Summer 1992

An Application of Neural Networks in Data Communication Real-Time Resource Reallocation

Qing Fan
Old Dominion University

AN APPLICATION OF NEURAL NETWORKS IN DATA COMMUNICATION REAL-TIME RESOURCE REALLOCATION

by

Qing Fan
B.S.E.E. August 1990, Old Dominion University

A Thesis submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

MASTER OF SCIENCE

ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
August, 1992

Approved by:

Mark D. Pardue (Director)

David L. Livingston

John W. Stoughton

Derya Alasya

ABSTRACT

AN APPLICATION OF NEURAL NETWORKS
IN DATA COMMUNICATION REAL-TIME RESOURCE REALLOCATION

Qing Fan
Old Dominion University, 1992
Director: Dr. Mark D. Pardue

This thesis presents an application of artificial neural networks in real-time resource reallocation, a methodology used in the implementation of an intelligent interface node in the Computer Integrated Manufacturing (CIM) environment. In particular, the problem is formulated using a Hopfield neural network model. The real-time reallocation problem is mapped into a two-dimensional matrix of neurons similiar to Hopfield and Tank's approach to the traveling salesman problem (TSP). An energy function is formulated in terms of the hard constraints and the solution cost. The interconnection weights and the input biases are determined by the energy function. It is shown through computer simulations that a deterministic Hopfield network does not always provide good solutions for the present problem. However, better solutions are obtained by using the Boltzmann machine with simulated annealing, although the long annealing schedules required for optimal solutions preclude its use for this problem application.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to Dr. Mark D. Pardue for his excellent guidance throughout this research. I would like to thank Dr. David L. Livingston, Dr. John W. Stoughton, and Dr. Derya Alasya for their time and effort in revising my thesis. Special thanks are due to my colleagues Jagadesh Gullapalli, Jeffrey Sung, and Srikumar Lakshmipathi for their special assistance. Finally, I would like to thank my parents, my brothers, and my relatives for their continuous support for this endeavor.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

Since its inception in 1985, IEEE 802.4 [13] (Token-Passing Access Method) has been the primary standard for local area networks in manufacturing environments. IEEE 802.4 has the following characteristics:

(1) Physically, it is a bus; logically, it is a ring.

(2) A special control frame called a "token" is passed around the logical ring, with the token holder being permitted to transmit data frames.

One major advantage of the token-passing scheme is that stations transmit in a predetermined order and collisions are avoided. However, one limitation of the token-passing scheme is that it is not flexible enough to deal with the case when a station suddenly has urgent data to send but does not hold the token [16].

Pardue [16] has made a few modifications to the 802.4 token bus standard by adding "an intelligent interface node capability" to suit the special needs of the Computer Integrated Manufacturing (CIM) environment, where there are occasionally critical real-time data transfer requirements

that have to be met in order to avoid high maintenance costs or catastrophies.

The intelligent interface node has two characteristics:

(1) the ability to deal with the uncertain conditions at each station and reprioritize its real-time data transfer requirements and

(2) the ability to reorder the token-passing sequence to guarantee the real-time data transfer using the real-time reallocation technique.

The real-time reallocation is an n/1 problem according to scheduling theory [3] and is an optimization problem in constraint satisfaction. This problem was addressed by Pardue using an expert system based on an algorithmic appoach. The underlying mechanics of the Pardue reallocation method is optimization by heuristically swapping the positions of a pair of stations if the swapping does not adversely affect the outcome. This reallocation algorithm is implemented on a serial computer. An analysis of the algorithm has shown that its execution time has a worst-case growth rate of order N^3 for an N-station token bus.

An artificial neural network is a computing device consisting of a number of processing elements loosely based on neurons in living things. In general, each neural network has a massive number of communication links between all processing elements to perform collective computation.

Neural networks are attractive due to the following:

(1) They have been shown to be successful in providing real-time response to combinatorial optimization problems and complex pattern recognition problems. The problem solving process of the artificial neural networks is non-algorithmic and parallel in nature, which is different from the process used in conventional programmed computers and rule-based expert systems [5].

(2) They are physically realizable since their architecture can take different physical forms: electronic, electro-optical, and entirely optical.

Neural networks have been successfully applied to solve a variety of problems. In particular, Hopfield neural networks have provided acceptable solutions to many optimization problems, such as A/D conversion and linear programming [18], the traveling salesman problem [12], and job-shop scheduling [4].

1.1 Objective

The goal of this thesis is to develop a neural network model for solving the real-time resource reallocation problem and evaluate its validity and performance.

1.2 Thesis Organization

The problem addressed in this research is introduced in chapter one. The thesis objective and its organization are

then outlined. The background on the real-time resource reallocation technique is presented in chapter two. The background on constraint satisfaction networks is presented in chapter three. Theorectical development of the current work is presented in chapter four. Simulations of using the proposed neural network model in solving real-time reallocation problems are presented in chapter five. The conclusions of this thesis and the possible future research efforts are presented in chapter six.

CHAPTER TWO

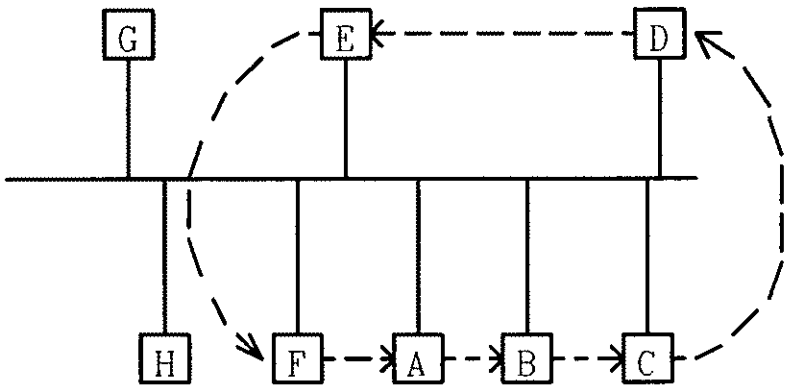
REAL-TIME RESOURCE REALLOCATION

The real-time resource reallocation problem was briefly introduced in chapter one. In this chapter, the background for the real-time resource reallocation technique is presented.

2.1 Introduction

Physically, a token bus network is a linear or tree-shaped cable onto which stations are attached. Logically, the stations are organized into a ring (see figure 2.1), with each station knowing the address of the station to its "left" and "right." When the logical ring is initialized, the highest numbered station may send the first frame. After it is finished, it passes permission to its immediate neighbor by sending the neighbor a special control frame called a "token." The token propagates around the logical ring, with only the token holder being permitted to transmit frames. Since only one station at a time holds the token and thus may transmit, collisions do not occur [13].

One major disadvantage of the token-passing scheme, as mentioned in chapter one, is its inability to respond to the case when a station has urgent data to send but does not hold the token, which can occur in a Computer Integrated Manufacturing environment [16].



Note: Stations G and H are non-transmitting stations.

Figure 2.1. Logical ring on physical bus.

2.2 Intelligent Interface Node

Pardue [16] has introduced a new architecture called "Intelligent Interface Node" to deal with the problem introduced above. The architecture is shown in figure 2.2. In its implementation, a "secondary channel" is used to pass control information between intelligent interface nodes. Upon the completion of the current data transfer and before the passing of the token, an intelligent interface node will look at the information gathered from all the nodes on the network, and perform reordering (when necessary) of the token-passing sequence based on priority and timing information. In order to perform reallocation, the following information is required by the interface node:

(1) interface node ID,

(2) time-to-live (TTL) of the real-time data to be transferred, in number of the data transfer opportunities,

(3) next data transfer opportunity (NT), and

(4) priority.

The TTL and NT are expressed in terms of time periods defined as the "maximum time between one node starting data transfer and the next node starting data transfer." Specifically, a time period equals the sum of Token Holding Time, Node Delay, and Transmission Delay, all of which are defined by IEEE 802.4 standard.

In the real-time resource reallocation scheme [16], the

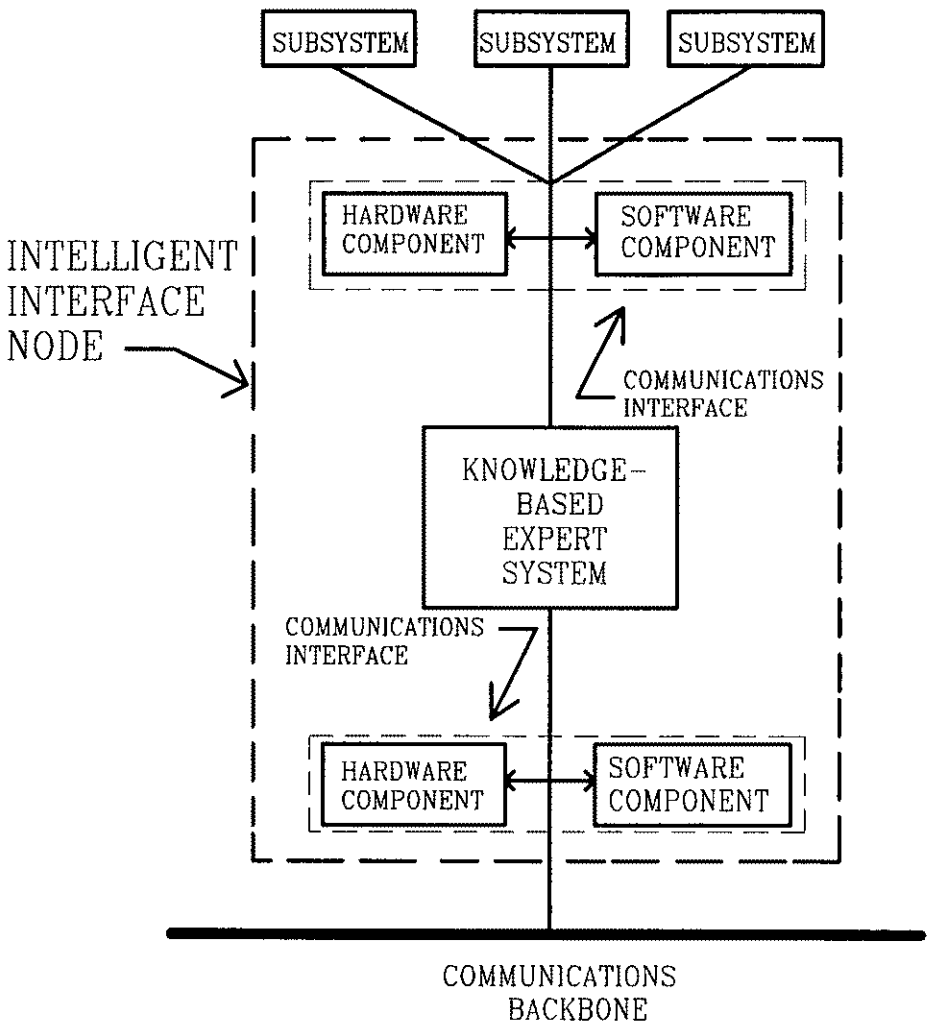


Figure 2.2. Architecture for an intelligent interface node.

utilization of communication resources is identified by calculating TTL - NT for each station. For a particular station, a positive result indicates a surplus of the communication resource, and a negative result indicates a shortfall which means the real-time data transfer requirement is not satisfied. When a shortfall occurs, the communication resources are reallocated by swapping the station positions subject to the following optimization criteria:

(1) reduce the number of shortfalls for higher priority data to a minimum before reducing the number of shortfalls for the lower priority data.

(2) reduce the total number of shortfalls with the first criterion satisfied.

2.3 Demonstration of Methodology

In this section, the following three cases will be used to demonstrate the real-time resource reallocation methodology:

(1) case 1 - no shortfall in the final solution (with no priority assignment),

(2) case 2 - shortfall in the final solution (with the first priority assignment), and

(3) case 3 - shortfall in the final solution (with the second priority assignment).

These cases are discussed in detail below.

Case 1: No shortfall in final solution (with no priority assignment)

Table 2.1(a) presents the initial token-passing sequence and timing information for a six-station network. Table 2.1(b) identifies the surplus or shortfall corresponding to the initial configuration. It can be seen that station F suffers a shortfall of 3 time periods. To eliminate the station F shortfall, F must be given the data transfer opportunity in one of the first 3 time periods. If F and C swap positions, there is no improvement overall because C now suffers a shortfall. However, if F and B swap positions, as indicated in table 2.1(c), both F and B will be guaranteed their required data transfer opportunity. Thus, the sequence in table 2.1(d), is the best solution.

