo(max*length,i*length);
   end;
gotoXY(50,1);writeln('No. of steps=',t);
```

Fig. 1-11 shows a Turbo Pascal program for simulating the synchronous parallel N-queen neural network model where the hysteresis McCulloch-Pitts neuron is used. UTP=3 and LTP=0 are used for hysteresis.

Fig. 1-12 depicts a Turbo Pascal program for simulating the asynchronous parallel N-queen neural network model where the same hysteresis McCulloch-Pitts neuron model is used. The difference between the sequential-simulation program in Fig. 1-10 and the synchronous-parallel-simulation program in Fig. 1-11 or the asynchronous-parallel simulation program in Fig. 1-12 lies in that the output values of the neurons are simultaneously updated outside of the motion equation loop for parallel simulation, while the output value of every neuron is individually computed as soon as the input of the neuron is evaluated inside of the motion equation loop for sequential simulation.

The asynchronous parallel system has a distinguished advantage over the synchronous parallel system. The asynchronous parallel system does not require any clock or synchronization mechanism. In Fig. 1-12 the program attempts to sequentially simulate the asynchronous parallel N-queen neural network model. The statement of "if (abs(random) mod 2 <1) then" in the output evaluation routine provides the pseudo asynchronous behavior in simulation.

Fig. 1-13 depicts several simulation results of N-queen problems using the illustrated programs in Fig. 1-10, Fig. 1-11, and Fig. 1-12. It takes about 11 minutes to solve a 100-queen problem on a Macintosh SE/30 where it only requires 36 iteration steps. If 10,000 processing elements are provided and assume that each processing element has the processing speed of one million state-updates per second (1 MSUPS), it will take 36 micro seconds to solve the 100-queen problem.

```
program sequential_queen;
uses memtypes, quickdraw,osintf,ToolIntf;
type twod_int= array[1..89,1..89] of integer;
var
A,B,C,t,h,i,j,k,sum_column,sum_row,diagonal1,diagonal2,conf,diag,seed,row,length
,max:integer;
U,V:twod_int; tempRect : Rect;
begin
    A:=1; B:=1; C:=1;
   writeln('Please define the queen problem size (6-89).'); readln(max);
   writeln('Please input a seed(0-9999).'); readln(seed);
   for i:=1 to seed do U[1,1]:=random; length:=trunc(270/max);
   for i:=1 to max do
     for j:=1 to max do
     begin U[i,j]:=-trunc(abs(random/4000)); if U[i,j]>0 then V[i,j]:=1 else V[i,j]:=0;
end;
{******** Main program ********}
   t:=0; diag:=1;
   while ((not keypressed) and (diag>0) and (t<2000)) do
     begin diag:=0;
       for i:=1 to max do
        for j:=1 to max do
         begin
           sum_column:=0;sum_row:=0;
 for k:=1 to max do begin sum_row:=sum_row+V[i,k];
 sum_column:=sum_column+V[k,j]; end;
 diagonal1:=0; k:=1;
while((j+k)<=max) and ((i-k)>=1)
do begin diagonal1:=diagonal1+V[i-k,j+k]; k:=k+1; end;
 k:=1;
while((j-k)>=1) and ((i+k)<=max)
do begin diagonal1:=diagonal1+V[i+k,j-k]; k:=k+1; end;
 diagonal2:=0; k:=1;
while((j+k)<=max) and ((i+k)<=max)
do begin diagonal2:=diagonal2+V[i+k,j+k]; k:=k+1; end;
 k:=1;
while((j-k)>=1) and ((i-k)>=1) do begin diagonal2:=diagonal2+V[i-k,j-k];
 k:=k+1; end;
h:=0;conf:=1;
```

```
        if (sum_column=0) then h:=1; if (sum_row=0) then h:=h+1;
       if((sum_column+sum_row=2) and (diagonal1<2) and (diagonal2<2)) then
        conf:=0;
       U[i,j]:=U[i,j]-A*(sum_row+sum_column-2)-B*(diagonal1+diagonal2)+C*h;
        if (U[i,j]>15) then U[i,j]:=15; if (U[i,j]<-5) then U[i,j]:=-5;
            if U[i,j]>0 then V[i,j]:=1 else V[i,j]:=0;
      diag:=diag+conf;
    end;{********* The end of i and j loop *********}
     t:=t+1; if (t mod 20 <5) then C:=4 else C:=1; write(t);
 end; {********* The end of while loop *********}
clearscreen;
for i:=1 to max do
 for j:=1 to max do
   if (V[i,j]=1) then
     begin
      SetRect(tempRect,0,0,length,length); OffsetRect(tempRect,(i-1)*length,(j-
1)*length); PaintRect(tempRect);
     end;
   MoveTo(0,0); LineTo(0,max*length); MoveTo(0,max*length);
LineTo(max*length,max*length);
   MoveTo(max*length,max*length); LineTo(max*length,0);
MoveTo(max*length,0); LineTo(0,0);
 for i:=1 to max do
   begin
    MoveTo(i*length,0); LineTo(i*length,max*length); MoveTo(0,i*length);
LineTo(max*length,i*length);
  end;
gotoXY(50,1);writeln('No. of steps=',t);readln;readln;
end. {********* The end of program *********}
```

Fig. 1-10 A program of the sequential N-queen neural network

```
program sync_parallel_queen;
uses memtypes, quickdraw,osintf,ToolIntf;
type twod_int= array[1..89,1..89] of integer;
var
A,B,C,t,h,i,j,k,sum_column,sum_row,diagonal1,diagonal2,conf,diag,seed,row,length
,max:integer;
U,V:twod_int; tempRect : Rect;
begin
   A:=1; B:=1; C:=1;
   writeln('Please define the queen problem size (6-89).'); readln(max);
   writeln('Please input a seed(0-9999).'); readln(seed);
   for i:=1 to seed do U[1,1]:=random; length:=trunc(270/max);
   for i:=1 to max do
     for j:=1 to max do
```

```
    begin U[i,j]:=-trunc(abs(random/1600)); if U[i,j]>0 then V[i,j]:=1 else V[i,j]:=0;
end;
{******** Main program ********}
   t:=0; diag:=1;
   while ((not keypressed) and (diag>0) and (t<2000)) do
     begin diag:=0;
       for i:=1 to max do
        for j:=1 to max do
         begin
           sum_column:=0;sum_row:=0;
           for k:=1 to max do begin sum_row:=sum_row+V[i,k];
sum_column:=sum_column+V[k,j]; end;
           diagonal1:=0; k:=1;
          while((j+k)<=max) and ((i-k)>=1) do begin diagonal1:=diagonal1+V[i-
k,j+k]; k:=k+1; end;
           k:=1;
          while((j-k)>=1) and ((i+k)<=max) do begin diagonal1:=diagonal1+V[i+k,j-
k]; k:=k+1; end;
           diagonal2:=0; k:=1;
          while((j+k)<=max) and ((i+k)<=max) do begin
diagonal2:=diagonal2+V[i+k,j+k]; k:=k+1; end;
           k:=1;
          while((j-k)>=1) and ((i-k)>=1) do begin diagonal2:=diagonal2+V[i-k,j-k];
k:=k+1; end;
       h:=0;conf:=1;
       if (sum_column=0) then h:=1; if (sum_row=0) then h:=h+1;
      if((sum_column+sum_row=2) and (diagonal1<2) and (diagonal2<2)) then
conf:=0;
      U[i,j]:=U[i,j]-A*(sum_row+sum_column-2)-B*(diagonal1+diagonal2)+C*h;;
       if (U[i,j]>15) then U[i,j]:=15; if (U[i,j]<-20) then U[i,j]:=-20;
       diag:=diag+conf;
    end;{******** The end of i and j loop ********}
for i:=1 to max do
    for j:=1 to max do
    begin if U[i,j]>3 then V[i,j]:=1; if U[i,j]<0 then V[i,j]:=0; end;
     t:=t+1; if (t mod 20 <5) then C:=4 else C:=1; write(t);
 end; {******** The end of while loop ********}
clearscreen;
for i:=1 to max do
 for j:=1 to max do
  if (V[i,j]=1) then
    begin
     SetRect(tempRect,0,0,length,length); OffsetRect(tempRect,(i-1)*length,(j-
1)*length); PaintRect(tempRect);
    end;
  MoveTo(0,0); LineTo(0,max*length); MoveTo(0,max*length);
LineTo(max*length,max*length);
  MoveTo(max*length,max*length); LineTo(max*length,0);
MoveTo(max*length,0); LineTo(0,0);
```

```
 for i:=1 to max do
   begin
    MoveTo(i*length,0); LineTo(i*length,max*length); MoveTo(0,i*length);
LineTo(max*length,i*length);
   end;
gotoXY(50,1);writeln('No. of steps=',t);readln;readln;
end. {******** The end of program ********}
```

Fig. 1-11 A program of the synchronous-parallel N-queen neural network

```
program async_parallel_queen;
uses memtypes, quickdraw,osintf,ToolIntf;
type twod_int= array[1..89,1..89] of integer;
var
A,B,C,t,h,i,j,k,sum_column,sum_row,diagonal1,diagonal2,conf,diag,seed,row,length
,max:integer;
U,V:twod_int; tempRect : Rect;
begin
   A:=1; B:=1; C:=1;
   writeln('Please define the queen problem size (6-89).'); readln(max);
   writeln('Please input a seed(0-9999).'); readln(seed);
   for i:=1 to seed do U[1,1]:=random; length:=trunc(270/max);
   for i:=1 to max do
     for j:=1 to max do
     begin U[i,j]:=-trunc(abs(random/1600)); if U[i,j]>0 then V[i,j]:=1 else V[i,j]:=0;
end;
{******** Main program ********}
   t:=0; diag:=1;
   while ((not keypressed) and (diag>0) and (t<2000)) do
     begin diag:=0;
       for i:=1 to max do
        for j:=1 to max do
         begin
          sum_column:=0;sum_row:=0;
          for k:=1 to max do begin sum_row:=sum_row+V[i,k];
sum_column:=sum_column+V[k,j]; end;
          diagonal1:=0; k:=1;
         while((j+k)<=max) and ((i-k)>=1) do begin diagonal1:=diagonal1+V[i-
k,j+k]; k:=k+1; end;
          k:=1;
         while((j-k)>=1) and ((i+k)<=max) do begin diagonal1:=diagonal1+V[i+k,j-
k]; k:=k+1; end;
          diagonal2:=0; k:=1;
         while((j+k)<=max) and ((i+k)<=max) do begin
diagonal2:=diagonal2+V[i+k,j+k]; k:=k+1; end;
          k:=1;
         while((j-k)>=1) and ((i-k)>=1) do begin diagonal2:=diagonal2+V[i-k,j-k];
```

```
k:=k+1; end;
        h:=0;conf:=1;
        if (sum_column=0) then h:=1; if (sum_row=0) then h:=h+1;
       if((sum_column+sum_row=2) and (diagonal1<2) and (diagonal2<2)) then
conf:=0;
       U[i,j]:=U[i,j]-A*(sum_row+sum_column-2)-B*(diagonal1+diagonal2)+C*h;;
        if (U[i,j]>15) then U[i,j]:=15; if (U[i,j]<-20) then U[i,j]:=-20;
        diag:=diag+conf;
      end;{******** The end of i and j loop ********}
for i:=1 to max do
     for j:=1 to max do if (abs(random) mod 2 <1) then
     begin if U[i,j]>3 then V[i,j]:=1; if U[i,j]<0 then V[i,j]:=0; end;
      t:=t+1; if (t mod 20 <5) then C:=4 else C:=1; write(t);
 end; {******** The end of while loop ********}
clearscreen;
for i:=1 to max do
 for j:=1 to max do
  if (V[i,j]=1) then
    begin
     SetRect(tempRect,0,0,length,length); OffsetRect(tempRect,(i-1)*length,(j-
1)*length); PaintRect(tempRect);
    end;
  MoveTo(0,0); LineTo(0,max*length); MoveTo(0,max*length);
LineTo(max*length,max*length);
  MoveTo(max*length,max*length); LineTo(max*length,0);
MoveTo(max*length,0); LineTo(0,0);
for i:=1 to max do
  begin
    MoveTo(i*length,0); LineTo(i*length,max*length); MoveTo(0,i*length);
LineTo(max*length,i*length);
  end;
gotoXY(50,1);writeln('No. of steps=',t);readln;readln;
end. {******** The end of program ********}
```

Fig. 1-12 A program for simulating the asynchronous-parallel N-queen neural network model

1.6 REFERENCES

Abramson B., and Yung M., (1989), Divide and conquer under global constraints: a solution to the N-queens problem, J. of Parallel and Distributed Computing, 6, 649-662.

Akiyama Y., Yamashita A., Kajiura M., and Aiso H., (1989), Combinatorial

optimization with **Gaussian machines.** Proc. of the Int. Joint Conf. on Neural Networks.

Bitner J, and Reingold **E. M., (1975), Backtrack programming techniques,** Comm. ACM 18, 651-655.

Filman R. E., and **Friedman D. P., (1984),** Coordinated computing: tools and techniques for distributed software, **(McGraw-Hill, New York).**

Finkel R., and Manber **U., (1987), DIB-a distributed implementation of backtracking,** ACM Trans. Programming Language Systems, **9, 235-256.**

Hebb D. O, (1949), **The organization of behavior (New York: Wiley).**

Hopfield J. J., and **Tank D. W., (1985), Neural Computation of Decisions in** Optimization Problems. **Biological Cybernetics, 52, 141-152.**

Kale L.V., (1990), **An almost perfect heuristic for the N nonattacking queens** problem, Information Processing Letters, **34, 173-178.**

McCulloch W. S., and Pitts **W. H., (1943), A logical calculus of ideas immanent in** nervous activity. Bulletin of Mathematical Biophysics, **5, 115.**

Minsky M, and Papert **S., (1969),** Perceptrons **(MIT Press).**

Page E., and Tagliarini **G. A, (1987), Solving constraint satisfaction problems with** neural networks, Proc. of IEEE first Int. Conf. on Neural Networks.

Paielli R. A., (1988), **Simulation tests of the optimization method of Hopfield and** Tank using neural networks, NASA Technical Memorandom 101047.

Rosenblatt F., (1962), **Principles of neurodynamics (New York: Spartan).**

Stone H., and Stone **J. M., (1987), Efficient search techniques-an empirical study of** N-queens problem, IBM J. Res. Develop., **31, 4.**

Widrow B, and Hoff **M. E., (Sept. 1960), Adaptive switching circuits.** Proc. of IREWESCON Convention Record.

Wilson G. V. and **Pawley G. S., (1988), On stability of the Travelling Salesman** **Problem** Algorithm **of Hopfield and Tank.** Bio. Cybern., **58, 63-70.**

Yaglom A. M., and Yaglom I. M., (1964), Challenging mathematical problems with elementary solutions, (Holden-Day, San Francisco).

1.7 EXERCISES

1. Simulate the 5-queen neural network using the sequential program in Fig. 1-10 in order to observe whether the system has the local minimum or not. If yes, give an example of the local minima. Create some of the local minima in thc 8-queen neural network using the program in Fig. 1-10.

2. Simulate the 8-queen neural network using the sequential, thc synchronous parallel, and the asynchronous parallel simulation program. Collect the data on the frequency of failures and on the average number of iteration steps to converge to solutions in the successful cases. Investigate the behavior of every system with or without hysteresis, and with or without the hill-climbing term. Observe the effect of hysteresis by changing the value of UTP and LTP respectively. Explain a role of the statement of "if (t mod 20 <5) then C:=4 else C:=1;" in every program in Fig. 1-10, Fig. 1-11, and Fig. 1-12. Observe the effect of the hill-climbing term by changing the value of the coefficient C and the period in the motion equation.

3. Why is the statement of "if (U[i,j]>15) then U[i,j]:=15; if (U[i,j]<-20) then U[i,j]:=-20;" used in the program in Fig. 1-11 and Fig. 1-12? Remove the similar statement from the program in Fig. 1-10 and investigate the behavior of the system in terms of the number of iteration steps.

4. Plot the relationship between the problem size and the number of average iteration steps where at least 100 simulation runs must be performed in every problem. The range of the problem size is from 6 to 80.

5. Compare some of the conventional algorithms with the neural network algorithm for N-queen problems in terms of the computation time and the system space.

6. On an 8 x 8 chessboard we want to command 64 squares by using the minimum number of queens.

6.1 Give the neural representation and the motion equation.

6.2 Write a program based on the result of 6.1.

6.3 Collect the data on the frequency of failures and the average number of iteration steps to converge to the solution.

7. Examine the same problem using the minimum number of knights on an 8 x 8 chessboard.

8. Write a program for solving the N-superqueen problem where a solution of the N-queen problem gives a solution of the (N-1)-queen problem by removing the top row and leftmost column from the N x N chessboard. This leaves an (N-1) x (N-1) chessboard with (N-1) mutually nonattacking queens.

