1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Produit en collaboration avec

Université de l'Ontario

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Parallel Computational Geometry on Analog Hopfield Networks

by

Natana Valiveti

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute of Computer Science

**School of Computer Science
Carleton University
Ottawa, Ontario
November 23, 1992**

© Copyright
1992, Natana Valiveti

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Votre titre / Votre référence

Our title / Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-84093-5

Canada

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis,
**Parallel Computational Geometry
on Analog Hopfield Networks**
submitted by
Natana Valiveti
in partial fulfilment of the requirements
for the degree of Master of Science

Director, School of Computer Science

Thesis Supervisor

Thesis Co-Supervisor

Carleton University

December 11, 1992

Abstract

This thesis presents a novel approach to solving Computational Geometry problems in parallel, by using *Analog Hopfield* Neural Networks, which are simplified models of the human brain. The use of massively parallel analog networks requires a radically different approach to geometric problem solving because (i) time is continuous instead of the usual discretized time step used for sequential or parallel processing, and (ii) geometric data is represented by analog components instead of the usual digital representation. We present *analog network* algorithms for the following geometrical problems:

- Minimum weight triangulation of planar point sets or of polygons with holes.
- Finding the smallest ε so that two given point sets are ε -congruent via translation.
- Partition a set of points into k clusters such that the maximum diameter of the clusters is minimized.

We also present an improvement to Hopfield's solution for the Euclidean Travelling Salesman Problem.

For each network, we present a detailed analysis of the network's parameters, together with proofs that the networks indeed produce feasible solutions. Experimental results presented in each chapter demonstrate the performance of our networks.

Acknowledgements

I have been very fortunate to have two excellent supervisors for my master's thesis, Prof. F. Dehne and J.-R. Sack. Their enthusiasm and encouragement was the sole reason for me to apply to the Master's program in Carleton. I am extremely grateful for their constant guidance and support extended during these two and half years as a graduate student. I thank you both whole heartedly.

I wish to acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada and the Faculty of Graduate Studies, Carleton University, for the duration of this Masters degree.

I also wish to thank Profs. R. Thomas (School of Business) and R. Sitter (Department of Math. and Statistics), for letting me use their SUN SPARC workstations. But for their help, some of the TSP experiments would still be running! Thanks are also due to Micheal Gastaldo and Daryl Graf for preliminary discussions on Neural Networks. I thank Prof. F. Fiala for giving me a PASCAL implementation of the 2 - *OPT* TSP heuristic. I wish to acknowledge Andrew Rau-Chaplin for valuable discussions and support. I extend my appreciation to Rosemary Carter, Judy Charlton, Peter Choynowski, Barbara Coleman, Brian Fultz, Janet Jansen, Jane MacArthur, Murali Thiyagarajah and Marlene Wilson, the superb Technical and Support Staff of the School of Computer Science who make this place a second home to all the graduate students.

I take this opportunity to thank Radha, who has always been a constant source of inspiration and guidance and to him I owe everything. My parents have been

very considerate during these years and but for my mother's stay here, I would still be completing the course requirements for the Master's degree! I thank Anjana for taking excellent care of Kashyap while I was able to put my neurons to work. Last, but not the least, I apologize to little Kashyap for not being able to provide him with undivided mother's attention.

Contents

Abstract	iii
Acknowledgements	iv
List Of Tables	ix
List Of Figures	x
1 Introduction	1
1.1 Neural Networks	1
1.1.1 The “Real” Neural System	2
1.1.2 Processing in the Human Brain	3
1.1.3 The “Artificial” Neural System	4
1.1.4 Threshold Functions	5
1.1.5 Interconnection Schemes	8
1.1.6 Learning in Artificial Neural Networks	11
1.1.7 Collective Computation	13
1.2 Sequential Computational Geometry	15
1.3 Parallel Computational Geometry	17
1.4 Analog Parallel Computational Geometry—Overview of Results	20
1.5 Organization of the Thesis	23

2	Hopfield Neural Networks	24
2.1	Discrete Hopfield Neural Networks	24
2.2	Analog Hopfield Neural Networks	28
2.3	Relation Between Stable States of the Two Models	35
2.4	Combinatorial Optimization Using Analog Hopfield Neural Networks	38
3	Minimum Weight Triangulation of a Planar Point Set	40
3.1	Some Sequential Algorithms	41
3.1.1	Greedy Triangulation	42
3.1.2	Delaunay Triangulation	43
3.1.3	Random Triangulation	44
3.2	Our Neural Network Solution	46
3.2.1	The Analog Network Design and Analysis	46
3.2.2	Experimental Results	51
3.2.3	Related Problems	54
4	Point Congruence	56
4.1	Our Neural Network Solution	57
4.1.1	Initialization of Network Parameters	58
4.2	Experimental Results	65
5	Geometric k-clustering problem	68
5.1	Problem Definition	69
5.2	Our Neural Network Solution	70
5.3	Experimental Results	76

6	The Travelling Salesman Problem	80
6.1	Hopfield's Neural Network Solution	82
6.2	Discussion of Hopfield's TSP Solution	84
6.3	Modifications to the Hopfield and Tank TSP formulations	87
6.4	Other Neural Network Models	91
6.5	Our Neural Network Solution to the TSP	93
6.6	Experimental Results	98
7	Conclusions	105
7.1	Summary of the Thesis	105
7.2	Unanswered Questions	106
	References	108

List of Tables

2.1	$[u_1, u_2]$ when β is varied between 0 and 1.	33
4.1	Performance of our neural network based ε -congruence	67
5.1	An example for grouping 8 points into 3 clusters.	71
5.2	Performance of our neural network based clustering	78
6.1	Performance of our neural network based TSP compared to Hopfield & Tank's Solution.	103
6.2	Mean percentage difference between the tour lengths produced by our neural network and the other sequential heuristics.	104

List of Figures

1.1	A Simplified Biological Neuron.	2
1.2	An Artificial Neuron.	4
1.3	A linear threshold function.	6
1.4	A Non-linear Ramp threshold function.	6
1.5	A step function.	7
1.6	The hyperbolic tangent function.	8
1.7	The logistic function.	9
1.8	A single layer neural network.	9
1.9	Feedforward and Feedback networks.	10
2.1	The mean firing rate of action potentials in a biological neuron.	28
2.2	A preliminary electrical model of an analog neuron.	29
2.3	A resistor-capacitor circuit.	29
2.4	The logical model of an artificial neuron.	30
2.5	The physical model of an artificial neuron.	30
2.6	Fixed points for a 2-neuron network.	33
2.7	High and low gain sigmoidal responses.	37
3.1	A counter example to the "greedy" solution.	44
3.2	A counter example to the Delaunay triangulation.	45
3.3	A 3-dimensional presentation of the triangulation network $TN(S)$	52