Table 2.1. Application of reallocation technique to case 1.

(a) Beginning states.

Station ID	Time-to-live	Next Transfer Opportunity
A	3	1
B	6	2
C	4	3
D	5	4
E	6	5
F	3	6

Table 2.1. Application of reallocation technique to case 1 (continued).

(b) 1st attempt to alleviate station F shortfall

Station ID	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall
A	3	1	2
B	6	2	4
C	④	③	1
D	5	4	1
E	6	5	1
F	③	⑥	-3

(c) Operation to alleviate station F shortfall

Station ID	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall
A	3	1	2
B	⑥	②	4
C	4	3	1
D	5	4	1
E	6	5	1
F	③	⑥	-3

Table 2.1. Application of reallocation technique to case 1 (continued).

(d) Final token-passing sequence

Station ID	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall
A	3	1	2
F	3	2	1
C	4	3	1
D	5	4	1
E	6	5	1
B	6	6	0

Case 2: Shortfall in final solution (with the first priority assignment)

Table 2.2(a) presents the initial token-passing sequence, timing information, and priority assignment for a six-station network for case 2. Table 2.2(b) identifies the surplus and shortfall corresponding to the initial configuration. It can be seen that both stations D and F suffer from shortfalls. Since station F has priority 4, which is higher than station D, the positions between F and D are swapped. Table 2.2(c) shows the result after the swapping. It can be seen that no further improvement is possible.

Table 2.2. Application of reallocation technique to case 2.

(a) Beginning states

Station ID	Priority	Time-to-live	Next Transfer Opportunity
A	2	3	1
B	0	2	2
C	1	3	3
D	0	3	4
E	0	5	5
F	4	4	6

(b) Operation to alleviate station F shortfall

Station ID	Priority	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall (TTL-NT)
A	2	3	1	2
B	0	2	2	0
C	1	3	3	0
D	0	③	④	-1
E	0	5	5	0
F	4	④	⑥	-2

Table 2.2. Application of reallocation technique to case 2 (continued).

(c) Final token-passing sequence

Station ID	Priority	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall (TTL-NT)
A	2	3	1	2
B	0	2	2	0
C	1	3	3	0
F	4	4	4	0
E	0	5	5	0
D	0	3	6	-3

Case 3: Shortfall in final solution (with the second priority assignment)

Table 2.3(a) presents the initial token passing sequence, timing information and priority assignment for a six-station network for case 3. Table 2.3(b) identifies the surplus and shortfall corresponding to the initial configuration. It can be seen that both stations F and D suffer from shortfalls. Since station F has higher priority, a swap between F and D will reduce the number of shortfalls down to 1, indicated in table 2.3(c). By inspection, it can be seen that not all shortfalls can be eliminated. However, this is not the optimal solution because it still violates the optimization criterion 1: station D with priority 1 must be satisfied before station B with priority 0. Therefore, a swap between D and B will result in the optimal solution in table 2.3(d).

Table 2.3. Application of reallocation technique to Case 3.

(a) Beginning states

Station ID	Priority	Time-to-live	Next Transfer Opportunity
A	2	3	1
B	0	2	2
C	1	3	3
D	1	3	4
E	0	5	5
F	4	4	6

(b) Operation to alleviate station F shortfall

Station ID	Priority	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall (TTL-NT)
A	2	3	1	2
B	0	2	2	0
C	1	3	3	0
D	1	③	④	-1
E	0	5	5	0
F	4	④	⑥	-2

Table 2.3. Application of reallocation technique to case 3 (continued).

(c) Operation to alleviate station D shortfall

Station ID	Priority	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall (TTL-NT)
A	2	3	1	2
B	0	2	2	0
C	1	3	3	0
F	4	4	4	0
E	0	5	5	0
D	1	3	6	-3

(d) Final token-passing sequence

Station ID	Priority	Time-to-live	Next Transfer Opportunity	Surplus/ Shortfall (TTL-NT)
A	2	3	1	2
D	1	3	2	1
C	1	3	3	0
F	4	4	4	0
E	0	5	5	0
B	0	2	6	-4

CHAPTER THREE

CONSTRAINT SATISFACTION NEURAL NETWORKS

The background of constraint satisfaction neural networks is presented in this chapter. General concepts of artificial neural networks are introduced first in section 3.1. Then, the Hopfield neural network model and the Boltzmann machine with simulated annealing is introduced in section 3.2 and section 3.3, respectively. Finally, the neural network model suitable for this research is proposed in section 3.4.

3.1 General Concepts

An artificial neural network is a computing device consisting of a number of processing elements loosely based on neurons in human and other biological systems. In biological systems, each neuron is connected to a number of other neurons by dendrites and axons [19]. A neuron receives signals from other neurons at junction points called synapses. The signals received at each synapse are either excitatory or inhibitory. If the sum of these signals is above a threshold, then the cell fires. On the other hand, if the sum of these signals is below the threshold, then the cell is inhibited from firing.

Each neuron passes its state to a number of other neurons through output structures called axons. Similarly, in an artificial neural network, nonlinear threshold elements are used to mimic the biological neurons. Each threshold element receives a weighted sum of inputs from other elements. If the result is above the built-in threshold, the element fires.

According to directions of signal flow, artificial neural networks fall into two categories: feed-forward networks and recurrent networks [19]. In feed-forward networks, the signals flow from input layer to the output layer. In recurrent networks, signals not only flow from the input layer to the output layer, but also flow from output layer back into the input layer. Unlike feed-forward networks, which are always stable, recurrent networks have a dynamic reponse and may be unstable.

Recurrent networks have been studied and found to be useful in optimization. Cohen and Grossberg [2] devised a theorem that defines a subset of recurrent networks whose output can eventually reach a stable state. Hopfield and Tank [12] were one of the firsts to apply a recurrent network to solving combinatorial optimization problems. Their network configuration is often called the Hopfield network.

3.2 Hopfield Network

In Hopfield's original model [9][10], the artificial neurons are modeled using the McCulloch-Pitts input/output

function $V_i=f(U_i)$, where U_i and V_i are the net input and output of the ith neuron respectively, and

$$V_i = 0, \quad if\ U_i \leq 0$$

$$V_i = 1, \quad if\ U_i > 0. \tag{3.1}$$

The net input to the ith neuron can be computed using

$$U_i = \sum_{j \neq i} T_{ij} V_j + I_i \,. \tag{3.2}$$

The element T_{ij} represents the connection strength between the ith neuron and the jth neuron. The element V_j represents the output of the jth neuron. The element I_i is the external input bias applied to the ith neuron. The neuron states can be updated according to the rules above.

Hopfield has shown that a network with symmetric connections ($T_{ij} = T_{ji}$ and $T_{ii} = 0$) always leads to convergence to a stable state, which represents one of the local minima of the energy function

$$E = -\frac{1}{2}\sum_{i \neq j}\sum T_{ij} V_i V_j - \sum_i V_i I_i \,. \tag{3.3}$$

In general, the steps involved in solving an optimization problem using a Hopfild network are

(1) incorporate all the constraints in the form of an energy function,

(2) determine the weights and biases, and

(3) let the network relax to a stable state which is interpreted as a solution.

Since the Hopfield network provides a gradient descent method to minimize the energy function, the final state into which the network converges is one of the local minimum in the energy terrain, which may or may not be the global minima. Also, where the network converges is determined by the network initial condition. Inability to consistently provide global solutions is a serious drawback of the Hopfield network.

A class of networks known as Boltzmann machines aim at solving the local minima problem. The concepts of the Boltzmann machine are detailed in the next section.

3.3 Boltzmann Machine with Simulated Annealing

The Boltzmann machine, introduced by Hinton and Sejnowski [8], can be viewed as a generalization of the discrete Hopfield network. Like the discrete Hopfield model, the Boltzmann machine uses binary units; unlike the discrete Hopfield model, the Boltzmann machine uses a probabilistic state transition instead of a deterministic one. In the discrete Hopfield network discussed in section 3.2, each neuron i can compute the energy difference δE_i by

$$E_{i,off} - E_{i,on} = \delta E_i = \sum_{j \neq i} T_{ij} V_j + I_i \,, \tag{3.4}$$

where $E_{i,off}$ is the energy of the system when ith neuron is "off" or 0, and $E_{i,on}$ is the energy of the system when ith neuron is "on" or 1. In order to minimize the global energy, the ith neuron assumes 1 if δE_i is positive, and assumes 0 if δE_i is negative. In a Boltzmann machine, however, the ith neuron assumes 1 with probability p_i, where

$$p_i\,(unit = 1) = \frac{1}{1 + e^{-\frac{\delta E_i}{T}}} \,. \tag{3.5}$$

T is a parameter analogous to temperature and measures the noise introduced into the decision. It can be observed that a discrete Hopfield network is a special case of the Boltzmann machine when T is zero.

Annealing is a thermal process used in obtaining the ground state characteristics in solid. In an annealing process, a solid is first raised to a melting temperature and then gradually cooled down until it reaches the ground state. The solid in the ground state is considered to have the minimal system energy. In the annealing process, the cooling must be done carefully in order for the system to reach the minimum energy state. The concept of the annealing was applied to the combinatorial optimization by Kirkpatrick, et

al. [14]. The method is called simulated annealing. In simulated annealing, a fabricated energy is minimized through a "cooling" schedule similar to that of the annealing process.

In the Boltzmann machine with simulated annealing, the parameter T in equation 3.5 is used as the artificial temperature. T is initially a high value. At each temperature T, neurons are visited randomly, and each neuron i assumes 1 with the probability p_i according to equation 3.5. The temperature is reduced by a factor r ($0 < r < 1$) only when a constant statistical average has been reached. The process is stopped when a "freezing" temperature (which is near zero) has been reached.

The main limitation of the Boltzmann machine with simulated annealing is that global solutions are found at the expense of time. As the problem size gets larger, the annealing schedule required for convergence to the global solutions may be too long to be practical.

3.4 Proposed Model

In this chapter, the background of the constraint satisfaction networks has been presented. The discrete Hopfield model is shown to be appropriate in formulating the real-time resource reallocation problem addressed in chapter two. The problem will be formally treated in chapter four.

CHAPTER FOUR

PROBLEM FORMULATION

In this chapter, the application of neural networks to the real-time resource reallocation problem is formally presented. In section 4.1, a representation for the problem is presented. In section 4.2, an energy function is formulated according to the problem constraints. In section 4.3, the weights and biases of the neural network are determined from the energy function. A summary of this chapter is presented in section 4.4.

4.1 Representation

It is very important to find an appropriate respresentation when applying an artificial neural network to an optimization problem. The matrix representation used in solving the traveling salesman problem [12] is also used here because the present problem is very similar to the TSP problem. For a reallocation problem with N communication stations, an N-by-N matrix is used. Each row index represents the station's ID in the network, and each column index represents the next data transfer opportunities (NT's). The

N-by-N matrix which represents a valid token-passing sequence is called a permutation matrix. In a permutation matrix, each row should have one and only one neuron in the "on" state, and each column should have one and only one neuron in the "on" state. A permutation matrix is shown in table 4.1 for a problem with six communication stations. A six-station problem requires the use of 36 neurons. Table 4.1 illustrates the token-passing sequence C - A - D - F - E - B, one of the 6! = 720 possible stable states, which represent the valid token-passing sequences.

Table 4.1. Neural network representation of real-time reallocation problem for six-station case.

Node ID	Next Transfer Opportunity (NT)					
	0	1	2	3	4	5
A	off	on	off	off	off	off
B	off	off	off	off	off	on
C	on	off	off	off	off	off
D	off	off	on	off	off	off
E	off	off	off	off	on	off
F	off	off	off	on	off	off

4.2 Constraints and Energy Function

To use neural networks to solve the optimization problem, all of the constraints must be represented by an energy

function in which the lowest energy corresponds to the best solution. Similar to the traveling salesman problem, there are two types of constraints to the real-time reallocation problem: hard constraints and soft constraints.

4.2.1 Hard Constraints

The hard constraints are those constraints which must be satisfied by the neural network in order to produce valid solutions. The hard constraints of the real-time reallocation problem are exactly the same as those for the traveling salesman problem [7]:

(1) each row in a permutation matrix should have one and only one neuron with "on" state, and

(2) each column in a permutation matrix should have one and only one neuron with "on" state.

