An Artificial Neural Approach to the Decomposition Problem

Chandrashekar L. Masti
Old Dominion University

AN ARTIFICIAL NEURAL APPROACH TO THE DECOMPOSITION PROBLEM

by

Chandrashekar L. Masti
B. E., June 1987, Birla Institute of Technology, India

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

OLD DOMINION UNIVERSITY
June, 1990

Approved by:

David L. Livingston (Director)

Mark Pardue

Stephen A. Zahorian

ABSTRACT

AN ARTIFICIAL NEURAL APPROACH TO THE DECOMPOSITION PROBLEM

Chandrashekar L. Masti

The goal of this thesis is to develop an artificial neural approach toward addressing the intractability involved with the decomposition problem. The search for the lattice of substitution property (s. p.) partitions essential to decompositions is cast into the framework of constraint satisfaction. An artificial neural network is developed to provide solutions by performing optimization of a mathematically derived objective function over the problem space. The issue of transitivity is verified to belong to a class of problems beyond the scope of solvability for conventional quadratic-order constraint satisfaction neural networks. A theorem is stated and proved establishing that third-order correlations must be extracted by a neural network to generate s. p. partitions. A formalized method for the construction of constraint satisfaction neural networks is presented by exploiting abstract laws from relational and Boolean algebras. It is shown that a third-order Hopfield network fails to solve the s. p. partition problem, while impressive results are obtained by using a Boltzmann machine employing simulated annealing. Convergence to global solution states in large-sized problem domains have been obtained at fast rates by manipulation of the degenerate ground state characteristic of the network objective function.

ACKNOWLEDGEMENTS

I wish to express my gratitude to my advisor Dr. David L. Livingston for all his persistence, guidance and encouragement throughout this research work. His valuable contribution through numerous stimulating discussions backed by his profound insight into the subject matter is gratefully acknowledged. I would also like to express my sincere thanks to Dr. John W. Stoughton for all his continued and invaluable support through the course of my graduate study. Particular thanks are due to Dr. Vishnu K. Lakdawala for his understanding and cooperation towards the final phase of this work. I value my interaction and discussions with my friend and colleague Mr. Gursel Serpen as a cherishable and enjoyable experience. Thanks are due to the members of my thesis committee Dr. Stephen A. Zahorian and Dr. Mark Pardue, for taking their time to review my thesis. The financial support of this work by the National Aeronautics and Space Administration under Grant No. NAG-1-962 is sincerely acknowledged.

Finally I would like to take this as an opportunity to thank my parents for supporting my decision to attend Graduate school and pursue advanced studies.

TABLE OF CONTENTS

LIST OF TABLES

TABLE PAGE

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Decomposition techniques are extensively used in a variety of applications. In design engineering, top-down methods based on decomposition are emphasized for system synthesis. Humans regularly employ decomposition in problem solving and task planning. We are proficient in identifying underlying structures and dependencies within a given task description and are able to extemporaneously formulate a methodology for decomposing the initial task into sub-tasks of relatively lesser complexity.

Researchers from fields such as computer science, mathematics, psychology and engineering are currently examining the mechanics underlying the success of human intelligence in achieving contextually useful decompositions. Success has generally been limited to situations where an a-priori description of the problem space exists and unpredictable changes are known not to occur. The relatively simple common-sense prowess for decomposition by average human intelligence continues to remain beyond the reach of the most complex systems in existence today. Current artificial intelligence technology comprised of symbolic representation and heuristic search, implemented on serial computers does not appear to hold any sound promise of closing the gap between human performance and machine intelligence in the near future [1]. If a method exists for emulating human performance, it is our opinion that intelligent tasks such as decomposition must be addressed from a perspective that has a close connection with biological neural systems.

Artificial neural models have been developed and studied to contribute to a better understanding of biological neural systems and to provide a basis for engineering systems capable of achieving human-like performance [2]. These models, called artificial neural networks, are essentially composed of large numbers of nonlinear computational elements operating in parallel and have a high degree of connectivity reminiscent of biological neural networks. The interconnections between the computational elements of these nets are weighted depending on the specifics involved in their computations. In contrast to performing a program of instructions sequentially as in the traditional von Neumann computer, artificial neural nets are capable of verifying competing hypotheses simultaneously using their inherently massive parallelism. Thus, artificial neural net models are believed to have great potential in areas where many hypotheses are explored in parallel and high computation rates are required.

One example of the exploitation of neural networks in conjunction with decomposition is automatic task planning [3], [4]. High-order neural nets are used to plan decomposed tasks. In the problem of developing task structures, a significant portion of the effort involved consists of the combinatorialy explosive problem of finding the elements of the general structure from which the decompositions are obtained.

The problem of decomposition has been approached from an algebraic perspective that essentially relies on partition algebra. Hartmanis and Stearns [5] pioneered the use of partition algebra for analyzing the structure of sequential machines. Partitions generated over state machines determine the possibility of decompositions and provide an understanding of their structure. Currently existing techniques for obtaining machine decompositions are based on performing an exhaustive sequential search into the machine's transition characteristics governed by a given input environment. This process is known to become intractable with growth in problem size and consequentially imposes intemperate demands on computation resources.

The recent resurgence of artificial neural nets has drawn keen interest from groups of researchers who have observed that these models demonstrate capabilities in addressing computationaly "hard" problems including: the travelling salesperson problem [6], the graph partitioning problem [7] and the N-queens problem [8]. This new development offers a clear motivation for furthering a serious exploration into the potential of a neural approach to addressing the NP-hard machine decomposition problem.

1.1 Objective

The goal of this thesis is to explore the possibility of a neural approach toward a solution to the intractability involved in the partition search problem. It is suspected that a neural solution is a strong candidate for successfully addressing the machine decomposition problem in an efficient manner through the use of massive parallelism promised by forthcoming VLSI technology.

We cast the partition search problem into the framework of constraint satisfaction, our aim being to implement the solution of a constraint satisfaction problem using artificial neural networks. The technique incorporates the formulation of an "objective" or "energy" function serving as a measure of the degree of violation or compliance of plausible solutions to derived constraints. Properties that a partition is required to satisfy based on the laws of relational and Boolean algebras are incorporated into respectively required constraint terms in the network energy function. We also attempt to systematize the derivation procedure for the network energy function and provide a mathematical basis in an approach that has hitherto remained generally heuristic.

1.2 Thesis Structure

Chapter I introduces the main problem addressed in this work, states the objective and outlines the structure of the material presented in this thesis.

Chapter II presents a background view of past approaches to the decomposition problem and at the same time surveys the basic theoretical foundations on which the current work in this thesis is developed.

The theoretical framework underlying the formulation of the application of neural networks to the machine decomposition problem is detailed in chapter III. The issue of a proper representation scheme for mapping the decomposition problem into the neural domain is addressed from a perspective that has a mathematical basis in relational algebras. A theorem which algebraically shows that the interconnections in the network for solving the decomposition problem must be of third order is stated and proved.

With the theoretical basis provided for the network from chapter III, the final model is developed and tested via simulations in chapter IV. The results of simulations are discussed and necessary modifications to the strategies involved in the computation mechanism are also presented.

Chapter V summarizes the results and presents the conclusions that are drawn from this research endeavor. Research topics for further exploration and the scope for enhancements are also discussed in this chapter.

CHAPTER II

BACKGROUND

2.1 Overview

The predominant strategy behind decomposition techniques in existence today is exhaustive search executed in a sequential manner. Partition algebra provides the mathematical basis for addressing the problem of decompositions. Hartmanis and Stearns [5] exploited the theory of partition algebra and established a formalism for analyzing the structure of a suitably developed state machine and a computation of partitions over its state space. Partitions provide an understanding and evaluation for the feasibility of decompositions.

A basic approach for decomposing a system is comprised of the following steps. First, an algebraic description of the system is formulated. Using all known or pre-specified constraints, the complete state space of the system is then generated. The state space of the problem can be represented by a state table and thus a state machine with primitive operations serving as inputs. The state machine serves as a tool to find the elements of the basic structure from which the type of the decomposition: series, parallel or complex may be determined.

The manner in which a state machine is realized from a set of smaller component machines as well as the functional dependencies of its state and output variables defines the structure of the state machine. Performing a constrained parallel search into the transition properties of the state machine can generate partitions on the machine's states. The decomposability of the state machine is then determined by the properties exhibited by these partitions.

Constraint satisfaction neural networks naturally lend themselves to addressing problems of this kind due to their inherently parallel computation strategies for performing relaxation searches in a multiple constraint problem domain. Thus, a brief insight into the class of neural nets capable of addressing constraint satisfaction problems is now presented.

2.2 Constraint Satisfaction Neural Networks

General Concepts

Artificial neural nets have been investigated for a variety of applications. NP-complete optimization problems such as the Travelling Salesperson Problem

[9], signal decision, Analog to Digital conversion and linear programming circuit problems [10], and others have been addressed using constraint satisfaction neural nets. Phonemic feature extraction for speech-synthesis [11] and classification problems requiring arbitrarily complex decision regions [12] have been explored using neural nets. Vision, speech, language and motor control [13], difficult learning control problems like the balancing of a pole hinged to a movable cart [14] and bibliographic retrieval from large United States national information databases [15] have found good acceptance for neural net approaches. One of the motivations for these investigations stems from the fact that existing systems fall short of matching human performance in effecting high-speed searches in large-sized information databases. A simple example is the relative ease with which humans can generate vivid recapitulations of objects and images in response to weak, externally supplied stimuli. Our capabilities for locating isolated and specific, context-sensitive detail from an increasing reserve of knowledge within our brains continues to remain beyond the match of current best systems.

Artificial neural net models are also known as parallel distributed processing (PDP) or connectionist models [16]. The fundamental assumption for these models is that the information processing takes place through interactions between a large number of simple processing elements called units or neurons, where each can send either excitatory or inhibitory signals to other units in the system. In the application of neural nets to constraint satisfaction the individual neurons are themselves used to represent hypotheses. The activations of the neurons are analogous to the validity associated with the different possible hypotheses. The constraints known to exist between the different hypotheses are represented by weighted interconnections between the neurons. When neural net models are used in this fashion they demonstrate the capability for performing optimization.

The computation of solutions to constraint satisfaction problems by connectionist networks is performed by an iterative relaxation search which starts with a randomly chosen initial state. This state can be interpreted as a proposed solution which the network progressively improves by reducing a well defined "objective" or "energy" function. The eventual low-energy (minimal) states to which the network converges represent the required "good" or valid solutions. The energy function defined for the network measures the extent to which the current interpretation violates the stipulated constraints. Each possible state of activity of the network has an associated energy. The activation rule used for updating activity levels of the neurons is so chosen that this global network energy shows a general decline with every iteration.

Optimization problems belonging to the category of constraint satisfaction may be addressed using traditional techniques such as linear programming. When the variables in the problem formulation are constrained as to take on only the values 0 or 1, the linear programming approach is called zero-one programming. It is reported to have been shown by Hinton [17] that certain zero-one programming problems can be implemented as relaxation searches in parallel networks. This means that connectionist networks can find "good" solutions to problems in which there are discrete hypotheses that are either "true only" or "false only". Each constraint is encoded by an interconnection weight that enables measurement of the extent of violation or adherence to the constraint by the current values of activation. The network attempts to alter the activation values of the neurons to progressively reduce this violation. The network thereby achieves the ultimately desired low-energy (high-goodness) stable states representative of the desired solution points to the problem.

The opinion that connectionist networks effectively perform optimization of well defined objective functions basically stems from the investigations of Hummel and Zucker [18]. They have shown that the final state to which a parallel connectionist network converges using relaxation schemes is actually one of the local minima of it's associated energy function.

A variety of problem domains have been explored using constraint satisfaction neural networks [19], [20], [21]. For some simple cases, it can be guaranteed that these networks will settle to the best final state regardless of their initial starting values. However, for more difficult problems and especially the ones involving discrete decisions, a neural net with dense interconnections and multiple nonlinear decision elements is known to be generally not well behaved. Constraint satisfaction neural nets show a tendency to oscillate or get trapped in uninteresting states known as "local minima" of the objective function that do not represent the actually required global solutions to the problem.

Hopfield Networks

Hopfield [22] identified a global objective function for networks with symmetrically connected units at approximately the same time as Hummel and Zucker [18] published the results of their investigations on parallel connectionist networks. Hopfield's neural net models use deterministic update rules for neuron activity levels and progressively iterate to minimize the global network objective function. The term "energy" in reference to the objective function was coined by him when he noticed that symmetrically connected networks of neurons exhibit similar behavior to thermodynamic systems seeking minimum energy states. These states are called attractor states in physical systems. Establishing such an analogy with physical systems has been instrumental in providing an important conceptual tool for analysis of connectionist networks.

The interpretation of a Hopfield net used for constraint satisfaction is as follows: A positively weighted connection between units i and j represents a pair of mutually supportive hypotheses. This means that if one of these units is on, the other one is constrained to be on also. A negative connection weight implies that if one of the units is on, the other is required not to be. The weight magnitude itself corresponds to the penalty to be applied if the constraint is violated. The energy of any state in the case of a second order network is given by

$$E(\vec{V}) = -\frac{1}{2}\sum_i \sum_j V_i V_j W_{ij} + \sum_i V_i b_i, \quad i \neq j,$$

where W_{ij} is the weight or strength of the connection between neurons i and j, V_i is 1 if neuron i is "on" and 0 if "off" and b_i is a bias for neuron i.

The number of weighted neurons that occur in the product term reflect the order of the dependencies involved in the incorporation of constraints into the network energy function. A quadratic term for example, reflects second-order dependencies of constraints on the neuron activation states. When the dependencies involved are identified to be higher than two; triple, quad or even larger groups of neurons could participate in determining the activations.

Given Hopfield's quadratic definition of energy, each neuron i can locally determine the difference between the global energy of the network when it is in the "off" state and the global energy when it is in the "on" state, knowing the current states of all other units in the network. The energy gap is therefore given by

$$E_{i_{off}} - E_{i_{on}} = \Delta E_i = \sum_j W_{ij} V_j - b_i \,, \quad j \neq i \;.$$

This yields the following rule for determining the activation state at the neuronal level in the network: If the energy gap ΔE is positive, the unit should turn on (or stay on) to minimize the global energy; otherwise it should turn off (or stay off).

Optimization of the network energy function is effected by progressively achieving a reduction in the value of the discrete energy difference term ΔE_i and thereby performing a gradient descent in the discrete energy landscape. A deterministic update rule such as above, provides for a very fast gradient descent but, it can be expected to work well only for those few problems having an exceptionally smooth and continuously decreasing energy terrain. For any slightly hard problem having a complex or unknown energy landscape, an inevitable consequence of allowing purely downhill moves is to get trapped in local minima that do not represent the required global solution points. This is a serious drawback.

Interestingly enough, the dynamics governing the energy minimization mechanism by a parallel connectionist net are analogous to the bonding of atoms in a crystalline structure [23]. If an atom in a crystal is oriented in a particular direction, it will tend to influence the orientation of nearby atoms in the same direction to achieve an overall, optimal fit. This phenomenon occurs over the entire crystal so that some atoms in one part of the crystal can form a structure in one orientation while atoms in another part of the crystal can form a structure in another orientation. The points where these opposing orientations meet constitute flaws in the crystal that are analogous to the local minima in connectionist networks. Identification of such an analogy with crystals provides an important basis for an answer to the issue of local minima in connectionist networks.

Boltzmann Machine Networks and Simulated Annealing

Once the problem of constraint satisfaction has been cast as an energy minimization problem and the analogy of connectionist networks with crystals has been established, it is logical to expect that the solution to the problem of local minima can be solved in essentially the same way that the problem of flaws during crystal formation are dealt with. A standard method used in physical systems involves annealing. A plausible solution therefore is to add an annealing-like process to the networks and have them employ a kind of simulated annealing [19].

The Boltzmann machine architecture [24], [25] is essentially a Hopfield network that uses a simulated annealing search to escape from local minima. With some differences of emphasis, this same idea was also independently proposed by other contemporary research groups [26], [27].

Simulated annealing is a search technique that has been applied to a number of optimization problems [19]. The idea is to attempt to escape from local minima by the addition of a random element to the decision making process of each neuron. In most cases, the unit still takes downhill steps, but occasionally it is allowed to take a step uphill in the energy landscape. Mathematically stated, each unit i computes the energy difference ΔE_i exactly as given earlier, then assumes the "on" or "1" state with probability p_i given by

$$p_i(unit_i = 1) = \frac{1}{1 + e^{-\frac{\Delta E_i}{T}}} .$$

The term T is a parameter that determines the amount of randomness, or noise and is analogous to a temperature. For large T, p_i is about 0.5 and the system assumes states at random, essentially ignoring the constraints in the network; for $T = 0$, the random element is totally eliminated and the system behaves as in the pure Hopfield network, moving strictly downhill to the nearest local minimum. This local decision rule has been shown to ensure that in thermal equilibrium the relative probability of two global states is determined solely by their energy difference, and follows a Boltzmann distribution:

$$\frac{P_\alpha}{P_\beta} = e^{-\left(\frac{E_\alpha - E_\beta}{T}\right)} ,$$

where P_α is the probability of being in the α-th global state, and E_α is the energy associated with that state [25]. In other words, for a given temperature, the probability of convergence to a final state is inversely proportional to the associated energy of that state. The bias towards low-energy states is higher at lower temperatures, but the time required to get there, i.e., to reach thermal equilibrium, is much longer than what it takes at a higher temperature. The most widely used method for reaching equilibrium is therefore to employ simulated annealing. The technique is to start with a high temperature and gradually reduce it. Geman and Geman [26] have derived bounds that establish limits on the allowable speed of the annealing schedule.