3.4	Performance of the triangulation network compared to some known sequential heuristics.	53
4.1	A possible mapping between sets $\mathcal{A}$ and $\mathcal{B}$	58
4.2	Row r and Column c of <i>selected</i> neurons.	62
4.3	An example of a one-to-one mapping identified by the neural network.	66
5.1	An <i>unselected</i> column c	74
5.2	An example of grouping 35 points into 5 clusters which were exactly identified by the neural network. The co-ordinates are scaled by 100.	79
6.1	A possible TSP solution for 4 cities.	82
6.2	Tour of 10 cities produced by our neural network.	100
6.3	Tour of 10 cities produced by the HT neural network.	100
6.4	Tour of 10 cities produced by Lin-Kernighan heuristic.	101
6.5	Tour of 20 cities produced by our neural network.	101
6.6	A Random tour of 20 cities.	102
6.7	Mean tour length given by our neural network (NN) in comparison with (a) Lin-Kernighan algorithm (LK) (b) 2-OPT algorithm ($2 - OPT$) (c) A Random tour ($RAND$), for various sizes of n	103

Chapter 1

Introduction

1.1 Neural Networks

The nervous system in human beings and animals excels in processing sensory information and performs very difficult computational tasks with enormous speed and accuracy. Modern digital computers, on the other hand, can perform numerical computations, like multiplying huge numbers at blinding speed. Yet, they cannot perform “seemingly” simple tasks like understanding unconstrained human speech. Artificial neural systems, which are biologically inspired, are based on mathematical models of the activity of the brain and the nervous system. Artificial neural systems or simply, Neural Networks (NN), are also widely known as analog computers, neurocomputers, adaptive systems, connectionist models, and parallel distributed processors [Simpson 1989]. The primary goal of the research in this area has been to build machines that mimic the capability of human information processing in tasks such as vision, speech, olfaction, knowledge processing and motor-control. The theory of artificial neural networks is the outcome of the convergence of many disciplines, including psychology, mathematics, neuroscience, physics, engineering,

computer science, philosophy, biology and linguistics. The motive behind this unification among the sciences is the pursuit of a common goal — namely, to build intelligent machines that will eventually reproduce human information processing!

1.1.1 The “Real” Neural System

The mammalian nervous system consists of highly interconnected nerve cells called *neurons*. The neuron communicates information to and from the various parts of the body. A simplified representation of the biological neuron is shown in Figure 1.1.

Figure 1.1 A Simplified Biological Neuron.

The three important parts of the neuron are the cell body called the *soma*, several spine-like extensions of the cell body called *dendrites* and a single nerve fiber called the *axon* that branches out from the soma and connects to many other neurons. Input signals from other neurons are received by the dendrites through connection points called the “synapse”. This connection between neurons mediates the “strength” with which a signal crosses from one neuron to the other. On receiving

a signal at the synapse, these inputs are transmitted to the cell body, where they are summed and averaged with all other such signals. If the average is sufficiently large, the cell “fires” by sending a signal through the axon to other neurons. There are some inputs which tend to excite the cell by causing it to “fire” and there are some others which inhibit such firing. In its *simplest* form, the biological neuron can be considered a threshold unit – a processing element that collects inputs and produces an output only if the sum of the input exceeds an internal threshold value. A brief look at how the human brain processes information will provide an insight into the capabilities of the mind – specifically, its adaptiveness and resilience.

1.1.2 Processing in the Human Brain

In the human brain there are a massive number of neurons, estimated to be between 10^{11} and 10^{14} , all of which are operating in a highly parallel and distributed manner. Input is channeled from many sources, and computation is done in parallel by the millions of nerve cells. The time taken to process a single piece of information from input to output is typically in the millisecond range while a modern digital computer can process one step of a program in a few nanoseconds. But the reason why the brain is far superior in information processing is that the processing is done in parallel while a computer processes information serially. Knowledge is stored in the interconnections between neurons and new information is added to the brain by adjusting these interconnection strengths – this mechanism is believed to be the reason for our ability to *generalize* knowledge. The biological neuron is not merely a threshold unit but is rather a very complex computing device. The computations not only take place in the soma but the dendrites and the synaptic interconnections are also actively computing [Grossberg 1982, Grossberg 1986]. Information is stored in a distributed manner whereby each idea or concept is stored not just in one neuron, but is spread across many neurons and their interconnections. This makes it highly fault tolerant, which means that damage to individual neurons does not degrade its overall performance [Simpson 1989, Hopfield *et al.* 1983].

Artificial neural systems strive to model only a restricted form of the information processing capability in humans.

1.1.3 The "Artificial" Neural System

Analogous to the biological neuron in the nervous system, the basic unit of an artificial neural system is the "artificial" neuron. These artificial counterparts of a biological neuron also known as processing elements, nodes, short-term memory, threshold logic units, or simply neurons, they are the computing components in an "artificial" neural system. A model neuron is shown in Figure 1.2.

Figure 1.2 An Artificial Neuron.

The neuron can be described by of a set of *inputs*, the *processor* and an *output*. Inputs come from all other neurons. There is an additional input from the environment which is known as external input or simply, input bias. Each of the inputs carry (or, are associated with) a weight which corresponds to the strength of the synaptic interconnection. The artificial neuron mimics the basic threshold function performed by the biological neuron. Input signals are multiplied by their corresponding weights and the weighted aggregate is then compared to an internal

threshold value. An output signal is generated only if the collected input signal strength exceeds the threshold. Using vector notation¹, we represent the inputs by $\underline{s} = [s_1, s_2, \dots, s_i, \dots, s_N]^t$, where s_i is the output of neuron i , $i = 1 \dots N$. The collection of weights that adjoins neuron i forms a vector $\underline{T}_i = [t_{1i}, t_{2i}, \dots, t_{ji}, \dots, t_{Ni}]$, where t_{ji} represents the strength of the connection from neuron j to i . The external input to neuron i is represented by ϵ_i . The computation that is performed in each neuron i can be expressed as follows:

$$s_i = g \left(\sum_{j=1}^N s_j \cdot t_{ji} + \epsilon_i \right)$$

where, $g()$ is a function to be described shortly. A weighted sum of the inputs and external bias is calculated and passed through a threshold function $g()$, which determines the output of the neuron. We shall now discuss the various threshold functions $g()$ commonly used in artificial neural systems.