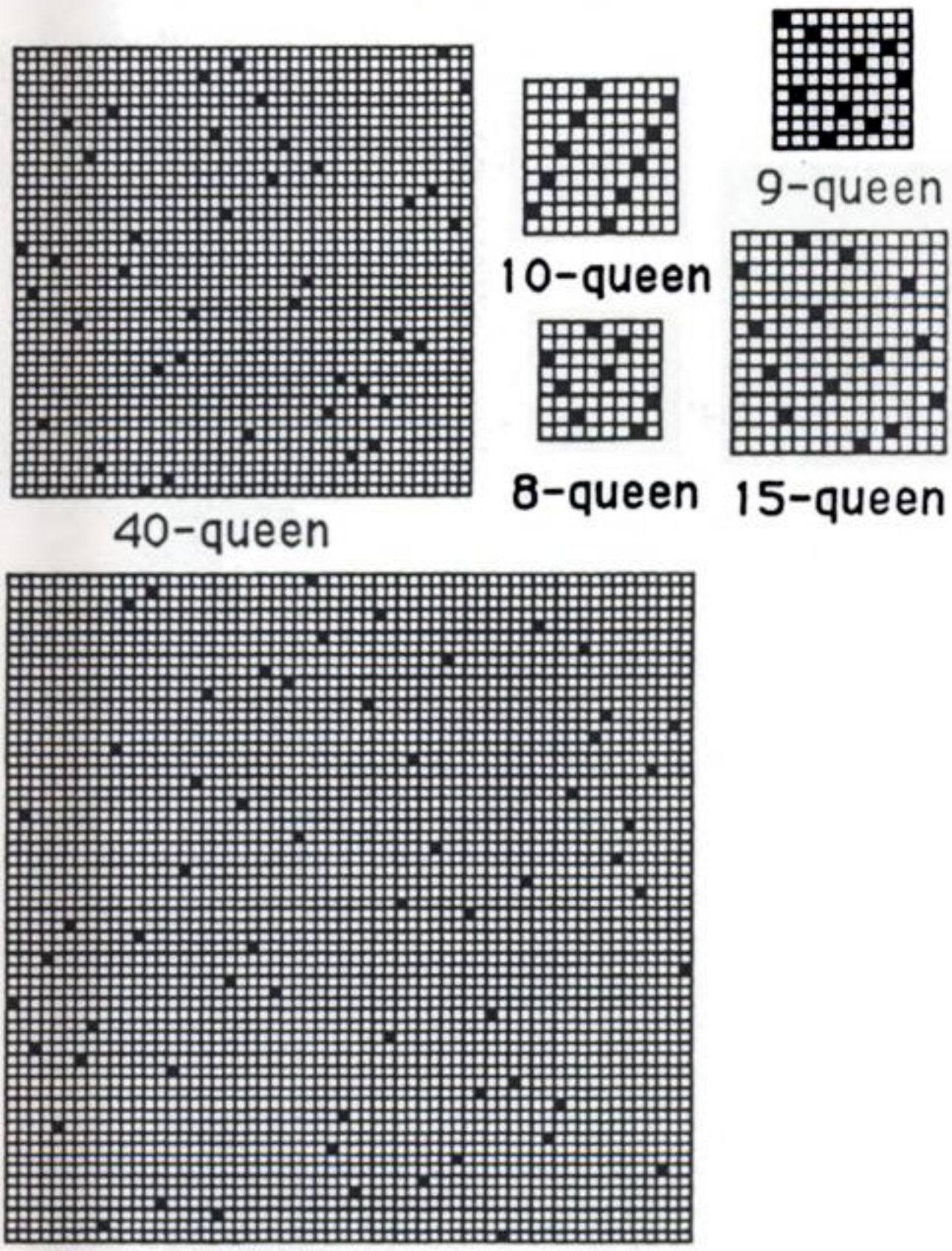
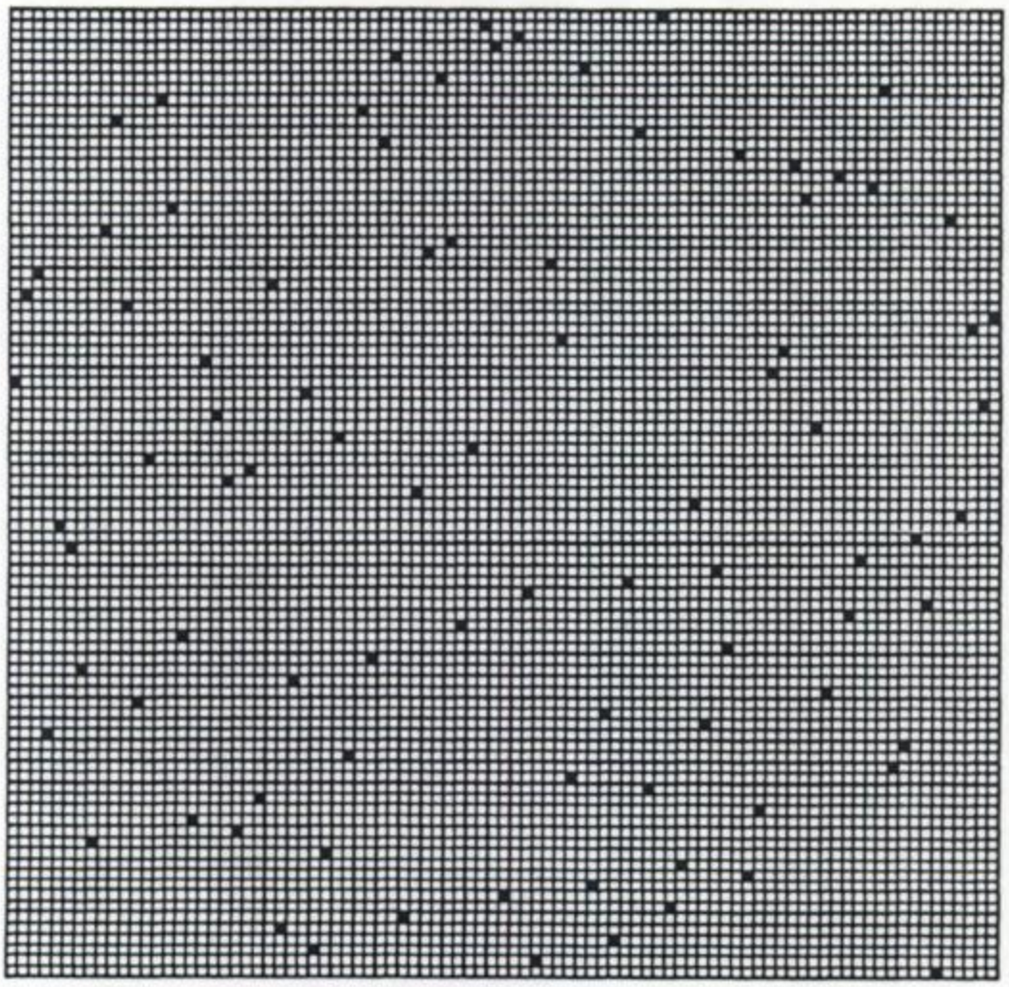


Fig. 1-13 (a) Solutions of N-queen problems



89-queen

Fig. 1-13 (b) A solution of 89-queen problems

Chapter 2

CROSSBAR SWITCH SCHEDULING PROBLEMS

A hysteresis McCulloch-Pitts neuron model is used in order to suppress the complicated oscillatory behavior of neural dynamics. The artificial hysteresis McCulloch-Pitts binary neural network is used for scheduling time-multiplex crossbar switches in order to demonstrate the effects of hysteresis. Time-multiplex crossbar switching systems must control traffic on demand such that packet blocking probability and packet waiting time are minimized. The system using n x n processing elements (neurons) solves an n x n crossbar-control problem within nearly O(1) time, while the best existing parallel algorithm requires O(n) time. The hysteresis McCulloch-Pitts binary neural network maximizes the throughput of packets through a crossbar switch. The solution quality of our system does not degrade with the problem size as far as we have observed the system behavior. The relationship between N-queen problems and crossbar switch scheduling problems is also given. This Chapter is largely based on a paper published in Biological Cybernetics (Takefuji and Lee 1991).

2.1 INTRODUCTION

In time-multiplex communication systems, crossbar packet switches route traffic from the input to output where a message packet is transmitted from the source to the destination. The randomly incoming traffic must be controlled and scheduled to eliminate conflict at the crossbar switch where the conflict is that two or more users

network, the state of the system is forced to converge to the local minimum. In other words, the solution quality drastically degrades with the problem size. Takefuji and Lee have successfully used the Hopfield neural network with McCulloch-Pitts binary neurons for solving the graph planarization problem (Takefuji and Lee 1989) and the tiling problem (Takefuji and Lee 1990a) where the state of the system converges to the near-global minimum in O(1) time. They proved that the state of the binary neural network system is guaranteed to converge to the local minimum (Takefuji and Lee 1990b). In 1986 Hoffman and Benson introduced sigmoid neurons with hysteresis for learning, where any changes in synaptic connection strengths are replaced by hysteresis (Hoffman and Benson 1986). Due to the hysteresis associated with each neuron, the system tends to stay in the region of phase space where it is located. They proposed the theory on a role for sleep in learning (Hoffman and Benson 1986).

Dynamic and static hysteresis in Crayfish stretch receptors was reported by Segundo and Martinez in 1985 (Segundo and Martinez 1985). They stated that hysteresis may be more widespread than suspected in sensory and perhaps other system. In 1989 Keeler, Pichler, and Ross presented the effects of hysteresis in pattern recognition and learning for improving the signal-to-noise ratio (Keeler et al. 1989). In this Chapter the hysteresis property is exploited in order to reduce the complicated oscillatory behaviors of neural dynamics for solving combinatorial optimization problems. The hysteresis with each neuron enhances the state of the system to stay in the region of phase space where it is located. In other words, it suppresses the oscillatory behaviors of neural dynamics so that the convergence time to the global minimum is drastically shortened.

The McCulloch-Pitts neuron model is a binary unit whose value depends on the linear sum of weighted inputs from the other neurons in the network (McCulloch and Pitts 1943). Fig. 1-1 in Chapter 1 shows the input/output relation of the McCulloch-Pitts neuron model. In 1982 Hopfield proposed the continuous input/output unit called the sigmoid neuron model (Hopfield 1982) as shown in Fig. 1-3. Simic presented the molecular electronic device with hysteresis in 1986 where it has the sigmoid hysteresis (Simic 1986).

In this Chapter a McCulloch-Pitts binary neuron model with hysteresis is highlighted. Fig. 1-2 shows the input/output function of the hysteresis McCulloch-

Pitts binary neuron model. A binary neural network with hysteresis is used for scheduling time-multiplex crossbar switches in order to demonstrate the effects of hysteresis. The complicated oscillatory behavior is one of the most undesirable phenomena in neural dynamics for solving optimization problems where we lack the mathematical tools to manipulate and understand them at a computational level (Hopfield and Tank 1986). Hysteresis suppresses the oscillatory behaviors of neural dynamics and consequently it shortens the convergence time to the global minimum. In other words, hysteresis in individual neurons allows the state of the proposed neural network to converge to the global minimum in nearly O(1) time. The system uses an N x N neural network array for solving an n x n crossbar switch problem where the output of the ijth hysteresis neuron V_{ij} is given by: V_{ij}=1 if U_{ij}>UTP (upper trip point), 0 if U_{ij}<LTP (lower trip point), and unchanged otherwise. Note that U_{ij} is the input of the *ij*th neuron. The output at any particular time does not depend only upon the present value of the input but also upon past values.

The proposed hysteresis McCulloch-Pitts binary neural network not only maximizes the throughput of packets through a crossbar switch but also minimizes packet blocking probability and packet waiting time. The constraints on an N x N crossbar switch are that no two inputs may be connected to the same output simultaneously and that no one input may be connected to more than one output simultaneously. In other words, no two packets should share the same row and the column of the traffic matrix.

The system simulator was developed based on the proposed model. A large number of simulation results are shown and demonstrated in order to support the effects of hysteresis.

2.2 CROSSBAR PROBLEMS AND N-QUEEN PROBLEMS

The constraints are considered that no two packets should share the same row and the column of the N x N traffic matrix. Our system uses an N x N neural network array where the motion equation of the *ij*th neuron is given by:

$$\frac{dU_{ij}}{dt} = -A\left(\sum_{k=1}^{N} V_{ik}-1\right)-A\left(\sum_{k=1}^{N} V_{kj}-1\right)+Bh\left(\sum_{k=1}^{N} V_{ik}\right)+Bh\left(\sum_{k=1}^{N} V_{kj}\right) \tag{2.1}$$

where h(x) is called the hill-climbing term, h(x) is 1 if x=0, 0 otherwise. Note that coefficients A and B are constant integers. The first term and the second term are the row constraint and the column constraint respectively. The first term forces one and only one neuron to be fired per row. If no neuron is fired per row or per column then it will perform excitatory forces. If more than one neuron are fired per row or per column then it will act inhibitory forces. The third term and the last term are the row hill-climbing term and the column hill-climbing term respectively. The hill-climbing terms are activated only when local conflicts (where no neuron is fired per row or column) are detected. If there is no conflict the hill-climbing terms will perform no operation. In other words, the local conflicts are resolved by the hill-climbing terms where the *ij*th neuron is encouraged to fire if the neuron has conflicts.

The reader might associate with the N-queen problem from the crossbar switch problem since they are very similar to each other except the diagonal constraints in N-queen problems. Eq. (1.8) with the hill-climbing term in Eq. (1.9) is exactly equivalent to Eq. (2.1) after removing B-term (diagonal terms) from Eq. (1.8). In other words, the crossbar switch scheduling problem is considered as N-rook problem where a rook commands horizontally and vertically. The positions of the rooks are equivalent to the state of the traffic matrix. The goal of the N x N crossbar switch scheduling problem is to place N mutually nonattacking rooks on an N x N chessboard. The maximum number of rooks is given by N^2.

The simulator has been developed on a Macintosh SE/30 and a DEC3100 workstation. Remember that the simulator can be easily implemented by Program1, Program2, or Program3 as shown in Chapter 1. The demonstrated simulator here is completely based on Program2 in Fig. 1-7 except the input/output function where the hysteresis McCulloch-Pitts neuron model is used. The simulator is to simulate the synchronous parallel neural network model. The motion equation in Eq. (2.1) is inserted in Program2. Fig. 2-2 shows the relationship between the average number of iteration steps, the problem size, and the band size of hysteresis. The hysteresis band size is given by the hysteresis band size=|UTP|=|LTP|. When no hysteresis is given to each neuron, it usually takes more than 5000 iteration steps or does not converge to the global minimum at all.

2.3 REFERENCES

Chen W., Mavor J., Denyer P. B., Renshaw D. (1990) Traffic routing algorithm for serial superchip system customisation. IEE Proc., 137, Pt. E, 1

Hoffman G. W., Benson M. W. (1986) Neurons with hysteresis form a network that can learn without any changes in synaptic connection strengths. Proc. of AIP Conf. on Neural Networks for Computing (Ed. J. S. Denker, AIP 1986)

Hopfield J. J. (1982) Neural networks and physical systems with emergent collective computational abilities. Proc. Natl. Acad. Sci. USA, 79, 2554-2558

Hopfield J. J., Tank D. (1986) Computing with neural circuits: A model. Science, 233, 625-633

Inukai T. (1979) An efficient SS/TDMA time slot assignment algorithm. IEEE Trans. on Commun., 27, 1449-1455

Keeler J. D., Pichler E. E., Ross J. (1989) Noise in neural networks: thresholds, hysteresis, and neuromodulation of signal-to-noise. Proc. Natl. Acad. Sci. USA, 86, 1712-1716

Marrakchi A., Troudet T. (1989) A Neural Net Arbitrator for Large Crossbar Packet-Switches. IEEE Trans. on Circuits and Systems, 36, 7

McCulloch W. S., Pitts W. (1943) A logical calculus of the ideas imminent in nervous activity. Bull. Math. Biophys., 5, 115-133

Rose C. (1989) Rapid Optimal Scheduling for Time-Multiplex Switches Using a Cellular Automaton. IEEE Trans. on Commun., 37, 500-509

Segundo J. P., Martinez O. D. (1985) Dynamic and Static hysteresis in Crayfish stretch receptors. Biological Cybernetics, 52, 291-296

Simic B. G. (1986) SPIE, 634, 195

Takefuji Y., Lee K. C. (1989) A near-optimum parallel planarization algorithm. Science, 245, 1221-1223

Takefuji Y., Lee K. C. (1990a) A parallel algorithm for tiling problems. IEEE Trans. on Neural Networks, 1, 1, 143-145

Takefuji Y., Lee K. C. (1990b) A super parallel sorting algorithm based on neural networks. IEEE Trans. on Circuits and Systems, 37, 11

Takefuji Y., Lee K. C. (1991) An hysteresis binary neuron: a model suppressing the

oscillatory behaviors of neural dynamics. Biological Cybernetics, 64, 353-356

2.4 EXERCISES

1. Generate several traffic matrices with different densities and observe the behavior of the system by changing the hysteresis band size with or without the hill-climbing term.
2. Compare the solution quality and the computation time of the proposed algorithm with that of the best known algorithm.
3. Build a simulator to simulate the synchronous parallel N x N crossbar model as if P x Q processing elements are used on a parallel machine for P, Q<N.