The preceding discussion examined the details of the use of neural nets to solve constraint satisfaction problems. The technique of obtaining decompositions relies on concepts from relational and Boolean algebras. In order to attempt a neural solution to the decomposition problem, the issue of a proper representation merits significant attention since it is important to achieve a proper mapping of the problem into the neural domain. In addition, an understanding of the algebra of relations, partitions and the theory of state machines is essential before developing a network to address the problem of decomposition. We thus present an overview of some of the important and relevant concepts in the following section. Further information may be obtained from references [5], [28], [29], [30], [31], [32].

2.3 Algebraic Theory

Relations

An ordered pair (a, b) is a pair of elements with a specific order associated with them. A binary relation R is a set of ordered pairs. If R is a binary relation and the pair (a, b) ϵ R, we write a R b to indicate that a is related to b by R. A binary relation from a set S to itself is called a relation in S, and is a subset of the cartesian product of the set S with itself, i.e., $R \subseteq S \times S$.

A relation R in a set S is said to be reflexive if it contains (a, a) $\forall$ a ϵ S. It is symmetric if the existence of an ordered pair (a, b) in R implies the existence of (b, a) in R $\forall$ a, b ϵ R. R is transitive if and only if the existence of ordered pairs (a, b) and (b, c) implies the existence of (a, c) also in R $\forall$ a, b, c ϵ S. A binary relation R in a set S is called an equivalence relation if it is reflexive, symmetric and transitive. Elements related by an equivalence relation are said to be equivalent.

An equivalence relation E partitions the elements s_i of a set S into disjoint subsets S_i called equivalence classes, such that all members of a subset are equivalent and members of different subsets are not equivalent. When a family of unary operations $\{\delta\}$ is defined on the set S, if the equivalence relation E demonstrates the characteristic of implying image equivalence in every situation where it can establish element equivalence, then E is said to satisfy the substitution property (s. p.) and is therefore called a congruence relation. Mathematically stated, consider an algebraic system $\mathfrak{V} = \langle V; \{\delta_1, \delta_2, ..., \delta_l\} \rangle$, where all the δ_i are unary operations; that is, $\delta_i: V \rightarrow V$, $(i = 1, 2, ..., l)$ and assume E to be an equivalence relation on V. We say that E satisfies the substitution property with respect to δ_i if, $\forall\, v_1, v_2 \in V$, $v_1 \, E \, v_2 \Rightarrow \delta_i(v_1) \, E \, \delta_i(v_2)$. When E satisfies the substitution property with respect to $\delta_i \; \forall\, i \in [1, l]$, then E is called a congruence relation on $\mathfrak{V}$.

Partitions

A partition π on a set S is a division of S into disjoint subsets $S_k \ni S_i \cap S_j = \varnothing, \forall\, i \neq j$ and $S = \cup S_k$. Each S_k is called a block of π. It is usual to represent a partition by writing overlines atop the disjoint subsets and separating them by semicolons so that the subsets appear as blocks of set members. An example of a partition is:

$$\pi = \{\, \overline{0, 2, 4}; \overline{1, 3, 5} \,\} .$$

The theory of finite state machine structure is based on laws derived from partition algebra. Mathematically defined, a finite state machine **M** is characterized by a three-tuple $\mathbf{M} \doteq \langle S, I; \{\delta\} \rangle$, where S is the state set of the machine **M**, I is its input set and $\{\delta\}$ is a set of state transition functions (or "next-state" mappings).

State machines may be decomposed as determined by certain special properties exhibited by the partitions generated over their state spaces, the most significant among them being the substitution property. With respect to a state machine characterized by a three-tuple as defined above, we present a formal definition of the property of substitution as satisfied by a partition over the state set of a finite state machine.

Definition

A partition on a state set S of a state machine **M** is said to satisfy the substitution property (s. p.) and is denoted by π if, $\forall$ s_i, s_j ϵ S which are in the same block of π and any input i_k ϵ I, the states $\delta(i_k, s_i)$ and $\delta(i_k, s_j)$ are also in a common block of π.

For any n-state set $S = \{s_1, s_2, ..., s_n\}$ of a machine **M**, there always exist two trivial s. p. partitions denoted by $\pi(0)$ and $\pi(I)$, where

$$\pi(0) = \{ \overline{s_1}; \overline{s_2}; \overline{s_3}; ...; \overline{s_n} \} \quad \text{and} \quad \pi(I) = \{ \overline{s_1, s_2, s_3, ..., s_n} \} .$$

Decomposition theory guarantees that if the product of any two s.p. partitions for a state machine **M** equals the trivial s.p. partition $\pi(0)$, then **M** can be directly decomposed into two independent machines $\mathbf{M_1}$ and $\mathbf{M_2}$ operating in parallel.

Classical method of generating s.p. partitions

S. p. partitions are currently obtained by performing a sequential exhaustive search into a given state machine's state transition characteristics that are determined by a specified input environment. This is the classical method, known to become intractable with growth in machine size [33]. An example is given below to illustrate the technique.

Example 2.1

Consider the state machine **M** whose state table is given below.

Present State (N)	**Next State**	
	First Input (I_1)	**Second Input (I_2)**
1	3	1
2	4	2
3	2	3
4	1	3

The machine has four states and is subjected to two inputs. The s. p. partitions for this machine are computed in several hierarchical levels. At the first level, s. p. partitions are generated by sequentially comparing two states of the machine at a time and applying the transitivity law at each step to merge implied pairs, eventually exhausting all the states. The partitions so obtained are then summed in accordance with the laws for addition of partitions, to obtain the second level of partitions. The second level partitions are then summed again to obtain a higher (third) level set of partitions. This process is continued until eventually no new partitions are generated. The procedure is illustrated below:

1, 2	⇒	3, 4; 1, 2	⇒	1, 2; 3, 4
1, 3	⇒	2, 3; 1, 3	⇒	1, 2, 3; 4
1, 4	⇒	1, 3 ; 1, 3	⇒	1, 3, 4 ; 2
2, 3	⇒	2, 4 ; 2, 3	⇒	2, 3, 4 ; 1
2, 4	⇒	1, 4 ; 2, 3	⇒	1, 2, 3, 4
3, 4	⇒	1, 2 ; 3	⇒	1, 2 ; 3, 4.

The complete set of non-trivial first level s.p. partitions is therefore:

$$\pi_1 = \{\, \overline{1,\ 2};\ \overline{3,\ 4} \,\},$$

$$\pi_2 = \{\, \overline{1,\ 2,\ 3};\ \overline{4} \,\},$$

$$\pi_3 = \{\, \overline{1,\ 3,\ 4};\ \overline{2} \,\},$$

$$\pi_4 = \{\, \overline{2,\ 3,\ 4};\ \overline{1} \,\},$$

and finally $\pi_5 = \pi(\mathrm{I})$ and $\pi_6 = \pi(0)$ are the two trivial s. p. partitions. Since no new partitions are generated for this machine by summing any of its first level partitions, we infer that the given machine **M** has four non-trivial s. p. partitions that may now be used to study its decomposability into smaller component machines in parallel, serial or complex interconnection schemes as applicable with the procedures of machine decomposition theory.

Representation

From the preceding discussion it is evident that there exists a one-to-one correspondence between partitions and relations. Relational algebra provides a well defined representation scheme for examining the characteristics of relations implied by given partitions. This representation is called a relation matrix and is used in the study of relational algebra [30].

Definition

Let X = $\{x_1, x_2, ..., x_m\}$ and Y = $\{y_1, y_2, ..., y_n\}$. Suppose R is a binary relation from X to Y. Then for a given order of the elements of X and Y the relation matrix of R denoted by M_R is defined as follows:

$$m_{ij} = 1, \text{ if } x_i \, R \, y_j$$
$$= 0, \text{ if } x_i \not R \, y_j,$$

where $m_{ij} \in M_R$ and $x_i \in X$ and $y_j \in Y$.

If the number of elements of X is equal to that of Y, i.e., n = m then M_R is a square matrix. As an example, letting R_e denote an equivalence relation given by

R_e = {(1, 1), (2, 2), (3, 3), (4, 4), (1, 2), (2, 3), (1, 3), (2, 1), (3, 2), (3, 1)},

the relation matrix M_R of R_e may be written from its definition as follows:

$$M_R = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

where the row and column indices stand for the elements of the set upon which R_e is defined. The matrix M_R is interpreted as follows. All diagonal elements of M_R are 1's, indicating that R_e is reflexive. M_R is symmetric about its diagonal, which is indicative of the symmetry in R_e. Further, M_R reflects the transitivity in R_e because $\forall$ (a, b), (b, c) $\in R_e$ and represented in M_R, the ordered pair (a, c) $\in R_e$ is also represented in M_R with a 1 entry in the appropriate row-column position. Thus, information about the characteristics of R_e is contained entirely in the relation matrix M_R.

2.4 Proposed Method

We now propose to cast the partition search problem into a framework of constraint satisfaction and develop a neural network capable of providing the desired solutions. A systematic and mathematical approach will be formulated for the derivation of the required network objective function. A detailed theoretical account of the proposed method is presented in the next chapter.

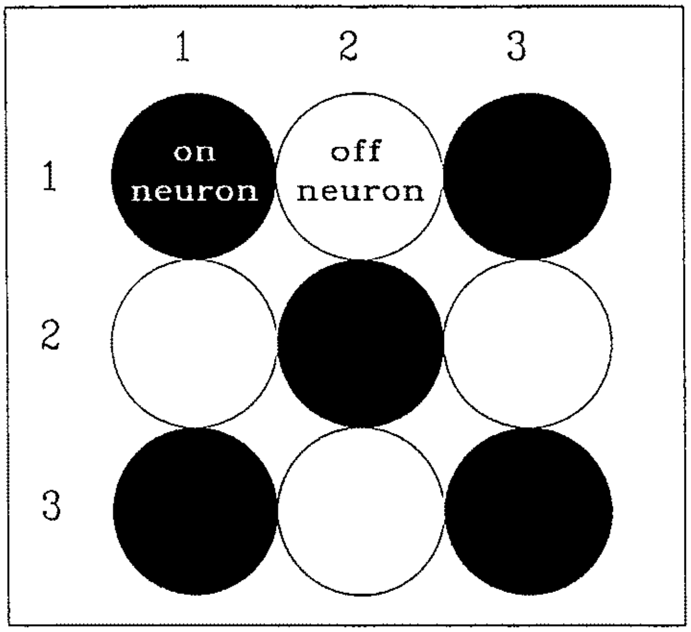
CHAPTER III

THEORETICAL DEVELOPMENT

An overview of the basic theoretical foundations underlying the work presented in this thesis was given in the last chapter. A method was proposed which is comprised of suitably casting the partition search problem into a framework of constraint-satisfaction and then developing a neural network to perform the optimization of a mathematically derived objective function over the problem space. This chapter discusses the theoretical aspects involved in realizing the proposed method.

3.1 Representation

In developing a neural approach to any problem, the issue of a proper representation merits primary attention. Partitions have a one-to-one correspondence with relations. We therefore use the relation matrix defined in chapter II to serve as the representation for partitions proposed by the network. The network is organized into a system of $N \times N$ neurons for a problem space of state dimension N. This means that a neuron in the "on" state at row-i, column-j is suggesting an equivalence between states i and j of the state machine subject to the incorporated constraints. As an illustration, consider the following final, stable state of activity for an $N \times N$ network of neurons with dimension $N = 3$:



Here the neurons in the "on" state are represented by solid circles. This representation, in the current state of the network corresponds to an equivalence between states 1 and 3, which is the s.p. partition

$$\pi\ \{\ \overline{1, 3};\ \overline{2}\ \}\ .$$

3.2 Derivation of the Energy Function

After addressing the issue of representation, the next step is to establish the interconnections between the neurons of the network. This is done by deriving the network energy function and then establishing a one-on-one correspondence of terms with those in the classical expression for the energy function of appropriate order. The derivation of the network energy function and a determination of its order are based on the identification of the constraint terms involved for the problem.

Constraints

Partitions satisfying the substitution property have been shown in chapter II to define uniquely corresponding congruence relations. By definition, congruence relations are implicitly equivalence relations with the added requirement that they imply image equivalence whenever states are established as equivalent. Thus, s. p. partitions must possess the equivalence relation properties: reflexivity, symmetry and transitivity as well as the (implication) properties of image equivalence.

The reflexive and symmetric relations can be implicitly encoded in the $N \times N$ representation scheme, by employing the following method:

1. Diagonal neurons in the $N \times N$ neuronal grid are clamped to remain in the "ON" state. Since this is representative of the equivalence of a state with

itself, the reflexive relation is satisfied before the network starts the computation process.

2. Symmetry can be incorporated by ensuring that the states of off-diagonal neurons in the in the upper triangular portion of the $N \times N$ grid maintain exact correspondence with the neuron states in the lower triangular portion.

With the above method the search space for solution points in given problem domains becomes noticeably reduced since the two constraint terms which would have been necessary to enforce the reflexive and symmetry properties have been obviated.

The remaining properties are therefore transitivity and the substitution property. Thus, the network energy function may be interpreted to consist of essentially two parts. The first part encodes the transitivity constraint. The second part encodes the state-image congruence constraint, i.e., the substitution property.

The second part of the network energy function must essentially map the complete state transition behavior of the state machine under its stimulus (input) set to the network dynamics. Thus, if we denote the energy function of the network by E, we may write

$$E = k_1 E_1 + k_2 E_2,$$

where E_1 is the energy term due to the transitivity constraint and E_2 is the energy term due to the state-image congruence (s. p. property or next state function) constraint. The parameters k_1 and k_2 are known as gain terms for the respective constraints. They determine the relative importance or weight that is assigned to each constraint.

Algebraic Basis of the Derivation

The neurons in the $N \times N$ network have only one of two possible states to which they can eventually converge: the "ON" state or the "OFF" state. This means that the ultimately stable solution states of the neural net are binary. We therefore use techniques from Boolean algebra to derive the required functional dependencies of the constraint terms on the neuron activation states, which we may regard as binary variables for our purpose of derivation. This is the basis that provides for a systematic and algebraic method of derivation for the network energy function.

The Transitivity Constraint

As defined in chapter II, a relation R is transitive iff a R b and b R c $\Rightarrow$ a R c, $\forall$ a, b, c $\in$ (state) set S over which R is defined. In other words, this means that if state "a" $\equiv$ state "b" and state "b" $\equiv$ state "c", then state "a" must also be $\equiv$ to state "c". Therefore, the third neuron responsible for representing equivalence between the states "a" and "c" in the $N \times N$ topological representation scheme for the network must be constrained to be "ON" whenever the pair of neurons representing equivalence between states "ab" and "bc" are in their "ON" states. This is a third-order functional dependence of the transitivity constraint term on the activation states of neurons in the network.

The above discussion leads to the conclusion that the transitivity constraint cannot be enforced into the network by a function that is of the more conventionally used quadratic order. We thus state a theorem which mathematically establishes that the transitivity constraint must be a third-order function of the neuron activation states.

Theorem

Third-order interconnections are required for an $N \times N$ topological neural net to verify the relation of transitivity in proposed partitions.