1.1.4 Threshold Functions

Also known as activation functions, squashing functions, or signal functions, these functions map the possibly infinite input domain of neuron inputs to a prespecified range of output – typically, between 0 and 1. We desire these threshold functions to be monotonic. Commonly studied examples of the threshold function $g()$ are one of the following [Simpson 1989]:

A linear function: $g(x) = \alpha x$, where α is a constant and regulates the magnification of the internal activity x of the neuron (Figure 1.3).

Non-linear ramp function: The output of the neuron is confined to some range $[-\gamma, +\gamma]$ and $g()$ takes the form (Figure 1.4):

$$g(x) = \begin{cases} +\gamma & \text{if } x \geq \gamma \\ x & \text{if } |x| < \gamma \\ -\gamma & \text{if } x \leq -\gamma \end{cases}$$

¹We use a superscript t to denote the transpose of a vector.

Figure 1.3 A linear threshold function. **Figure 1.4** A Non-linear Ramp threshold function.

A step function: The neuron emits $+\gamma$ if its activity x is positive and $-\delta$ otherwise (Figure 1.5). The following represents such a step function:

$$g(x) = \begin{cases} +\gamma & \text{if } x > 0 \\ -\delta & \text{otherwise} \end{cases}$$

where, γ and δ are positive scalars. Often $\gamma = 1$ and $\delta = 0$, giving a binary output.

Figure 1.5 A step function.

Sigmoid function: This is the most widely used squashing function. The sigmoid function is an *S*-shaped function that is bounded, monotonic, non-decreasing and provides a graded, non-linear response. Sigmoid functions are, for example, the hyperbolic tangent function (Figure 1.6)

$$g(x) = \tanh(x),$$

and the augmented ratio of squares

$$g(x) = \begin{cases} x^2 / (1 + x^2) & \text{if } x > 0 \\ 0 & \text{otherwise.} \end{cases}$$

Figure 1.6 The hyperbolic tangent function.

The sigmoid function that is widely used in statistics, physics, chemistry and sociology, and which will also be used in this thesis is the *logistic function* (Figure 1.7):

$$\begin{aligned} g(x) &= \frac{1}{2} \left\{ \tanh\left(\frac{u}{\beta}\right) + 1 \right\} \\ &= \frac{1}{1 + e^{-2u/\beta}} \end{aligned}$$

where the parameter β is positive and less than unity and determines the steepness of the curve. For the remainder of this thesis let $\Delta \geq 19$ be a “large” value for which $\tanh(\Delta) \approx 1$ and hence $g(\Delta\beta) \approx 1$.

All of the above threshold functions (with the exception of the simple linear function) constrain the output of the neuron to intervals such as $[0, 1]$ or $[-1, 1]$, which enables a simple encoding of the results of a neural network.

1.1.5 Interconnection Schemes

The computational power of an artificial neuron can be realized to its full potential by connecting these simple neurons into networks to achieve massive parallelism.

Figure 1.7 The logistic function.

The simplest of such networks is the single-layer neural network, where we have a group of neurons arranged in a layer as shown in Figure 1.8.

Figure 1.8 A single layer neural network.

The circles on the left serve only as points of distribution for input and no computation is performed here. More complex models are built similarly by combining several such layers to form multi-layer neural networks. The output from one layer becomes the input (along with some weights) for another layer. These layers are also known as slabs or fields. The information flow amongst the processing elements in different layers could either be **feedforward** or **feedback**. In the feedforward

networks, the interconnection information propagates only in one direction, from one layer to another, whereas in the feedback network, there is information flow amongst processing elements in either direction. An example of such feedback and feedforward networks is shown in Figure 1.9. Once the neurons are connected to each other to form a network, in a feedforward or feedback fashion, tremendous computational power results [Wasserman 1989].

Figure 1.9 Feedforward and Feedback networks.

(a) Feedforward network

(b) Feedback network

Two important traits of the human brain which have intrigued researchers for decades are its ability to *learn* through training and its ability to *retrieve* or recall information on the basis of associated cues. The pioneering attempts that were made

in building artificial neural networks started with modeling networks that could learn by experience. The distinction between the characteristics of feedforward and feedback networks is better understood in the light of learning.

1.1.6 Learning in Artificial Neural Networks

The history of learning follows very closely the history of artificial neural networks themselves and dates back to 1943 when McCulloch & Pitts introduced the first mathematical model of an artificial neuron [McCulloch and Pitts 1943]. Their work created a lot of excitement because they were able to show how such networks of artificial neurons could exhibit sophisticated behaviour by making use of very simple computations [McCulloch 1965]. The major drawback in these networks was the lack of learning capabilities. Rosenblatt then generalized the McCulloch-Pitts model by adding the learning ability. This was the birth of the well-known *Perceptron*. Rosenblatt was able to exhibit and mathematically prove the learning capabilities in 2-layered perceptrons only. His attempts to extend the learning procedure to multi-layered perceptrons were futile [Rosenblatt 1957, Rosenblatt 1958b, Rosenblatt 1958a].

Widrow introduced an idea similar to that of a perceptron, called the adaptive linear element ('adaline', for short) [Widrow 1959, Widrow 1960]. Learning in a 2-layered *adaline* was demonstrated in [Widrow and Hoff 1960]. All this initial excitement was dampened when Minsky & Papert proved that there are severe restrictions on what the single-layered perceptrons can represent and hence, on what they can learn! This work, along with other studies on perceptrons is described in their influential book *Perceptrons* [Minsky and Papert 1969]. Several years went by without a theoretically sound learning algorithm for multi-layered neural networks. The interest was once again restored with the invention of the back-propagation learning algorithm by Rumelhart, Hinton & Williams for systematically training multi-layer artificial neural networks. This is also commonly known as the Generalized Delta Rule with Back Propagation of Error [Rumelhart *et al.* 1986].

Let us understand what *learning* exactly means in our context. The learning part of the network consists of modifying the internal system to reflect experience and produce a desirable behaviour pattern. A set of inputs and an expected output, together called a *training pair*, are presented to the network. The learning algorithm constantly adjusts the internal weights so that the desired output is matched closely. The network is usually trained over a number of training pairs and after repeated iterations of learning, the error between the actual outputs and target outputs is reduced to an acceptable value. At this point, the network is said to have been trained and the correct set of weights has been learnt. The network can now be used for recognition, i.e., to classify unknown test cases.