4. Discuss why hysteresis in the neuron model is able to suppress the oscillatory behavior in neural dynamics.
5. Survey other crossbar switch problems and solve one of their problems.

Chapter 3

FOUR-COLORING AND K-COLORABILITY PROBLEMS

The computational energy is presented for solving a four-coloring map problem. The map-coloring problem is defined that one wants to color the regions of a map in such a way that no two adjacent regions (that is, regions sharing some common boundary) are of the same color. This Chapter presents a neural network parallel algorithm based on the McCulloch-Pitts binary neuron model. A 4 x n neural array is used to color a map of n regions where each neuron as a processing element performs the proposed motion equation. The capability of the proposed system is demonstrated through a large number of simulation runs. The parallel algorithm is modified for solving the k-colorability problem. This Chapter is based on our paper published in IEEE Transactions on Circuits and Systems (Takefuji and Lee 1991)

3.1 INTRODUCTION

A map maker colors adjacent countries with different colors so that they may be easily distinguished. This is not a problem as long as one has a large number of colors. However it is more difficult with a constraint that one must use the minimum number of colors required for a given map. It is still easy to color a map with a small number of regions. In the early 1850's Francis Guthrie was interested in

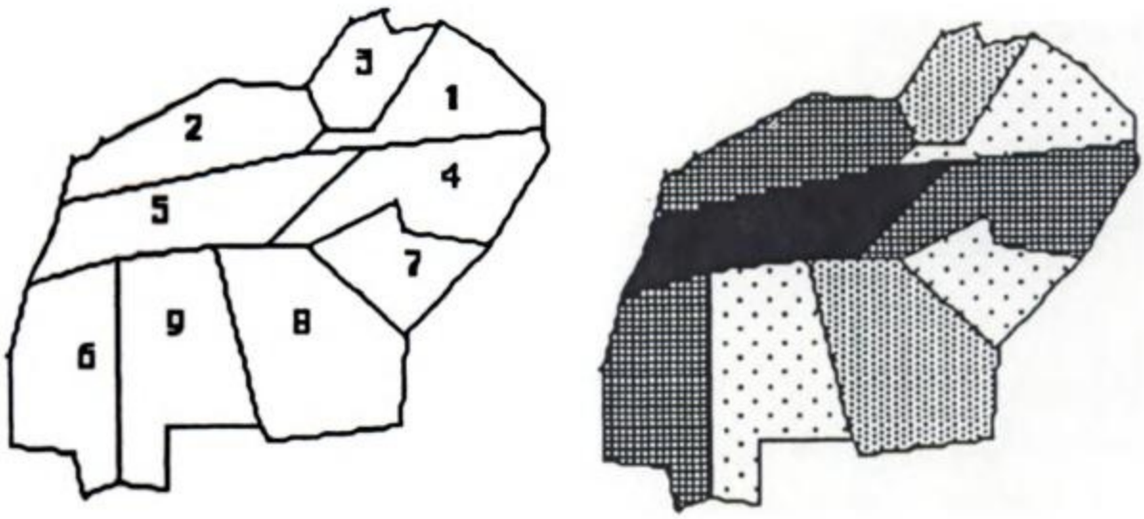


Fig. 3-1 Nine-region map and 4-coloring

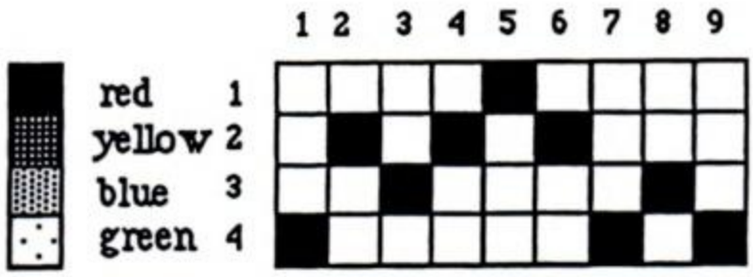


Fig. 3-2 Neural representation for 4-coloring map in Fig. 3-1

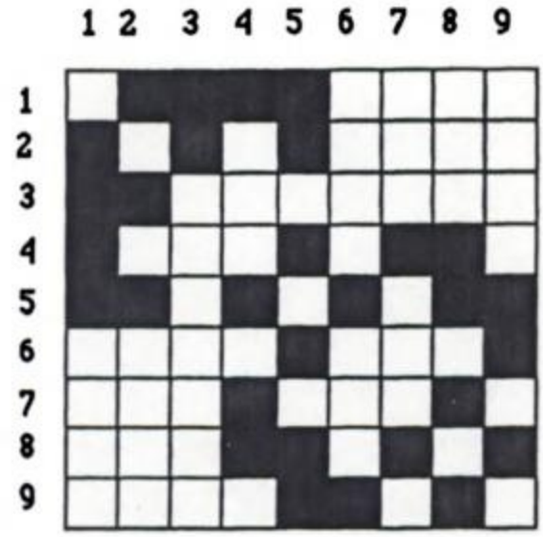


Fig. 3-3 Adjacency matrix of the map in Fig. 3-1

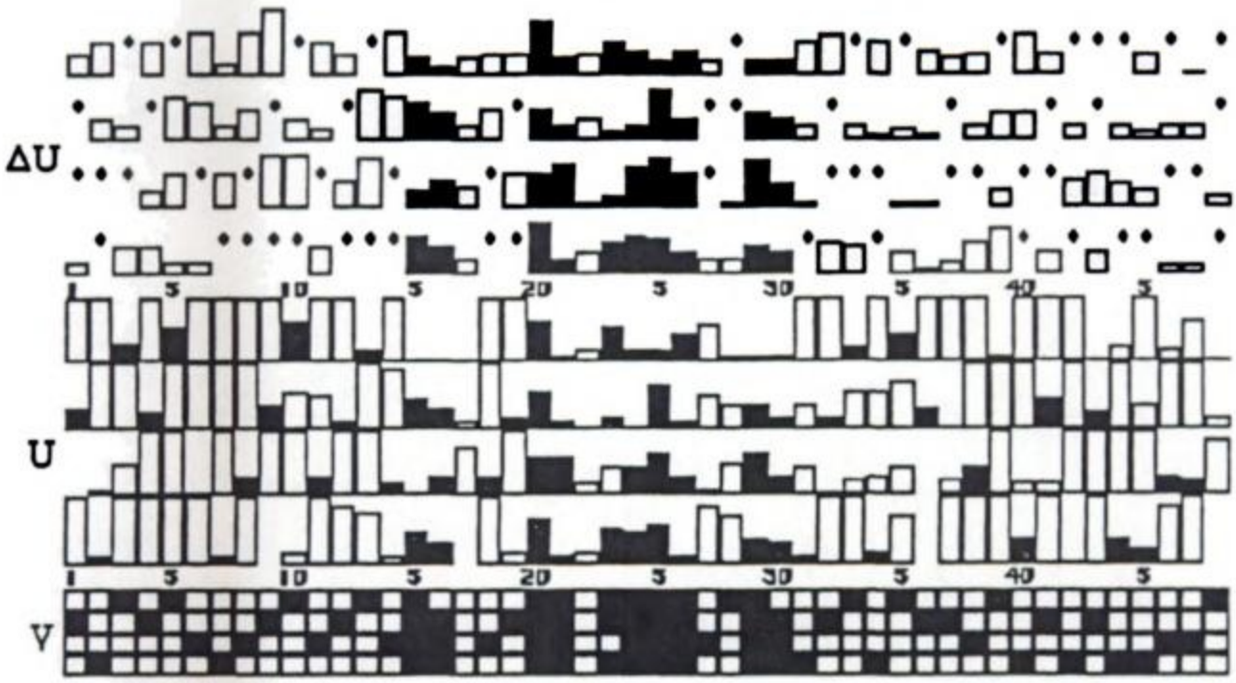


Fig. 3-5 (b) The convergence of the 48-state US map neural network to a four-coloring solution. This shows the intermediate state of 192 neurons after the tenth iteration.

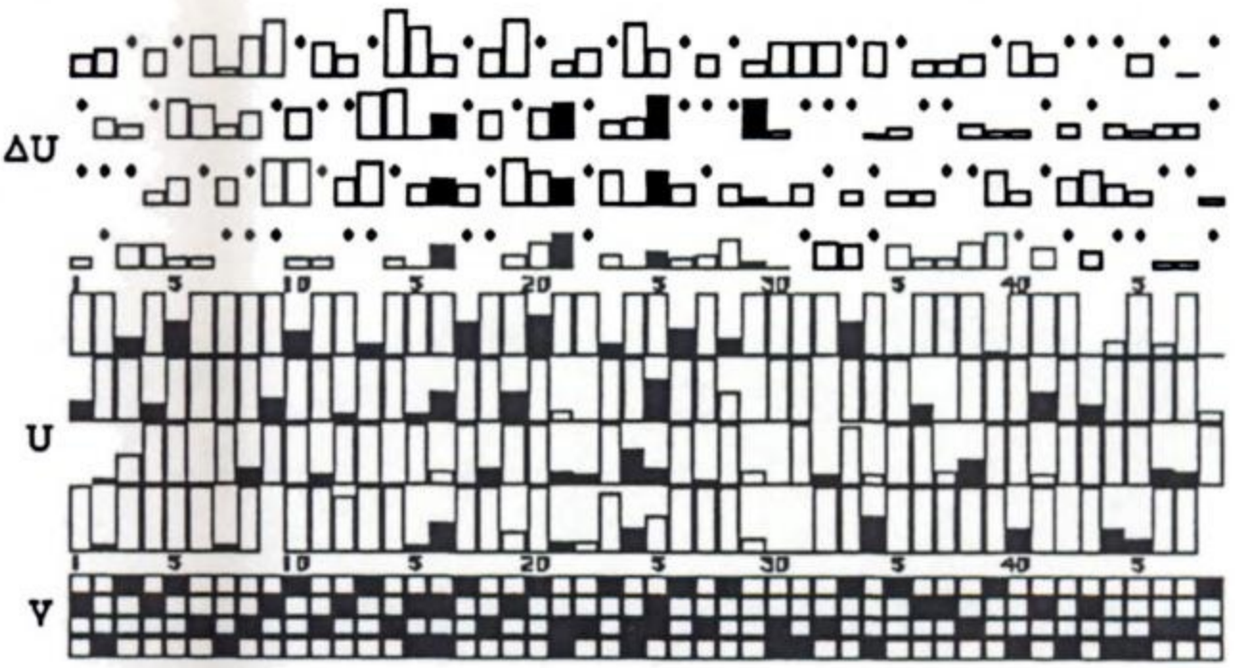


Fig. 3-5 (c) The convergence of the 48-state US map neural network to a four-coloring solution. This shows the intermediate state of 192 neurons after the twentieth iteration.

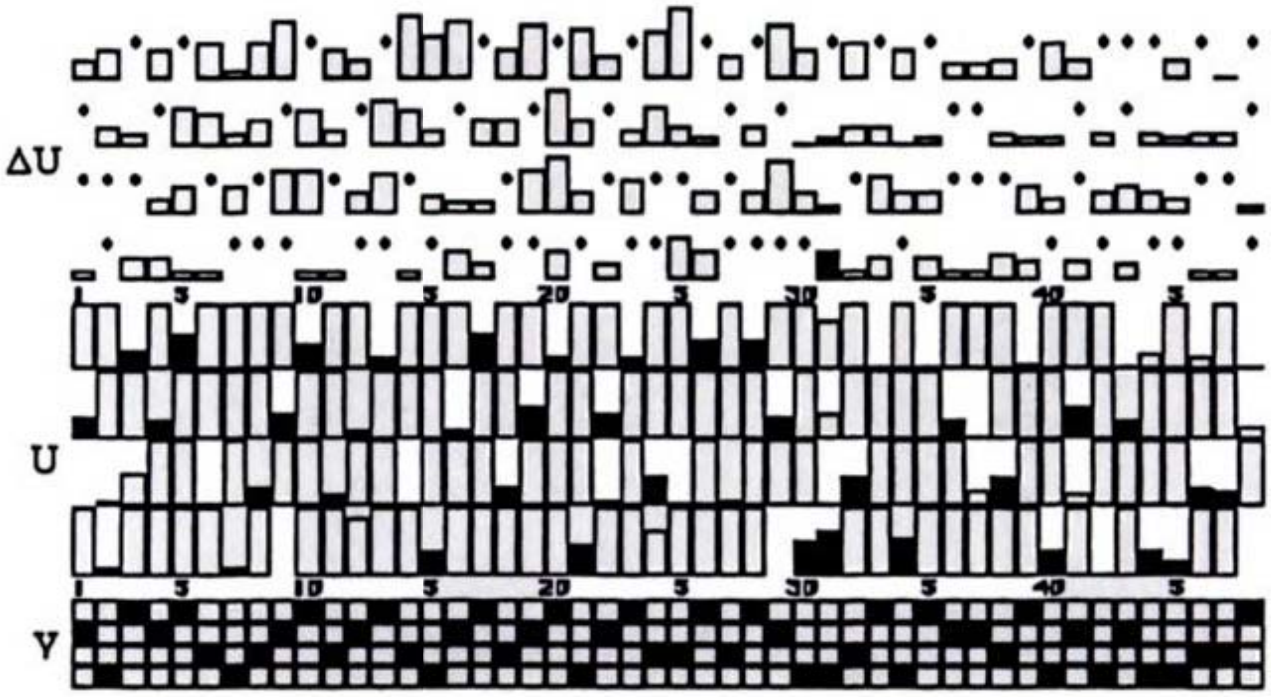


Fig. 3-5 (d) The convergence of the 48-state US map neural network to a four-coloring solution. This shows the final state of 192 neurons after the thirty fifth iteration.

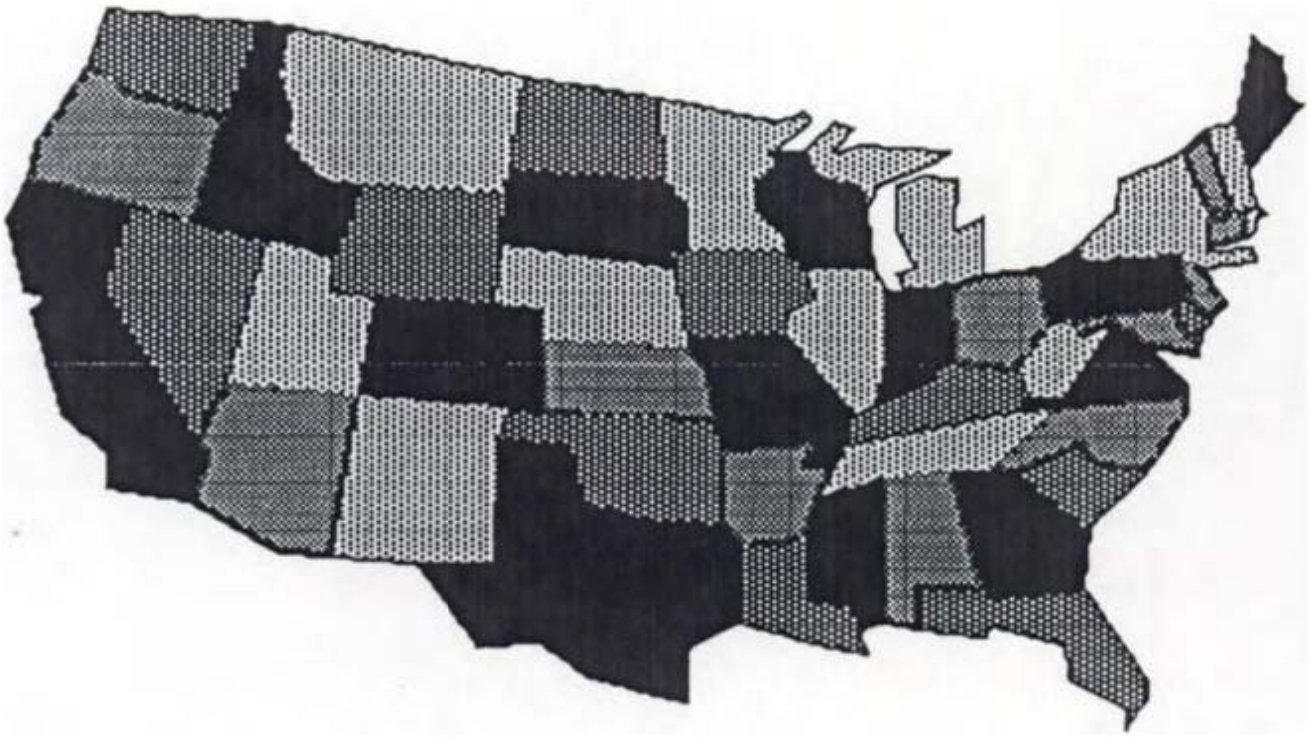


Fig. 3-6 One of the solutions

Fig. 3-6 depicts the solution which is decoded from the final state of 192 neurons in Fig. 3-5d. Fig. 3-7 describes one of the solutions for a 210-country map four-coloring problem where the problem is taken from the example of Appel and Haken's experiments. The state of the system always converged to the global minimum as far as we have observed through more than one thousand simulation runs. Fig. 3-8 shows the relationship between the frequency and the number of iteration steps using several hundred simulation runs in the 210-country map problem. The average number of iteration steps is 820.