Proof

Defining Boolean matrix multiplications over n-th order square matrices $\underline{A}$ and $\underline{B}$ [30] by

$$c_{ij} = \bigvee_{k=1}^{n} a_{ik} \wedge b_{kj} , \tag{1}$$

where the $\vee$ and $\wedge$ represent bit-ORing and bit-ANDing operations respectively, we see that the matrix $\underline{C} = \underline{AB}$ contains a "1" in the row-column position indexed ij whenever a "1" exists in row-i, column-k in matrix $\underline{A}$ and row-k, column-j in matrix $\underline{B}$ $\forall$ k ϵ [1, n]. For a relation that is reflexive and symmetric, its relation matrix M_R will encode transitivity in R [34] iff:

$$M^2 \equiv M \times M \equiv M_R . \tag{2}$$

The matrix M_R is a Boolean matrix because its elements are only 1's or 0's. Thus, using equation (1)

$$\tilde{m}_{ij} \in M^2 = M \times M \tag{3}$$

can be written

$$\tilde{m}_{ij} = \bigvee_{k=1}^{n} m_{ik} \wedge m_{kj} . \tag{4}$$

Equation (2) essentially means that $\forall$ k ϵ [1, n] with $m_{ij} \epsilon M_R$,

$$\begin{aligned} \tilde{m}_{ij} \oplus m_{ij} &\doteq 0 \\ \rightarrow\ (\tilde{m}_{ij} \wedge m_{ij}^{\prime}) \vee (\tilde{m}_{ij}^{\prime} \wedge m_{ij}) &\doteq 0\ . \end{aligned} \tag{5}$$

Since the variables involved are binary, we have the following three results that can map Boolean logic operations $\forall\ x, y \in \{0, 1\}$, into the domain of integer arithmetic operations: The logical complement of a Boolean (binary) variable is

$$x^{\prime} \equiv (1 - x)\ . \tag{i}$$

The expression

$$x \wedge y - x \times y \equiv xy \tag{ii}$$

equates the logical "AND" operation to the operation of multiplication in the integer-number domain. Finally, the equivalent expression for the logical "OR" operation is derived using the above two results (i) and (ii) with the second theorem of De-Morgan [29]:

$$(x^{\prime} \wedge y^{\prime})^{\prime} \equiv (x \vee y)\ ,$$

yielding

$$\begin{aligned} x \vee y &\equiv (x' \wedge y')' \\ &\equiv ((1-x)\wedge(1-y))' \\ &\equiv ((1-x)\times(1-y))' \\ &\equiv (1-y-x+x\times y)' \\ &\equiv (1-1+x+y-xy) \\ &\equiv (x+y-xy)\ . \end{aligned} \tag{iii}$$

Thus, equation (5) may be rewritten as

$$\tilde{m}_{ij}(1-m_{ij}) \vee (1-\tilde{m}_{ij})m_{ij} \doteq 0 \tag{6}$$

which when simplified using the results (i), (ii) and (iii) transforms to

$$\begin{aligned} (\tilde{m}_{ij} - \tilde{m}_{ij}m_{ij})\vee(m_{ij} - \tilde{m}_{ij}m_{ij}) &\doteq 0 \\ \rightarrow \qquad m_{ij} + \tilde{m}_{ij} - 2m_{ij}\tilde{m}_{ij} &\doteq 0\ . \end{aligned} \tag{7}$$

Thus, based on all the above results developed, equation (4) may be written in the form

$$\tilde{m}_{ij} = \sum_{k=1}^{n} m_{ik}m_{kj} \tag{8}$$

which upon substituting in equation (7), we finally obtain

$$m_{ij} + \sum_{k=1}^{n} m_{ik}m_{kj} - 2\sum_{k=1}^{n} m_{ij}m_{ik}m_{kj} = 0 .$$

This is an equation of third order in m ϵ M. ■

We now derive the transitivity constraint term using the result established by the above theorem and the laws of Boolean algebra. Since the activation states of the neurons in the system are binary variables, we enumerate all possible combinations of activation values considering triplets of neurons (V_{ij}, V_{jk}, V_{ik}) in the form of a classical truth table. This is shown in table 3. 1. The transitivity constraint term (E_1) assigns a penalty of (positive) unity to the network energy function (E) in situations where the required neuron combinations violate the definition of transitivity. Conformation to the definition of the relation of transitivity obviates the penalty.

To derive the functional dependency of the constraint term E_1 on the neuron triple (V_{ij}, V_{jk}, V_{ik}), we draw the Karnaugh-map [35] as shown in figure 3. 1.

Using the rules of Boolean algebra to obtain the classical sum-of-products form [36], [37], [38] for the constraint E_1, we may write $\forall$ *i, j, k*:

Neuron V_{ij}	**Neuron** V_{jk}	**Neuron** V_{ik}	**Transitivity Law: Violated ?**	**Constraint E_1 value (1 or 0)**
0	0	0	No	0
0	0	1	No	0
0	1	0	No	0
0	1	1	Yes	1
1	0	0	No	0
1	0	1	Yes	1
1	1	0	Yes	1
1	1	1	No	0

Table 3. 1. Transitivity function truth table $\forall$ V_{ij}, V_{jk}, $V_{ik} \in N \times N$ neuron grid.

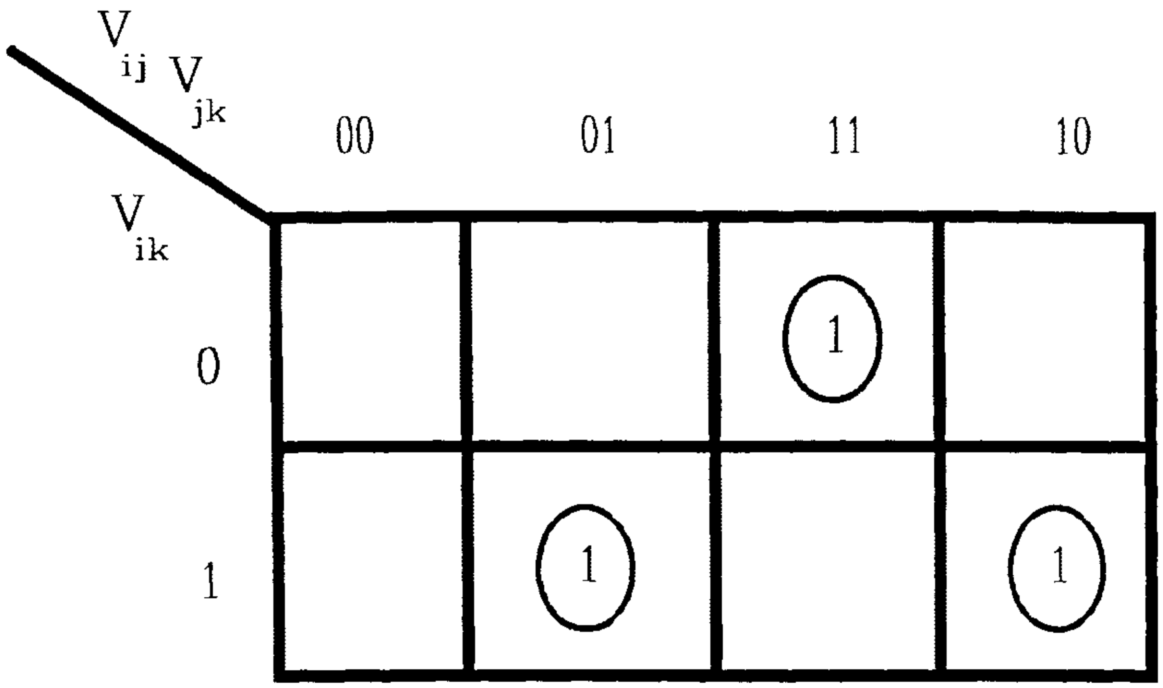


Figure 3. 1. Karnaugh Map of Transitivity Truth values.

$$\tilde{E}_1 = (V_{ij} \wedge V_{jk} \wedge V'_{ik}) \vee (V_{ij} \wedge V'_{jk} \wedge V_{ik}) \vee (V'_{ij} \wedge V_{jk} \wedge V_{ik}) ,$$

which upon using the results

$$\begin{aligned} x' &= (1 - x) , \\ x \wedge y &= xy , \\ x \vee y &= (x + y - xy) , \end{aligned}$$

may be written in the form

$$\tilde{E}_1 = V_{ij} V_{jk} + V_{jk} V_{ik} + V_{ij} V_{ik} - 3 V_{ij} V_{jk} V_{ik} .$$

Since the representation used is the $N \times N$ network topology, the indices i, j, k span only the upper triangular portion of the square grid of $N \times N$ neurons in the system (reference index starting at 0). Thus, the last equation may be compacted to the following overall form:

$$E_1 = \sum_{i=0}^{(n-3)} \sum_{j=(i+1)}^{(n-2)} \sum_{k=(j+1)}^{(n-1)} \tilde{E}_1 .$$

The equation for the transitivity constraint term E_1 is of third order as required. We note that the equation sums only for indices spanning the upper triangular portion of the $N \times N$ topology of neurons. This is useful for achieving a reduction in computation overheads. The equation ensures that E_1 generates a $+1$

contribution to the network energy $\forall$ triples of neurons (V_{ij}, V_{jk}, V_{ik}) in the upper triangular portion of the $N \times N$ grid that violate transitivity. E_1 evaluates to a value of zero energy for every triple that does not violate transitivity.

The Substitution Property Constraint

This constraint is responsible for motivating the network towards finding partitions satisfying the substitution property. To enforce this property into the generated partitions, the family of next-state functions determining the state transition behavior of the state machine must be incorporated into the constraint term. The state-transition function is defined for each present state by a pre-specified set of inputs. As a result, the constraint mapping this information into the neural net will need to scan the input set and determine the extent to which current solutions proposed by the net violate the requirement of preserving state-image congruence.

We interpret the neuron activation states as binary variables again, and generate a truth table enumerating all combinations for pairs of neurons (V_{ij} and its image neuron for a specific input "k" denoted $V_{\delta\text{-}k(i)\delta\text{-}k(j)}$). The subscript "$\delta\text{-}k(i)$" denotes the next-state determined by input "k", for the current (present) state "i".

A truth table generated by considering pairs of neurons in general terms is shown in table 3. 2. The s. p. constraint term (E_2), assigns a penalty of negative unity to the network energy for all cases where the required neuron combinations violate the definition of the substitution property. Conformation to the property of substitution obviates the penalty.

The entries in the truth table for constraint term E_2 show that its functional dependency on the states of the neurons V_{ij} and $V_{\delta-k(i)\delta-k(j)}$ has the same form as the classical "0110" combination corresponding to a two-variable logical "EXCLUSIVE-OR (XOR)" function. Thus, the sum-of-products form for the E_2 term $\forall$ i,j,k may be written from the truth values and the previously employed results as

$$\begin{aligned} \tilde{E}_2 &= V_{ij} V'_{\delta-k(i)\delta-k(j)} \vee V'_{ij} V_{\delta-k(i)\delta-k(j)} \\ &\equiv V_{ij} + V_{\delta-k(i)\delta-k(j)} - 2 V_{ij} V_{\delta-k(i)\delta-k(j)} \, . \end{aligned}$$

As in the case of the transitivity constraint, since the indices span only the upper triangular half of the $N \times N$ neuron grid, the above equation may be compacted to the concise form:

Neuron V_{ij}	**Next-state Neuron** $V_{\delta-k(i)\delta-k(j)}$	**Violation of Substitution Property ?**	**Value for** E_2
0	0	No	0
0	1	Yes	1
1	0	Yes	1
1	1	No	0

Table 3. 2. Substitution Property constraint term E_2 truth table.

$$E_2 = \sum_{i=0}^{n-2} \sum_{j=(i+1)}^{n-1} \sum_{\delta-k(i)\delta-k(j)=\delta-k_1(i)\delta-k_1(j)}^{\delta-k_{k_{\max}}(i)\delta-k_{k_{\max}}(j)} \tilde{E}_2 \, .$$

The above equation is quadratic; it is computationaly efficient to the extent of offering the benefit of scanning only a total of $[N(N-1)] \div 2$ neurons from the $N \times N$ grid.

Combining the two constraint terms E_1 and E_2 derived so far into one expression, the overall equation for the network energy function may now be written as

$$E = k_1E_1 + k_2E_2.$$

The gain parameters k_1 and k_2 are set relatively equal to each other so that the constraint enforcing transitivity shares equal importance with the one influencing the substitution property. This is not only beneficial in eliminating "tuning" of the gains, but is also necessary due to the characteristic of s. p. partitions.

Ground State Characteristic of the Energy Function

The energy function (E) for the network has degenerate ground states. In other words, the solution points for the s. p. partition problem represent a value of zero to the network energy function. Neural net solutions to constraint satisfaction/optimization problems of the class of the TSP [6], graph partitioning problem [7] and others, have always been compared for merit with solutions generated by classical techniques before accepting their "optimality". The degenerate ground-state characteristic of the network we have developed obviates the comparison exercise due to the fact that solution states are now uniquely identified by their associated zero-energy values. A further use of this characteristic is the easy detection of local minima since unlike global minima, these states will not have zero energies associated with them. Thus, the energy function more tangibly reflects the merit of suggested solutions by the network: If the partition energy is zero, the proposed solution by the network in its current state of activity is the required global solution, else the computation needs to be continued and the present interpretation of the net needs to be improved.

3.3 Interconnection strengths and Activation rule

The classical form of the energy function (E) for a third-order network is written as [39]

$$E(\vec{V}) = -\frac{1}{3}\sum_{i\neq j\neq k}\sum_{j\neq k\neq i}\sum_{k\neq j\neq i} W^{(3)}{}_{ijk}V_iV_jV_k - \frac{1}{2}\sum_{i\neq j}\sum_{j\neq i} W^{(2)}{}_{ij}V_iV_j - \frac{1}{1}\sum_{i} W^{(1)}{}_{i} \,,$$

where $W^{(3)}_{ijk}$ is the weight matrix storing the interconnection strengths between triples of neurons $(V_iV_jV_k)$, $W^{(2)}_{ij}$ contains the strengths between pairs of neurons (V_iV_j) and $W^{(1)}_i$ represents the bias for each neuron.

Comparing this expression with the derived equation (E) for the network energy function and equating the coefficients of like terms, we obtain the entries to the three weight matrices.

The rule for activation of neurons in the net is based on the classical Hopfield-like constraint satisfaction neural net mechanism. Each neuron takes turns in evaluating the difference in the global energy of the network when it is in the "ON" state and when it is in the "OFF" state. The state of activity of the neuron that lowers the global energy of the network is then assumed by the neuron. This decision is made locally by each neuron and iterated enough number of times till further changes cease to affect the network globally. The network is thereupon understood to have performed the intended gradient descent in a $2^{N(N-1)/2}$ dimension landscape and effected an optimization of the energy function. The final, stable state of the neurons in the network then reflect the solution computed by the net.

This chapter discussed the theoretical aspects involved in the formulation of the neural net developed for addressing the s. p. partitioning problem for machine decomposition. The next chapter discusses the results obtained by implementation of the method via software simulations on a serial computer.

CHAPTER IV

SIMULATIONS AND RESULTS

The theoretical basis for a neural approach to the decomposition problem was developed in the previous chapter. Issues of network architecture and topology were addressed from a mathematical perspective. This is a relatively new approach which has hitherto been generally heuristic. A formal algebraic method of derivation for the network energy function was developed. This chapter discusses the actual implementation of the network via software simulation and the presents the ensuing results.

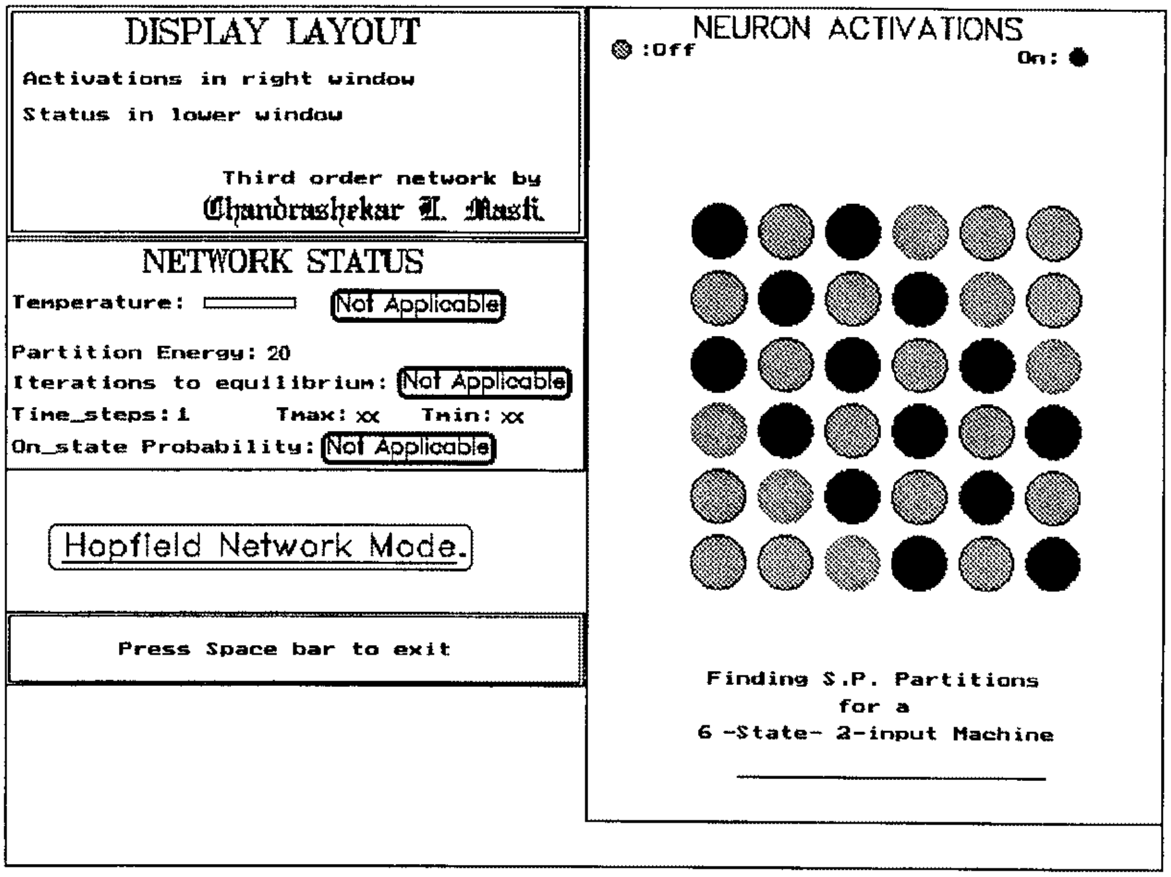
Network simulations were performed on an IBM-compatible personal computer (PC) equipped with a math-coprocessor and running at a 20 Mhz clock rate. The detailed code is written in the "C" programming language, a copy of which is provided in the appendix. Initial network simulations were performed by employing the Hopfield architecture. This was later changed to a Boltzmann machine net implementation after observing that the inherent performance limitations of the Hopfield net yielded unsatisfactory results for our problem.