The constraints guarantees that each station is visited once and only once. Therefore, the energy function E_1 corresponding to the two hard constraints is:

$$E_1=\frac{A}{4}\left(\sum_X\left(1-\sum_i V_{Xi}\right)^2+\sum_i\left(1-\sum_X V_{Xi}\right)^2\right), \qquad (4.1)$$

where A is a positive constant. V_{Xi} represents the neuron state in the Xth row and ith column. Similarly, V_{Yj} represents the neuron state in the Yth row and jth column. The energy E_1 is zero if the token-passing sequence is a valid sequence.

4.2.2 Soft Constraints

In chapter two, the real-time reallocation technique was described and demonstrated using three cases. The following aspects can be observed:

(1) any shortfall associated with any priority is undesirable, and

(2) a shortfall associated with the higher priority is much more undesirable than that associated with the lower priority.

The optimization process can be considered in terms of "cost." First, each shortfall is associated with a cost, which increases as the priority increases. When a token-passing sequence does not result in any shortfall, the cost associated with the sequence is zero. When a token-passing sequence does result in one or more shortfalls, the cost associated with the sequence is nonzero and is computed by summing the costs of all the shortfalls. Therefore, the real-time resource reallocation problem can be restated as:

> Given time-to-live TTL_i and priority P_i for station i (i = 1, 2, ... , N), find a configuration C that minimizes the cost
>
> $$\sum_{i=1}^{N} f(P_i)\ S_i\ , \qquad (4.2)$$

where f is an injective mapping from priority domain to cost domain such that $f(P_i)$ represents the cost for a station i shortfall corresponding to its priority. S_i is the shortfall

for station i, where S_i=0 if $TTL_i - NT_i \geq 0$ and S_i=1 otherwise. NT_i is the next data transfer opportunity for station i in configuration C.

The energy function corresponding to the soft constraint is

$$E_2 = D\sum_X \sum_i C_X V_{Xi} \Theta(i - T_X) \ , \qquad (4.3)$$

where $\Theta(x) = 1$ if $x > 0$, and $\Theta(x) = 0$ if $x \leq 0$. If neuron V_{Xi} is on, that means the next transfer opportunity for node X is i. T_X is the time-to-live for node X's data frames. C_X is the "cost" for the shortfall. The value of C_X depends on the priority of station X. The total cost E_2 is zero if there is no shortfall.

An example is used to illustrate the correctness of the energy function formulation of the soft constraint. The example is the case 2 from section 2.3. In the example, the constant coefficient D in the equation 4.3 is assumed to be 1. An exponential function is used to assign the cost: $y = 2^x$, where variable x represents the priority and y represents the cost. The cost assignment is illustrated in table 4.2. This assignment is used because it can give enough penalty for any token-passing sequence that might satisfy the real-time data transfer requirement of lower priority before that of higher priority. Table 4.3 shows the energy corresponding to different configurations. Since the sequence A-B-C-F-E-D is

associated with the lowest energy, it is the best solution. The result is the same as that of table 2.2.

Table 4.2. Cost assignment for illustrations.

Priority	Cost Per Shortfall
4	16
3	8
2	4
1	2
0	1

Table 4.3. Illustration of energy corresponding to the different configurations for case 2.

Token Passing Sequence	Number of Shortfalls for priority					Total Energy (cost)	Global Minimum?
	4	3	2	1	0		
A-B-C-D-E-F	1	0	0	0	1	17	No
A-C-B-D-E-F	1	0	0	0	2	18	No
B-A-C-D-E-F	1	0	0	0	1	17	No
F-B-C-D-E-A	0	0	1	0	1	5	No
A-B-C-F-E-D	0	0	0	0	1	1	Yes

4.3 Weights and biases

In general, the quadratic terms in the energy function (see equation 3.3) define a connection matrix and the linear terms define input biases. By expanding the total energy function of the real-time reallocation problem,

$$E = \frac{A}{4}\left(\sum_X \left(1-\sum_i V_{Xi}\right)^2+\sum_i \left(1-\sum_X V_{Xi}\right)^2\right)$$

$$+D\sum_X \sum_i C_X V_{Xi}\Theta(i-T_X) \ , \qquad (4.4)$$

the connection matrix and the biases can be found. A connection can be described using a general form

$$T_{Xi,Yj} = -A\delta_{XY}\,[1 - \delta_{ij}] \ -A\delta_{ij}\,[1 - \delta_{XY}] \ , \qquad (4.5)$$

where $T_{Xi,Yj}$ denotes the connection strength between the neuron on the Xth row and the ith column and the neuron on the Yth row and the jth column. The function δ_{ab} is 0 if $a \neq b$, and the function δ_{ab} is 1 if a=b. It is shown by equation 4.5 that each neuron has an inhibitory connection of "-A" with every other neuron which is in the same row or the same column. A bias can be described as

$$I_{Xi} = A - DC_X\Theta(i - T_X) \ , \qquad (4.6)$$

where I_{Xi} is the bias to the neuron on Xth row and jth column. The function $\Theta(x)$ is 1 if $x > 0$, and the function $\Theta(x)$ is 0 if

$x \leq 0$. The second part of the equation is an extra inhibition applied to those neurons which satisfy the condition $i > T_x$, which is essentially the same as the expression TTL - NT < 0, a shortfall condition. This means that the neural network uses extra inhibition to those neurons which would result in shortfalls if turned on. The amount of inhibition applied is determined by the priority.

4.4 Summary

In this chapter, the real-time reallocation problem has been formulated for neural network implementation. This neural network model is evaluated through computer simulations which are detailed in the chapter five.

CHAPTER FIVE

SIMULATIONS AND RESULTS

The neural network model for solving the real-time resource reallocation was developed in chapter four. In this chapter, the effectiveness of the neural network model is evaluated through computer simulations. The simulations were developed on an IBM-compatible personal computer. All simulation programs were written in Pascal, and are listed in the appendix. First, extensive simulations were performed using a discrete Hopfield model. The simulation results are presented in Section 5.1. Later, simulations using a Boltzmann machine were performed and the comparisons between the two methods are made in Section 5.2.

5.1 Hopfield Model

There are two objectives in the simulations of the Hopfield network. The first objective is to investigate the convergence characteristics of the Hopfield model. The second objective is to evaluate the performance of real-time reallocation using the present model through combined simulations on token bus networks.

5.1.1 Convergence Characteristics

In these simulations, a McCulloch-Pitts input/output function was used to simulate the artificial neurons. Since the constant coefficient A is the only parameter in the weight matrix (see equation 4.5), it was assumed to be 1 for convenience. With A=1, the relative importance of the hard constraint and the soft constraint in the energy function (see equation 4.4) can be determined solely by D, the coefficient of the cost term. The cost assignment for shortfalls was determined by an exponential function $y = 2^x$ (see table 4.2), where y represents the cost and x represent the priority.

Simulations were performed on two cases:

(1) case 1 - no shortfall in the optimal solution and

(2) case 2 - shortfall in the optimal solution.

For each case, 100 independent trials using different initial conditions and random update orders were performed when D assumed one of the six different values: 0.01, 0.1, 0.2, 0.3, 0.8, and 1.2. The results are presented in detail below.

Case 1: No shortfall in the optimal solution

Table 5.1 illustrates the initial information that must be present on the control channel of a token bus network with 10 stations. In this case, all the real-time data transfer requirements can be satisfied and the cost of the optimal solution is zero. The results produced by the neural network are converted to corresponding costs so that they can be

compared. A negative value "-5" is used to represent the cost of an invalid solution for convenience. The distributions of solution costs are plotted in histograms shown in figure 5.1(a) through (f). The results are summarized in table 5.2. First, it is shown in figure 5.1(a) through (f) that the Hopfield network failed to escape the local minima most of the time. Second, there are trade-offs in choosing the parameter D. When D was 0.01 (a small value), the neural network could always produce the valid solutions. However, the qualities of the valid solutions were not good since the range of the costs were from 0 to 30. As the parameter D was increased, more invalid solutions were produced, but the quality of the valid solutions was getting better because of the lower solution costs.

Table 5.1. Initial information for case 1.

Station ID	Priority	Time-to-live	Next Transfer Opportunity
A	1	11	1
B	1	4	2
C	2	3	3
D	0	7	4
E	0	11	5
F	4	2	6
G	2	11	7
H	0	11	8
I	3	2	9
J	2	11	10

Table 5.2. Summary for the experiments on case 1.

Figure	D	Number of Valid Solutions	Costs for Valid Solutions		Number of Optimal Solutions
			Min.	Max.	
5.1(a)	0.01	100	0	30	2
5.1(b)	0.1	56	0	15	2
5.1(c)	0.2	20	0	9	2
5.1(d)	0.3	6	0	3	3
5.1(e)	0.8	5	0	1	4
5.1(f)	1.2	4	0	0	4

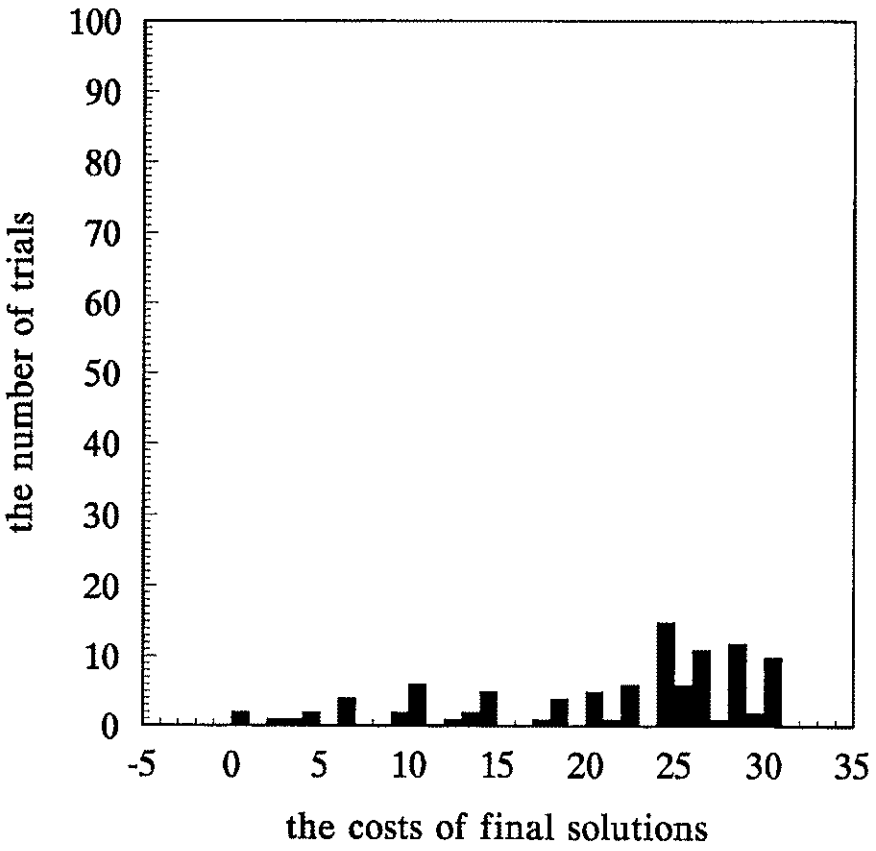


Figure 5.1(a). Histogram of the number of different solutions between cost 0 and 35 (D=0.01).