The learning procedure therefore consists of starting the network with a random set of weights and using the various training pairs to arrive at a set of weights in a feedforward manner. Feedforward networks have been traditionally used for tasks for which the desired output is known, for e.g., in pattern recognition and speech recognition [Pao 1989].

Learning could be classified further into *supervised* and *unsupervised* learning. Supervised learning is presented a training set consisting of an input vector and a desired output. As the network learns, its outputs are compared to the "ideal" output. In the unsupervised learning, the training set consists only of an input vector. The networks developed by Rosenblatt and Widrow & Hoff, incorporated supervised learning only. On the other hand, D.O. Hebb introduced a model for unsupervised learning [Hebb 1949]. This concept is based on the self-organization properties of the brain and hence is a far more plausible model of the biological learning exhibited in human beings. With an input vector presented to it, the network learns by self-organizing the data and the output of such a network must be generally transformed into a comprehensible form after the learning process. Unsupervised learning can be useful in situations where only a problem representation is known with no knowledge of the desired solution. For example, in areas like constrained optimization, the aim is to arrive at a feasible solution by choosing from a very large space of possible solutions. It is infeasible to come with such large training sets if only feedforward

networks are available. The main aim is to learn, i.e., solve problems without ever going through a training phase. For such problems, feedback networks have been successfully used [Hopfield and Tank 1985, Hopfield 1986], where the set of weights is fixed and learning involves in arriving at a feasible solution (or, an equilibrium point) while satisfying the constraints of the problem.

Feedback networks are highly interconnected and could be easily described in terms of an undirected graph, where the vertices represent the neurons and the edges represent the synaptic connections. They form a complete graph where every neuron is connected to every other neuron. These networks were originally proposed as a model of computation for solving signal decision and decoding problems in pattern recognition. The research in artificial neural networks took a new direction when Hopfield & Tank showed successful applications of such feedback networks to solve problems in combinatorial optimization [Hopfield 1982, Hopfield 1984, Hopfield and Tank 1986a, Tank and Hopfield 1987]

In summary, we have two types of learning, one, where the set of weights are learnt with the help of the desired output (supervised learning), and the other, where the set of weights is fixed and the network arrives at the desired output using these weights (unsupervised learning).

1.1.7 Collective Computation

In this section, the following question is addressed: "How do such neural networks compute?" The answer lies in the manner in which the artificial neurons are connected to one another. The computational behaviour exhibited by neural networks is significantly different from the computations performed by a traditional computer. The rich interconnection among the neurons allows them to participate in the computation process as a whole. The basic process of 'computation' is a series of state changes, starting at an initial state and terminating at a state that corresponds very closely, if not exactly, to an expected answer. In a traditional computer, this process can be pictured as a path from the initial state to the desired state, through the configuration space of the computer, as it evolves in time. The initial state is provided

by the program and the input data, and state changes take place in a step-by-step (i.e., discrete time steps) manner until the final state is reached. In neural circuits, this computation process is significantly different. The neurons, being connected to one another interact in a collective manner continuously, until the required result is obtained.

Once the final state is reached, the network is in a stable state and is said to have converged. Stability occurs when all the inputs are mapped to fixed points (also called convergence points, equilibrium points or limit points). The information about the solution is implicit in the fixed point. Stability in neural networks is described by three important theorems [Cohen and Grossberg 1983], all of which are based on a method due to Lyapunov. The Lyapunov's method calls for designing a function of the variables of a dynamical system (i.e., the neural network), where the derivatives of the function with respect to time, exhibit certain properties. [Simpson 1989] gives a detailed discussion on the global stability theorems and the Lyapunov method.

The whole process of starting at an initial state and gradually moving to a stable state can be visualized as a path through a terrain of hills and valleys, which we call the *Energy* terrain. The Energy will be explicitly defined as a mathematical quantity in Chapter 2. In 1982, John Hopfield showed that the dynamics of a single-layer of artificial neurons can be described by means of an energy equation [Hopfield 1982]. He also showed that this cleverly crafted energy equation would always force a system to converge to a local minimum, under certain circumstances, for all possible inputs. Using this model, Hopfield was able to build artificial neural systems for applications of great interest, e.g., content addressable-memory and finding near-optimal solutions to NP-complete problems [Hopfield and Tank 1986b, Tank and Hopfield 1986].

Hopfield introduced two neural network models, namely, the *Discrete Hopfield Neural Networks* [Hopfield 1982] and the *Analog Hopfield Neural Networks* [Hopfield 1984]. In this thesis, we will study *Analog Hopfield Neural Networks*. In addition to being a feedback network for supervised learning, this model has

the property that the collective computation process on the network's analog data proceeds in *continuous* time. In Chapter 2 we will describe the Hopfield model in detail.

1.2 Sequential Computational Geometry

In this section we present a brief overview of this fairly new discipline of Computer Science, which deals with the computational complexity of geometric problems. The name "Computational Geometry" was given by M.I. Shamos in 1975 [Shamos 1975] and since then it has seen an enormous growth of research. It is now regarded as an important discipline in the area of Design and Analysis of Algorithms. The development of several interesting algorithmic techniques are attributed to this field, some of which are incremental construction, plane-sweep, geometric transformation, prune-and-search and dynamization. We now discuss briefly five major areas of geometric problems.

Convex Hulls: Given a set S of n points in 2-dimensional space, the convex hull of S , denoted as $CH(S)$ is the smallest convex polygon that contains the set S . The computation of convex hulls has applications in areas like pattern recognition [Duda and Hart 1973] and image-processing [Rosenfeld 1969]. Numerous algorithms for computing the convex hull of a set of points exist and their complexity is either $O(n \log n)$, $O(nH)$, or $O(n \log H)$ where n is the number of points and H is the number of points on the hull [Graham 1972, Jarvis 1973, Akl and Toussaint 1978, Kirkpatrick and Seidel 1986].

Intersections: Computing the intersection or detecting the intersection of convex polygons, star-shaped polygons and line segments is a basic computational problem in computer graphics, VLSI design, pattern recognition and architectural design. This problem also arises in finding the kernel of a polygon, or computing the visibility of a polygon from a point or an edge. An optimal algorithm for computing the intersection of two

convex polygons is due to [Shamos and Hoey 1976]. Chazelle presents an $O(n \log^2 n / \log \log n + k)$ time and $O(n + k)$ space algorithm to find all intersecting pairs of n line segments [Chazelle 1984]. The optimal algorithm to compute the intersection of n line segments in $O(n \log n)$ time (plus output) is due to [Chazelle and Edelsbrunner 1988]. A collection of some intersection problems with their reported solutions can be found in the survey by [Lee and Preparata 1984].