4.1 Hopfield network simulations

Hopfield networks employ a deterministic update rule to deduce the neuron activation states. In terms of performing optimization of the objective function derived over a given problem space, this amounts to taking strictly downhill moves in a complex, unknown terrain. A natural and inevitable consequence of such a technique is to get trapped in local minima, an example of which is shown in figure 4. 1. A third-order Hopfield network was simulated in this example to search for s. p. partitions over the state space of a 'modulo-six counter'. The state table of this machine is given in table 4. 1. Since the dimension of the energy landscape for an N-state machine is given by the expression $2^{N(N-1)/2}$, no convenient graphical representation exists to clearly show that the state the network is currently trapped in is indeed a local minimum. The problem addressed here has no more than six states (N = 6), but the dimension of its energy landscape is $2^{15} = 32768$. Table 4. 2 therefore shows the energies of all states at a unit Hamming distance [40] from the local minimum into which the modulo-six counter neural net settles. The table shows that the energy associated with the current state of the net is less than or at most equal to the energy of every other state at a unit Hamming distance from this point in the landscape of the function. By definition, this by itself means that the net is currently trapped in a local minimum. A further assertion is provided by observing that the value displayed for the partition energy is non-zero. Since Hopfield nets always update states in the direction of decreasing gradient in the energy space, it

Present State	**Input I_1**	**Input I_2**
0	1	5
1	2	4
2	3	3
3	4	2
4	5	1
5	0	0

Table 4. 1. Modulo-six counter state table.



```
DISPLAY LAYOUT
Activations in right window
Status in lower window
                    Third order network by
                    Chandrashekar L. Masti

NETWORK STATUS
Temperature:            Not Applicable
Partition Energy: 20
Iterations to equilibrium: Not Applicable
Time_steps: 1       Tmax: xx    Tmin: xx
On_state Probability: Not Applicable

Hopfield Network Mode.

Press Space bar to exit

NEURON ACTIVATIONS
▨ :Off                          On: ●

● ▨ ● ▨ ▨ ▨
▨ ● ▨ ● ▨ ▨
● ▨ ● ▨ ● ▨
▨ ● ▨ ● ▨ ●
▨ ▨ ● ▨ ● ▨
▨ ▨ ▨ ● ▨ ●

Finding S.P. Partitions
for a
6 -State- 2-input Machine
```

Figure 4. 1. Third-order Hopfield net trapped in a local minimum.

Partition	Associated Energy	Remarks
τ_{local}	20	*local minimum*
τ_1	40	unit Hamming distance away
τ_2	25	unit Hamming distance away
τ_3	45	unit Hamming distance away
τ_4	25	unit Hamming distance away
τ_5	70	unit Hamming distance away
τ_6	30	unit Hamming distance away
τ_7	45	unit Hamming distance away
τ_8	20	unit Hamming distance away
τ_9	55	unit Hamming distance away
τ_{10}	30	unit Hamming distance away
τ_{11}	50	unit Hamming distance away
τ_{12}	50	unit Hamming distance away
τ_{13}	20	unit Hamming distance away
τ_{14}	45	unit Hamming distance away

Table 4. 2. Energies of states at unit Hamming distance from local minimum.

cannot be expected that the network will escape from the state it is trapped in, implying that the net will stay in the local minimum for infinite time. The current interpretation by the net cannot be improved.

The Boltzmann machine architecture was the next logical choice for exploration. Simulated annealing was used with the Boltzmann machine, with a view to addressing the problem of local minima. A stochastic machine with a Boltzmann net architecture employing simulated annealing is known to asymptotically converge to global solution points of problem domains generally without regard to the complexity involved with the energy terrain [1].

4.2 Boltzmann Machine with Simulated Annealing

A third-order Boltzmann machine network employing simulated annealing was developed and simulated to solve for s. p. partitions. Amongst several algorithms that exist for incorporating the technique of simulated annealing into parallel connectionist nets, the logarithmic rule derived by Geman and Geman [26] mathematically guarantees asymptotic convergence to global minima. Their rule is called classical simulated annealing (CSA). We used this technique in our simulations for Boltzmann machines. Three example problems are illustrated here that provide information for performance evaluation over a spectrum of problem sizes.

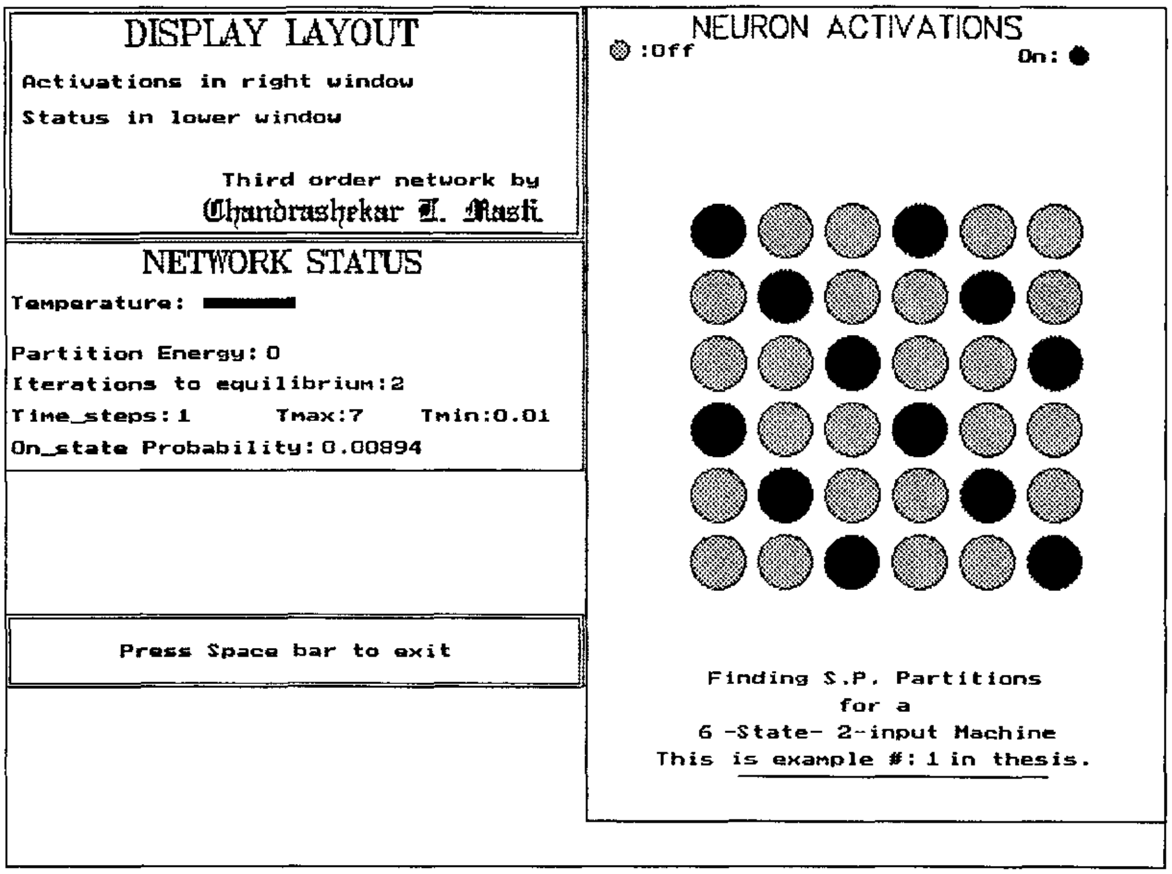
Example 1.

The modulo-six counter that trapped the Hopfield network into a local minimum was addressed again, using a third-order Boltzmann machine architecture. The state table for the problem is the same as the one given earlier in table 4.1 used for analyzing the Hopfield net. The performance of the network is impressive. A snap-shot of the network in simulation illustrated in figure 4. 2 shows a case where the problem has been solved in one discrete-time step. The network generated the partition

$$\pi_1 = \{ \overline{0, 3}; \overline{1, 4}; \overline{2, 5} \},$$

over the state space of the counter. The above solution is decoded by noticing that in the upper triangular portion of the square $N \times N$ grid, neurons are in the "on" states in the first row, fourth column, i.e., row-0, column-3, and second row, fifth column, i.e., row-1, column-4 and third row, sixth column, i.e., row-2, column-5 positions. Only one other non-trivial s. p. partition exists for this machine:

$$\pi_2 = \{ \overline{0, 2, 4}; \overline{1, 3, 5} \}.$$



DISPLAY LAYOUT

Activations in right window

Status in lower window

Third order network by
Chandrashekar L. Masti

NETWORK STATUS

```
Temperature:

Partition Energy: 0
Iterations to equilibrium:2
Time_steps:1        Tmax:7     Tmin:0.01
On_state Probability: 0.00894
```

Press Space bar to exit

NEURON ACTIVATIONS

▨ :Off   On: ●

```
● ▨ ▨ ● ▨ ▨
▨ ● ▨ ▨ ● ▨
▨ ▨ ● ▨ ▨ ●
● ▨ ▨ ● ▨ ▨
▨ ● ▨ ▨ ● ▨
▨ ▨ ● ▨ ▨ ●
```

Finding S.P. Partitions
for a
6 -State- 2-input Machine
This is example #: 1 in thesis.

Figure 4. 2. Third-order Boltzmann machine solving the modulo-six counter problem.

In order to obtain a more general idea of the overall performance of the net over an ensemble of random starting points in the 32768 integer-point discrete energy landscape, fifteen mutually independent trials were made and results examined for the possible success rate of the net in finding valid solutions. Table 4. 3 lists the results obtained. It is evident from the table that the net always converged to a valid solution state. The initial network temperature was set to seven, a number that was determined upon ensuring that the probabilities of updating the activation states of the neurons in the net displayed initial values in the neighborhood of 0.5. Every neuron in the net is randomly chosen and visited for stochastically updating its activation state. Two iterations were arbitrarily determined to be satisfactory for allowing the net to reach thermal equilibrium before updating the network temperature. A new discrete time step is regarded to have started each time the network temperature is updated. The simulation is written to provide an audio-visual indication to the user when a solution state is approached by the net. A zero value displayed for the partition energy as illustrated in the snap-shot figure visually indicates solution; a soft four-tone chime is also incorporated.

Example 2.

In this example a larger state space was examined. A state machine with nine states in its state set and three inputs in its input set was presented to the network. The state table describing the machine is shown in table 4. 4. Since the dimension

Trial number	Partition	# Time Steps needed for solution
1	$\pi(I)$	2
2	$\pi(0)$	3
3	π_1	1
4	π_2	1
5	π_1	2
6	$\pi(0)$	3
7	$\pi(0)$	2
8	$\pi(0)$	3
9	$\pi(0)$	3
10	$\pi(0)$	3
11	$\pi(0)$	6
12	π_2	2
13	π_2	1
14	$\pi(I)$	2
15	$\pi(0)$	3

Table 4. 3. Results of the network compiled for fifteen trial runs.

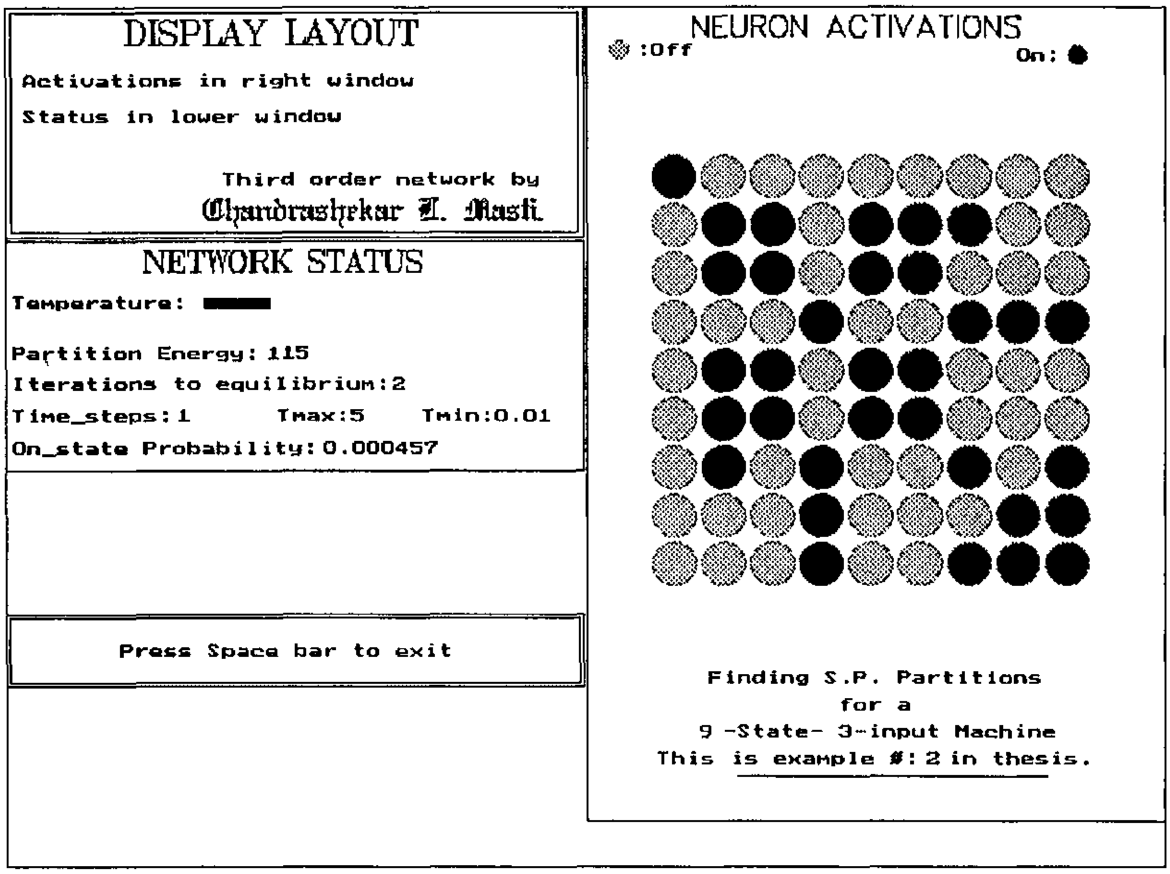
Present State	**Input I_1**	**Input I_2**	**Input I_3**
0	0	0	0
1	1	1	7
2	2	5	2
3	3	3	6
4	4	4	4
5	2	5	5
6	6	3	6
7	1	7	7
8	8	8	8

Table 4. 4. State table for a nine-state three-input machine.

of the state space being addressed is nine (N=9), the network is organized into a square grid of 9 × 9 neurons. Amongst several valid solutions, one s. p. partition