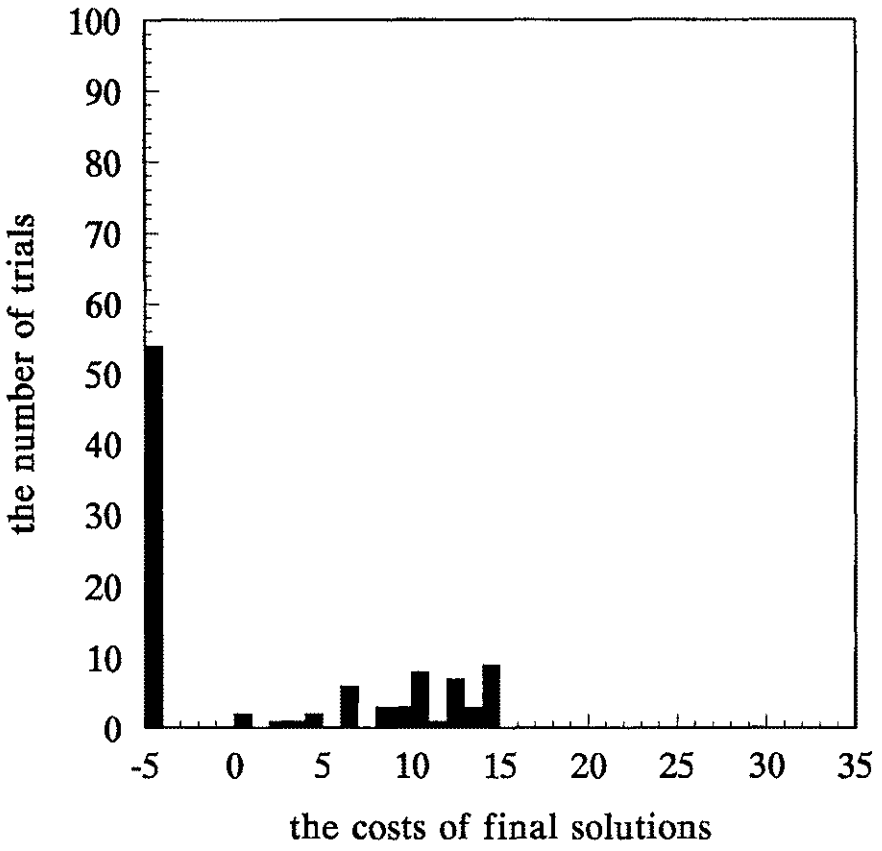


Figure 5.1(b). Histogram of the number of different solutions between cost 0 and 35 (D=0.1).

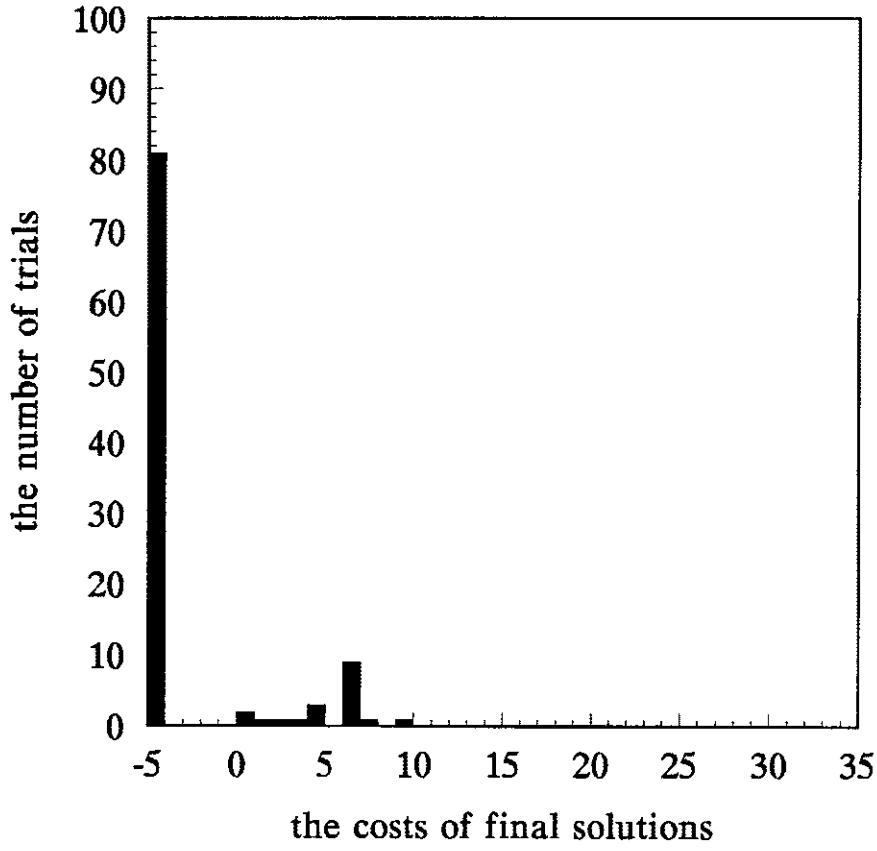


Figure 5.1(c). Histogram of the number of different solutions between cost 0 and 35 (D=0.2).

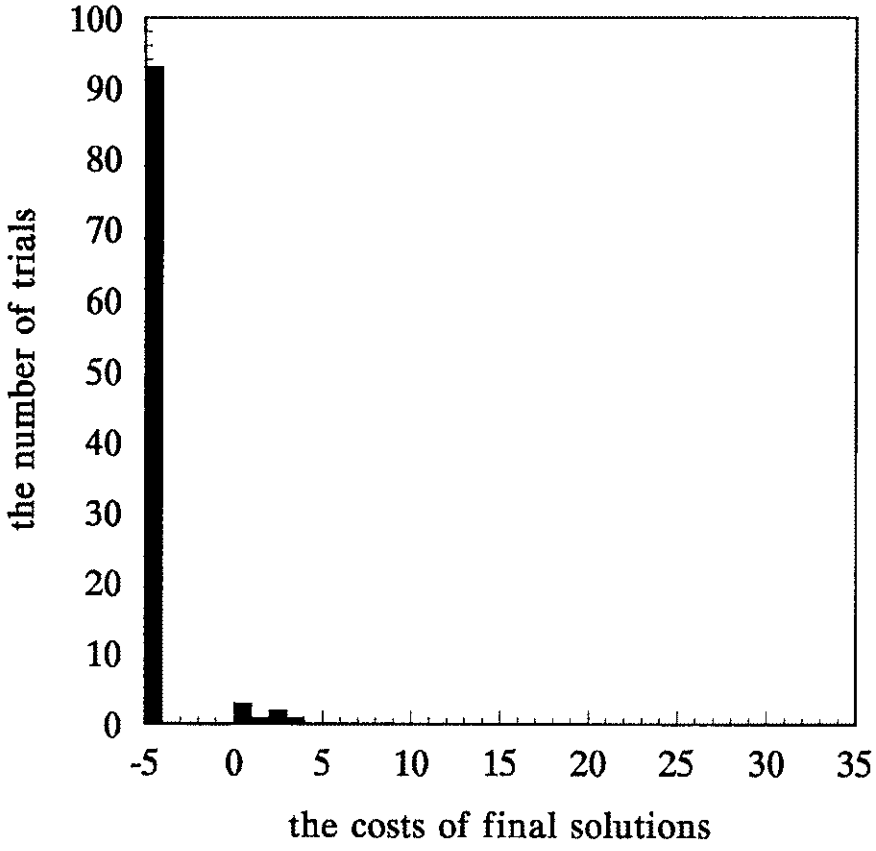


Figure 5.1(d). Histogram of the number of different solutions between cost 0 and 35 (D=0.3).

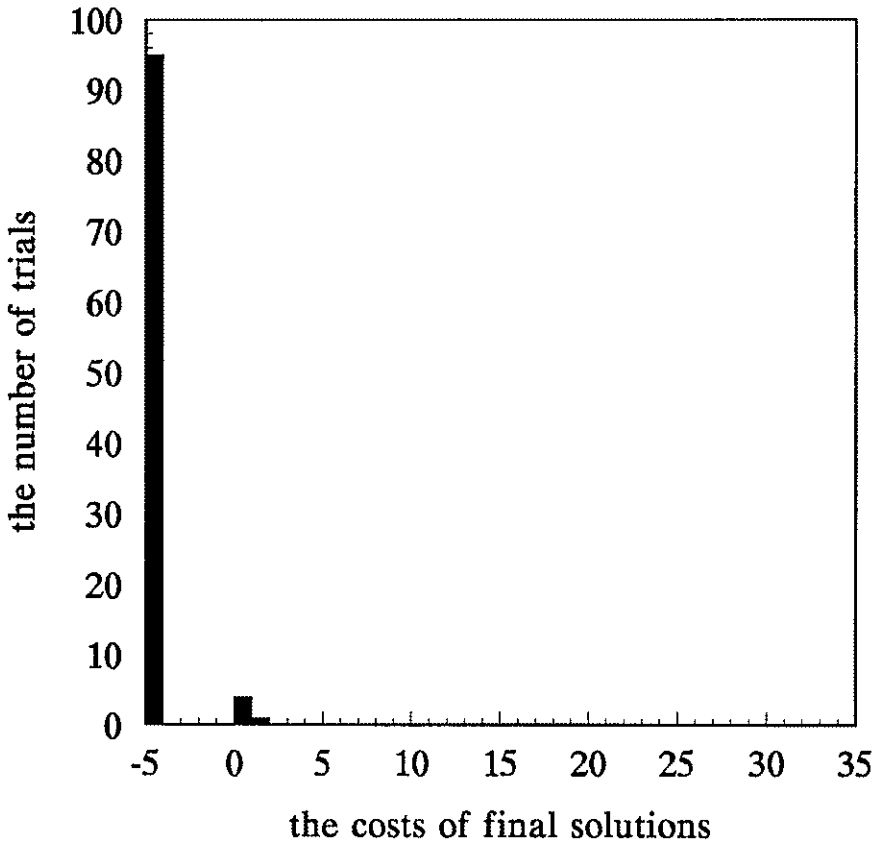


Figure 5.1(e). Histogram of the number of different solutions between cost 0 and 35 (D=0.8).

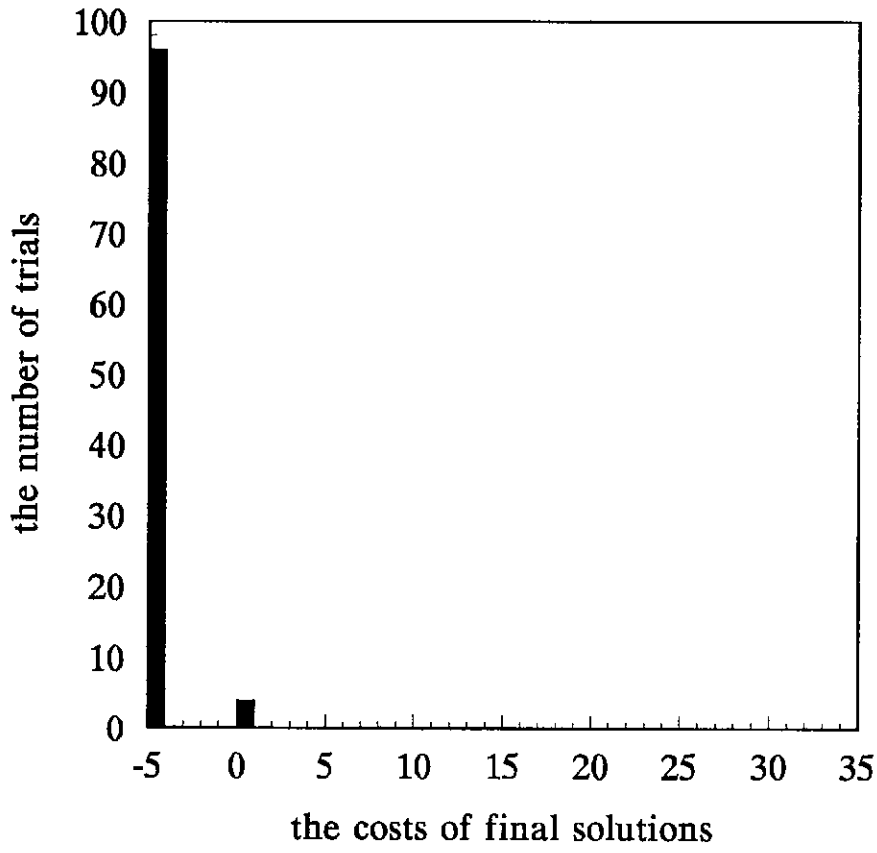


Figure 5.1(f). Histogram of the number of different solutions between cost 0 and 35 (D=1.2).

Case 2: Shortfall in the optimal solutions

Table 5.3 illustrates the initial information that must be present on the control channel of a token bus network with 10 stations. In this case, not all real-time data transfer requirement can be satisfied, and the cost of the optimal solution is 1. Similar to case 1, the distributions of solution costs are plotted in histograms shown in figure 5.2(a) through (f). The results are summarized in table 5.4. The trend shown in case 1 is also shown here. One difference is that no valid solution could be found when D was 1.2.