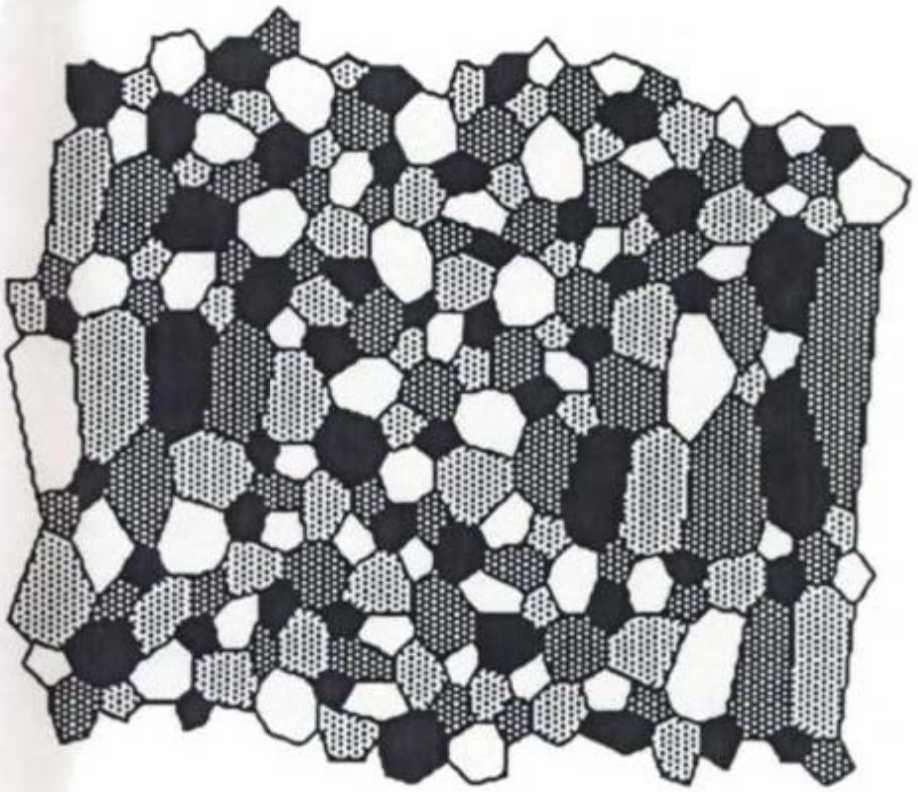


Fig. 3-7 One of the solutions for the 210-country map problem

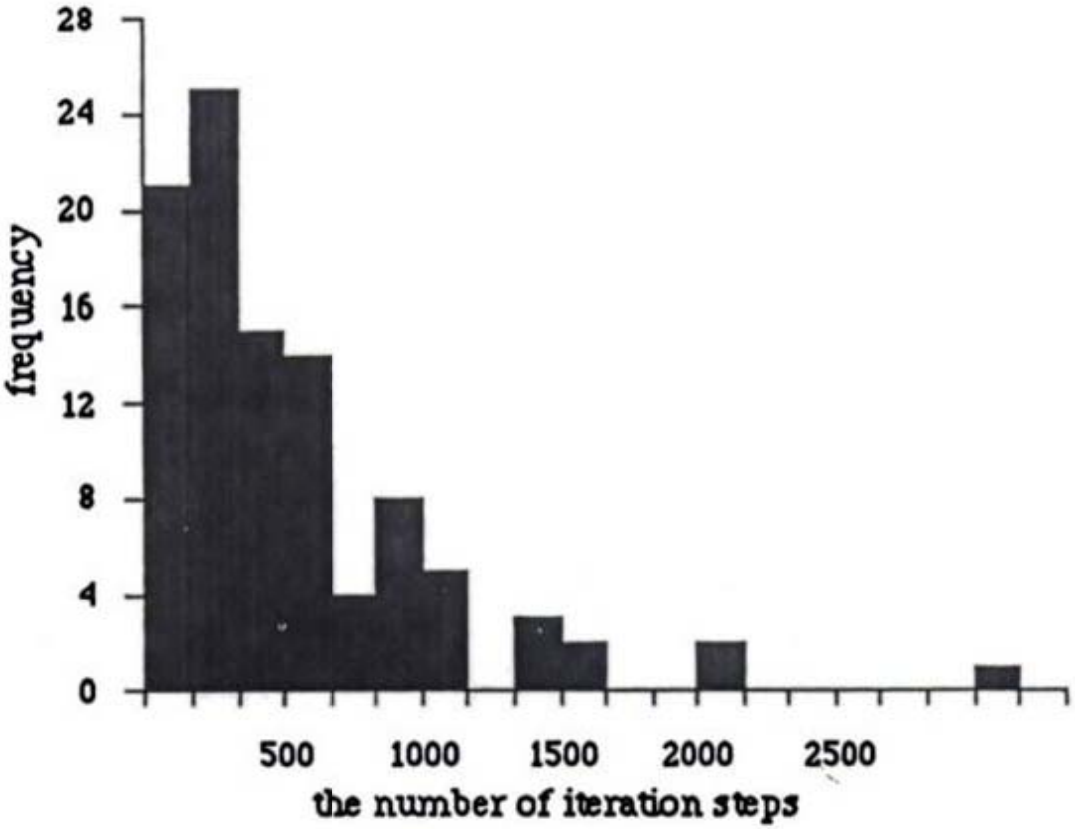


Fig. 3-8 Relationship between the frequency and the number of iteration steps in the 210-country map coloring problem

Fig. 3-9 shows one of the solutions for a 430-country map problem. Fig. 3-10 depicts the frequency versus the number of iteration steps using several hundred simulation runs for the 430-country map problem. The average number of iteration steps is 1000. Fig. 3-8 and Fig. 3-10 indicate that the problem size does not strongly influence the number of iteration steps to converge to the global minimum.

Through more than one thousand sets of simulation runs we have observed that the problem size does not strongly influence the number of iteration steps to converge to the global minimum. The proposed scaling method for the coefficients in the motion equation is quite effective which allows the state of the system to escape from the local minimum and to converge to the global minimum.

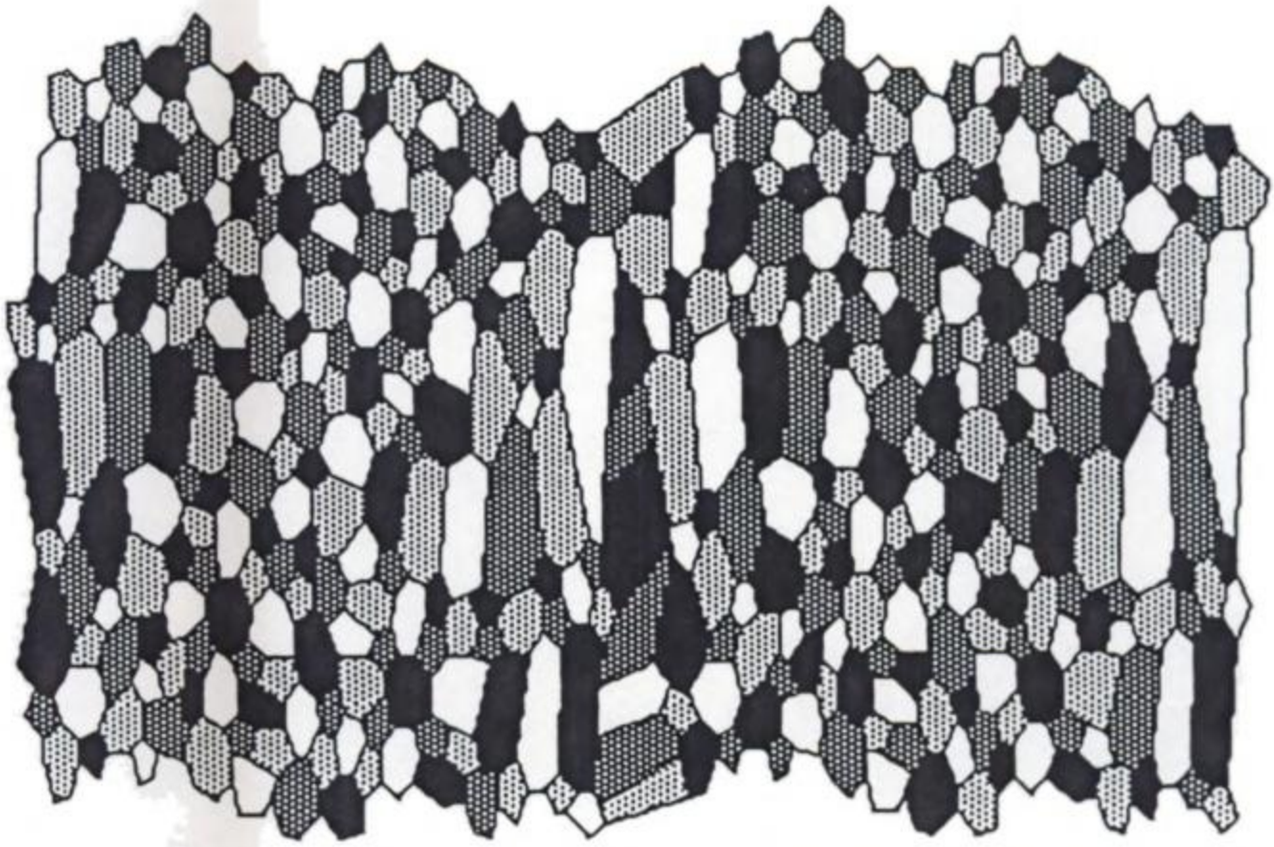


Fig. 3-9 One of the solutions for the 430-country map problem

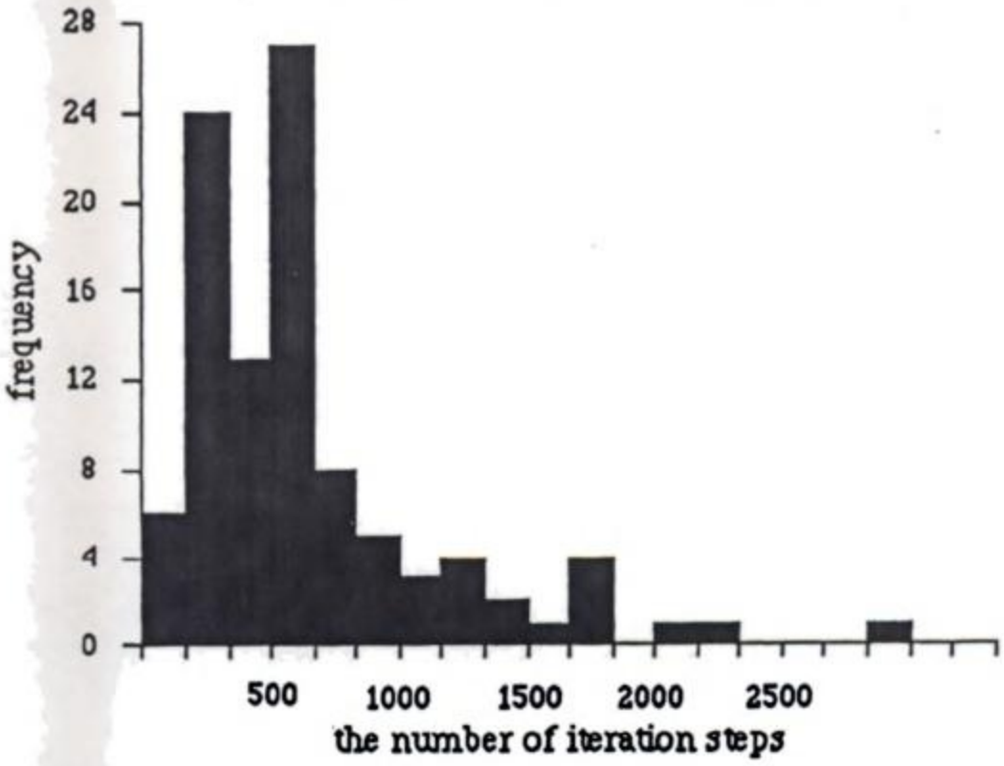


Fig. 3-10 Relationship between the frequency and the number of iteration steps in the 430-country map coloring problem

Chapter 4

GRAPH PLANARIZATION PROBLEMS

In this Chapter a near-optimum parallel planarization algorithm is introduced. The proposed system composed of an N x N neural network array not only generates a near maximal planar subgraph from a nonplanar graph or a planar graph but also embeds the subgraph on a single plane within O(1) time. The algorithm can be used in designing printed circuit boards and routing very large scale integrated (VLSI) circuits. This Chapter is based on a paper published in Science (Takefuji and Lee 1989).

4.1 INTRODUCTION

Maximal planarization of a planar or nonplanar graph is an important problem in designing printed circuit boards and routing VLSI circuits. A graph is planar if it can be drawn on a single plane with no two edges crossing each other except at their end vertices. If a given circuit is planar, it can be wired on a single layer. If a given circuit is nonplanar, we would like to maximize the number of edges to be planarized and minimize the number of edges to be removed from a nonplanar graph. To yield a maximal planar subgraph from a nonplanar graph is an NP-complete problem (Garey and Johnson 1979).

Two tasks must be accomplished to solving the graph planarization problem.

Vup_{12}	Vup_{13}	Vup_{14}	Vup_{23}	Vup_{24}	Vup_{34}
$Vdown_{12}$	$Vdown_{13}$	$Vdown_{14}$	$Vdown_{23}$	$Vdown_{24}$	$Vdown_{34}$

1	1		1		
		1		1	1

Fig. 4-2 Single-row representation of the graph in Fig. 4-1d

There are two kinds of forces performing in the artificial neural network; excitatory and inhibitory forces. In the graph planarization problem the excitatory force means that if an edge (i,j) exists in a given graph then the *ij*th neuron to represent embedding the connection line in a plane is encouraged to fire. The inhibitory force means that the neurons which violate the two-edge-crossing condition are discouraged to fire. The two-edge-crossing violation condition is expressed by the following:

If Vup_{lm}=1 and (l<i<m<j or i<l<j<m) then the *ij*th up-neuron should not be fired. In other words, Vup_{ij} should not be 1. If $Vdown_{lm}$=1 and (l<i<m<j or i<l<j<m) then $Vdown_{ij}$ should not be 1. Fig. 4-3 describes the edge-violation conditions for Vup_{ij} and $Vdown_{ij}$. The necessary and sufficient conditions of our violation conditions are given by the concise functions:

$\sum_{l} \sum_{\substack{m \\ l<m}} f(l,i,m)f(i,m,j)Vup_{lm}$ for Vup_{ij} neuron is shown in Fig. 4-3a where the function f(L,M,R) is 1 if L<M<R, 0 otherwise.

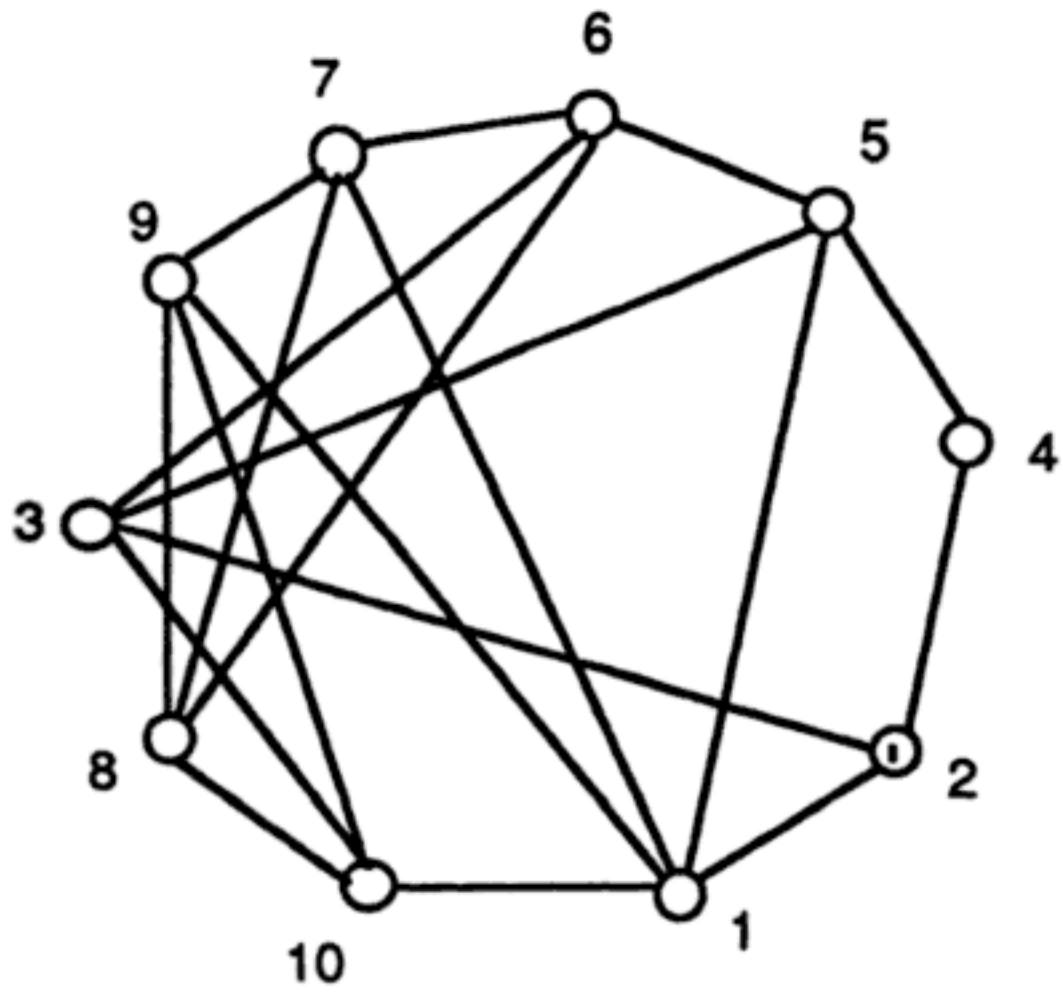


Fig. 4-5 Maximal planar subgraph by Jayakumar

We have developed the simulator based on Program2 as discussed in Chapter 1.4 in Fig. 1-8. First we solved the same nonplanar graph problem as shown in Fig. 4-4. When the coefficients A=2 and B=1, the unit time $\Delta t=10^{-5}$ were used for Eq. (4-1) and (4-2), and the initial values of $Uup_{ij}(t)$ and $Udown_{ij}(t)$ where i=1,...,N and j=1,...,N were randomized in the range of -1/10000 to 0, the state of our system converged to the global minimum in the 14th iteration. Fig. 4-6a and 4-6b describe the state of the system at the first and fourteenth iteration. Squares in the upper and the lower triangle indicate V_{upij} and V_{downij} respectively. The linear dimension of each rectangle is proportional to the value of ΔU_{upij}, ΔU_{downij}, U_{upij}, U_{downij}, V_{upij}, and V_{downij}. Black and white rectangles indicate positive and negative values respectively. Fig. 4-6a shows the intermediate state of 44 neurons after the first iteration where 7 edges are embedded but some edges intersect each other. Fig. 4-6b shows the final state of 44 neurons after the fourteenth iteration where 20 edges are embedded and no edges intersect each other. Our simulator discovered that the new maximal planar subgraph contains 20 edges instead of 19 edges, which contradicts to the result of Jayakumar.