$$\pi = \{ \overline{0, 2, 4, 5}; \overline{1, 7}; \overline{3, 6, 8} \}$$

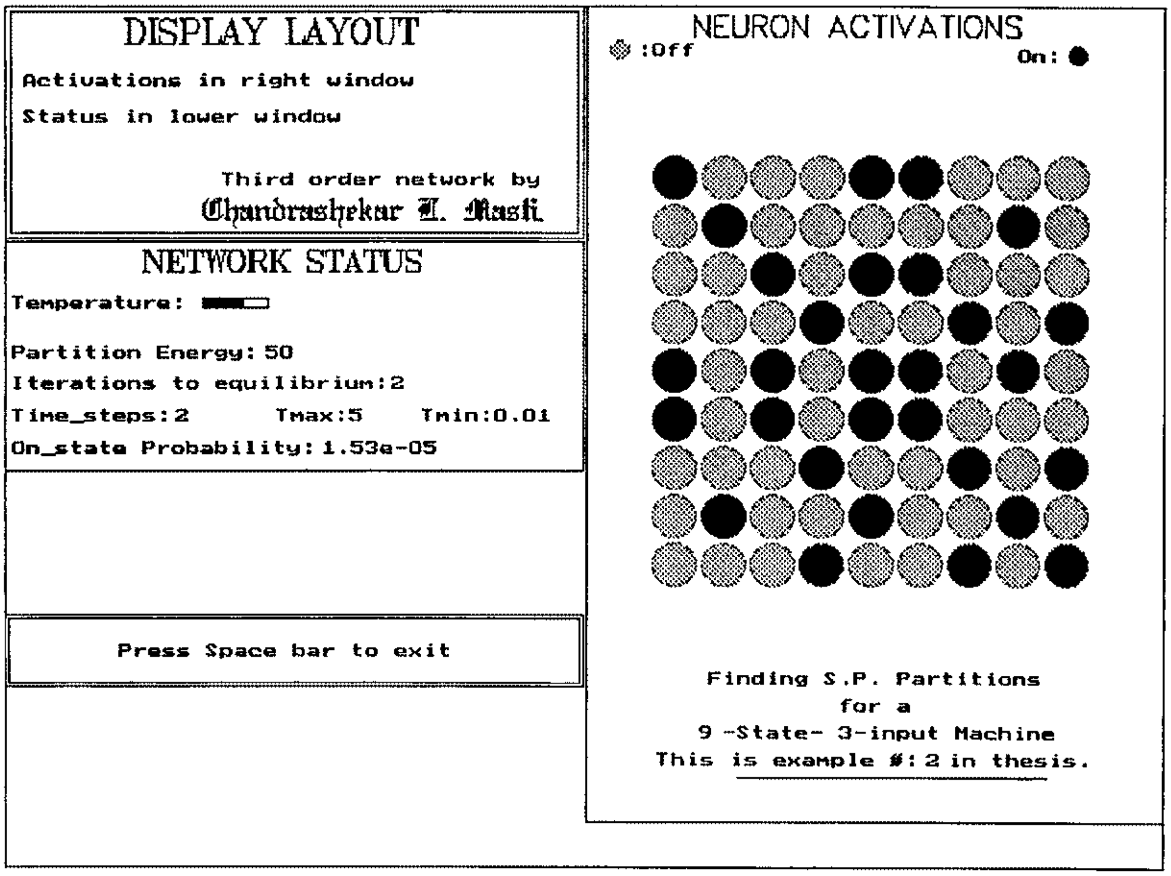
was generated by the net in three discrete-time steps. The computation process of the net as it generated the above solution is illustrated in sequence via snap-shot figures 4. 3 - 4. 5. A random initial starting state of the network is shown in figure 4. 3 where the value of the current partition energy is displayed as 115 units and the starting temperature is five units. It is evident from the figure that neither the relation of transitivity nor the substitution property is satisfied by the net in its current state of activity. Since an s. p. partition must have zero energy, it is inferred that the present partition as proposed by the net at this time step is not a solution. Figure 4. 4 shows the state of the network one discrete-time step later. The displayed value of 50 units for the current partition energy again indicates that the present state of the net does not represent a solution, a fact which is easily verifiable by a manual examination for either the transitive relation or the substitution property as governed by the next-state function determined by the machine's given input set. The network converged to the state shown in figure 4. 5 in the third time step. The fact that the partition energy is zero asserts that the net in its current state now represents solution. With a view to obtaining an overall picture of the network's performance to a state space of the current size, fifteen mutually independent trials were once again performed, the results of which are presented in table 4. 5. It is



```
DISPLAY LAYOUT
Activations in right window
Status in lower window
                    Third order network by
                    Chandrashekar L. Masti

NETWORK STATUS
Temperature:
Partition Energy: 115
Iterations to equilibrium:2
Time_steps:1       Tmax:5     Tmin:0.01
On_state Probability: 0.000457

Press Space bar to exit
```

```
NEURON ACTIVATIONS
▨ :Off                         On: ●

● ▨ ▨ ▨ ▨ ▨ ▨ ▨ ▨
▨ ● ● ▨ ● ● ● ▨ ▨
▨ ● ● ▨ ● ● ▨ ▨ ▨
▨ ▨ ▨ ● ▨ ▨ ● ● ●
▨ ● ● ▨ ● ● ▨ ▨ ▨
▨ ● ● ▨ ● ● ▨ ▨ ▨
▨ ● ▨ ● ▨ ▨ ● ▨ ●
▨ ▨ ▨ ● ▨ ▨ ▨ ● ●
▨ ▨ ▨ ● ▨ ▨ ● ● ●

      Finding S.P. Partitions
              for a
     9 -State- 3-input Machine
  This is example #: 2 in thesis.
```

Figure 4. 3. A random initial starting state of the 9 × 9 net.



DISPLAY LAYOUT

Activations in right window

Status in lower window

Third order network by

Chandrashekar L. Masti

NETWORK STATUS

```
Temperature:

Partition Energy: 50
Iterations to equilibrium:2
Time_steps:2        Tmax:5     Tmin:0.01
On_state Probability: 1.53e-05
```

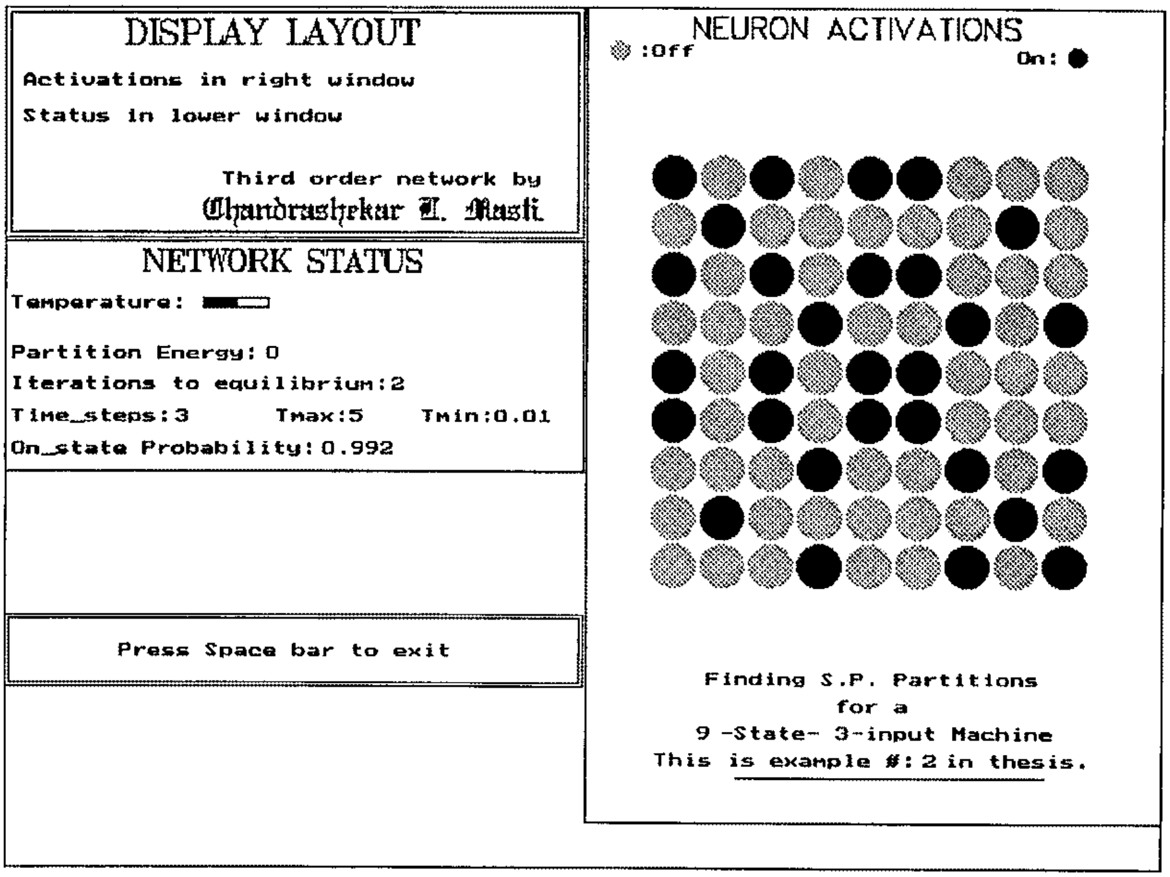
Press Space bar to exit

NEURON ACTIVATIONS

▨ :Off   On: ●

Finding S.P. Partitions
for a
9 -State- 3-input Machine
This is example #: 2 in thesis.

Figure 4. 4. A snap-shot of the net one time-step later.



DISPLAY LAYOUT

Activations in right window

Status in lower window

Third order network by
Chandrashekar L. Masti

NETWORK STATUS

Temperature:

Partition Energy: 0

Iterations to equilibrium: 2

Time_steps: 3   Tmax: 5   Tmin: 0.01

On_state Probability: 0.992

Press Space bar to exit

NEURON ACTIVATIONS

▨ :Off   On: ●

```
● ▨ ● ▨ ● ● ▨ ▨ ▨
▨ ● ▨ ▨ ▨ ▨ ▨ ● ▨
● ▨ ● ▨ ● ● ▨ ▨ ▨
▨ ▨ ▨ ● ▨ ▨ ● ▨ ●
● ▨ ● ▨ ● ● ▨ ▨ ▨
● ▨ ● ▨ ● ● ▨ ▨ ▨
▨ ▨ ▨ ● ▨ ▨ ● ▨ ●
▨ ● ▨ ▨ ▨ ▨ ▨ ● ▨
▨ ▨ ▨ ● ▨ ▨ ● ▨ ●
```

Finding S.P. Partitions
for a
9 -State- 3-input Machine
This is example #: 2 in thesis.

Figure 4. 5. Solution state approached by the net in time-step three.

Trial number	Partition	Time steps needed for solution
1	π_1	4
2	π_2	2
3	π_3	3
4	π_4	2
5	π_5	3
6	π_6	2
7	π_7	2
8	π_8	2
9	π_9	1
10	π_{10}	3
11	π_{11}	2
12	π_{12}	2
13	π_{13}	1
14	π_{14}	2
15	π_{15}	3

Table 4. 5. Performance of the net to 15 independent trials.

apparent that the net always found a valid solution. The logarithmic annealing schedule (CSA) used earlier was employed for updating the network's temperature. It was determined that no more than two steps were needed for allowing the net to reach thermal equilibrium at each temperature. The legend for the various s. p. partitions generated by the net and tabulated in table 4. 5 is provided in figure 4. 6.

<u>**Example 3.**</u>

A state space of twice the dimension examined in the previous problem was addressed in this example. A state machine with 18 states and three inputs was presented to the network. The state table for this machine is given in table 4. 6. Figures 4. 7 - 4. 10 provide an illustration of the network as it generated a valid solution

$$\pi = \{\overline{0, 2, 4};\ \overline{1, 9, 17};\ \overline{3, 5};\ \overline{6, 8, 10};\ \overline{7, 15};\ \overline{11};\ \overline{12, 13, 14, 16}\}\ .$$

Figure 4. 7 shows a snap-shot of the 18 × 18 net in a random initial starting state. Two intermediate states of the net as it evolved ultimately to the final solution configuration are illustrated by figures 4. 8 and 4. 9. The valid global (required) solution state to which the network converged in four discrete time-steps is shown in figure 4. 10. The performance of the network to fifteen independent runs was observed and is tabulated in table 4. 7. The success rate of the net is impressive. A valid solution was always obtained - the fastest in as little as two time steps.

$$\pi_1 = \{ \overline{0, 1, 7}; \overline{2, 5}; \overline{3, 6}; \overline{4}; \overline{8} \},$$

$$\pi_2 = \{ \overline{0, 2, 5}; \overline{1, 3, 6, 7}; \overline{4}; \overline{8} \},$$

$$\pi_3 = \{ \overline{0, 1, 4, 7}; \overline{2, 3, 5, 6}; \overline{8} \},$$

$$\pi_4 = \{ \overline{0}; \overline{1, 7}; \overline{2, 5, 8}; \overline{3, 6}; \overline{4} \},$$

$$\pi_5 = \{ \overline{0, 1, 7, 8}; \overline{2, 4, 5}; \overline{3, 6} \},$$

$$\pi_6 = \{ \overline{0}; \overline{1, 7}; \overline{2, 3, 5, 6}; \overline{4, 8} \},$$

$$\pi_7 = \{ \overline{0, 3, 6}; \overline{1, 7}; \overline{2, 4, 5}; \overline{2, 4, 5}; \overline{8} \},$$

$$\pi_8 = \{ \overline{0, 3, 4, 6}; \overline{1, 2, 5, 7, 8} \},$$

$$\pi_9 = \{ \overline{0, 1, 7}; \overline{3, 6, 8}; \overline{2, 5}; \overline{4} \},$$

$$\pi_{10} = \{ \overline{0}; \overline{1, 7, 8}; \overline{2, 4, 5}; \overline{3, 6} \},$$

$$\pi_{11} = \{ \overline{0}; \overline{1, 4, 7, 8}; \overline{2, 3, 5, 6} \},$$

$$\pi_{12} = \{ \overline{0, 2, 5}; \overline{1, 4, 7}; \overline{3, 6, 8} \},$$

$$\pi_{13} = \{ \overline{0, 4, 8}; \overline{1, 2, 3, 5, 6, 7} \},$$

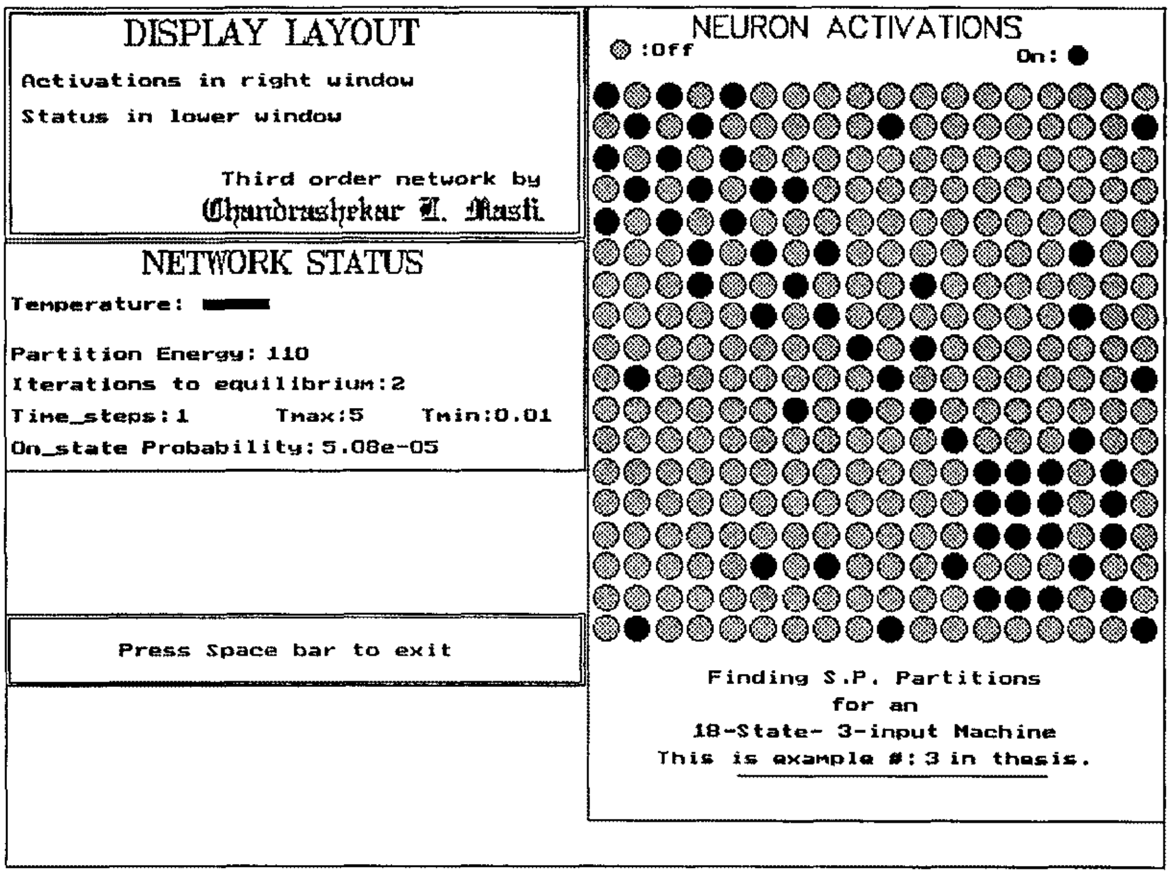
$$\pi_{14} = \{ \overline{0, 2, 5}; \overline{1, 7, 8}; \overline{3, 6}; \overline{4} \},$$

$$\pi_{15} = \{ \overline{0, 3, 6}; \overline{1, 2, 5, 7, 8}; \overline{4} \}.$$

Figure 4. 6. Legend for the s. p. partitions listed in table 4. 5.

Present state	Input I_1	Input I_2	Input I_3
0	0	2	4
1	1	9	17
2	0	2	4
3	3	3	3
4	0	2	4
5	5	5	5
6	6	8	10
7	7	7	7
8	6	8	10
9	1	9	17
10	6	8	10
11	11	11	11
12	12	14	16
13	13	13	13
14	12	14	16
15	15	15	15
16	12	14	16
17	1	9	17

Table 4. 6. 18-state 3-input state machine table.