Table 5.3. Initial information for case 2.

Station ID	Priority	Time-to-live	Next Transfer Opportunity
A	1	11	1
B	1	4	2
C	2	3	3
D	0	2	4
E	0	11	5
F	4	2	6
G	2	11	7
H	0	11	8
I	3	2	9
J	2	11	10

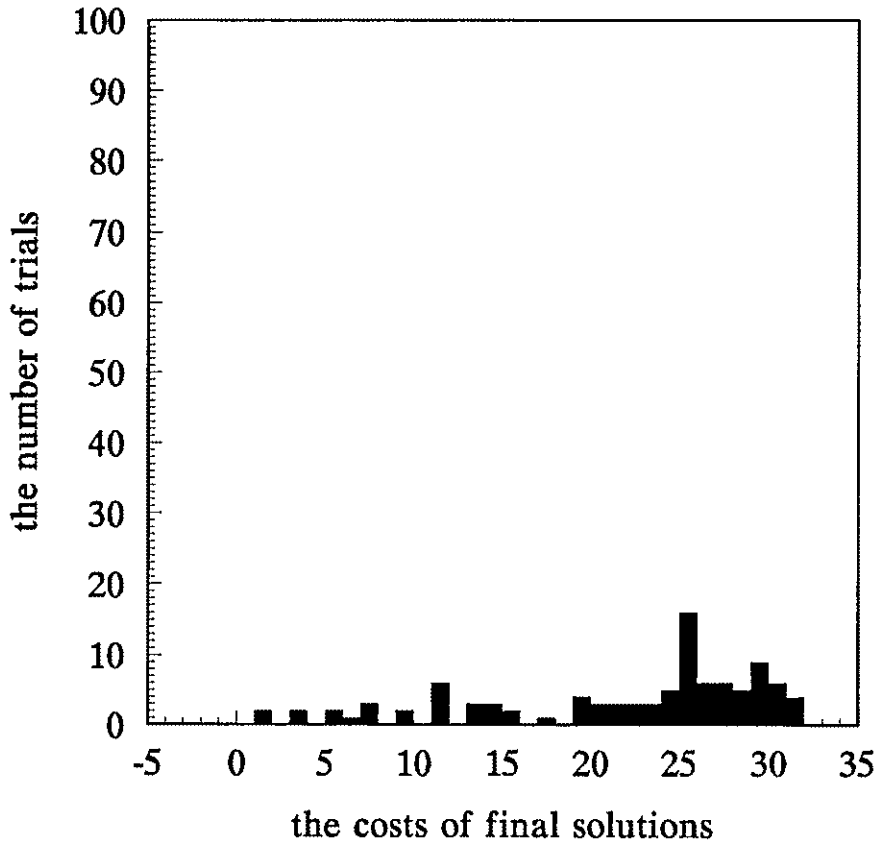


Figure 5.2(a). Histogram of the number of different solutions between cost 0 and 35 (D=0.01).

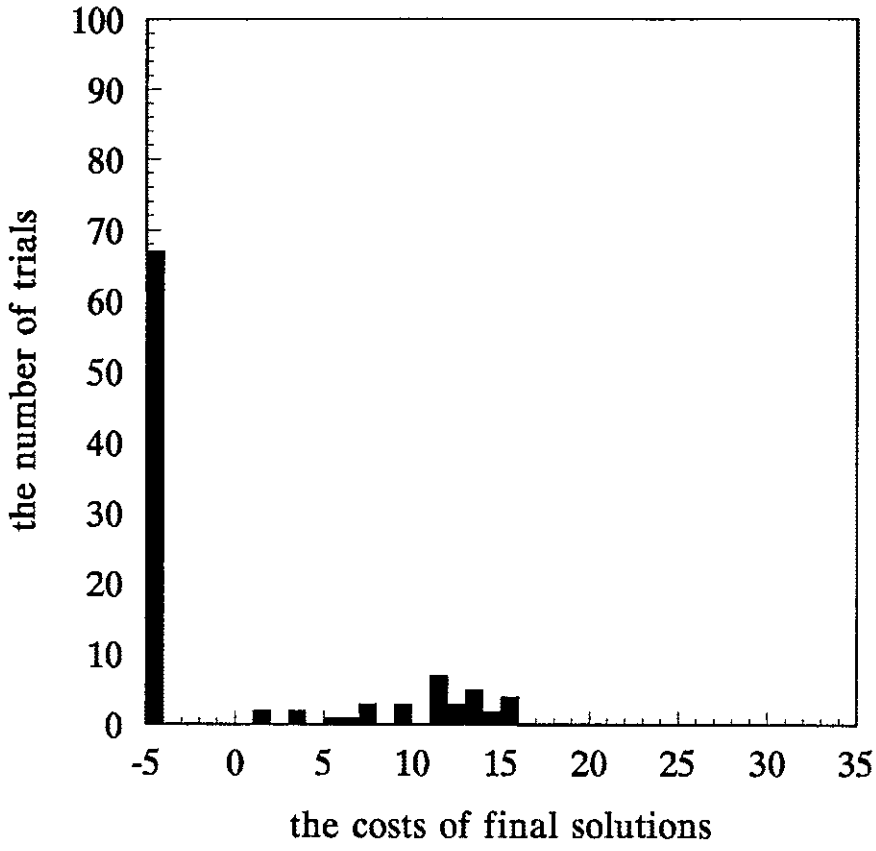


Figure 5.2(b). Histogram of the number of different solutions between cost 0 and 35 (D=0.1).

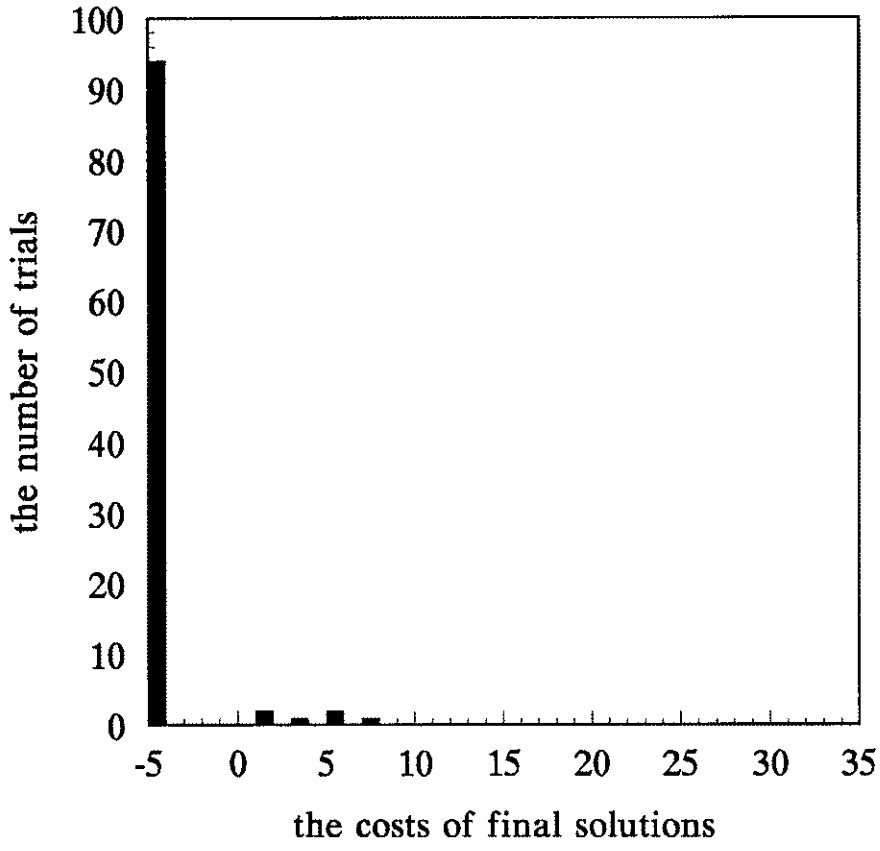


Figure 5.2(c). Histogram of the number of different solutions between cost 0 and 35 (D=0.2).

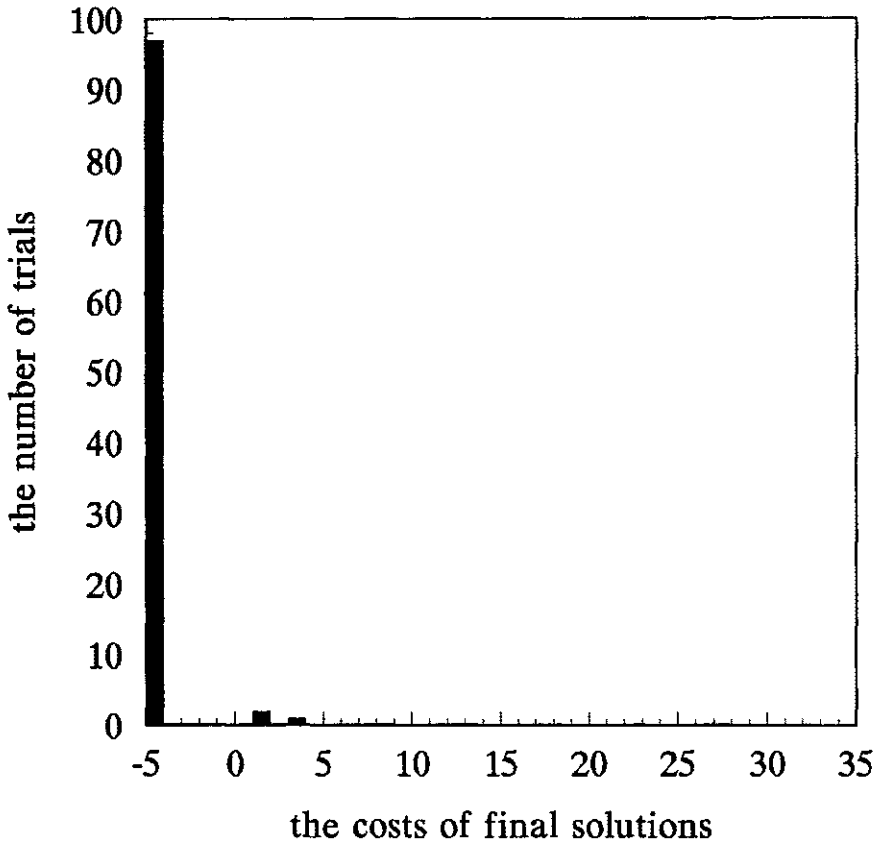


Figure 5.2(d). Histogram of the number of different solutions between cost 0 and 35 (D=0.3).

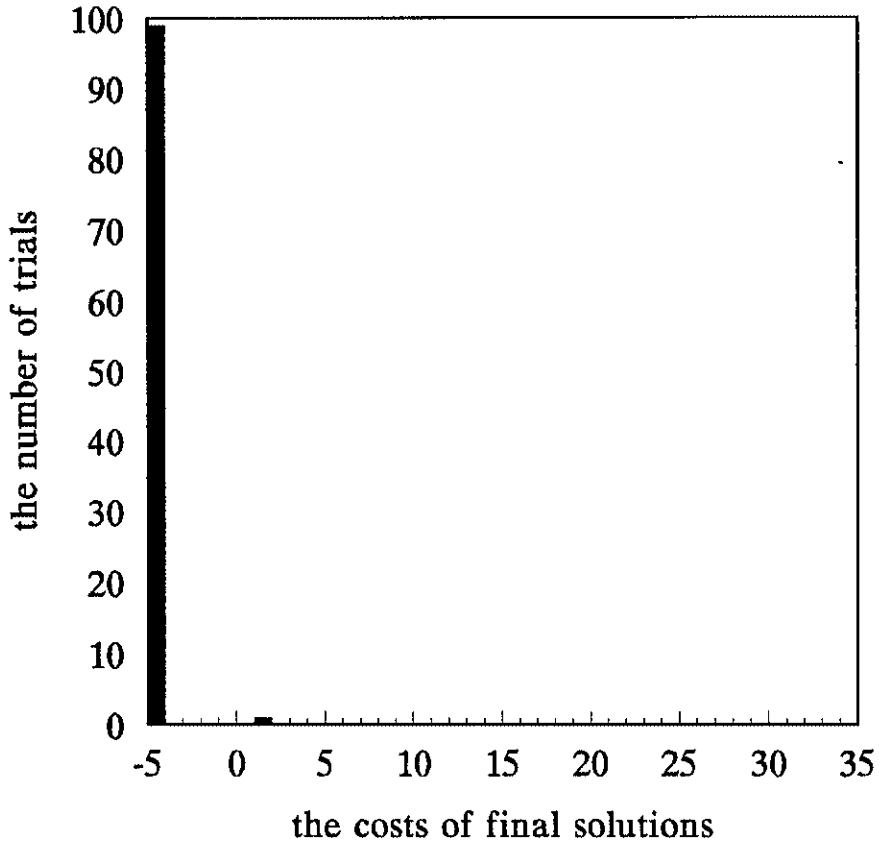


Figure 5.2(e). Histogram of the number of different solutions between cost 0 and 35 (D=0.8).