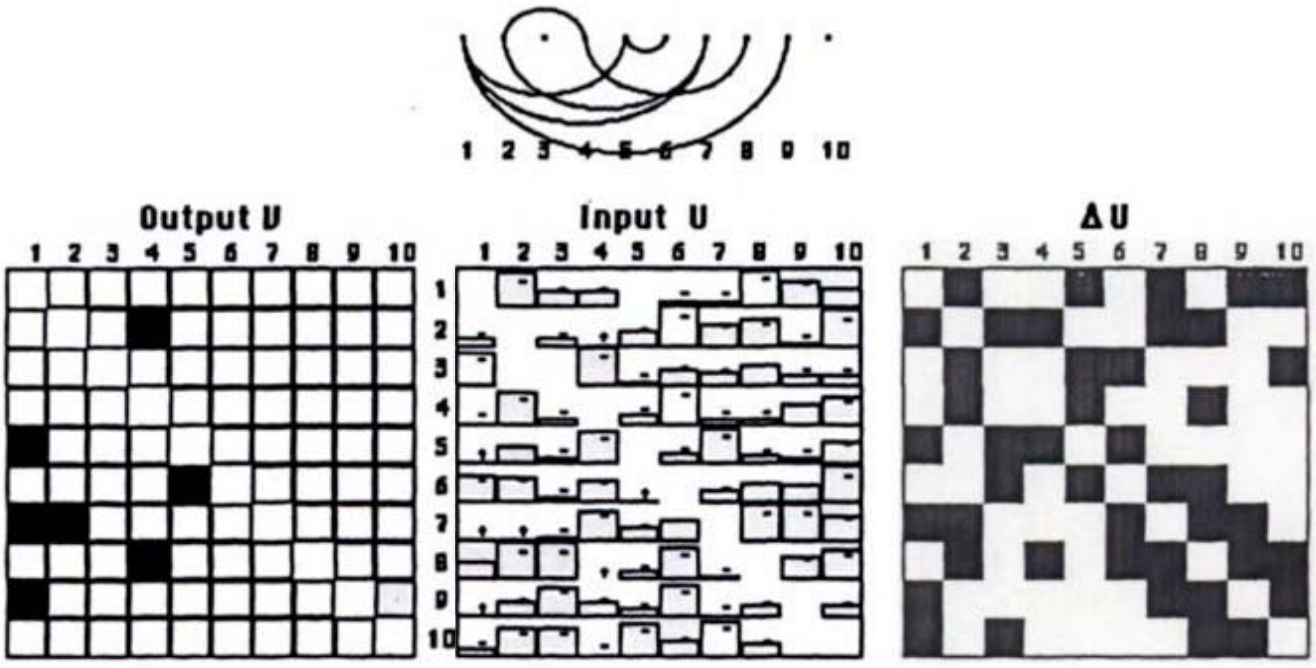


Fig. 4-6 (a) The convergence of the graph planarization neural network to a solution

Fig. 4-7 shows the simulation result where several sets of the coefficients were experimented. It indicates that either 20 edges or 19 edges out of 22 edges can be consistently embedded in a single plane.

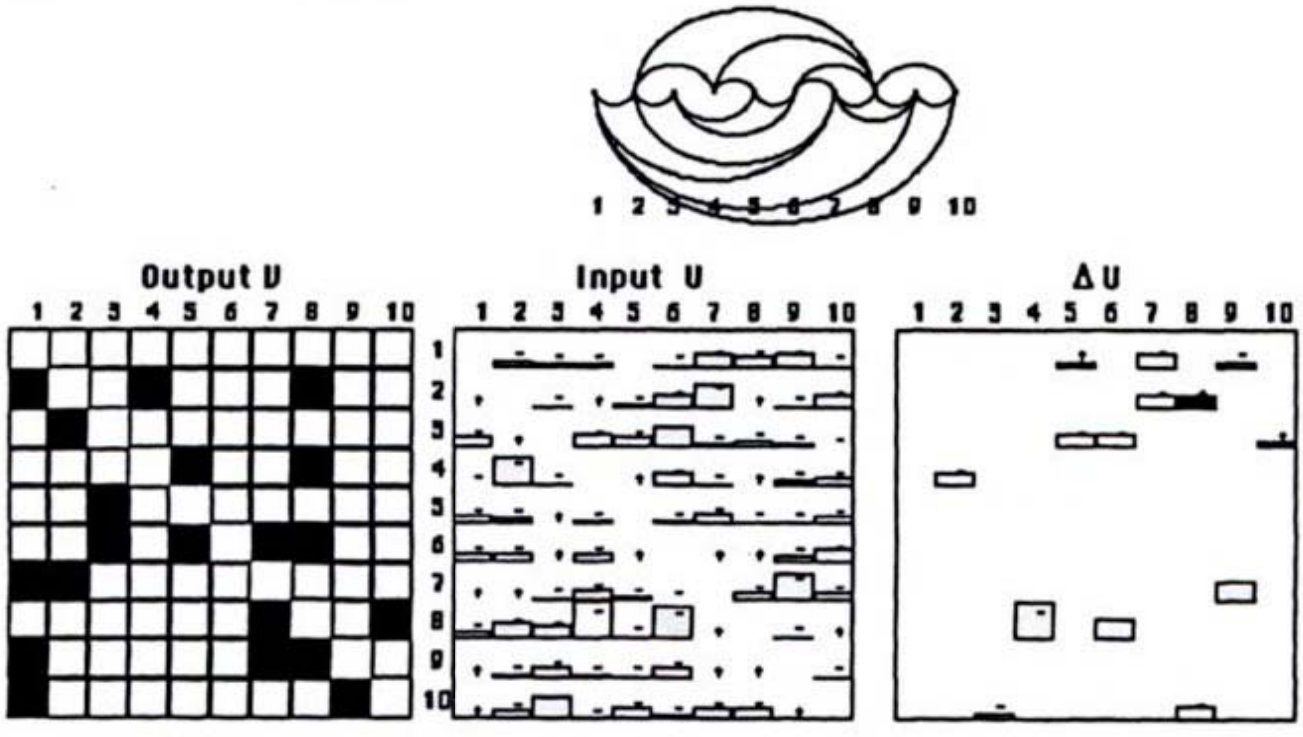


Fig. 4-6 (b) The convergence of the graph planarization neural network to a solution

no action.

When A=B=1, C=5, Dt=10^{-6}, and the initial values of $Uup_{ij}(t)$ and $Udown_{ij}(t)$ were randomized in a range of 0 to -1/2000, the state of the system converged to the global minimum in the 206th iteration. The solution is depicted in Fig. 4-8c. Several experiments using different sets of the coefficients showed the consistent improvement in the solutions. We also changed the numbering of state regions, our system produced the planar subgraph with more than 100 edges in 20 iteration steps.

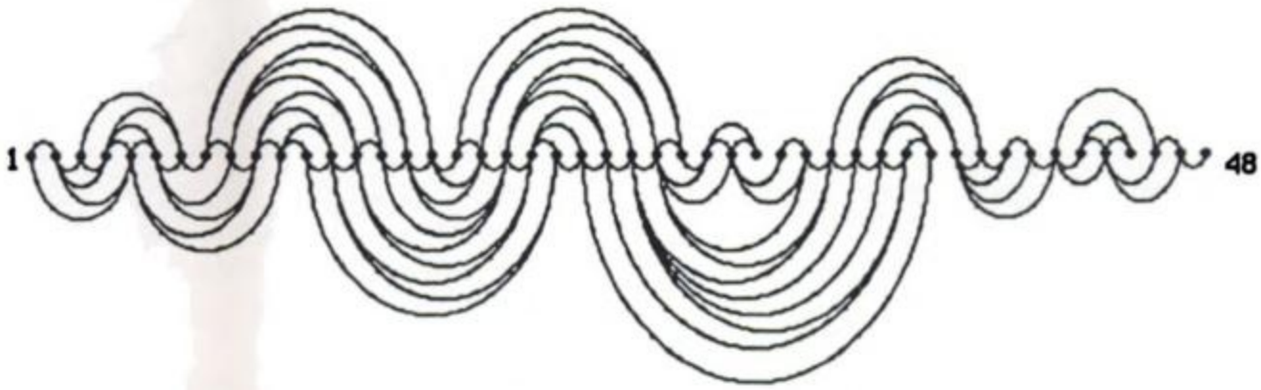


Fig. 4-8 (c) Solution of the graph in Fig. 4-8b

Based on the proposed parallel planarization algorithm we have developed the simulator. The developed simulator is currently running on a Macintosh and on an HP Apollo 3500 computer. Within O(1) time the algorithm not only generates a near-optimum planar subgraph from the nonplanar or planar graph but also embeds the subgraph on a plane. The algorithm can be implemented by an N x N 2-dimensional neural network array where N is the number of vertices in a given graph. Among N^2 neurons only 2M neurons are used to obtain the solution where M is the number of edges in the graph. Remember that the number of required neurons for M-edge graph planarization is actually less than 2M where it is actually determined by the number of intersected edges. As far as we have observed the behavior of the simulator the state of the system always converges to the good solution within 20 or 30 iteration steps.

5.1 INTRODUCTION

The channel routing problem in a multi-layer channel is very important in automatic layout design of VLSI circuits and printed circuit boards. A channel consists of two parallel, horizontal rows of points which are called terminals. The terminals are placed at regular intervals and identify the columns of the channel. A net consists of a set of terminals that must be interconnected through some routing paths. Some nets may have a connection point at one or both ends (top and/or bottom) of the channel. The channel routing problem is not only to route the given nets or interconnections between the terminals on the multi-layer channel, but also to minimize the channel area.

The sequential algorithms for the two-layer channel routing problems have been extensively studied by many researchers (Hashimoto A. and Stevens J. 1971) (Kernighan B. W., Schweikert D. G., and Persky G. 1973) (Deutsch D. N. 1976) (Sahni S. and Bhatt A. 1980) (LaPaugh A. S. 1980) (Dolev D. et al. 1981) (Yoshimura T., and Kuh E. S. 1982) (Rivest R. L. and Fiduccia C. M. 1982) (Leong H. W. and Liu C. L. 1983) (Burstein M. and Pelavin R. 1983a) (Burstein M. and Pelavin R. 1983b) (Szymanski T. G. 1985) (Reed J., Sangiovanni-Vincentelli A., and Santomauro M. 1985) (Leong H. W., Wong D. F., and Liu C. L. 1985) (Joobbani R. and Siewiorek D. P. 1985). The existing algorithms have the following common features. The routing paths consist of the horizontal segments which are parallel to the terminals of the channel and the vertical segments. All the horizontal segments of the routing paths are assigned on one layer and all the vertical segments of them are assigned on another layer. The connections between the horizontal segments and the vertical segments are made through the contact windows which are called via holes. For the integrated circuits, typically the horizontal segments are embedded on a metal layer while the vertical segments are embedded on a polysilicon and/or diffusion layer. Any two routing paths on the same layer cannot be placed within some distance of each other which is called separation condition. For

convenience, a unit grid is superimposed on the channel where the size of one unit satisfies the separation condition and all the terminals are located at the grid points. All the routing paths on the channel must follow the grid lines. The horizontal segments are called tracks and the vertical segments are called columns. In this model, the separation condition is that any two nets must be embedded neither on the same track nor on the same column if they overlap there, which is called overlapping condition. When the width of the channel is fixed, minimizing the channel area is equivalent to minimizing the number of required tracks where all the given nets must be embedded.

In some sequential algorithms, doglegging is introduced where a routing path of a net is split into more than two horizontal segments on different tracks (Deutsch D. N. 1976) (Yoshimura T. and Kuh E. S. 1982) (Rivest R. L. and Fiduccia C. M. 1982) (Leong H. W. and Liu C. L. 1983) (Burstein M. and Pclavin R. 1983a) (Burstein M. and Pelavin R. 1983b) (Szymanski T. G. 1985) (Reed J., Sangiovanni-Vincentelli A., and Santomauro M. 1985) (Leong H. W., Wong D. F., and Liu C. L. 1985) (Joobbani R. and Siewiorek D. P. 1985). Doglegging is sometimes effective to reduce the number of tracks of the channel and to solve the cyclic conflict. The cyclic conflict occurs when one net interconnects a top terminal on the i-th column with a bottom terminal on the j-th column while another net interconnects a bottom terminal on the i-th column with a top terminal on the j-th column. Note that a top terminal means a terminal at the top end of a channel and a bottom terminal means a terminal at the bottom end of a channel. However, this cyclic conflict occurs infrequently and it can often be avoided by rearranging the terminal placement. Doglegging requires additional via holes which reduce the reliability of the VLSI system and increase the manufacturing cost. It is desirable either to reduce the number of dogleggs or to eliminate dogleggs completely for the practical use.

To further reduce the channel area, several sequential algorithms for two-layer-and-over-the-cell channel routing problems (Deutsch D. N. and P. Glick 1980) (Krohn H. E. 1983) (Shiraishi Y. and J. Sakemi J. 1987) (Gudmundsson G. and Ntafos S. 1987) (Cong J. and Liu C. L. 1988) (Cong J. and Liu C. L. 1990) and for three-layer channel routing problems have been also proposed (Chen Y. K. and M. L.

#1 net (T2,T5) is assigned on the second track of the first two-layer channel and the #2 net (B1,B6) is assigned on the first track of the second two-layer channel, and so on. Note that the two-layer channel means a pair of two layers for the horizontal segments and vertical segments. Fig. 5-1c shows the routing solution corresponding to Fig. 5-1b.