DISPLAY LAYOUT

```
Activations in right window
Status in lower window

                    Third order network by
```

Chandrashekar V. Masti

NETWORK STATUS

```
Temperature: 

Partition Energy: 110
Iterations to equilibrium:2
Time_steps:1        Tmax:5     Tmin:0.01
On_state Probability:5.08e-05
```

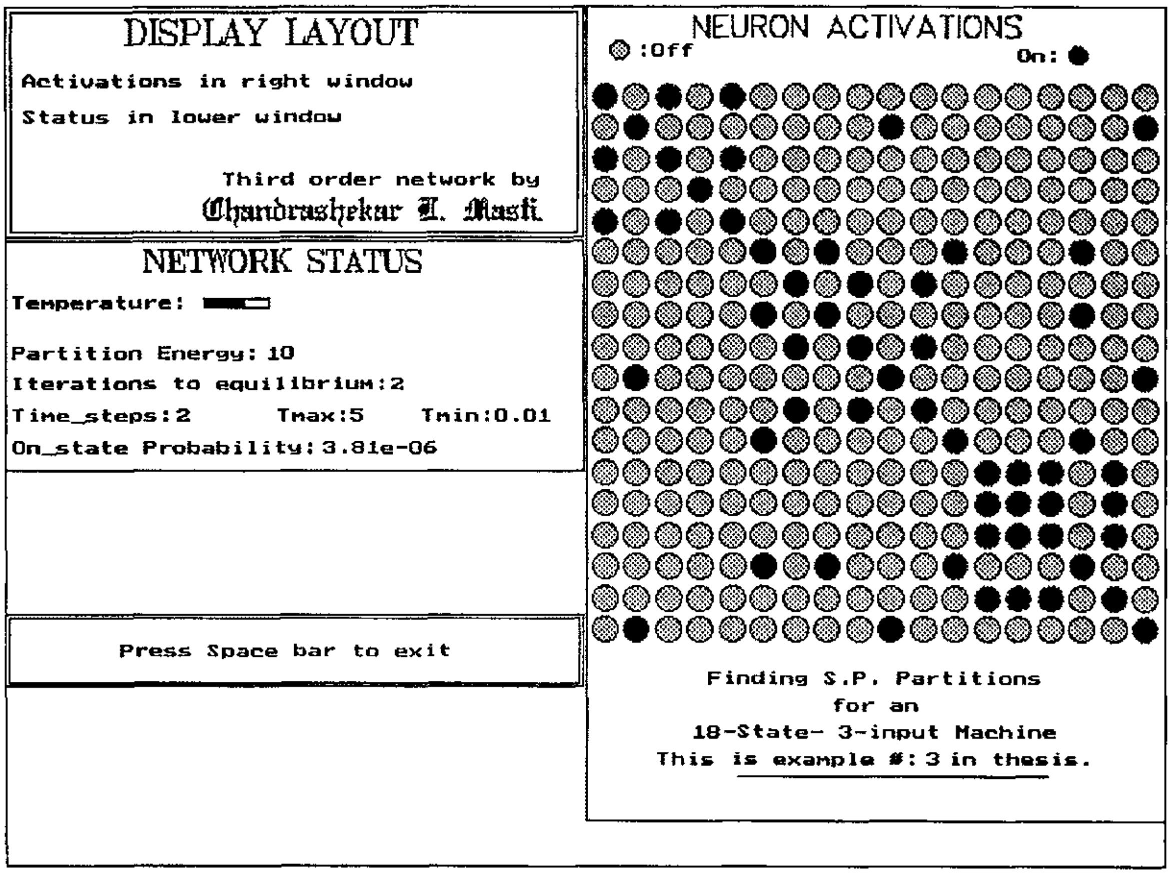
```
Press Space bar to exit
```

NEURON ACTIVATIONS

```
◍ :Off                    On: ●
```

```
Finding S.P. Partitions
        for an
18-State- 3-input Machine
This is example #: 3 in thesis.
```

Figure 4. 7. A random starting state of the 18 × 18 network.



DISPLAY LAYOUT

Activations in right window

Status in lower window

Third order network by

Chandrashekar L. Masti

NETWORK STATUS

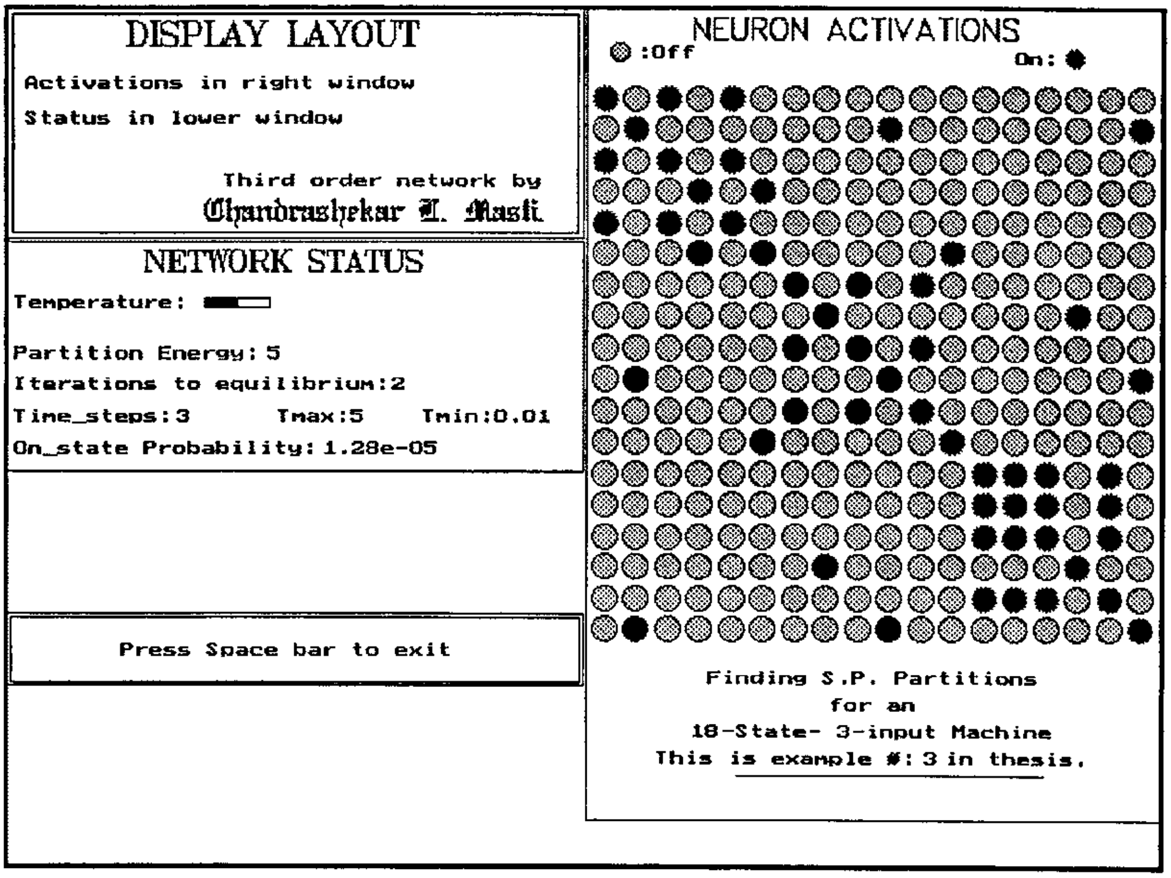
```
Temperature:

Partition Energy: 10
Iterations to equilibrium:2
Time_steps:2        Tmax:5     Tmin:0.01
On_state Probability: 3.81e-06
```

Press Space bar to exit

NEURON ACTIVATIONS

Finding S.P. Partitions
for an
18-State- 3-input Machine
This is example #: 3 in thesis.

Figure 4. 8. State of the network at the second time-step.



DISPLAY LAYOUT

Activations in right window

Status in lower window

Third order network by

Chandrashekar L. Masti

NETWORK STATUS

Temperature:

Partition Energy: 5

Iterations to equilibrium:2

Time_steps:3 Tmax:5 Tmin:0.01

On_state Probability: 1.28e-05

Press Space bar to exit

NEURON ACTIVATIONS

:Off On:

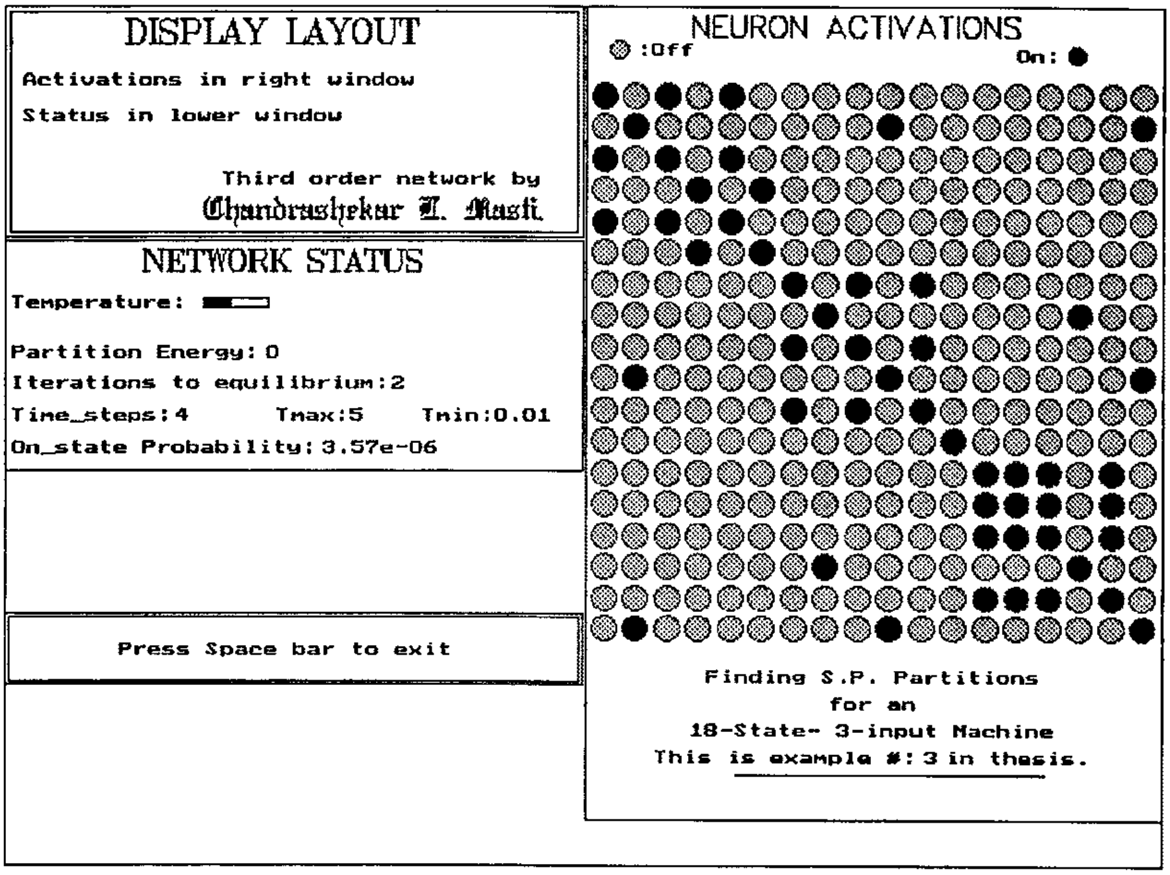
Finding S.P. Partitions

for an

18-State- 3-input Machine

This is example #: 3 in thesis.

Figure 4. 9. Time lapse of the network states.



DISPLAY LAYOUT

Activations in right window

Status in lower window

Third order network by
Chandrashekar L. Masti

NETWORK STATUS

Temperature:

Partition Energy: 0
Iterations to equilibrium:2
Time_steps:4   Tmax:5   Tmin:0.01
On_state Probability: 3.57e-06

Press Space bar to exit

NEURON ACTIVATIONS

▨ :Off   On: ●

Finding S.P. Partitions
for an
18-State- 3-input Machine
This is example #: 3 in thesis.

Figure 4. 10. Final solution state of the 18 × 18 net.

Trial number	S. p. partition	Time steps needed for solution
1	π_1	2
2	π_2	2
3	π_3	5
4	π_4	2
5	π_5	5
6	π_6	3
7	π_7	3
8	π_8	2
9	π_9	7
10	π_{10}	2
11	π_{11}	3
12	π_{12}	4
13	π_{13}	2
14	π_{14}	5
15	π_{15}	3

Table 4. 7. Results of the network performance to 15 trial runs.

Figure 4. 11 is the legend for the various s. p. partitions listed in table 4. 7. The annealing schedule used was the same as the one employed in previous examples. Other relevant statistics are illustrated within the figures.

This chapter presented the results of the simulations performed on a third-order neural network developed for addressing the s. p. partition problem in machine decomposition. The conclusions drawn from this research endeavor and the scope for future enhancements are discussed in the next chapter.

$$\pi_1 - \{\, \overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 13,\ 17};\ \overline{3,\ 7};\ \overline{5};\ \overline{6,\ 8,\ 10};\ \overline{11};\ \overline{12,\ 14,\ 15,\ 16} \,\},$$

$$\pi_2 - \{\, \overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 13,\ 17};\ \overline{3,\ 11};\ \overline{5};\ \overline{6,\ 8,\ 10};\ \overline{7};\ \overline{12,\ 14,\ 15,\ 16} \,\},$$

$$\pi_3 - \{\, \overline{0,\ 2,\ 4,\ 15};\ \overline{1,\ 9,\ 17};\ \overline{3,\ 6,\ 7,\ 8,\ 10};\ \overline{5};\ \overline{11};\ \overline{12,\ 13,\ 14,\ 16} \,\},$$

$$\pi_4 - \{\, \overline{0,\ 2,\ 4};\ \overline{1,\ 7,\ 9,\ 15,\ 17};\ \overline{3,\ 5,\ 11};\ \overline{6,\ 8,\ 10};\ \overline{12,\ 14,\ 16};\ \overline{13} \,\},$$

$$\pi_5 - \{\, \overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 17};\ \overline{3,\ 5,\ 15};\ \overline{6,\ 8,\ 10};\ \overline{7,\ 11};\ \overline{12,\ 14,\ 16};\ \overline{13} \,\},$$

$$\pi_6 - \{\, \overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 17};\ \overline{3,\ 6,\ 8,\ 10,\ 13,\ 15};\ \overline{5};\ \overline{7};\ \overline{11};\ \overline{12,\ 14,\ 16} \,\},$$

$$\pi_7 - \{\, \overline{0,\ 1,\ 2,\ 4,\ 6,\ 7,\ 8,\ 10,\ 13,\ 17};\ \overline{3,\ 5,\ 11};\ \overline{12,\ 14,\ 16};\ \overline{15} \,\},$$

$$\pi_8 - \{\, \overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 13,\ 17};\ \overline{3,\ 5,\ 11};\ \overline{6,\ 8,\ 10};\ \overline{7,\ 12,\ 14,\ 16};\ \overline{15} \,\},$$

$$\pi_9 - \{\, \overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 12,\ 14,\ 16,\ 17};\ \overline{3,\ 15};\ \overline{5,\ 13};\ \overline{6,\ 8,\ 10};\ \overline{7};\ \overline{11} \,\},$$

$$\pi_{10} - \{\, \overline{0,\ 2,\ 4,\ 15};\ \overline{1,\ 9,\ 13,\ 17};\ \overline{3};\ \overline{5,\ 11};\ \overline{6,\ 7,\ 8,\ 10,\ 12,\ 14,\ 16} \,\}$$

Figure 4. 11. Legend for s. p. partitions π_1 through π_{10} tabulated in table 4. 7.

(Contd.)

$$\boxed{\begin{aligned}
\pi_{11} &- \{\,\overline{0,\ 2,\ 4,\ 15};\ \overline{1,\ 9,\ 17};\ \overline{3,\ 7};\ \overline{5,\ 13};\ \overline{6,\ 8,\ 10};\ \overline{11};\ \overline{12,\ 14,\ 16}\,\}\,, \\
\pi_{12} &- \{\,\overline{0,\ 2,\ 4};\ \overline{1,\ 3,\ 7,\ 9,\ 17};\ \overline{5,\ 3,\ 15};\ \overline{6,\ 8,\ 10};\ \overline{11,\ 12,\ 14,\ 16}\,\}\,, \\
\pi_{13} &- \{\,\overline{0,\ 2,\ 4,\ 12,\ 14,\ 16};\ \overline{1,\ 9,\ 17};\ \overline{3,\ 7,\ 15};\ \overline{5};\ \overline{6,\ 8,\ 10};\ \overline{11,\ 13}\,\}\,, \\
\pi_{14} &- \{\,\overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 12,\ 14,\ 15,\ 16,\ 17};\ \overline{3};\ \overline{5,\ 7};\ \overline{6,\ 8,\ 10};\ \overline{11}\,\}\,, \\
\pi_{15} &- \{\,\overline{0,\ 2,\ 4};\ \overline{1,\ 9,\ 13,\ 15,\ 17};\ \overline{3};\ \overline{5};\ \overline{6,\ 8,\ 10,\ 11,\ 12,\ 14,\ 16};\ \overline{7}\,\}\,.
\end{aligned}}$$

Figure 4. 11. (Contd.) Legend for s. p. partitions π_{11} through π_{15} in table 4. 7.

CHAPTER V

CONCLUSIONS AND SCOPE FOR FUTURE RESEARCH

5.1. Conclusions

Present approaches to the machine decomposition problem have relied on a sequential and exhaustive search technique for generating partitions satisfying the substitution property. This is known to become intractable with growth in problem size and therefore imposes intemperate demands on computation resources. We have developed an intrinsically parallel approach to addressing the intractability involved in the partition search problem by the use of an artificial neural network model that has potential for offering a viable alternative to existing sequential search methods.

The problem of decompositions was cast into the framework of constraint satisfaction and a neural net model was developed to solve constraint satisfaction problems through optimization of a mathematically derived objective function over the problem space. The formulation strategies governing the derivation of objective functions for constraint satisfaction neural nets have so far been based on heuristics.

We have presented a formalized method for the construction of constraint satisfaction networks by exploiting useful theories from Boolean and relational algebras.

We have verified that the issue of decompositions belongs to a class of problems that are beyond the scope of solvability for second-order networks. A theorem was stated and proved establishing that third-order correlations must be extracted by a neural net to generate substitution property partitions. A third-order deterministic network such as a Hopfield net has been shown to fail in solving the s. p. partition search problem due to the lack of an adequate technique for escaping from any local minima of its associated objective function. Impressive results were obtained by using a third-order Boltzmann machine with simulated annealing.

A spectrum of problem sizes was addressed using a third-order Boltzmann machine network. In many cases, solutions were obtained in under five discrete time-steps. Due to the excellent convergence properties of the higher-order scheme in connection with the ground state characteristic of the derived energy function, it was possible to introduce intelligent engineering methods in identifying global solution states in arbitrary problem domains at fast rates. The network performance scales favorably with the state dimension of the problem space. However, the network as simulated currently does degrade in performance with respect to the input dimension. Intrinsic limitations behind the technique of a software simulation of an essentially parallel (neural) approach on a serial computing machine have precluded any substantial immediate improvements.