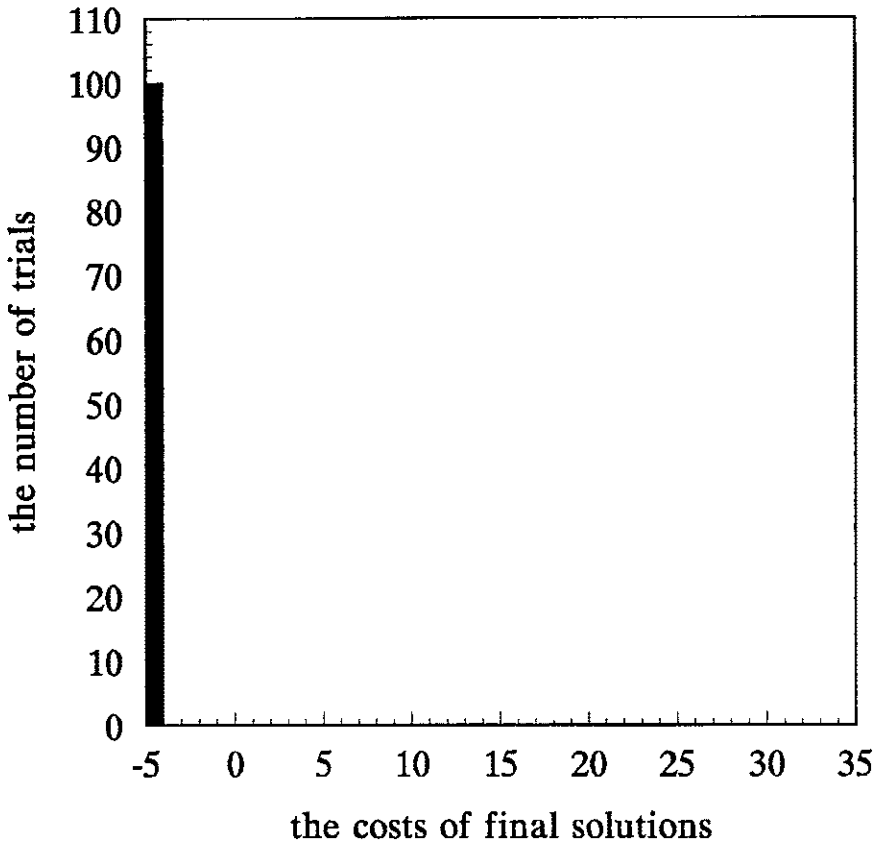


Figure 5.2(f). Histogram of the number of different solutions between cost 0 and 35 (D=1.2).

Table 5.4. Summary for the experiments on case 2.

Figure	D	Number of Valid Solutions	Costs for Valid Solutions		Number of Optimal Solutions
			Min.	Max.	
5.2(a)	0.01	100	1	31	2
5.2(b)	0.1	33	1	15	2
5.2(c)	0.2	6	1	7	2
5.2(d)	0.3	3	1	4	2
5.2(e)	0.8	1	1	1	1
5.2(f)	1.2	0	N/A*	N/A	0

* Not available.

Discussion

A bound for D can be determined so that all valid solutions are local minima [15]. In particular, a bound for D for the present problem can be determined by

$$A > D \cdot c^4 , \tag{5.1}$$

where A and D are the coefficients for the hard constraint and the soft constraint, respectively (see equation 4.4). The constant c is the base value for the general exponential function

$$y = c^x , \tag{5.2}$$

where y represents the cost and x represents the priority. Particularly, with A=1 and c=2 in our experiments, the condition becomes D < 0.0625, which explains why all 100 trials converged to valid solutions in both cases when D=0.01. Therefore, not all the valid solutions are local minima when D > 0.0625. According to chapter three, the net input to the neuron in X-th row and i-th column can be expressed as

$$net\ input = \sum T_{Xi,Yj} V_{Yj} + I_{Xi} \ . \tag{5.3}$$

Since

$$\sum T_{Xi,Yj} V_{Yj} \leq 0 \ , \tag{5.4}$$

we have

$$net\ input \leq I_{Xi} \ . \tag{5.5}$$

Therefore, a sufficient condition to turn a neuron off is

$$I_{Xi} \leq 0 \ . \tag{5.6}$$

It has been shown by equation 4.6 that an extra inhibition is applied to the neuron in the X-th row and the i-th column if a shortfall condition $i > T_x$ exists. It can be seen that the sufficient condition for this neuron to be turned off is

$$I_{Xi} = A - D \cdot C^p \leq 0 \ , \tag{5.7}$$

where C^p is the cost of shortfall with priority p according to equation 5.2.

With D=0.1, $I_{Xi} = 1 - (0.1) \cdot 2^p \leq 0$ when p = 4. Therefore, the network produced either invalid solutions or the valid solutions which eliminated the shortfalls of priority 4.

With D=0.2, $I_{xi} = 1 - (0.2)\cdot 2^p \leq 0$ when $p \geq 3$. The network produced either invalid solutions or the valid solutions which eliminated the shortfalls of priorities 4 and 3.

With D=0.3, $I_{xi} = 1 - (0.3)\cdot 2^p \leq 0$ when $p \geq 2$. The network produced either invalid solutions or the valid solutions which eliminated the shortfalls of priorities 4, 3, and 2.

With D=0.8, $I_{xi} = 1 - (0.8)\cdot 2^p \leq 0$ when $p \geq 1$. The network produced either invalid solutions or the valid solutions which eliminated the shortfalls of priorities 4, 3, 2, and 1.

With D=1.2, $I_{xi} = 1 - (1.2)\cdot 2^p \leq 0$ when $p \geq 0$. The network produced either invalid solutions or the valid solutions which eliminated the shortfalls of all priorities. Especially, no valid solution was found in case 2 because such a solution did not exist.

5.1.2 Simulations for token bus networks

The combined simulations of real-time reallocation using the Hopfield neural network model were performed for a 4-station token bus, a 6-station token bus, an 8-station token bus, and a 10-station token bus. 25 instances were generated for each token bus. Five priority levels of data transfer were assumed. The distribution of the data with different priority is presented in table 5.5.

Table 5.5. Selected distribution of data transfer priorities for use in simulations.

Priority	% of Data Transfers
4	2%
3	8%
2	20%
1	30%
0	40%

Note: Priority 4 is the highest priority and priority 0 is the lowest priority

Again, the cost assignment was the same as that of the previous experiments. The parameter A was assumed to be 1. Since any invalid solution was unacceptable, the parameter D was set to 0.01 (according to section 5.1.1), which was small enough to produce valid solutions at all times. 100

independent trials were performed for each of the 25 instances. The best, the worst and the average solutions were recorded. Then, the shortfalls for each priority were summed over the 25 best solutions. Similarly, the shortfalls for each priority were summed over the 25 worst solutions, and the shortfalls for each priority were summed over the 25 average solutions. The statistics were compared with that before any reallocation and that after using Pardue's reallocation method. The statistics for the 4-station token bus, the 6-station token bus, the 8-station token bus, and the 10-station token bus were plotted in figure 5.3, figure 5.4, figure 5.5, and figure 5.6, respectively. It is shown in all cases that the best solutions produced by neural network were close to those obtained using Pardue's method. However, the worst solutions were worse than those before using any reallocation. In general, the Hopfield network did not perform well.

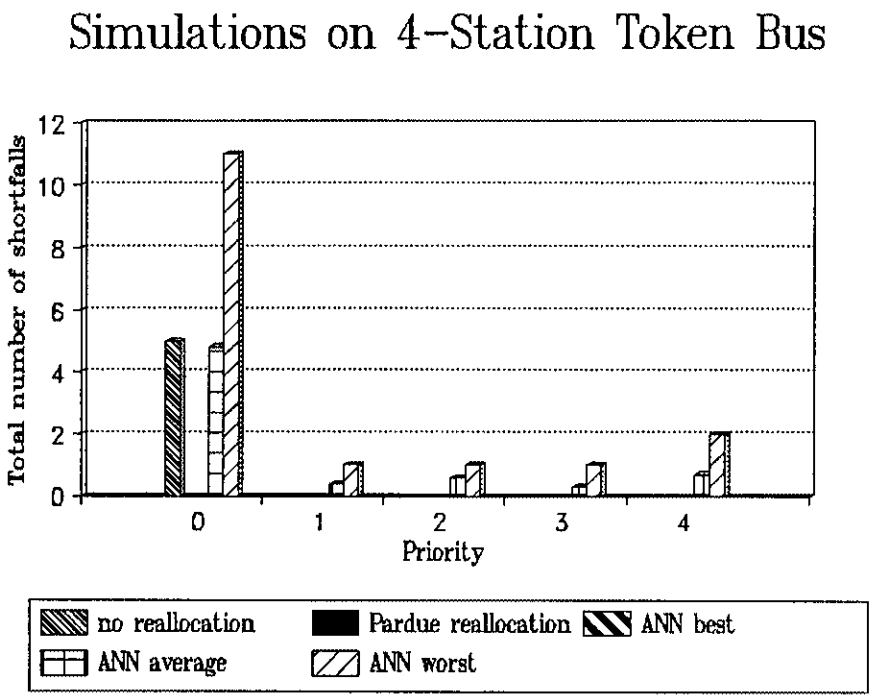


Figure 5.3. Summary statistics for the simulations on the 4-station token bus.

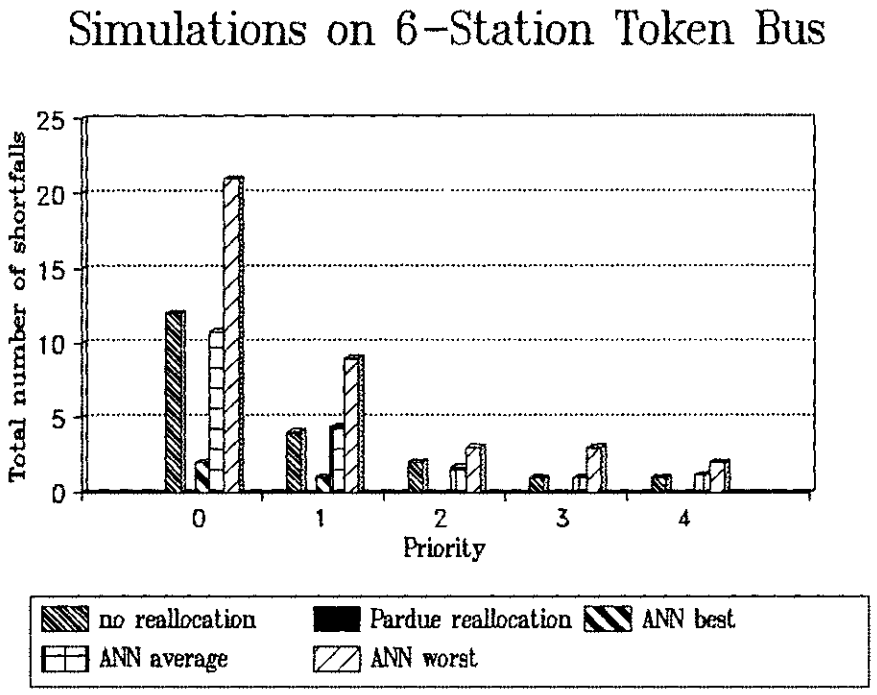


Figure 5.4. Summary statistics for the simulations on the 6-station token bus.