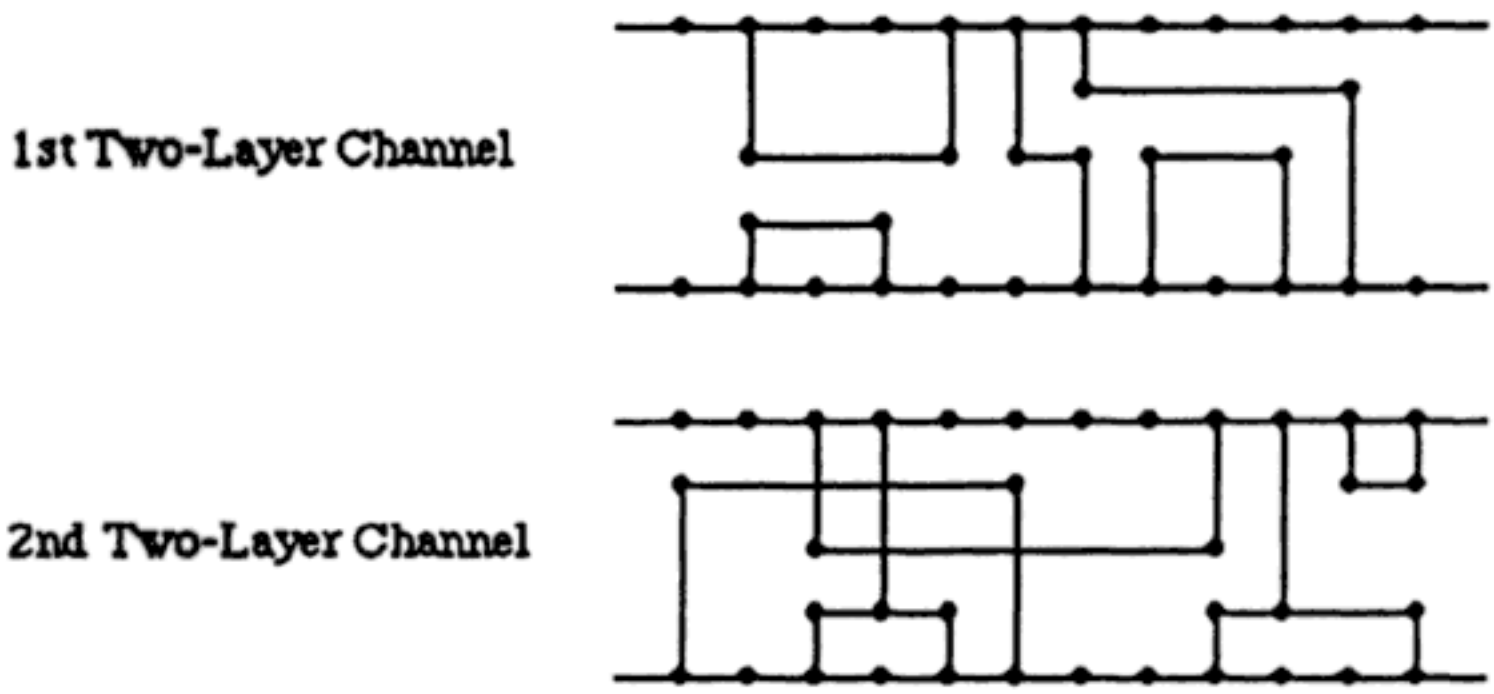


Fig. 5-1 (c) The routing solution corresponding to Fig. 5-1b

Each net must satisfy the separation conditions, in other words, any two different nets must not violate the overlapping conditions. Fig. 5-2 shows the overlapping conditions for the horizontal segments of the nets where $head_i$ indicates the left-most column number of the i-th net and $tail_i$ indicates the right-most column number of the i-th net. The horizontal overlapping conditions for the i-th-net-j-th-track-k-th-layer processing element are given by:

$$\sum_{\substack{p=1 \\ p\neq i \\ head_i \le head_p \le tail_i}}^{n} V_{pjk} + \sum_{\substack{p=1 \\ p\neq i \\ head_p \le head_i \le tail_p}}^{n} V_{pjk} \qquad (5.2)$$

This horizontal condition is nonzero if the horizontal segments of the other nets overlap the horizontal segment of the i-th net on the j-th track of the k-th layer.

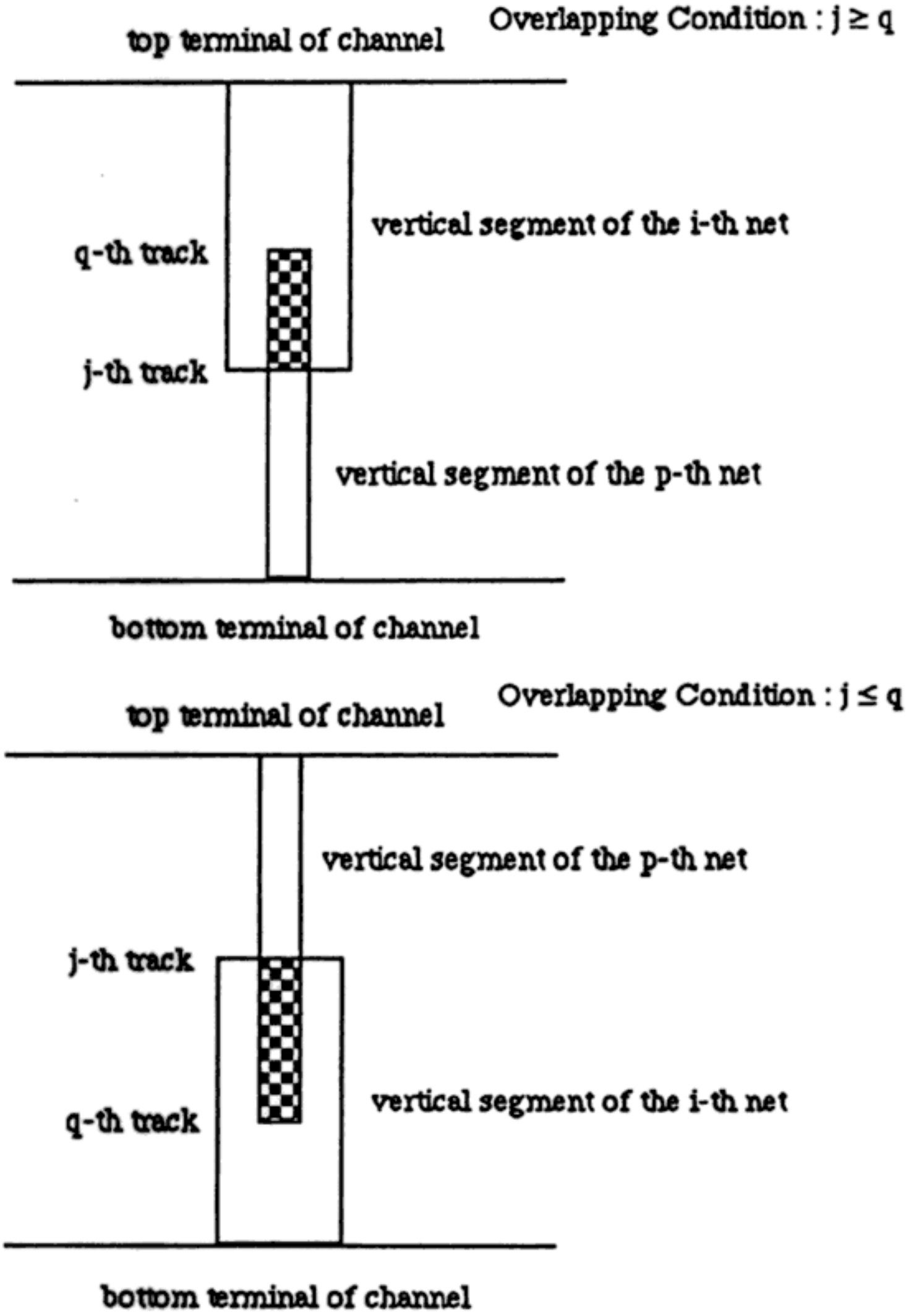


Fig 5-3 Overlapping conditions for vertical segments

The termination condition is given by: If $V_{ijk}(t)=1$ and $\Delta U_{ijk}(t)=0$ for i=1,...,n,

Table 5-1 Comparisons of track numbers with other no-doglegging routers

problem #	number of nets	our six-layer solutions	two-layer solutions by Yoshimura & Kuh	three-layer solutions by Chen & Liu
the example 1	21	3	12	7
the example 3a	45	4	15	8
the example 3b	47	5	17	10
the example 3c	54	5	18	9
the example 4b	55	5	17	13
the example 5	61	5	20	10
Deutsch's difficult example	72	7	28	23

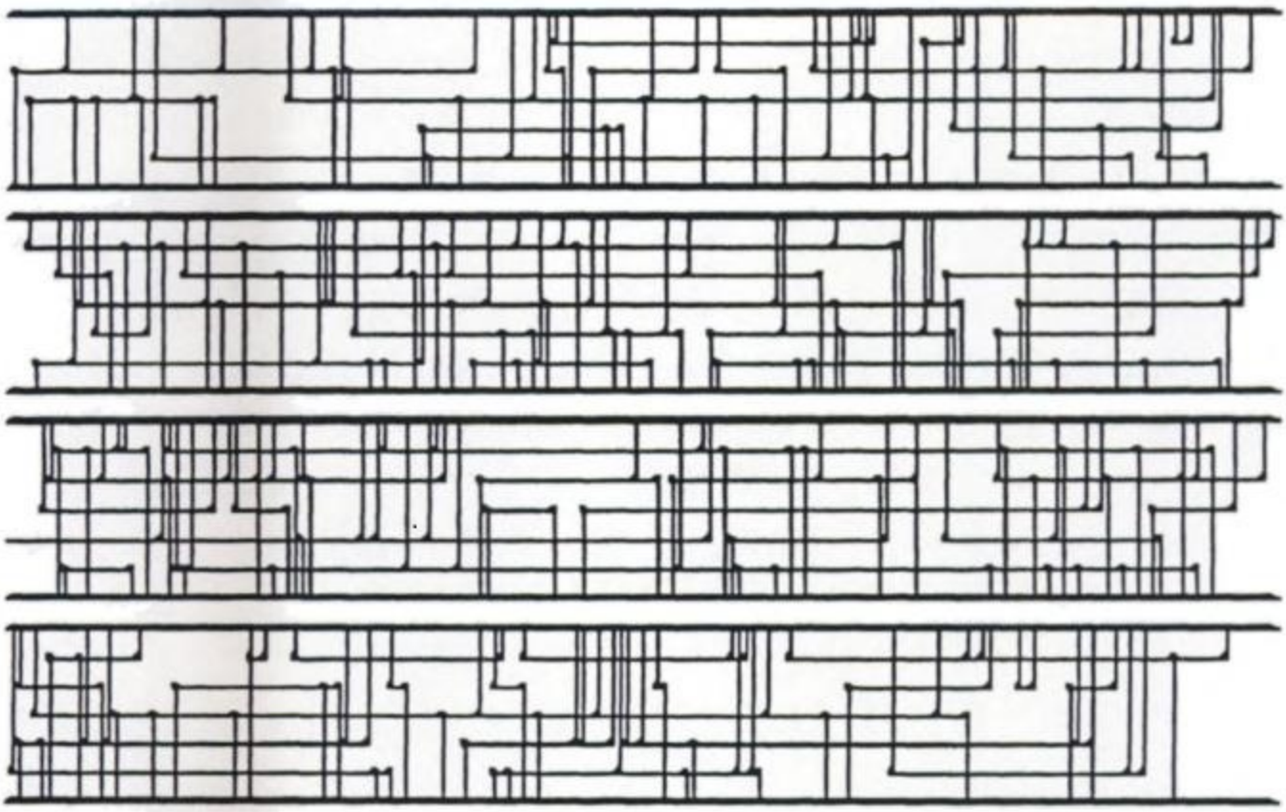


Fig. 5-8 Eight-layer solution of Deutsch's difficult example

Table 5-3 Track numbers in 4-10 layer solutions of Deutsch's difficult example

Number of Layers	Number of Tracks
4	11
6	7
8	5
10	4

This Chapter proposes the parallel algorithm for the four-layer channel routing

Chapter 6

RNA SECONDARY STRUCTURE PREDICTION

In this Chapter two neural network models are used for solving RNA secondary structure prediction problems. One is for solving the maximum independent set problem and is slightly modified for predicting the RNA secondary structure. The other is modified from the graph planarization algorithm discussed in Chapter 4 and it takes advantage of molecular thermodynamics using a part of Tinoco's model. Both simulators were developed on a Macintosh machine, an HP Apollo 3500 workstation, and DEC 3100 workstations. A large number of simulation runs have been performed to test the proposed parallel algorithms and to observe the behavior of the proposed system. The simulation results demonstrated that the proposed parallel algorithms are promising and worthwhile for practical use. This Chapter is based on a paper published in IEEE Trans. on Neural Networks (Takefuji et al 1990b), a paper published in Biological Cybernetics (Takefuji, Lin, and Lee 1990a), and a paper published in Journal of Intelligent Manufacturing (Takefuji 1991).

6.1 INTRODUCTION

This Chapter first introduces a parallel algorithm for finding a near-maximum independent set of a circle graph within several hundred iteration steps. The proposed algorithm is modified for predicting the secondary structure of ribonucleic acids (RNA) where the circle graph is very suited for computing the secondary structure. Non-intersected edges in the circle graph provides information on the base pairs for

set of a circle graph but also predicts the secondary structure of ribonucleic acids. It requires n processing elements where n is the number of edges in the circle graph or the number of possible base pairs. Our simulator based on the proposed algorithm discovered the new structures in a sequence of 38 bases, a sequence of 55 bases, and a sequence of 359 bases from the potato spindle tuber viroid (PSTV), two of which are more stable than the formerly proposed structures. The simulator was tested by solving other problems.

We believe this is the first parallel/distributed processing attempt to solving RNA secondary prediction problems. This Chapter presents the clear comparison between the conventional RNA folding algorithms, the backpropagation algorithm by Quan and Sejnowski (Quan and Sejnowski 1989) or by Holley and Karplus (Holley and Karplus 1990), and our algorithm. Although the proposed algorithm is parallel computing, the simulator is currently running on sequential machines including a Macintosh machine, an HP Apollo 3500 computer and a DEC 3100 computer under Unix operating system. The state of the system can usually converge to the near-optimum solution within about 500 iteration steps. The algorithm uses n processing elements where each processing element performs the McCulloch-Pitts binary neuron.

6.2 MAXIMUM INDEPENDENT SET PROBLEMS

An independent set in a graph is a set of vertices, no two of which are adjacent. A maximum independent set is an independent set whose cardinality is the largest among all independent sets of a graph. The problem of finding a maximum independent set for arbitrary graphs is NP-complete (Karp 1972) (Garcy and Johnson 1979). Gavril developed a $\theta(n^3)$ time algorithm for finding a near-maximum independent set in a circle graph where n is the number of edges in the circle graph (Gavril 1974). Supowit proposed an $O(n^2)$ time algorithm in the circle graph (Supowit 1987). Hsu gave an $O(m^4)$ time algorithm on planar perfect graphs where m is the number of vertices (Hsu 1988). Choukhmane (Choukhmane and Granco 1986) and Burns (Burns 1989) proposed an algorithm on cubic planar graphs. Masuda

where d_{xy}=1 if the *x*th edge and the *y*th edge intersect each other in the circle graph, 0 otherwise. Note that A and B are constant coefficients. Edge-intersection conditions between the *i*th and the *j*th edges in the circle graph are given by: head(i)<head(j)<tail(i)<tail(j) and head(j)<head(i)<tail(j)<tail(i) where tail(i) and head(i) are two end vertices of the ith edge. Note that distance(i) is given by distance(i)=min(|head(i) -tail(i)|, |n+head(i) -tail(i)|) where tail(i)>head(i) is always satisfied. The function h(x) is 1 if x=0, 0 otherwise.

The first term is the inhibitory force in order to remove the edges which intersect with the *i*th edge in the circle graph. If the ith edge is removed from the circle graph, the first term will not be activated at all, because the state of the ith neuron should be V_i=1. In order to keep the *i*th edge in the circle graph, the first term should not have any edge-intersection violation. Whenever the *i*th edge have any edge-intersection violation, it will be eventually removed from the circle graph. The last term is the encouragement force to embed the *i*th edge in the circle graph. If the *i*th edge is removed but does not intersect with any other edges, the last term will force the *i*th neuron to be V_i=0. In other words, the *i*th edge is encouraged to exist in the circle graph. Fig. 6-6a shows the circle graph with 50 vertices and 122 edges. One of the solutions in Fig. 6-6a is shown in Fig. 6-6b.

G-C base pairs and A-U base pairs. The possible base-pair must also satisfy the hairpin-loop constraint such as |head(i) -tail(i)|>3. Because Tinoco (Tinoco et al. 1971) stated that it is sterically impossible to organize the hairpin loop with less than three bases. The circle graph is fed to the neural network simulator in order to find the near-maximum independent set.

Our simulator was tested by solving several secondary structure prediction problems in ribonucleic acids. In this Chapter three examples are only shown. A sequence of 38 bases from residues 1118-1155 of E. coli 16S rRNA given by Stern (Stern et al. 1988) was used. Fig. 6-7a shows the secondary structure proposed by Stern where the strength of structure stability is computed based on Tinoco's stability number (Tinoco et al. 1971). The stability number of the secondary structure in Fig. 6-7a is +7. Fig. 6-7b shows the circle graph with 38 vertices and 151 edges. Remember that each edge represents a possible base-pairing.