5.2 Future Research

Real world problems in decomposition can be expected to have large-sized state as well as input dimensions. As a result, a possible strategy for addressing the issue of scaling network performance in adequate proportion to the input dimension is to exploit the theory of the intersection of lattices of sub-groupoids [28], [29]. A suggested technique is to develop a separate network for each dimension in the input set of the problem space being addressed and incorporate a parallel communication link (a constraint) between the individual nets as they descend toward their respective global minima [41]. Common intersection points of individual global minima that reflect a valid, globally acceptable solution as determined overall by the complete dimension of the input space would then represent the required solution. A little insight suggests that such an approach has its potential in essentially a hardware implementation, since the larger the communication bandwidth simulated, the greater the number of iterative loops a single-CPU von Neumann-machine would be required to execute, rendering it an increasingly inadequate choice for the purpose. Our attempts at software simulations based on such a "decomposed approach to the decomposition problem" has provided corroborative results favoring our conviction toward emphasizing a future attempt on VLSI based hardware.

The majority of optimization/constraint satisfaction problems addressed to this date use neural nets with the order of interconnections between the neurons in the network being no higher than quadratic. The potential of higher order neural nets has been known for some time, but hardware limitations have so far constrained their use [42]. With the advent of high-speed, multi-level VLSI devices specially designed for parallel computation algorithms such as neural networks, we speculate that the foundations we have laid for a neural approach to the decomposition problem will one day obviate sequential methods altogether.

REFERENCES

[1] Fahlman, S. E., and Hinton, G. E., "Connectionist Architectures for Artificial Intelligence," *IEEE Computer, January 1987,* 100-109, IEEE Press, Piscataway, New Jersey.

[2] Feldman, J. A., "Connectionist Models and their Applications: Introduction," *Cognitive Science, 9,* 1-2, 1985.

[3] Livingston, D. L. and Serpen, G. S., "Evaluation of Lattice Theoretic Techniques for Task Decomposition Formalism," *Proceedings of IEEE SouthEastCon'89, Vol. III,* 1989.

[4] Serpen, G. S. and Livingston D. L., "A High Order Boltzmann Machine for Transfer Sequence Searches in Decomposed State Machines," *Proceedings of IJCNN-89 Washington D. C., Vol. II,* 598, 1989.

[5] Hartmanis, J. and Stearns, R. E., *Algebraic Structure Theory of Sequential Machines.* New Jersey: Prentice-Hall Inc., Englewood Cliffs, 1966.

[6] Hopfield, J. J. and Tank, D. W., "'Neural' Computation of Decisions in Optimization Problems," *Biological Cybernetics, 52,* 141-152, Springer-Verlag, 1985.

[7] Ramanujam, J. and Sadayappan, P., "Optimization by Neural Networks," *Proceedings of IJCNN-88 SanDiego, Vol. II,* 325-332, IEEE-INNS Press, June 1988.

[8] Tagliarini, G. A. and Page, E. W., "Solving Constraint Satisfaction Problems with Neural Networks," *IEEE First International Conference on Neural Networks, SanDiego, CA, Vol. III,* 741-747, IEEE-INNS Press, June 21-24, 1987.

[9] Hopfield, J. J. and Tank, D. W., "Computing with Neural Circuits: A Model," *Science, Vol. 233,* 625-633, August 1986.

[10] Tank, D. W. and Hopfield, J. J., "Simple 'Neural' Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Transactions on Circuits and Systems, CAS-33,* 533-541, 1986.

[11] Sejnowski, T and Rosenberg, C. R., "NETtalk: A Parallel Network that Learns to Read Aloud," *Johns Hopkins University Technical Report, JHU/EECS-86/01,* 1986.

[12] Rumelhart, D. E., Hinton, G. E. and Williams, R. J., "Learning Internal Representations by Error Propagation," in D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations.* MIT Press, 1986.

[13] Grossberg, S., *The Adaptive Brain I: Cognition, Learning, Reinforcement, and Rhythm, and The Adaptive Brain II: Vision, Speech, Language and Motor Control.* Amsterdam: Elsevier/North-Holland, 1986.

[14] Barto, A. G., Sutton, R. S. and Anderson, C. W., "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems," *IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-13, No. 5,* September/October, 1983.

[15] Lippmann, R. P., Gold, B. and Malpass, M. L., "A Comparison of Hamming and Hopfield Neural Nets for Pattern Classification," *MIT Lincoln Laboratory Technical Report, TR-769,* Cambridge: Massachusetts, MIT Press, 1988.

[16] Rumelhart, D. E. and McClelland, J. L., *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, 1986.

[17] Hinton, G. E. and Sejnowski, T. J., "Learning and Relearning in Boltzmann Machines," in D. E. Rumelhart and J. L. McClelland (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Vol. 1: Foundations.* MIT Press, 1986.

[18] Hummel, R. A. and Zucker, S. W., "On the Foundations of Relaxation Labelling Processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-5,* 267-287, 1983.

[19] Feldman, J. A. and Ballard, D. H., "Connectionist models and their properties," *Cognitive Science, Vol. 6, No.3,* 205-254, 1982.

[20] Rosenfield, A., Hummel, R. A. and Zucker, S. W., "Scene Labelling by Relaxation Operations," *IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-6,* 420-433, 1976.

[21] Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences, USA, 79,* 2554-2558, 1982.

[22] 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,* 3088-3092, May 1984.

[23] Kirkpatrick, S., Gelatt, C. D. Jr. and Vecchi, M. P., "Optimization by Simulated Annealing," *Science, 220,* 671-680, 1983.

[24] Fahlman, S. E., Hinton, G. E. and Sejnowski, T. J.,"Massively Parallel Architecture for AI: NETL, Thistle,and Boltzmann Machines," *Proceedings of the National Conference on Artificial Intelligence, AAA-83,* William Kaufman Inc., 1983.

[25] Ackley, D. H., Hinton, G. E. and Sejnowski, T. J., "A Learning Algorithm for Boltzmann Machines," *Cognitive Science, Vol. 9, No.1,* 147-169, 1985.

[26] Geman, S. and Geman, D., "Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-6, No.6,* 721-741, Nov 1984.

[27] Smolensky, P., "Schema Selection and Stochastic Inference in Modular Environments," *Proceedings of the National Conference on Artificial Intelligence, AAAI-83,* 109-113, William Kaufman Inc., 1983.

[28] Birkhoff, G., *Lattice Theory, American Mathematical Society Colloquium Publications, Vol. XXV,* Providence, Rhode Island: American Mathematical Society, 1967.

[29] Birkhoff, G. and Bartee, T. C., *Modern Applied Algebra.* New York: McGraw-Hill Book Company, 1970.

[30] Gill, A., *Applied Algebra for the Computer Sciences.* Prentice-Hall Series in Automatic Computation, New Jersey: Prentice-Hall Inc., Englewood Cliffs, 1976.

[31] Kohavi, Z., *Switching and Finite Automata Theory.* New York: McGraw-Hill Inc., 1970.

[32] Trembley, J. P. and Monahar, R., *Discrete Mathematical Structures with Applications to Computer Science.* New York: McGraw-Hill Inc., 1975.

[33] Aigner, M., *Combinatorial Theory.* New York: Springer-Verlag New York Inc., 1979.

[34] Ginzburg, A., *Algebraic Theory of Automata.* ACM Monograph series, New York: Academic Press, 1968.

[35] Karnaugh, M., "The Map Method for Synthesis of Combinational Logic Circuits," *Trans. AIEE, pt. I, Vol. 72, no. 9,* 593-599, 1953.

[36] Shannon, C. E., "A Symbolic Analysis of Relay and Switching Circuits," *Tans. AIEE, Vol. 57,* 713-723, 1938.

[37] Shannon, C. E., "The Synthesis of Two-Terminal Switching Circuits," *Bell System Technical Journal, Vol. 28,* 59-98, 1949.

[38] Keister, W., Ritchie, S. A. and Washburn, S., *The Design of Switching Circuits.* New York: D. Van Nostrand Company, 1951.

[39] Sejnowski, T. J., "Higher Order Boltzmann Machines," *AIP Conference Proceedings 151,* Snowbird: Utah, 1986.

[40] Hamming, R. W., "Error Detecting and Error Correcting Codes," *Bell System Technical Journal, Vol. 29,* 147-160, April, 1950.

[41] Masti, C. L. and Livingston, D. L., "Neural Networks for Addressing the Decomposition Problem in Task Planning," *Expert Systems and Other Real World Applications Track of the Proceedings of the International Joint Conference on Neural Networks, IJCNN-90-WASH-DC,* Washington, D. C., Jan. 15-19, IEEE-INNS Press, 1990.

[42] Lee, Y. C., Doolen, G., Chen, H. H., Sun, G. Z., Maxwell, T., Lee, H. Y. and Giles, L. C., "Machine Learning a Higher Order Correlation Network," *Physica 22D,* 276-306, North-Holland, Amsterdam, 1986.