Geometric Searching: Given a geometric database and a sample data item, *searching* consists of locating this data item in the database. The cost measures associated with searching are the query time, storage space, preprocessing time and update time. The two major paradigms of geometric searching are:

1. **Location:** The database represents a partition of the geometric space into regions and the question is to identify the region in which a query point lies. Algorithms of varying complexity exist for solving this problem. The pioneering ideas were introduced in [Dobkin and Lipton 1976], who give an $O(\log n)$ time, using $O(n^2)$ storage and $O(n^2)$ preprocessing time algorithm for locating a point in a planar-straight-line-graph [Preparata and Shamos 1985] on n vertices. There are several optimal search algorithms, one of which is due to [Kirkpatrick 1983] and runs in $O(\log n)$ time, using $O(n)$ storage and $O(n \log n)$ preprocessing time.
2. **Range-search problems:** Given a database, the query is in the form of a standard geometric shape, for e.g., a rectangle. Given n points, a possible query is to report all the points that lie in a given rectangle. Numerous data structures were introduced for solving range-searching problems (for e.g., Bentley's *range tree*) and are discussed in [Bentley 1975, Bentley and Maurer 1980, Bentley and Shamos 1977, Lee and Wong 1977].

Proximity and Related Problems: One basic proximity problem can be stated as follows: Given a set of n points in the plane, find the two points that

are closest. Some of the more difficult problems are computing the Euclidean Minimum Spanning Tree [Garey *et al.* 1976] and the Triangulation of n points in the plane [Lloyd 1977]. The Voronoi diagram [Roger 1964] is an important tool for solving some proximity problems like nearest-neighbour search or k nearest neighbours search [Dickerson *et al.* 1992]. Other problems include, finding the diameter of a set of points, triangulation of a simple polygon, quadrilateralization of a rectilinear polygon and decomposition of polygons into convex parts [Lee and Preparata 1984].

Geometric Optimization Problems: They are a class of geometric problems which are of a combinatorial nature. There are a number of problems in this class which are NP-complete or NP-hard [Garey and Johnson 1979]. Some of the geometric optimization problems are finding the smallest enclosing circle [Megiddo 1983], finding the largest empty rectangle [Chazelle *et al.* 1984], p -center and p -median problems (both of which are shown to be NP-hard [Kariv and Hakimi 1979a, Kariv and Hakimi 1979b]), the Euclidean Traveling Salesman Problem [Papadimitriou 1977] and the disc-cover problem [Papadimitriou 1981].

A comprehensive coverage of all the above problem areas can be found in the first book in this field, *Computational Geometry: An Introduction* [Preparata and Shamos 1985]. Other books which cover special topics in Computational Geometry are [Edelsbrunner 1987], [Mehlhorn 1984] and [Du and Hwang 1993].

1.3 Parallel Computational Geometry

Parallel Computational Geometry deals with the design and analysis of algorithms for geometric problems, based on *parallel* models of computation. The surge of interest in *parallel algorithms* for geometric problems is due to the nature of some of the applications which often require very fast and real time computation. Parallel computation involves dividing the problem amongst several processors where each one

of the processors co-operates and contributes to the final solution. This results in a major reduction in the computation time. In contrast to the traditional von Neumann computer, several models of computation have been proposed. The computer architecture being employed strongly influences the design of efficient algorithms. Some of the commonly used architectures are the Parallel Random Access Machine (PRAM), the Hypercube, the Linear Array, the Mesh-connected Computer, Butterfly Network, Cube-Connected Cycles, and the Pyramid. A detailed discussion of the various computational models can be found in [Akl 1989].

The pioneering work in the field of *Parallel Computational Geometry* was done by Chow [Chow 1980] and since then scores of problems have been solved in all the major areas of Computational Geometry. The field of Parallel Computational Geometry has been well surveyed in a recent book [Akl and Lyons 1993] and a number of recent papers [Goodrich and Yap 1991, Dehne and Sack 1988]. In this section we shall not attempt a survey of this field but briefly mention some of the important problems studied. For a thorough discussion of each of these topics, the interested reader is urged to refer to [Akl and Lyons 1993].

1. Computing the convex hull of a set of points is probably the most widely studied problem in parallel computational geometry. Numerous algorithms to compute the convex hull for a variety of parallel computational models have been proposed, some of which are due to [Chow 1981, Dehne *et al.* 1986, Wang and Chen 1990, Holey and Ibarra 1990].
2. The class of intersection problems has been extensively studied for the PRAM, Mesh, Linear Array and Cube-Connected Cycles. For example, the detection of line segment intersections in parallel can be done in $O(\sqrt{n})$ time using $O(n)$ processors on a Mesh [Miller and Stout 1989].
3. Reif & Sen present an $O(\log n)$ time randomized algorithm for locating n points in an arrangement of line segments using a Butterfly machine with $O(n)$ processors [Reif and Sen 1990], while Cole gives an algorithm for the CREW PRAM for locating a point in a triangulated subdivision in $O(\log n)$ query

- time using $O(n/\log n)$ processors, $O(n)$ space and $O(\log n)$ pre-processing time [Cole and Zajicek 1990].
4. A parallel algorithm to compute the portion of a polygon P that is visible from a point p inside P is due to [Dehne 1988] for a Mesh of processors of size $O(n)$ running in $O(\sqrt{n})$ time.
 5. Parallel algorithms have also been designed to solve the all-nearest neighbours problem. For example, an $O(n \log n / \log N)$ running time algorithm for a linear array of size $O(N)$ is presented by [Atallah and Tsay 1989]. Querying the nearest-neighbour can be accomplished on a Mesh of size $O(n)$ in $O(\sqrt{n})$ time [Miller and Stout 1989] and determining the closest pair of points from a set S on a hypercube can be done in $O(\log^2 n)$ time using $O(n)$ processors [Stojmenovic 1988].
 6. [Jeong and Lee 1990] present an $O(\sqrt{n})$ time algorithm to compute the Voronoi diagram of planar point sets on a Mesh. An algorithm running on a CRCW PRAM model can compute the Voronoi diagram of a planar point set in $O(\log^3 n)$ time using $O(n)$ processors [Evans and Stojmenovic 1989]. [Goodrich *et al.* 1989] give an $O(\log^2 n)$ time algorithm for computing the Voronoi diagram of a set of line segments on the CREW PRAM model using $O(n)$ processors.
 7. An algorithm to compute the trapezoidal decomposition and triangulation a simple polygon on a hypercube of size $O(n \log n)$ in time $O(\log^2 n)$ is given in [Dehne and Rau-Chaplin 1990], whereas [Jeong and Lee 1990] solve the same problem on a Mesh in time $O(\sqrt{n})$ using $O(n)$ processors. In [Atallah *et al.* 1989] cascading divide-and-conquer is used to construct a trapezoidal decomposition of a simple polygon in $O(\log n)$ time using $O(n)$ processors for a CREW PRAM.