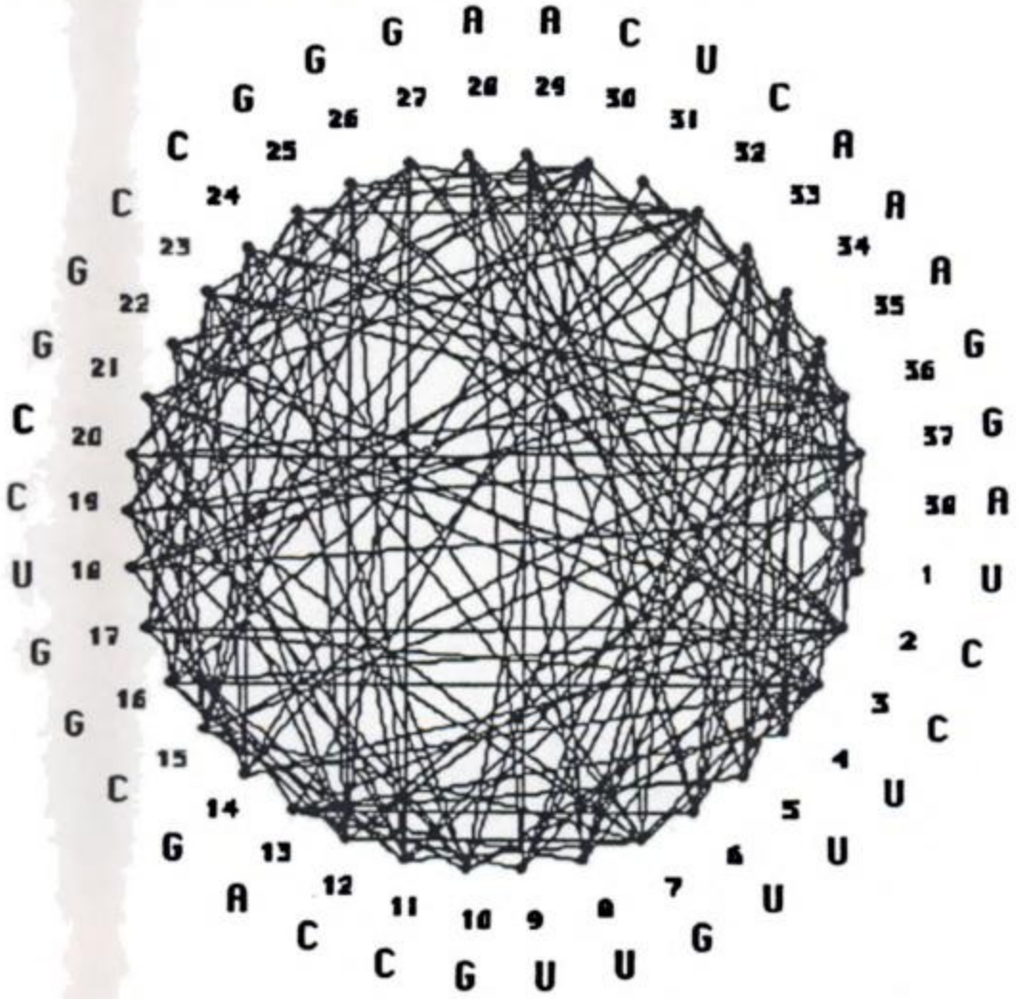


Fig. 6-7 (b) Circle graph with 38 vertices and 151 edges

Finally a sequence of 359 bases from the potato spindle tuber viroid (PSTV) was used to verify our algorithm. Gross (Gross et al. 1978) proposed the secondary structure of the PSTV where the stability number is +62. The circle graph with 359 vertices and 1017 edges was generated where possible base pairs (i and j) were given by the following condition: $350<i+j<370$. The state of the system converged to the solution in the 240th iteration step with A=1, B=0.01, and $U_i(0)=-5$ for i=1,..., 1017. The simulation result was obtained when A=1, B=0.01, and small negative random numbers are assigned to $U_i(0)$ for i=1,..., 1017. The secondary structure predicted by our algorithm is composed of 359 vertices and 128 edges where the stability number is +65. Sanger proposed another secondary structure of the PSTV where the stability number is +64 (Sanger 1984). It indicates that our simulator found the most stable structure of the PSTV. Our simulation result shows that within about 500 iteration steps the state of the system can converge to the solution in the PSTV secondary structure prediction problem. The details of the PSTV experiment are depicted in (Takefuji 1991).

In this Section we have shown the parallel algorithm for the maximum independent set problem which is modified for predicting the secondary structure of ribonucleic acids. The algorithm uses n processing elements where n is the number of edges in the circle graph or the number of possible base-pair in ribonucleic acids. Our simulation result shows that the state of the system converges to the solution within several hundred iteration steps. The simulator discovered the most stable structures in a sequence of 38 bases and a sequence of 359 bases from the PSTV within 500 iterations.

6.4 PLANARIZATION AND RNA STRUCTURE PREDICTION

The stability number for a given RNA secondary structure is the sum of the contributions of the loops, bulges, and helices. The structure with the highest number is the most stable, called optimal folding. The mathematical problem to compute an optimal folding based on free-energy minimization is mapped onto a

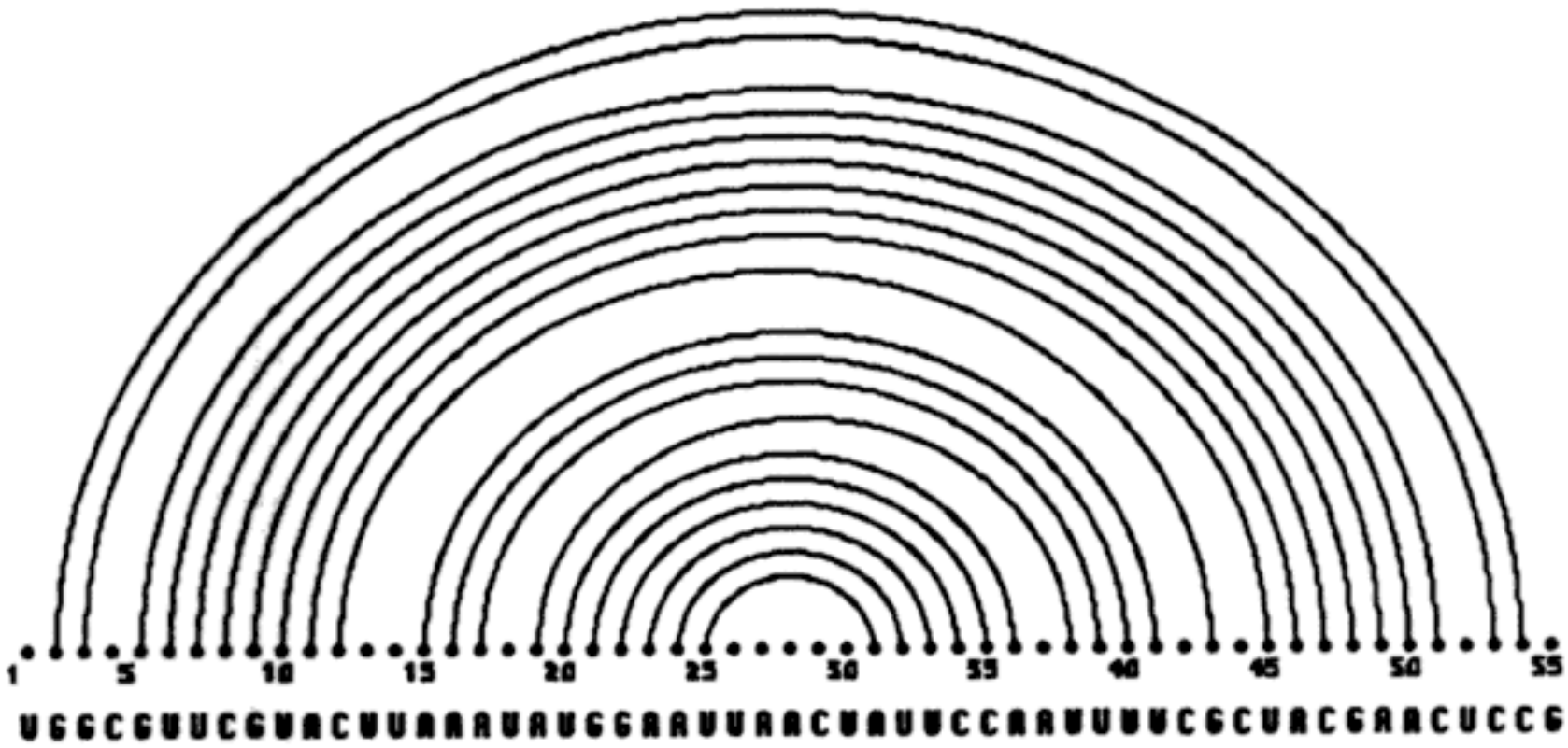


Fig. 6-10 (b) State of the system after the sixty-first iteration step

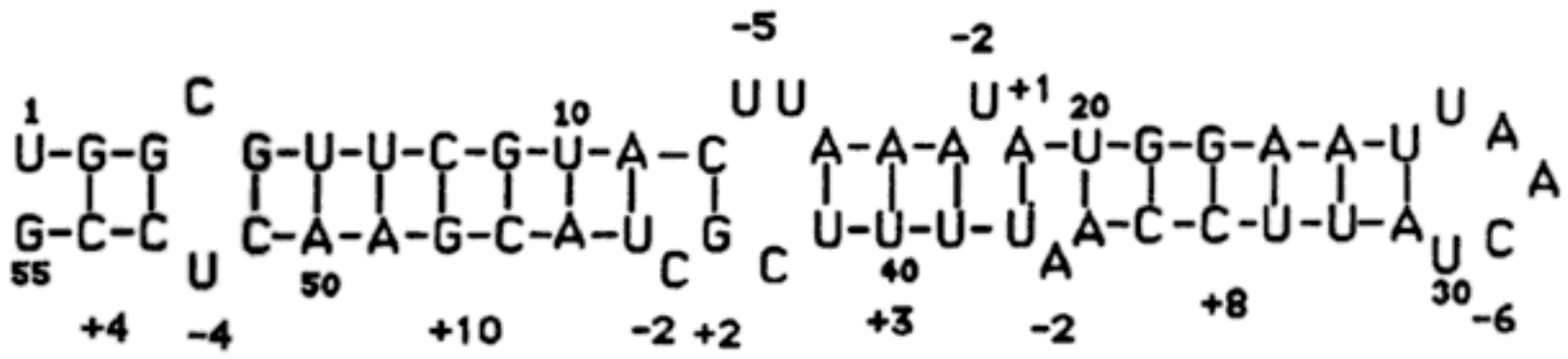


Fig. 6-11 Secondary structure predicted by our algorithm

6.5 REFERENCES

Burns J. E., 1989, The maximum independent set problem for cubic planar graph. Networks, 19.

Choukhmane E., and Franco J., 1986, An approximation algorithm for the maximum independent set problems in cubic planar graphs. Networks, 16.

Diener T. O., 1987, The Viroid (Plenum Press, New York).

Fresco J. R., Alberts B. M., and Doty P., 1960, Some molecular details of the secondary structure of ribonucleic acid. Nature, 188, 98.

Garey M. R., and Johnson D. S., 1979, Computers and Intractability: A Guide to the theory of NP-completeness (W. H. Freeman and Company).

Gavril F., 1974, Algorithms on circular-arc graph. Networks, 4, 357-369.

Fig. 7-1 shows where the knight moves in an L-shape route. In a 3 x 4 chessboard knight's tour problem, 12 squares are numbered from left to right and from top to bottom as shown in Fig. 7-2a. From #2 square there are three valid moves: $V_{2,8}$, $V_{2,9}$ and $V_{2,11}$ where two moves are only needed to find the Hamiltonian circuit.

Fig. 7-2b shows the p(p-1)/2=66 neurons for this problem where the black squares indicate their outputs are one's. Fig. 7-2b depicts the two closed loops as shown in Fig. 7-2c which is not the Hamiltonian circuit. Unfortunately there is no solution for the 3 x 4 knight's tour problem (17). In 1943 Fred. Schuh stated that the number of squares has to be even which is necessary but not sufficient (17).

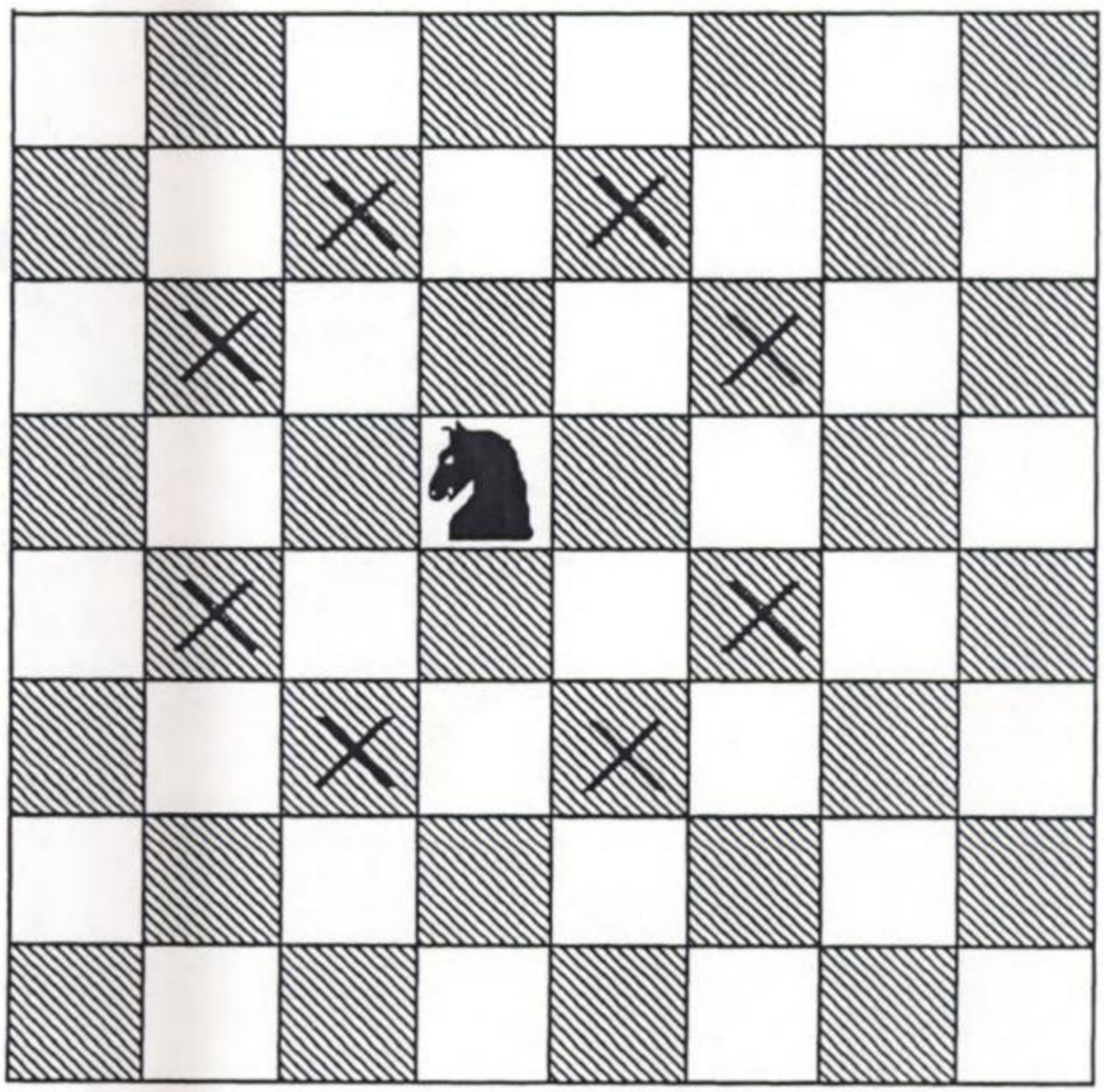


Fig. 7-1 (a) The knight moves in an L-shape route

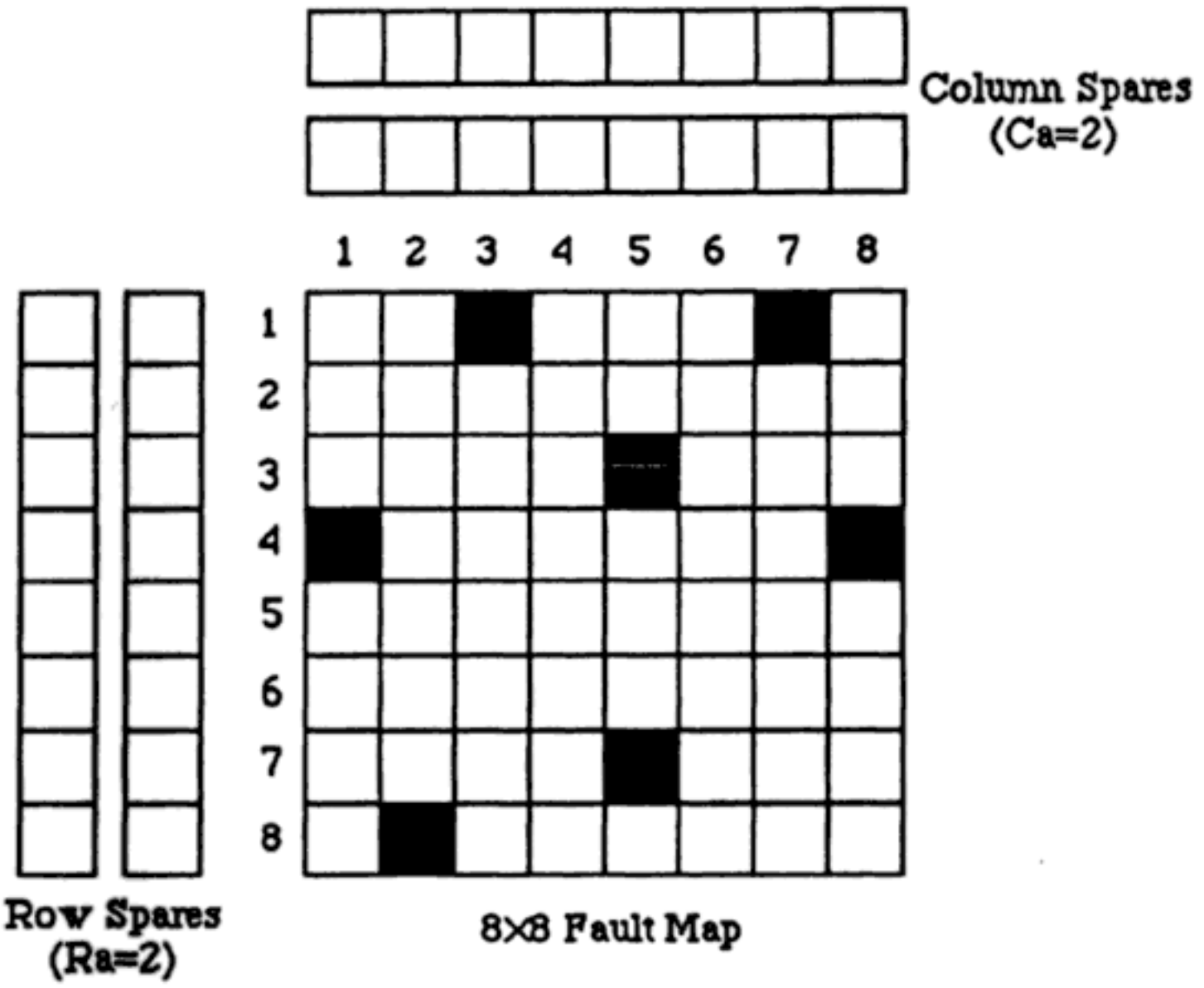


Fig. 8-1 8×8 fault map problem with two spares per row and two spares per column

time sequential algorithms for this problem have been proposed in RRAM applications (Tarr M., Boudreau D., and Murphy R. 1984), (Day J. R. 1985), (Kuo S-Y. and Fuchs W. K. 1987), (Wey C-L. and Lombardi F. 1987), (Haddad R. W. and A. T. Dahbura A. T. 1987), (Huang W. K., Shen Y-N., and Lombardi F. 1990). In 1984, Tarr et al. proposed two algorithms ; the broadside approach and the repair-most approach (Tarr M., Boudreau D., and Murphy R. 1984). The broadside approach is to scan the memory and to replace each faulty cell with a spare memory per row or a spare memory per column whichever is available where no optimization is performed. The repair-most approach is to repeatedly repair the row or column which has the most faulty cells. In 1985, Day proposed the fault-driven approach which is to repair the faulty cells according to user-defined preferences (Day J. R. 1985). In 1987, Kuo and Fuchs proposed the branch-and-bound approach and the heuristic polynomial time approximation approach (Kuo S-Y. and Fuchs W. K. 1987). In 1987, Wey and Lombardi proposed another approach which determines both

from 20×20 to 200×200.