APPENDIX

PROGRAM LISTING

```
/*
            FUNCTION PROTOTYPES FOR EXAMPLE3.LIB FILE.
**********************************************************************/
        void setup_for_graphics_display(void);
        void frame_the_screen(void);
        void main_menu_in_first_quadrant(void);
        void setup_activations(void);
        void setup_status_below_menu(void);
        void setup_exiting_info_below_status(void);
        void update_activations(void);
        void update_status(int );
        void display_probability(float );
/**********************************************************************
                        BOLTZMANN MACHINE
                    Graphics Display Routines file
*/
#include <graphics.h>
#include <dos.h>
#include "EXAMPLE3.H"
struct videoconfig
      {
      int      GraphicDriver;
      int      GraphMode;
      int      MaxX, MaxY;
      int      Maxcolors;
      double   AspectRatio;
      struct   palettetype palette;
       };
struct videoconfig config;
/**********************************************************************
void setup_for_graphics_display()
{
   extern struct videoconfig config;
   int    ErrorCode, xasp, yasp;
   config.GraphicDriver = DETECT;
   initgraph(&config.GraphicDriver,&config.GraphMode,"");
```

```
  ErrorCode = graphresult();
  if(ErrorCode != grOk)
    {
    printf("Graphics Initialization Error: %s\n",grapherrormsg(ErrorCode));
    exit(1);
    }
  config.MaxX      = getmaxx();
  config.MaxY      = getmaxy();
  config.Maxcolors = getmaxcolor() + 1;
  getpalette(&config.palette);
  getaspectratio(&xasp, &yasp);
  config.AspectRatio = (double)xasp/(double)yasp;
       return;
}
/*********************************************************************/
void frame_the_screen()
{
  void box(int, int, int, int, int);
  setbkcolor(BLACK);
        clearviewport();
        box(0,0,config.MaxX,config.MaxY,WHITE);
        return;
}
/*********************************************************************/
void main_menu_in_first_quadrant()
{
               struct viewporttype viewinfo;
               int w,h;
               void box(int,int,int,int,int);
               void changetextstyle(int,int,int);
               int gprintf(int *w,int *h,char *fmt,int);
               getviewsettings(&viewinfo);
               box(0,0,viewinfo.right/2,viewinfo.bottom/3-30,GREEN);
               box(3,3,(viewinfo.right/2)-3,(viewinfo.bottom/3)-30-3,WHITE);
               setfillstyle(INTERLEAVE_FILL,DARKGRAY);
               floodfill(4,4,WHITE);
               changetextstyle(TRIPLEX_FONT,HORIZ_DIR,2);
               settextjustify(CENTER_TEXT,TOP_TEXT);
               w = (viewinfo.right-viewinfo.left)/2 - 170;
               h = 3;
               gprintf(&w,&h,"DISPLAY LAYOUT",WHITE);
               h = textheight("x")+20;
               changetextstyle(DEFAULT_FONT,HORIZ_DIR,1);
               settextjustify(LEFT_TEXT,TOP_TEXT);
```

```
            w = viewinfo.left +10;
            gprintf(&w,&h,"Activations in right window",WHITE);
            h = textheight("x")+50;
            gprintf(&w,&h,"Status in lower window",WHITE);
            h = textheight("x")+55;
            w += 110;
            h = textheight("x")+85;
            gprintf(&w,&h,"Third order network by",CYAN);
            w = viewinfo.left +100;
            h = textheight("x")+95;
            changetextstyle(GOTHIC_FONT,HORIZ_DIR,1);
            gprintf(&w,&h," Chandrashekar L. Masti.",CYAN);
            changetextstyle(DEFAULT_FONT,HORIZ_DIR,1);
            return;
}
/*********************************************************************/
void box(int startx, int starty, int endx, int endy, int color)
{
   setcolor(color);
   line(startx,starty,startx,endy);
   line(startx,endy,endx,endy);
   line(endx,endy,endx,starty);
   line(endx,starty,startx,starty);
}
/*********************************************************************/
void changetextstyle(font,direction,charsize)
int font,direction,charsize;
{
            int ErrorCode;
            graphresult();
            settextstyle(font,direction,charsize);
            ErrorCode = graphresult();
            if(ErrorCode != grOk)
                        {
                        closegraph();
      printf("Graphic font loading error: %s\n",grapherrormsg(ErrorCode));
                        exit(1);
                        }
            return;
}
/*********************************************************************/
int gprintf(int *xloc,int *yloc,char *fmt,int color)
{
            char str[200];
            setcolor(color);
            strcpy(str,fmt);
            outtextxy(*xloc,*yloc,str);
            *yloc = textheight("x")+1;
            return;
}
/***********************************************************************/
void setup_activations()
{
            struct viewporttype viewinfo;
            int w,h;
            int x1,y1,x2;
            void box(int,int,int,int,int);
            void changetextstyle(int,int,int);
            int gprintf(int *w,int *h,char *fmt,int);
            int gulprintf(int *w,int *h,unsigned long fmt,int );
            getviewsettings(&viewinfo);

box((viewinfo.right/2)+2,0,viewinfo.right,viewinfo.bottom-25,MAGENTA);
            setfillstyle(SOLID_FILL,LIGHTGRAY);
            floodfill((viewinfo.right/2)+4,1,MAGENTA);
            changetextstyle(SANS_SERIF_FONT,HORIZ_DIR,1);
            settextjustify(CENTER_TEXT,TOP_TEXT);
            w = (viewinfo.right/2)+155;
            h = 0+1;
            gprintf(&w,&h,"NEURON ACTIVATIONS",YELLOW);
            setcolor(BLUE);
            circle(w-135,textheight("X")+8,5);
            setfillstyle(SOLID_FILL,LIGHTBLUE);
            floodfill(w-135+2,textheight("X")+8+2,BLUE);
            w -= 110;
            h = textheight("X") + 5;
            changetextstyle(DEFAULT_FONT,HORIZ_DIR,1);
            gprintf(&w,&h,":Off",BLACK);
            w = viewinfo.right-70;
            h += 15;
            gprintf(&w,&h,"On:",BLACK);
            setcolor(RED);
            circle(w+22,h+18,5);
            setfillstyle(SOLID_FILL,LIGHTMAGENTA);
            floodfill(w+22+2,h+18+2,RED);
            w = (viewinfo.right/2)+160; h = viewinfo.bottom-100;
            settextjustify(CENTER_TEXT,BOTTOM_TEXT);
            gprintf(&w,&h,"Finding S.P. Partitions",BLACK);
```

```
		w = (viewinfo.right/2)+160; h = viewinfo.bottom-85;
		settextjustify(CENTER_TEXT,BOTTOM_TEXT);
		gprintf(&w,&h,"for an",BLACK);
		w = (viewinfo.right/2)+160; h = viewinfo.bottom-70;
		settextjustify(CENTER_TEXT,BOTTOM_TEXT);
		gprintf(&w,&h,"  -State-  -input Machine",BLACK);
		w = (viewinfo.right/2) + 67; h = viewinfo.bottom-70;
		gulprintf(&w,&h,n,BLACK);
		w += 77; h = viewinfo.bottom-70;
		gulprintf(&w,&h,no_of_inputs,BLACK);
		w = (viewinfo.right/2)+160; h = viewinfo.bottom-55;
		settextjustify(CENTER_TEXT,BOTTOM_TEXT);
		gprintf(&w,&h,"This is example #:  in thesis.",BLACK);
		w = (viewinfo.right/2)+192; h = viewinfo.bottom-55;
		gulprintf(&w,&h,example_number,BLACK);
		x1 = (viewinfo.right/2)+85; y1 = viewinfo.bottom-50;
		x2 = x1 + 170;
		line(x1,y1,x2,y1);
		return;
}
/*********************************************************************/
void setup_status_below_menu()
{
		struct viewporttype viewinfo;
		int w,h;
		extern float Tmax, Tmin;
		void box(int,int,int,int,int);
		void changetextstyle(int, int, int);
		int  gprintf(int *w,int *h,char *fmt,int);
		int  ggprintf(int *w,int *h,float fmt,int ,int);
		int gulprintf(int *w,int *h,unsigned long fmt,int );
		getviewsettings(&viewinfo);
box(0,viewinfo.bottom/3-28,viewinfo.right/2,viewinfo.bottom/3 +100,GREEN);
		changetextstyle(TRIPLEX_FONT,HORIZ_DIR,1);
		settextjustify(CENTER_TEXT,TOP_TEXT);
		w = viewinfo.left+155;
		h = viewinfo.bottom/3 - 27;
		gprintf(&w,&h,"NETWORK STATUS",LIGHTGREEN);
		w = viewinfo.left +4;
		h += 145;
		changetextstyle(DEFAULT_FONT,HORIZ_DIR,1);
		settextjustify(LEFT_TEXT,TOP_TEXT);
		gprintf(&w,&h,"Temperature:",LIGHTGREEN);
		h = viewinfo.bottom/3 + 31;
		gprintf(&w,&h,"Partition Energy:",LIGHTGREEN);
		h = viewinfo.bottom/3 + 48;
		gprintf(&w,&h,"Iterations to equilibrium:",LIGHTGREEN);
		w = 213; h = viewinfo.bottom/3 +48;
	gulprintf(&w,&h,(unsigned long)Max_Equilibrium_steps,LIGHTGREEN);
		w = viewinfo.left +4;
		h = viewinfo.bottom/3 + 66;
		gprintf(&w,&h,"Time_steps:",LIGHTGREEN);
		w = 150; h = viewinfo.bottom/3 + 66;
		gprintf(&w,&h,"Tmax:",LIGHTGREEN);
		w += 40; h = viewinfo.bottom/3 + 66;
		ggprintf(&w,&h,Tmax,3,LIGHTGREEN);
		w += 40; h = viewinfo.bottom/3 + 66;
		gprintf(&w,&h,"Tmin:",LIGHTGREEN);
		w += 40; h = viewinfo.bottom/3 +66;
		ggprintf(&w,&h,Tmin,3,LIGHTGREEN);
		w = viewinfo.left +4; h = viewinfo.bottom/3 + 84;
		gprintf(&w,&h,"On_state Probability:",LIGHTGREEN);
		return;
}
/*************************************************************************/
void setup_exiting_info_below_status()
{
		struct viewporttype viewinfo;
		int w,h;
		void box(int,int,int,int,int);
		void changetextstyle(int,int,int);
		int gprintf(int *w,int *h,char *fmt,int);
		getviewsettings(&viewinfo);
box(0,viewinfo.bottom/2 +100,viewinfo.right/2,viewinfo.bottom-100,GREEN);
box(2,viewinfo.bottom/2 +102,viewinfo.right/2 -2,viewinfo.bottom-102,WHITE);
		setfillstyle(INTERLEAVE_FILL,DARKGRAY);
		floodfill(4,viewinfo.bottom/2 +104,WHITE);
		changetextstyle(DEFAULT_FONT,HORIZ_DIR,1);
		settextjustify(CENTER_TEXT,BOTTOM_TEXT);
		w = (viewinfo.right-viewinfo.left)/2 - 165;
		h = viewinfo.bottom/2 +125;
		gprintf(&w,&h,"Press Space bar to exit",LIGHTGRAY);
		return;
}
/*************************************************************************/
void update_activations()
{
		struct viewporttype viewinfo;
```

```
            int i,j;
            int x_center,y_center,radius;
            getviewsettings(&viewinfo);
            x_center = (viewinfo.right/2)+12; y_center = 50; radius = 7;
            for(i=0;i<n;i++)
                  {
                  for(j=0;j<n;j++)
                        {
                        if(V[i][j]==0)
                              {
                              setcolor(BLUE);
                              circle(x_center+j*17.5,y_center+i*17.5,radius);
                              setfillstyle(SOLID_FILL,LIGHTBLUE);
                  floodfill(x_center+j*17.5-2,y_center+i*17.5-2,BLUE);
                              }
                        else if(V[i][j]==1)
                              {
                              setcolor(RED);
                              circle(x_center+j*17.5,y_center+i*17.5,radius);
                              setfillstyle(SOLID_FILL,LIGHTMAGENTA);
                  floodfill(x_center+j*17.5-2,y_center+i*17.5-2,RED);
                              }
                        }
                  }
            return;
}
/*********************************************************************/
void update_status(int incomming_Energy)
{
            int w,h,y1,y2;
            double x1,x2,x2_box;
            int gulprintf(int *w,int *h,unsigned long fmt,int);
            void changetextstyle(int,int,int);
            changetextstyle(DEFAULT_FONT,HORIZ_DIR,1);
            settextjustify(LEFT_TEXT,TOP_TEXT);
            x1 = 110;     y1 = 163;
            x2_box = 5*Tmax/log(1+1);
            x2 = 110 + x2_box-1; y2 = 168;
            setfillstyle(SOLID_FILL,BLACK);
            setcolor(BLACK);
            bar(x1,y1,x2,y2);
            setcolor(RED);
            x2_box = 109 + x2_box; y1 = 163;
            x1 = 109; y2 = 168;
            rectangle(x1,y1,x2_box,y2);
            x1 = 109;     y1 = 164;
            x2 = 109 + 5*Temperature;    y2 = 167;
            setfillstyle(SOLID_FILL,LIGHTRED);
            setcolor(LIGHTRED);
            bar(x1,y1,x2,y2);
            x1 = 144;; y1 = 190;
            x2 = 169; y2 = 200;
            setfillstyle(SOLID_FILL,BLACK);
            setcolor(BLACK);
            bar(x1,y1,x2,y2);
            w = 144; h = 190;
            gulprintf(&w,&h,(unsigned long)incomming_Energy,LIGHTGRAY);
            w = 94;
            h = 225;
            gulprintf(&w,&h,step_number-1,BLACK);
            w = 94;
            h = 225;
            gulprintf(&w,&h,step_number,LIGHTGRAY);
            return;
}
/***********************************************************************/
int gulprintf(int *xloc,int *yloc,unsigned long fmt,int color)
{
            char str[1000];
            setcolor(color);
            ultoa(fmt,str,10);
            outtextxy(*xloc,*yloc,str);
            *yloc = textheight("x")+1;
            return;
}
/***********************************************************************/
int ggprintf(int *xloc,int *yloc,float fmt,int ndec,int color)
{
            char str[1000];
            setcolor(color);
            gcvt(fmt,ndec,str);
            outtextxy(*xloc,*yloc,str);
            *yloc = textheight("x")+1;
            return;
}
/***********************************************************************/
void display_probability(float Probability)
{
        int w,h;
        int x1,x2,y1,y2;
        int  ggprintf(int *w,int *h,float fmt,int ,int);
        x1 = 175; y1 = 243;
        x2 = 245; y2 = 258;
        setfillstyle(SOLID_FILL,BLACK);
        setcolor(BLACK);
        bar(x1,y1,x2,y2);
        w = 175; h = 243;
        ggprintf(&w,&h,Probability,3,LIGHTGRAY);
        return;
}
/*********************************************************************
                    THIRD-ORDER BOLTZMANN MACHINE
                          Using CSA schedule
For demonstrating solvability of Example_3 discussed in thesis.......

NOTE 1:     Reflexivity and Symmetry merit terms are unnecessary due to implicit
            benefits of current representation scheme being based on the relation
            matrix.
NOTE 2:     Trivial partitions can be avoided by using the right value for
            equal_size_blocks_enforcer and setting k_trivial not equal to zero.
            This is currently removed from the program to reduce the associated
            computation overhead.
NOTE 3:     Ensure that Tmax = 5.0 in the input file netpmts3.txt, to provide
            enough randomness during first few iterations.
NOTE 4:     Following include files are needed for displaying the graphics involved-
            EXAMPLE3.LIB and EXAMPLE3.H
NOTE 5:     Input files are -
            NETPMTS3.txt and
            STATES3.txt
NOTE 6:     This is an example drawn from the first 3 colums of Towers of Hanoi
            second level decomposition state table.
NOTE 7:     For this example, ensure that n = # states in the state machine = 18;
            no_of_inputs = 3 and finally, example_number = 3. These are the
            first three '#define' statements in this program.
*********************************************************************
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>
#include <math.h>
#include <string.h>
#include <values.h>
```

```
#define example_number 3      /* To display this on the run-time screen*/
#define    n      18            /* no_of_states in state m/c       */
#define no_of_inputs 3
#define Max_shuffle_no        10
#define Max_Equilibrium_steps 2
#define k_trnstvty   1*5    /* Transitivity scaling factor       */
#define k_NS               1*5    /* NSMPM scaling factor            */
#define shuffle_mat_max_dim     n*(n-1)/2
int shuffle_matrix[shuffle_mat_max_dim];
int          V[n][n];
int edge_matrix[no_of_inputs][n];
float  Tmax;
float  Tmin;
long   step_number;
float  Temperature;
#include "EXAMPLE3.LIB"
/*****************************************************************/
void main(void)
{
        int     ii,jj;
        int     Energy;
        int     Equilibrium_steps;
        int     shuffle_matrix_index;
        int     diff;
        int     net_input_V_ii_jj;
        int     Energy_pass_accounter[2];
        div_t x;
        void get_network_temperature_specs(void);
        void get_state_table_information(void);
        void randomly_initialize_neurons(void);
        void initialize_shuffle_matrix(void);
        void shuffle_neuron_indices_for_visit(void);
        void Evaluate_Energy(int *Energy);
        void Decide_on_updating_the_neuron(int *ii,int *jj,int );
        void check_for_user_termination(void);
        void Audio_signal_zero_energy_to_user(int );
        get_network_temperature_specs();
        get_state_table_information();
        randomly_initialize_neurons();
        initialize_shuffle_matrix();
        setup_for_graphics_display();
        frame_the_screen();
        main_menu_in_first_quadrant();
        setup_exiting_info_below_status();
        setup_activations();
        setup_status_below_menu();
        step_number = 0;
        do
                {
                step_number++;
                Temperature = Tmax/log(1+step_number);
for(Equilibrium_steps=0;Equilibrium_steps<Max_Equilibrium_steps;Equilibrium_
steps++)
                        {
                        shuffle_neuron_indices_for_visit();
                        shuffle_matrix_index=0;
                        while(shuffle_matrix_index<shuffle_mat_max_dim)
                                {
                                x = div(shuffle_matrix[shuffle_matrix_index],n);
                                jj = x.rem;ii = x.quot;
                                V[jj][ii] = V[ii][jj] = 1;
                                for(diff=0;diff<2;diff++)
                                        {
                                        Evaluate_Energy(&Energy);
                                        Energy_pass_accounter[diff] = Energy;
                                        V[jj][ii] = V[ii][jj] = 0;
                                        check_for_user_termination();
                                        }
        /* for loop for E@ "ON" state - E@ "OFF" state */
net_input_V_ii_jj = (Energy_pass_accounter[0]-Energy_pass_accounter[1]);
        Decide_on_updating_the_neuron(&ii,&jj,net_input_V_ii_jj);
        shuffle_matrix_index++;
                        }
        /* loop for reading shuffle_matrix         */
                        }
                /* loop for doing equilibrium steps */
                Evaluate_Energy(&Energy);
                update_status(Energy);
                update_activations();
                Audio_signal_zero_energy_to_user(Energy);
                } while(Temperature>=Tmin);
}
/***********************************************************************/
void get_network_temperature_specs()
{
                FILE *input_file1   = NULL;
                if(NULL != (input_file1 = fopen("netpmts3.txt","r")))
                        {
```

```
                        fscanf(input_file1,"%f\n",&Tmax);
                        fscanf(input_file1,"%f\n",&Tmin);
                        }
                else
                        {
                        puts(" Error opening input_file1");
                        puts(" Press any key to continue");
                        getche();
                        exit(0);
                        }
                if (fclose(input_file1)!= NULL)
                        {
                        puts(" Error closing input_file1");
                        puts(" Press any key to continue");
                        getche();
                        exit(0);
                        }
        return;
}
/*********************************************************************/
void get_state_table_information()
{
        FILE *input_file2   = NULL;
        int i,j;
        int dummy;
        char comma;
                if (NULL != (input_file2 = fopen("states3.txt","r")))
                {
                for (i=0;i<(no_of_inputs);i++)
                        {
                        for (j=0;j<n;j++)
                                {
                                fscanf(input_file2,"%d%c",&dummy,&comma);
                                edge_matrix[i][j] = dummy;
                                }
                        fscanf(input_file2,"\n");
                        }
                }
        else    {
                                puts(" Error opening input_file2");
                                puts(" Press any key to continue");
                                getche();
                                exit(0);
                                }
        if (fclose(input_file2)!= NULL)
                {
                puts(" Error closing input_file2");
                puts(" Press any key to continue");
                getche();
                exit(0);
                }
        return;
}
/**********************************************************************/
void randomly_initialize_neurons()
{
        int i,j;
        srand((int)time(NULL));
        for(i=0;i<(n-1);i++)
                {
                for(j=(i+1);j<n;j++)
                        {
                        V[j][i] = V[i][j] = (int)rand()%2;
                        }
                }
        for(i=0;i<n;i++)    V[i][i] = 1;
        return;
}
/**********************************************************************/
void initialize_shuffle_matrix()
{
        int shuffle_mat_index;
        int row,col;
        for(row=0,shuffle_mat_index=0;row<(n-1);row++)
                {
                for(col=(row+1);col<n;col++,shuffle_mat_index++)
                        {
                        shuffle_matrix[shuffle_mat_index] = (row*n + col);
                        }
                }
        return;
}
/**********************************************************************/
void shuffle_neuron_indices_for_visit()
{
int shuffle_matrix_index_one,shuffle_matrix_index_two,temporary;
int shuffle_no;
for(shuffle_no=0;shuffle_no<Max_shuffle_no;shuffle_no++)
```

```
        {
shuffle_matrix_index_one = ((int)rand()%shuffle_mat_max_dim);
shuffle_matrix_index_two = ((int)rand()%shuffle_mat_max_dim);
temporary = shuffle_matrix[shuffle_matrix_index_one];
shuffle_matrix[shuffle_matrix_index_one] = shuffle_matrix[shuffle_matrix_index_two];
shuffle_matrix[shuffle_matrix_index_two] = temporary;
        }
        return;
}
/***********************************************************************/
void Evaluate_Energy(int *Result)
{
        int      term3;
        int      NSM_term;
        void   Evaluate_NS_map_merits(int *NSM_term);
        void   Evaluate_transitivity_merit(int *term3);
               Evaluate_NS_map_merits(&NSM_term);
               Evaluate_transitivity_merit(&term3);
               *Result = (k_trnstvty*term3)+(k_NS*NSM_term);
               return;
}
/***********************************************************************/
void Evaluate_NS_map_merits(int *NSM_term_total)
{
               int i,j;
               int input;
               int isum, jsum;
               int index1, index2;
               *NSM_term_total = 0;
               for(input=0;input<no_of_inputs;input++)
                       {
                       isum = 0;
                       for(i=0;i<(n-1);i++)
                               {
                               jsum = 0;
                               index1 = edge_matrix[input][i];
                               for(j=(i+1);j<n;j++)
                                       {
                               index2 = edge_matrix[input][j];
        jsum += (V[i][j] + V[index1][index2] -(2*V[i][j]*V[index1][index2]));
                                       }
                               isum += jsum;
                               }
                       *NSM_term_total += isum;
```

```
                    }
            return;
}
/**********************************************************************/
void Evaluate_transitivity_merit(int *Total)
{
        int     i,j,k;
        int     tempterm3jsum,tempterm3ksum;
                *Total = 0;             /* transitivity merit */
                for(i=0;i<(n-2);i++)
                        {
                        tempterm3jsum = 0;
                        for(j=(i+1);j<(n-1);j++)
                                {
                                tempterm3ksum = 0;
                                for(k=(j+1);k<n;k++)
                                        {
tempterm3ksum += (V[i][j]*V[i][k]+V[j][k]*V[i][k]+V[i][j]*V[j][k] -
3*V[i][j]*V[i][k]*V[j][k]);
                                        }
                                tempterm3jsum += tempterm3ksum;
                                }
                        *Total += tempterm3jsum;
                        }
                return;
}
/**********************************************************************/
voidDecide_on_updating_the_neuron(int*row_index,int*col_index,intnet_input_V)
{
        float   parameter;
        float   Probability,Some_random_decision_number;
                parameter = ((float)net_input_V/(float)Temperature);
                if(parameter >= 709)
                        {
                        Probability = 1/(1+MAXDOUBLE);
                        }
                else    {Probability = 1/(1+exp(parameter));}
                display_probability(Probability);
                Some_random_decision_number = (rand()/(float)RAND_MAX);
                if(Probability>Some_random_decision_number)
V[*col_index][*row_index] = V[*row_index][*col_index] = 1;
                else    V[*col_index][*row_index] = V[*row_index][*col_index] = 0;
                return;
}
```

```
/*********************************************************************/
void check_for_user_termination()
{
        if(kbhit())
         {
         closegraph();
         restorecrtmode();
         clrscr();
         puts("Abnormal termination through keystroke!!!..........");
         puts("Type 'example3' again at the DOS prompt to re-run;");
         puts("Press a key now to exit to DOS.");
         getche(); getche();
         exit(0);
         }
   return;
}
/*********************************************************************/
void Audio_signal_zero_energy_to_user(int Energy)
{
        if(Energy= =0)
          {
          sound(2000); delay(70); nosound(); delay(70);
          sound(2000); delay(70); nosound(); delay(70);
          sound(2000); delay(70); nosound(); delay(70);
          sound(2000); delay(70); nosound(); delay(70);
          }
          return;
}
/*********************************************************************/
```

DATE DUE

JAN			
MAY 1 2 1997			