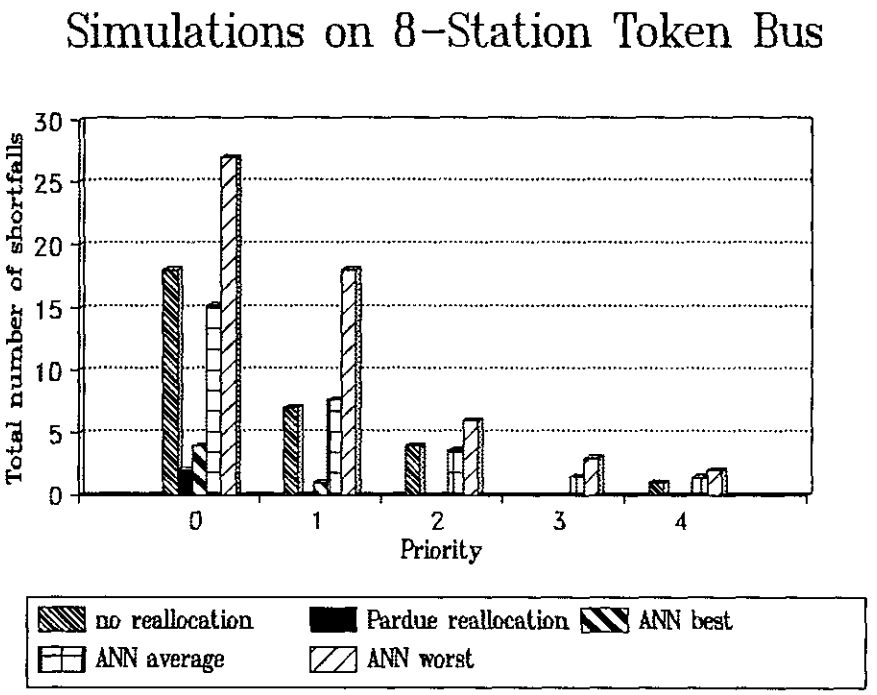


Figure 5.5. Summary statistics for the simulations on the 8-station token bus.

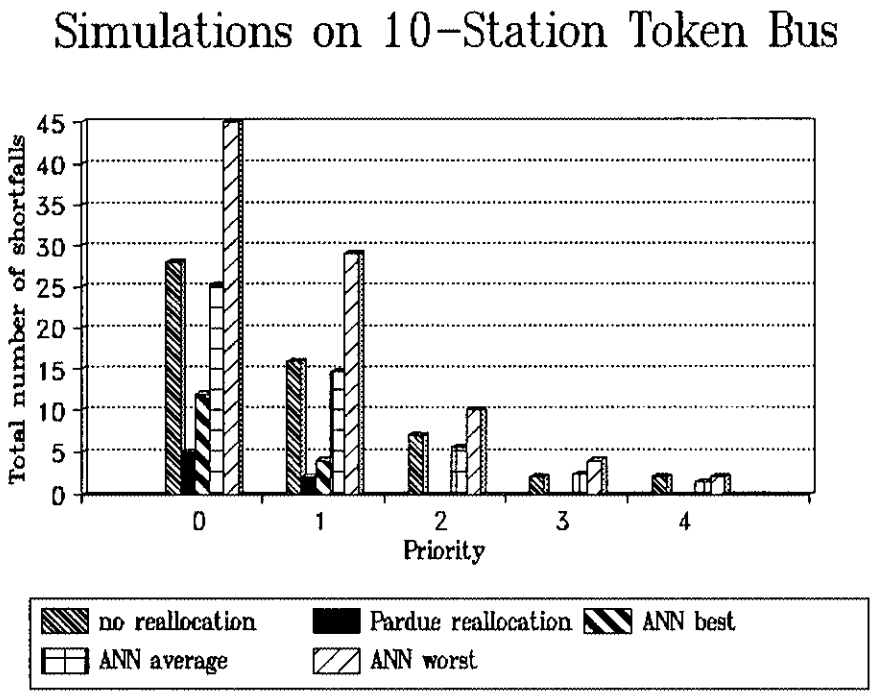


Figure 5.6. Summary statistics for the simulations on the 10-station token bus.

5.2 Boltzmann machine with Simulated Annealing

Simulations of the Boltzmann machine with simulated annealing were performed using the annealing schedule shown in table 5.6. It was assumed that the number of repetitions were sufficient for reaching equilibrium at each temperature. The parameter A was assumed to be 1. The parameter D was assumed to be 0.01, which is small enough for all the solutions to be valid. The two cases used in section 5.1.1 were used again here. 100 independent trials were performed for each case. The results are compared with those produced by the Hopfield network using histograms.

Table 5.6. The annealing schedule for the experiments.

Boltzmann machine with simulated annealing

Step 1. Set $T \leftarrow T_{max}$.

Step 2. Select a neural element i randomly and evaluate the energy difference δE_i according to equation 3.4. Set the neural element to 1 with probability p_i according to equation 3.5. Repeat this step k times.

Step 3. Set $T \leftarrow r \cdot T$. If $T \geq T_{min}$, go to step 2, otherwise stop.

Parameters

Symbol	Value	Description
T_{max}	10	Starting temperature
T_{min}	0.002	Minimum temperature
r	0.8	Temperature decay rate
k	2000	Time per temperature

The results were shown to be improved by the Boltzmann machine with simulated annealing in both cases. The comparison for case 1 is presented in figure 5.7. It is shown that 26% of the solutions produced by the Hopfield network had costs below 15. However, 71% of the solutions produced by the Boltzmann machine were in the same range. The comparison for case 2 is presented in figure 5.8. It is shown that 27% of the solutions produced by the Hopfield network had costs below 15. However, 66% of the solutions produced by the Boltzmann machine were in that range.

In the simulations of the Boltzmann machine with simulated annealing, not all solutions were global solutions. The problem may lie in the assumption that the repetitions in the experiments were sufficient for reaching the equilibrium at each temperature. It is expected that more repetitions at each temperature are needed for reaching a true equilibrium.

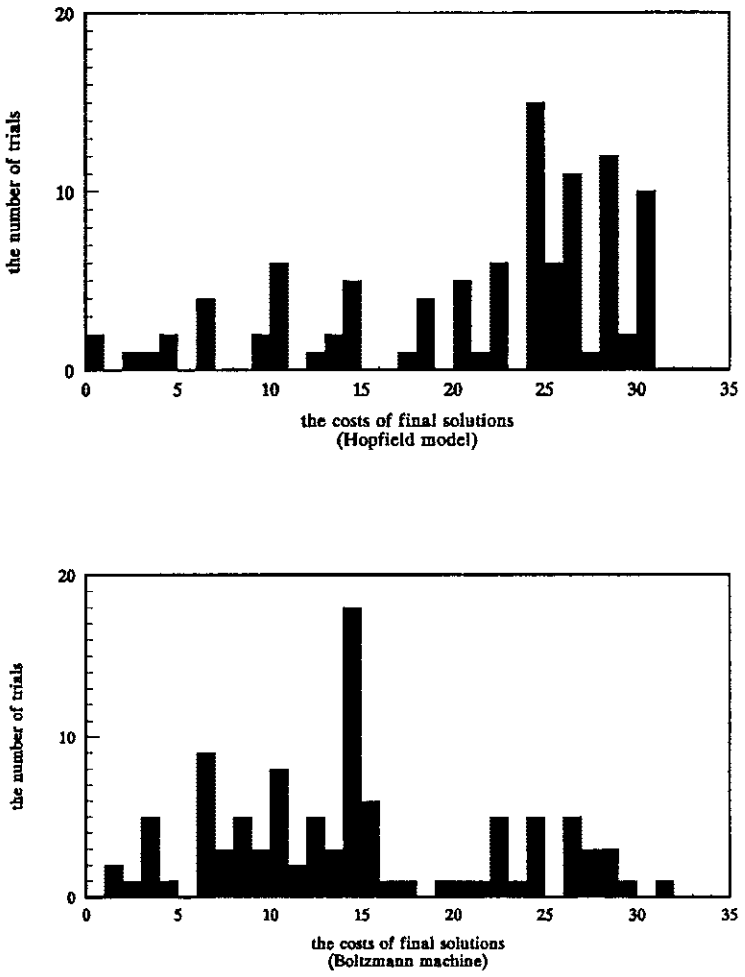


Figure 5.7. Histograms of the number of different solutions between cost 0 and 35 for case 1 (D=0.01).

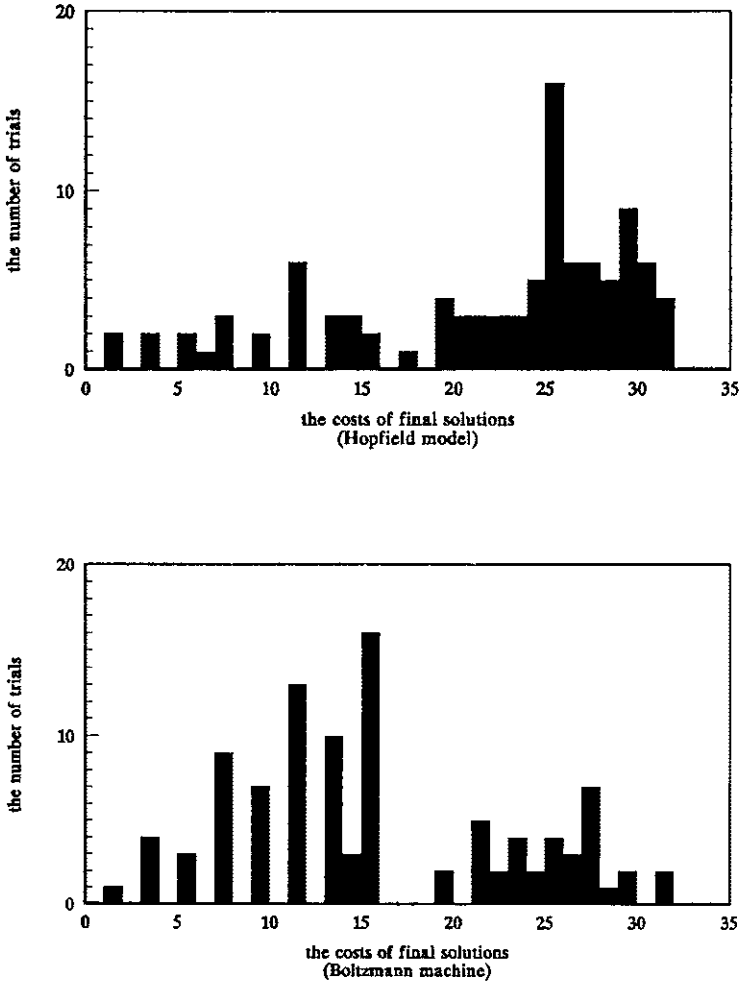


Figure 5.8. Histograms of the number of different solutions between cost 0 and 35 for case 2 (D=0.01).

CHAPTER SIX

CONCLUSIONS AND RECOMMENDATIONS

Real-time resource reallocation is a data communication scheduling technique. Past implementation of this technique was based on a sequential and exhaustive search algorithm, which could be executed in polynomial time. In this thesis, an implementation of the real-time reallocation technique has been developed using artificial neural networks which employ massive parallelism. This chapter concludes the thesis and recommends future research.

6.1 Conclusions

The real-time reallocation problem was first mapped into a two-dimensional matrix of neurons similar to Hopfield and Tank's approach to the traveling salesman problem. An energy function was then formulated in terms of the problem constraints. Specifically, a cost assignment scheme was introduced in order to incorporate the optimization criteria into the energy function. In this thesis, an exponential function $y=c^x$ $(c>1)$ was used for the cost assignment because this assignment could give enough penalty for any token-

passing sequence that might satisfy the real-time data transfer requirement of lower priority before that of higher priority.

Using the artificial neural network in real-time resource reallocation was first simulated using a discrete Hopfield network. It was shown that the discrete Hopfield network did not perform well. The primary cause was that the Hopfield network model does not provide for a technique for escaping any local minima of its associated energy function. The problem representation and the cost assignment might also have some effect on the bad performance of the neural network.

Experiments were later performed using the Boltzmann machine with simulated annealing, and it was shown that better solutions were obtained statistically. Since the repetitions k in the experimental assumption were not sufficient for reaching a true equilibrium at each temperature, many of the results were still local minima. However, any further increase in k will substantially increase the time for convergence and slow down the simulations. Since the resource reallocation problem must be solved in real-time, the Boltzmann machine is not an appropriate solution.