8. Computing the shortest path from a point inside a simple polygon P to all the vertices in P can be accomplished on a CREW PRAM model in $O(\log^2 n)$ time using $O(n)$ processors [ElGindy and Goodrich 1988].

To summarize, the above results are based on either the PRAM or the network model of parallel computation. We now present a new model of parallel computation in which the computing elements are not digital components, but rather analog electronic circuits [Millman and Halkias 1975]. This model is typically referred to as the *Analog Computer* – based mainly for solving simultaneous differential equations.

1.4 Analog Parallel Computational Geometry— Overview of Results

Many geometric problems require considerable algorithmic effort but are solved by humans with surprising speed and ease. Consider, for example, the problems of determining whether two planar point sets are ε -congruent, or determining a collision-free motion of an object in the presence of obstacles. The human brain is able to solve these problems with the least amount of effort, whereas, complicated algorithms are required for a conventional machine to arrive at a desirable solution. In the previous literature, sequential and/or parallel computational geometry has focussed on processing *discrete data* in *discrete time steps*. The main idea in this thesis is to employ *Analog Hopfield Neural Networks* to solve computational geometry problems. In contrast to the traditional algorithmic methods to solve problems (sequential and/or parallel), our approach manipulates *analog data* in *continuous time*. The two approaches are radically different and require both different architectures and different problem solving techniques. The manipulation of analog geometric data in continuous time is how the brain is believed to solve geometric problems. Maybe it is in this distinction that we will find the explanation to the human brain's superior performance in geometric problem solving.

The contribution of this thesis is:

1. To present analog network designs for the following geometrical problems:

- minimum weight triangulation of planar point sets or of polygons with holes,
- finding the smallest ϵ so that two given point sets are ϵ -congruent via the geometric transformation of translation,
- partition a set of points into k clusters such that the maximum diameter of the clusters is minimized.

2. To present an improvement to Hopfield's solution for the Euclidean Travelling Salesman Problem.

In the standard discrete model, most of these problems either have high polynomial time solutions or are conjectured/known to be NP-hard/complete.

Efficient solutions to computationally hard problems are by nature heuristics. The main advantage of *Analog Hopfield* Neural Networks in contrast to standard digital heuristic methods is that analog networks produce (in general) virtually instantaneous results. The disadvantage for the designer is that the solution must be expressed within the particular framework of the system of differential equations describing the dynamic behaviour of such a network. This poses an additional challenge compared to using standard heuristics or optimization techniques.

There are two possible approaches to solving optimization problems using Analog Networks: one which strives at finding the best approximation to the "optimal" solution, but may produce "infeasible" solutions; while the other guarantees feasibility of the solution but may not necessarily produce an equally good approximation. A particular application may determine which approach is more appropriate.

In the first method, the objective function of the optimization problem has to be expressed in the form of the Lyapunov energy function designed by Hopfield (this is discussed in detail in Chapter 2). The function is carefully constructed so that its global minimum corresponds to the solution of the original optimization problem. The weights of the neural network are derived by a term-by-term comparison

of the formulated energy function and the “standard form” suggested by Hopfield [Hopfield 1984]. This is the approach used by Hopfield & Tank in their solution to the TSP, and has been adopted by several researchers who applied analog Hopfield nets to solve other optimization problems. The advantage of this method is that the *global* minimum of the analog network exactly yields the *optimal* solution to the optimization problem. The output vector of the network, if it reaches the global minimum, is certainly one that meets the constraints posed. On the other hand, the syntactic correctness of the solution for the *local* minima *cannot* be guaranteed. Given this uncertainty, the network produces very good solutions if its output is syntactically correct. But the associated risk is that we may get invalid solutions. For example, Hopfield’s solution for the Traveling Salesman Problem (TSP) [Hopfield 1984, Hopfield and Tank 1985, Hopfield and Tank 1986a] neither guarantees that the reported tour is optimal nor that it is even valid (and sometimes the network does actually report “invalid” tours).

We have taken a different approach in this thesis. We have placed great emphasis on always producing feasible solutions. Given this stringent requirement, the method of working “backward” from the energy function does not appear promising. We have instead used the problem constraints to decide the weights of the interconnections. The details of the objective function have been incorporated into either the weights or the bias, depending on the problem. The result is that the energy in a stable state is proportional to the objective function for the problem and we establish this in the Lemmas. It appears that the accuracy vs. feasibility tradeoff is often unavoidable. However, a synthesis of these two approaches may be valuable and may perform better than any one approach in isolation.

In our solution to the TSP, we establish that our network *always* produces a *valid* tour. We give in our network analysis, rigorous proofs of the feasibility of the solution. We also show that our minimum weight triangulation network always produces a valid triangulation and that the energy of the network, which is known to be minimized when the network reaches a local minimum in a stable state, is proportional to the total length of the selected edges (plus a fixed constant). We

have not found any such analysis of the network in the existing literature describing analog networks for other problem areas.

1.5 Organization of the Thesis

The chapters of this thesis are organized as follows. In Chapter 2 we discuss in detail the two models introduced by Hopfield, namely, the *Discrete Hopfield Neural Networks* and *Analog Hopfield Neural Networks*. Chapter 3 contains a detailed description of our analog network design for the minimum weight triangulation problem. Analytical derivations of the sufficient conditions for the network's parameters to always produce a feasible solution, and proofs of correctness of the solution are also presented. The design of analog networks for other related geometric problems is discussed in Section 3.2.3. The analog neural network solution for the approximate point congruence problem is presented in Chapter 4. It includes a description of the analog network, derivation of the network parameters and also proofs of correctness of our solution. Chapter 5 discusses the method for solving the geometric k -clustering problem using *Analog Hopfield Networks*. Finally, we present a novel approach to solve the well known Travelling Salesman Problem in Chapter 6. A literature survey of earlier solutions to the TSP, and proofs of correctness of our solution are also presented.

Chapter 2

Hopfield Neural Networks

The artificial neural network model proposed by Hopfield in 1982 is the discrete model in which all the neurons have only two states as output [Hopfield 1982]. This was later refined to include neurons whose output was continuous and was confined to the interval $[0, 1]$ [Hopfield 1984].