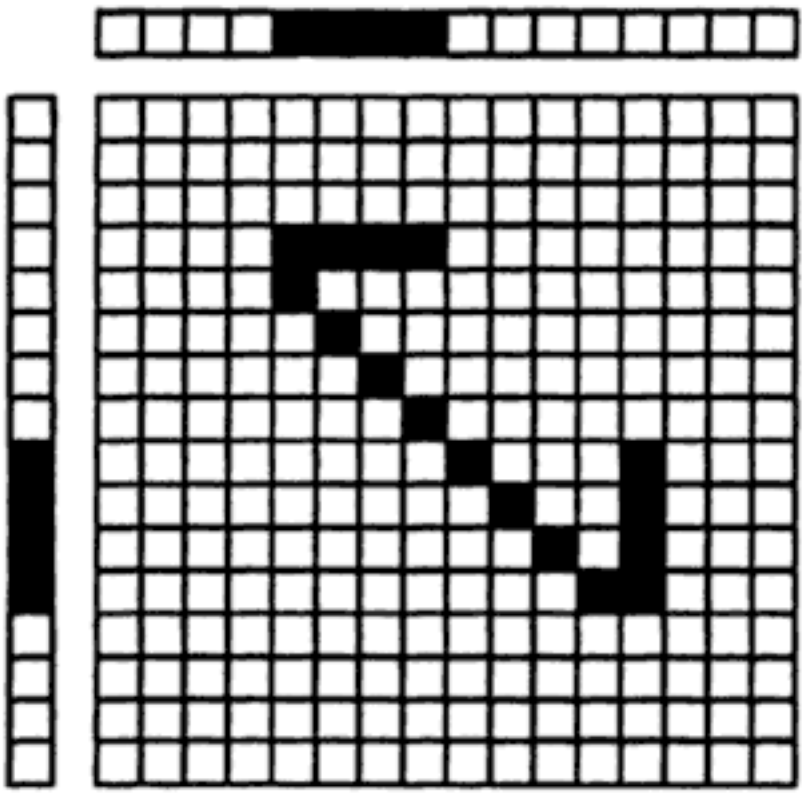


Fig. 8-3 16 x 16 fault map problem with Ra=Ca=4

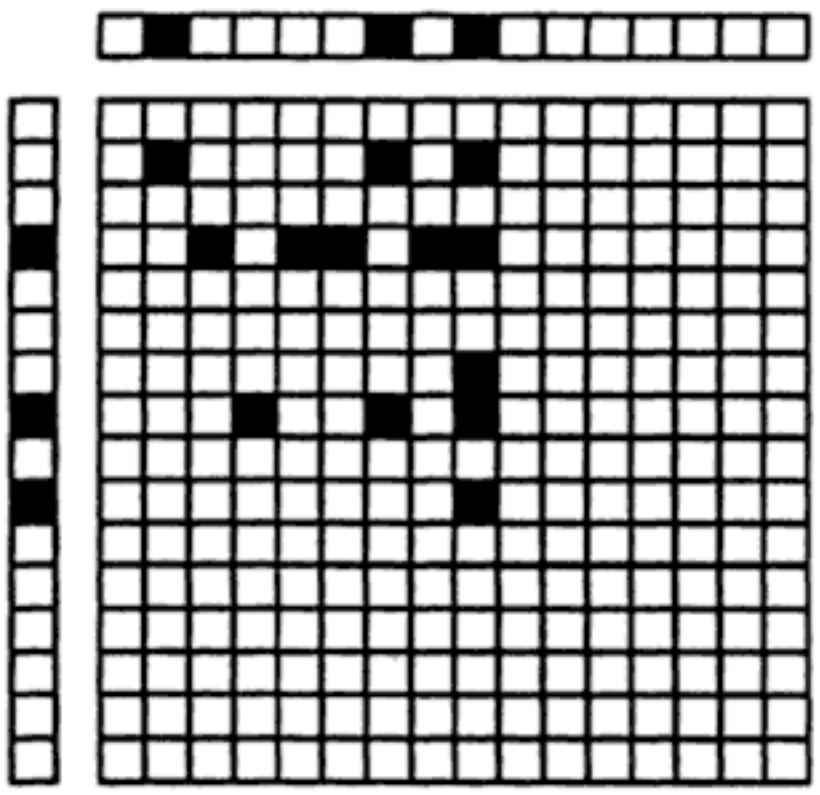


Fig. 8-4 16 x 16 fault map problem with Ra=Ca=3

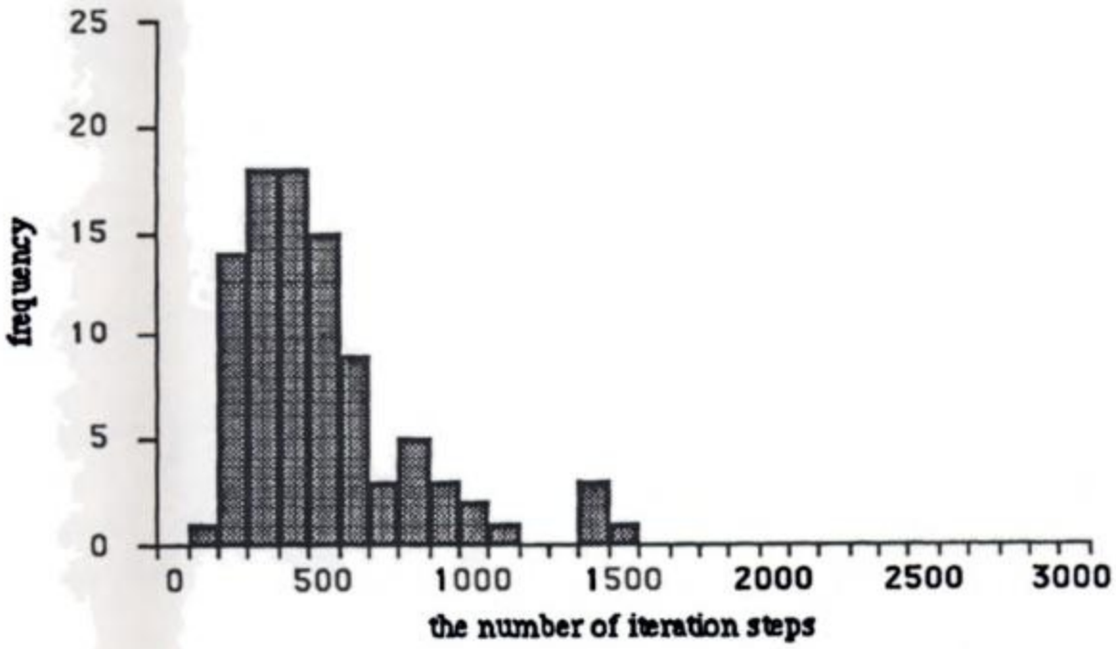


Fig. 8-7 Relationship between the number of iteration steps and the frequency for the 40 x 40 fault map problem

8.4 EXERCISES

1. Calculate the energy function E for the spare allocation problems from Eqs. (8. 3) and (8. 4).
2. Write a subroutine to generate a faulty cell pattern with a specified probability of a faulty cell occurring.
3. Generate a faulty cell pattern and solve it with your simulator.
4. The proposed motion equations in Eqs. (8. 3) and (8. 4) find a spare allocation pattern by using exact Ra spares per row and exact Ca spares per column. However, it is actually sufficient to find a spare allocation pattern by using Ra or less spares per row and Ca or less spares per column. How should the motion equations be modified in order to satisfy this condition ?
5. Modify your simulator with the modified motion equations and run it. Summarize the simulation results.

encourages the exact matching patterns to remain while the first term is zero.

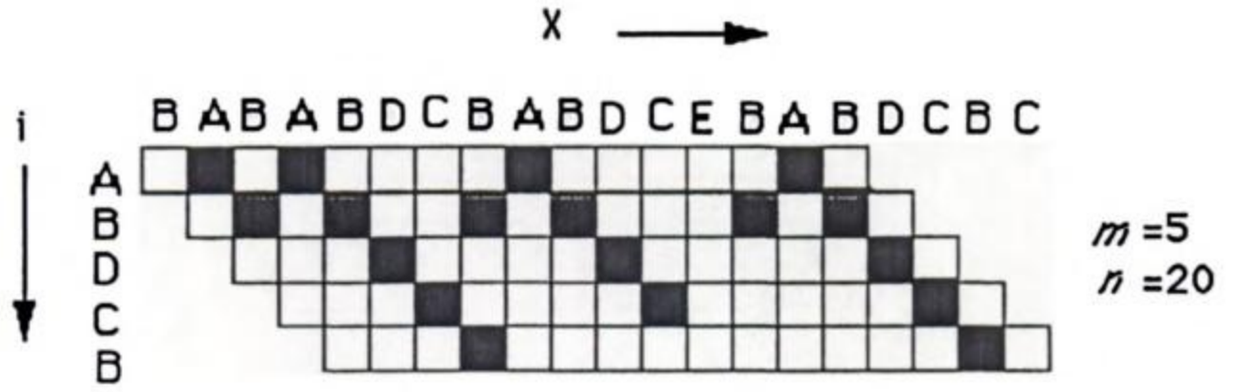


Fig. 9-4 (a) The state of the system after the first iteration step

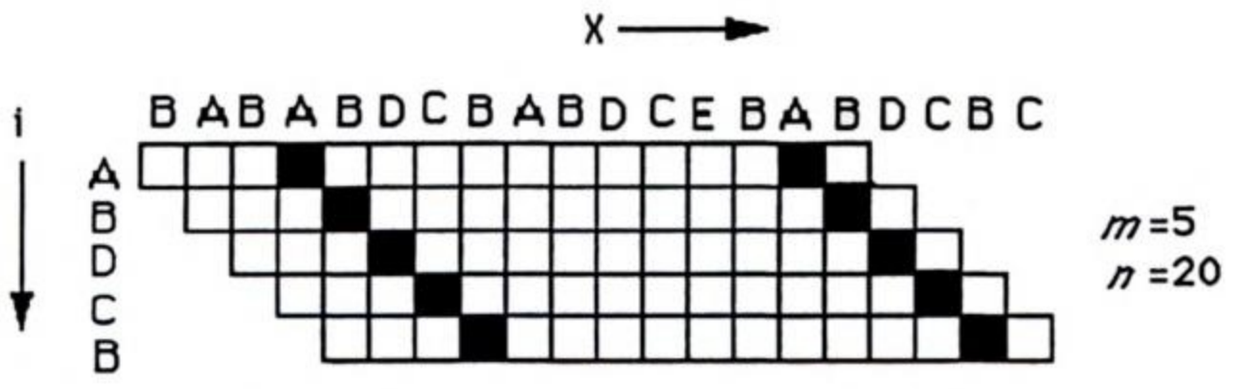


Fig. 9-4 (b) The final state of the system after the second iteration step

A large number of simulation runs were performed with various length of texts and patterns on Macintosh SE/30. Fig. 9-5 shows the simulation result of the text length 100. The characters of the text and the pattern in Fig. 9-5 were randomly generated. In Fig. 9-5, the pattern "cdac" is located at two places in the 100-length text.

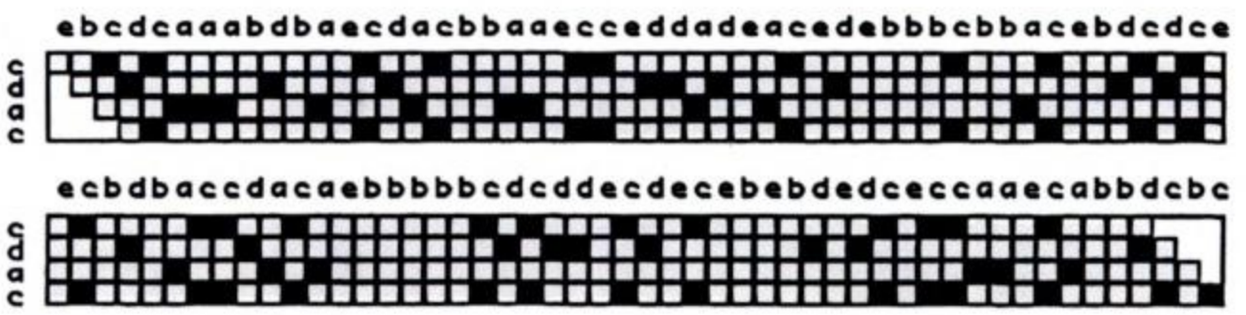


Fig. 9-5 (a) The state of the system after the first iteration step

neural network array. The second term forces no two markers to be placed in the same position on the checkerboard. The third term is always inhibitory which describes the overlap violation between polyominoes where the ten violation functions for ten different polyominoes: f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), f(9), and f(10) are given by respectively:

$$f(1)=\sum_{\substack{q=1\\ q\neq 1}}^{11}\left(V'_{qjk}+V'_{q,j+1,k}+V'_{q,j,k+1}+V'_{q,j,k+2}+V'_{q,j+1,k+2}\right)$$

$$f(2)=\sum_{\substack{q=1\\ q\neq 2}}^{11}\left(V'_{qjk}+V'_{q,j+1,k}+V'_{q,j+1,k+1}+V'_{q,j+1,k+2}+V'_{q,j+2,k+2}\right)$$

$$f(3)=\sum_{\substack{q=1\\ q\neq 3}}^{11}\left(V'_{qjk}+V'_{q,j,k+1}+V'_{q,j,k+2}+V'_{q,j+1,k+2}+V'_{q,j+2,k+2}\right)$$

$$f(4)=\sum_{\substack{q=1\\ q\neq 4}}^{11}\left(V'_{qjk}+V'_{q,j+1,k}+V'_{q,j+2,k}+V'_{q,j+1,k+1}+V'_{q,j+1,k+2}\right)$$

$$f(5)=\sum_{\substack{q=1\\ q\neq 5}}^{11}\left(V'_{qjk}+V'_{q,j+1,k}+V'_{q,j+1,k-1}+V'_{q,j+1,k+1}+V'_{q,j+2,k}\right)$$

$$f(6)=\sum_{\substack{q=1\\ q\neq 6}}^{11}\left(V'_{qjk}+V'_{q,j+1,k}+V'_{q,j+2,k}+V'_{q,j+3,k}+V'_{q,j+4,k}\right)$$

$$f(7)=\sum_{\substack{q=1\\ q\neq 7}}^{11}\left(V'_{qjk}+V'_{q,j+1,k}+V'_{q,j+1,k-1}+V'_{q,j+2,k-1}+V'_{q,j+2,k-2}\right)$$

Printed in the United States
54584LVS00002B/250

Distributors for North America:
Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:
Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

Takefuji, Yoshiyasu, 1955-
Neural network parallel computing / by Yoshiyasu Takefuji.
p. cm. -- (The Kluwer international series in engineering and computer science ; SECS 0164)
Includes bibliographical references (p.) and index.
ISBN 0-7923-9190-X (acid free paper)
1. Neural networks (Computer science) 2. Parallel processing (Electronic computers) I. Title. II. Series.
QA76.87.T35 1992
006.3--dc20 91-42280
CIP

Printed on acid-free paper.

Printed in the United States of America