It can be seen that the disadvantages outweighed the advantages in both the discrete Hopfield model and the Boltzmann machine. Therefore, the neural network approach to the real-time reallocation problem as developed in this thesis is not a good approach. This may be due to the specific

problem representation and cost assignment function used in this research.

6.2 Recommendations for Future Research

The future research effort can be made in several areas. First, it was mentioned in chapter four that representation is important in mapping the real-time reallocation problem into the neural network. Therefore, effort can be made in searching for better representations for the present problem. Second, an exponential function was used as a cost assignment function throughout this thesis. Effect of the different cost assignment on the solution quality can also be studied. Third, it was claimed by Hopfield [12] that the neural network using the analog implementation could produce much better solutions than those of the discrete implementation. Therefore, the analog implementation of the Hopfield neural network model can be investigated. Fourth, the possibility of speeding up the simulated annealing process using parallel Boltzmann machines [1] or Cauchy machines [17] can be investigated. Finally, genetic algorithms based on the mechanics of natural selection and natural genetics [6] can also be investigated for solving the real-time reallocation problem.

REFERENCES

[1] Aarts, Emile and Jan Korst. *Simulated Annealing and Boltzmann Machines*, John Wiley & Sons: New York, 1989.

[2] Cohen, M.A., and S.G. Grossberg. "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks." *IEEE Transaction on Systems, Man and Cybernetics*," 13:815-26, 1983.

[3] Conway, R.W., W.L. Maxwell and L.W. Miller. *Theory of Scheduling*, Addison-Wesley Publishing Company: Reading, Massachusetts, 1967.

[4] Foo, Y.P.S. and Y. Takefuji. "Stochastic Neural Networks for Solving Job-Shop Scheduling," *IEEE International Conference on Neural Networks*, San Diego, California, July 1988.

[5] Giarratano, J., and G. Riley. *Expert Systems*, PWS-KENT Publishing Company: Boston, Massachusetts, 1989.

[6] Goldberg, D.E. *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc.: Reading, Massachusetts, 1989.

[7] Hertz, John, A. Krogh, and R.G. Palmer. *Introduction to the Theory of Neural Computation*, Addison-Wesley Publishing Company: Redwood City, California, 1991.

[8] Hinton and Sejnowski. "Optimal perceptual inference," *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, Washington, D.C., 448-453, 1983.

[9] Hopfield, J.J. "Neural networks and physical systems with emergent collective computational abilities." *Proceedings of the National Academy of Science*. 79:2554-58, 1982.

[10] Hopfield, J.J. "Neurons with graded response have collective computational properties like those of two-state neurons," *Proceedings of the National Academy of Sciences USA*, Vol.81,1984, pp. 3088-3092.

[11] Hopfield, J.J. and D.W. Tank. "Computing with Neural Circuits: A Model," *Science*, Vol. 233, Aug. 1986.

[12] Hopfield, J.J. and D.W. Tank. "Neural Computation of Decisions in Optimization Problems," *Biological Cybernetics*, Vol. 52, pp. 141-152, 1985.

[13] IEEE Computer Society. *802.4 Token-Passing Bus*, The Institute of Electrical and Electronics Engineers, Inc: New York, 1985.

[14] Kirkpatrick, S., C.D. Gelatt Jr., and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, No. 4598, May 1983.

[15] Livingston, D.L. Private communication.

[16] Pardue, M.D. *An Intelligent Interface Node for Use in CIM*, Ph.D. Dissertation, George Mason University: Fairfax, VA, August, 1988.

[17] Szu, Harold and Ralph Hartley. "Fast Simulated Annealing," *Physics Letters A*, Vol. 122, No. 3,4, pp. 157-162, 1987.

[18] Tank, D.W. and J.J. Hopfield. "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit and Linear Programming Circuit," *IEEE Transaction on Circuits and Systems*, Vol. CAS-33, No. 5, pp. 533-541, 1986.

[19] Wasserman, Philip D. *Neural Computing*, Van Nostrand Reinhold: New York, 1989.

APPENDIX A

SIMULATION PROGRAM FOR THE HOPFIELD NETWORK

```
program reallocation_with_hopfield_net(input, output);

{*************************************************************}
{* This program simulates the real-time reallocation using   *}
{* the deterministic Hopfield network                        *}
{*************************************************************}

const
   n = 10;    (* the number of nodes on token bus network *)
   a = 1;     (* gain constant for hard constraint *)
   d = 0.01;  (* constant coefficient for the cost term *)
   a0= 10;    (* base for the exponential function *)
   b0= 0;

type
   nodes    = 1..n;
   weight   = array [nodes, nodes, nodes, nodes] of real;
   matrixi  = array [nodes, nodes] of integer;
   matrixr  = array [nodes, nodes] of real;
   vector   = array [nodes] of real;

var
   u        : matrixr; (* analog voltage of a single node *)
   v        : matrixi; (* neuron states "1" or "0" *)
   t        : weight;  (* weight between two nodes *)
   bias     : matrixr; (* input bias to a single node *)
   ttl      : vector;  (* time-to-live *)
   pri      : vector;  (* current priority of each node *)
   cost     : vector;  (* cost per shortfall *)
   delta_e  : real;    (* energy difference *)
   x,i,y,j  : integer; (* row and column indexes *)
   cycles   : ineger;

function cost_function(x : real): real;
begin
   cost_function := exp(x * ln(a0)) + b0;
end;

procedure init_timing_info;
var
```

```
   x1, i1   : integer;
   p        : real;
begin
   for x1 := 1 to n do
      begin
         writeln('input ttl[', x1, ']:');
         readln(ttl[x1]);
      end;
   for x1 := 1 to n do
      begin
         writeln('input pri[', x1, ']:');
         readln(pri[x1]);
      end;
   for x1 := 1 to n do
      cost[x1] := cost_function(pri[x1]);
end;

procedure init_neuronet;
var
   p : real;
begin
   for x := 1 to n do
      for i := 1 to n do
         begin
            p := random(100);
            if (p>=50)
               then
                  v[x,i] := 1
               else
                  v[x,i] := 0
         end;
end;

function delta(a : integer; b : integer) : integer;
begin
   if (a = b)
      then
         delta := 1
      else
         delta := 0
end;

function theta(x : real) : integer;
begin
   if (x > 0)
      then
         theta := 1
      else
         theta := 0
end;

procedure weights_biases;
```

```
begin
   for x := 1 to n do
      for i := 1 to n do
         for y := 1 to n do
            for j := 1 to n do
               begin
                  if ((x <> y) or (i <> j))
                     then
                                          t [ x , i , y , j ]    : =
-a*delta(x,y)*(1-delta(i,j))-a*delta(i,j)*(1- delta(x,y))
                     else
                        t[x,i,y,j] := 0
               end;
   for x := 1 to n do
      for i := 1 to n do
         begin
          bias[x,i] := a - d * cost[x] * theta(i - ttl[x])
         end;
end;

procedure energy_difference;
var
   net_input : real;
begin
   net_input := 0;
   for y := 1 to n do
      for j := 1 to n do
         begin
            net_input := net_input + t[x,i,y,j] * v[y,j]
         end;
   net_input := net_input + bias[x,i];
   delta_e := net_input
end;

procedure deterministic_update;
var
   period : integer;
begin
    for period := 1 to 1000 do
       begin
            x := random(n) + 1;
            i := random(n) + 1;
            energy_difference;
            if delta_e > 0
               then
                  v[x,i] := 1
               else
                  v[x,i] := 0;
       end
end;
```

```
begin {main}
   init_timing_info;
   init_neuronet;
   deterministic_update;
end.
```

APPENDIX B

SIMULATION PROGRAM FOR THE BOLTZMANN MACHINE

```
program reallocation_with_Boltzmann(input, output);

{*************************************************************}
{*  This program simulates the real-time reallocation using  *}
{*  the Boltzmann machine with simulated annealing           *}
{*************************************************************}

const
   n = 10;   (* number of node on token bus network *)
   a = 1;    (* constant coefficient for row inhibition *)
   d = 0.01;(* constant coefficient for the soft constraint *)

   (* The following are the parameters for the simulated
      annealing *)
   T_max       = 10;        (* starting temperature *)
   T_min       = 0.002;     (* minimum temperature *)
   r           = 0.8;       (* temperature decay rate *)
   k           = 2000;        (* time per temperature *)

type
   matrix      = array [1..n, 1..n] of integer;
   vector      = array [1..n] of integer;
var
   v          : matrix;     (* neuron states *)
   (* The following are some inputs to the neural network *)
   ttl        : vector;     (* time-to-live *)
   pri        : vector;     (* priority of each node *)
   cost       : vector;     (* cost per shortfall *)

   energy     : real;      (* total energy *)
   e_hard1    : real;      (* energy for row inhibition *)
   e_hard2    : real;      (* energy for column inhibition *)
   e_soft     : real;      (* energy for soft constraint *)
   e_previous: real;       (* previous lowest energy *)

   temp       : real;      (* temperature *)

procedure init_timing;
var
   x : integer;
```

```
begin
   for x := 1 to n do
      begin
         writeln('Input ttl [',x, ']:');
         readln(ttl[x]);
      end;

   for x := 1 to n do
      begin
         writeln('Input pri [',x, ']:');
         readln(pri[x]);
      end;
   for x := 1 to n do
      begin
         if (pri[x] = 4)
            then
               cost[x] := 16;
         if (pri[x] = 3)
            then
               cost[x] := 8;
         if (pri[x] = 2)
            then
               cost[x] := 4;
         if (pri[x] = 1)
            then
               cost[x] := 2;
         if (pri[x] = 0)
            then
               cost[x] := 1
     end
end;

procedure init_neural_network;
var
   x, i : integer;
   p : real;
begin
   for x := 1 to n do
      for i := 1 to n do
         begin
            p := random(100);
            if (p > 50)
               then
                  v[x, i] := 1
               else
                  v[x, i] := 0;
         end;
   energy := 0;
   e_previous := 0;
end;

function theta(x : integer): integer;
```

```
(* this is a threshold function *)
begin
   if (x>0)
      then
         theta := 1
      else
         theta := 0
end;

procedure evaluate;
var
   x, i : integer;
   total_row : integer;
   total_col : integer;
   a1, a2 : integer;
   b1, b2 : integer;

begin
   e_hard1 := 0;
   e_hard2 := 0;
   e_soft  := 0;
   a1 := 0;
   a2 := 0;

   (* evaluate e_hard1 *)
   for x := 1 to n do
      begin
         total_row := 0;
         for i := 1 to n do
            total_row := total_row + v[x, i];
         a1 := 1 - total_row;
         a1 := a1 * a1;
         e_hard1 := e_hard1 + a1;
      end;

   (* evaluate e_hard2 *)
   for i := 1 to n do
      begin
         total_col := 0;
         for x := 1 to n do
            total_col := total_col + v[x, i];
         a2 := 1 - total_col;
         a2 := a2 * a2;
         e_hard2 := e_hard2 + a2;
      end;

   (* evaluate e_soft *)
   for x := 1 to n do
      for i := 1 to n do
         e_soft := e_soft + cost[x] * v[x,i] * theta(i -
ttl[x]);
   e_soft := e_soft * d;
```

```
   (* evaluate total energy *)
   energy := (e_hard1 + e_hard2) * a / 4 + e_soft;
end;

procedure Boltzmann_machine;
var
   x, i        : integer;
   repetition  : integer;
   rand        : real;
   p           : real;
   delta_e     : real;
   times       : integer;
   ratio       : real;
begin
   temp := T_max;
   repeat
      for repetition := 1 to k do
         begin
            x := random(n) + 1;
            i := random(n) + 1;
            v[x, i] := 1 - v[x, i];
            evaluate;
            delta_e := energy - e_previous;
            ratio := delta_e/temp;
            if (ratio < 10)
               then
                  p := 1/(1 + exp(ratio))
               else
                  p := 0;
            rand := random(100)/100;
            if (rand < p)
               then
                  begin
                     v[x, i] := v[x, i];
                     e_previous := energy
                  end
               else
                  v[x, i] := 1 - v[x, i] ;
         end;
      temp := temp * r;
   until (temp < T_min)
end;

{main lines}
begin
   init_timing;
   init_neural_network;
   evaluate;
   e_previous := energy;
   Boltzmann_machine;
end.
```