2.1 Discrete Hopfield Neural Networks

The building block used in these networks is identical to the artificial neuron shown in Figure 1.2. Each neuron has only two states, '0' and '1', as outputs, like those originally modeled by McCulloch & Pitts [McCulloch and Pitts 1943]. If the output is a '0', the neuron is said to make an inhibitory connection, a '1' indicates an excitatory connection. The input to each neuron i comes from two sources, namely, outputs from all other neurons, and an external (bias) input. The total input into neuron i , denoted as u_i , is therefore:

$$u_i = \sum_{j \neq i} s_j \cdot t_{ij} + \epsilon_i$$

where, t_{ij} is the strength of the connection from neuron i to neuron j . Unconnected neurons have a strength of 0. Computation using a network of such neurons involves changing the states of these neurons in time. The discrete model describes such a system as changing the states stochastically. Each neuron samples its input at

random times with a fixed mean attempt rate of W . Also, each neuron i has a fixed internal threshold Θ_i . (Unless otherwise stated, Θ_i is assumed to be 0). The computation performed in each neuron is simple. An aggregate of the inputs is compared with the internal threshold and according to the following rule an output is produced:

$$s_i = \begin{cases} 1 & \text{if } \sum_{j \neq i} s_j \cdot t_{ij} + \epsilon_i > \Theta_i \\ 0 & \text{if } \sum_{j \neq i} s_j \cdot t_{ij} + \epsilon_i < \Theta_i \end{cases} \quad (2.1)$$

Thus, each neuron randomly and asynchronously evaluates whether the weighted sum of inputs is above or below its internal threshold and produces an outputs accordingly. The important differences between this model and the Perceptron are, firstly, the Perceptrons have only feedforward connections and secondly, they require synchrony, like the conventional digital computer. This model, on the contrary, is based on a strong feedback recall and the asynchrony was introduced to represent the real neural system in which delays, jitter and noise are present. It was shown in [Hopfield 1982] that a network of such discrete asynchronous neurons exhibit very useful collective computational properties, one of which is Content-Addressable-Memory (CAM) or Associative Memory. The memory in a conventional random-access machine is significantly different from an associative memory which has the following characteristics [Pao 1989]:

1. The ability to store many associated pattern pairs, namely, a stimulus and a response.
2. Storage of patterns is distributed and accomplished via a self-organization process.
3. When a stimulus pattern is received, either in complete or distorted form, the memory should be able to reproduce the associated response pattern.

A function called the *energy function* was formulated, whose minima correspond to some stored patterns or memory. The network is started with some random initial state, which represents partial information about some memory pattern, and in the

process of minimizing the energy function, a final stable state is reached, which corresponds to the full information about that pattern. The important point is that the memory is reached not by knowing an address, as in conventional computers, but by knowing a part of the memory in the initial state. That is, the memory is addressable by *content*, rather than by *location*. Computer simulations to study the effectiveness of the discrete network of neurons as an associative memory were conducted and for detailed experimental results, the interested reader is referred to [Hopfield 1982].

The energy function which represented the state space of the discrete Hopfield model is the following:

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j - \sum_i \epsilon_i s_i + \sum_i \Theta_i s_i. \quad (2.2)$$

where, t_{ij} is the interconnection strength and represents a portion of some memory. For example, to store a vector $\underline{g}$ of N states, the matrix of interconnection strengths is obtained as follows:

$$t_{ij} = \sum_{k=1}^N (2s_i^k - 1)(2s_j^k - 1)$$

and $t_{ii} = 0$. The condition that the T matrix be symmetric with zero-diagonal elements is essential for the above energy function to converge to a stable state [Hopfield 1982]. This requirement, though it seems restrictive, is not unrealistic as the brain does exhibit connections of the kind implied by a symmetric matrix [McCarrager and Chase 1985, Essen and Maunsell 1983].

Theorem 2.1 *For a network of two-state neurons with symmetric, loop-free interconnections (no self-connections), the energy function given by*

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j - \sum_i \epsilon_i s_i + \sum_i \Theta_i s_i.$$

is always decreased if one neuron of the network changes its state.

Proof:

Consider the change ΔE in E when neuron k changes its state by Δs_k with all other neurons j remaining in the same state.

$$s_k \rightsquigarrow s_k + \Delta s_k$$

$$s_i \rightsquigarrow s_i, \quad i \neq k$$

We now calculate the value of the energy function E' ,

$$\begin{aligned} E \rightsquigarrow E' &= -\frac{1}{2} \left\{ \sum_i \sum_j t_{ij} s_i s_j + 2 \sum_i T_{ik} s_i \Delta s_k \right\} - \left\{ \sum_i \epsilon_i s_i + \epsilon_k \Delta s_k \right\} \\ &\quad + \left\{ \sum_i \Theta_i s_i + \Theta_k \Delta s_k \right\} \end{aligned}$$

regrouping the terms.

$$\begin{aligned} E' &= -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j - \sum_i \epsilon_i s_i + \sum_i \Theta_i s_i \\ &\quad - \sum_i T_{ik} s_i \Delta s_k - \epsilon_k \Delta s_k + \Theta_k \Delta s_k \\ &= E - \Delta s_k \left(\sum_i T_{ik} s_i + \epsilon_k - \Theta_k \right) \end{aligned}$$

$$\text{therefore, } \Delta E = -\Delta s_k \left(\sum_i T_{ik} s_i + \epsilon_k - \Theta_k \right).$$

$$\text{Let } Z_k = \sum_i T_{ik} s_i + \epsilon_k - \Theta_k.$$

$$\text{then, } \Delta E = -\Delta s_k \cdot Z_k.$$

The term Z_k determines the sign of Δs_k . If $Z_k \geq 0$, then neuron k is set to 1, hence $\Delta s_k \geq 0$, and ΔE becomes negative (i.e., $\Delta E \leq 0$). If $Z_k \leq 0$, then neuron k is set to 0, hence $\Delta s_k \leq 0$, and once again ΔE becomes negative (i.e., $\Delta E \leq 0$). Hence ΔE is always negative. This means that the rate of change of the energy function E using two-state neurons is always negative, implying that in equilibrium, E will settle in a stable state. $\square$

It was shown in [Kamgar-Parsi and Kamgar-Parsi 1987] that the same energy function will not converge to a stable state for a T matrix with non-zero diagonal elements. In such a case, it could get trapped in cycles. They also show that, even with a symmetric T matrix with zero diagonals, the *Discrete Hopfield Network*, can

be trapped in a cycle if *more* than one neuron changes its state at the same time (synchronous change).

2.2 Analog Hopfield Neural Networks

The analog model is an extension of the discrete model in the sense that the outputs of the neurons are no longer restricted to being either 0 or 1, but that they are in the continuous interval of $[0, 1]$. The motivation behind this is to build an artificial system which realistically models the biological nervous system. The biological neurons are known to communicate with each other by means of pulses of electrical activity called "action potentials". What we desire is that the relation between the input u_i of neuron i and its output s_i be described in terms of a function $g(u_i)$ which is typically a sigmoid, because it is known that the rate at which action potentials are generated, is in general, a smooth sigmoid function [Perkel and Bulloch 1969] (shown in Figure 2.1) of the input of that particular neuron [Hopfield 1986]. This choice of $g(u_i)$ can be electrically modeled since it resembles the input-output characteristic of a typical non-linear amplifier. Thus, a preliminary electrical model of a continuous-state neuron looks as shown in Figure 2.2.

Figure 2.1 The mean firing rate of action potentials in a biological neuron.

Figure 2.2 A preliminary electrical model of an analog neuron.

In a fully interconnected network formed with these neurons as building blocks, the input u_i of neuron i is derived from the output of other neurons. The biological neurons produce action potentials at a rate determined by the *total* synaptic input current. Therefore, it is required that u_i does not immediately follow the changes in the output of other neurons, but lags behind in time. This lag can be electrically modeled using the classical *resistor-capacitor* circuit (RC circuit, shown in Figure 2.3) because, the time delay is its typical characteristic.

Figure 2.3 A resistor-capacitor circuit.

Using the well known Kirchoff's Current Law [Millman and Halkias 1975], it can be shown that the output voltage u obeys the differential equation

$$RC \cdot \frac{du}{dt} + u = V$$

or, $C \cdot \frac{du}{dt} + \frac{u}{R} = \frac{V}{R}$ (2.3)

With this background, the *logical* model for an artificial neuron i can be refined to be as shown in Figure 2.4. It can be seen that the total input to neuron i is $\sum t_{ij} \cdot s_j + \epsilon_i$. The corresponding *physical* model is shown in Figure 2.5.

We can now derive the behaviour of a network of artificial neurons as shown in Figure 2.5, using Kirchoff's Laws. The output voltage u_i follows the usual behaviour

Figure 2.4 The logical model of an artificial neuron.

Figure 2.5 The physical model of an artificial neuron.

given in equation (2.3). That is,

$$\begin{aligned} \sum_{j \neq i} \left(\frac{s_j - u_i}{R_{ij}} \right) + \epsilon_i &= C_i \cdot \frac{du_i}{dt} + \frac{u_i}{\rho_i} \\ C_i \cdot \frac{du_i}{dt} + u_i \left\{ \frac{1}{\rho_i} + \sum_{j \neq i} \frac{1}{R_{ij}} \right\} &= \epsilon_i + \sum_{j \neq i} \frac{1}{R_{ij}} \cdot s_j \\ \text{substituting, } \frac{1}{R_i} &= \frac{1}{\rho_i} + \sum_j \frac{1}{R_{ij}} \\ \text{and, } t_{ij} &= \frac{1}{R_{ij}} \end{aligned} \quad (2.4)$$

$$\text{we get, } C_i \cdot \frac{du_i}{dt} = -\frac{u_i}{R_i} + \sum_{j \neq i} t_{ij} s_j + \epsilon_i \quad (2.5)$$

Note that in the system of equations (2.5), R_{ij} is the resistor connecting the output of neuron j to the input line of neuron i , C_i is the total input capacitance of the amplifier i and ρ_i is the input resistance of the amplifier i . Therefore, the complete behaviour of a network of N neurons can be described by a system of differential equations given in the following equation (2.6) with one equation per neuron. This system of equations predicts the *dynamics* of the neural network as time elapses.

$$C_i \cdot \frac{du_i}{dt} = \sum_{j=1}^N t_{ij} s_j - \frac{u_i}{R_i} + \epsilon_i \quad i = 1, \dots, N \quad (2.6)$$

In a system of equations given in (2.6), the outputs of the neurons can lie anywhere in the continuous interval of $[0, 1]$. The main challenge is to encode the problem in such a way that the network's behaviour is predictable and that it actually solves the given problem, i.e. it converges to an equilibrium state which encodes a solution to the problem. The advantage of this approach is that, even with many neurons, the time for an analog network to settle into a stable state is typically extremely small.

For ease of representation, it is customary to normalize the differential equations given in (2.6) to the one given in the following equation (2.7). Note that the term u_i/R_i will be denoted simply by u_i , and $1/R_i$ is as defined in equation (2.4).

$$\dot{\underline{u}} = -\underline{u} + T\underline{s} + \underline{\epsilon} \quad (2.7)$$

where,

$$\begin{aligned}\underline{u} &= [u_1, \dots, u_N]^t, & \underline{\dot{u}} \text{ denotes the vector of time derivatives of } \underline{u} \\ \underline{s} &= [s_1, \dots, s_N]^t, & 0 \leq s_i \leq 1 \\ \underline{\epsilon} &= [\epsilon_1, \dots, \epsilon_N]^t.\end{aligned}$$

T is the $N \times N$ matrix of t_{ij} values, and

$$s_i = g(u_i) = \frac{1}{2} \left\{ \tanh\left(\frac{u_i}{\beta}\right) + 1 \right\}, \quad 0 < \beta < 1, \quad (2.8)$$

Now, equation (2.7) can be used to derive the equation which governs the stable state $\underline{u}$. In the stable state $\underline{\dot{u}} = 0$, therefore,

$$\begin{aligned}\underline{\dot{u}} &= -\underline{u} + T\underline{s} + \underline{\epsilon} = 0 \\ \text{or, } \underline{u} &= T\underline{s} + \underline{\epsilon}.\end{aligned} \quad (2.9)$$

Let us now illustrate the idea of stable points or fixed points for the simple case of a 2-neuron network. For simplicity, we assume that the bias vector is 0. We use for illustration, the T matrix used by Hopfield in his experiments on a 2-neuron network [Hopfield 1984]:

$$T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

For this system, in equilibrium, equation (2.9) implies:

$$\begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} g(u_1) \\ g(u_2) \end{bmatrix}$$

$$\text{or, } u_1 = g(u_2)$$

$$u_2 = g(u_1).$$

Graphically, the solution can be visualized as shown in Figure 2.6. The parameter β in the g function determines the steepness of the curve. The closer β is to 0, the steeper the curve is and the stable state values are closer to 1 or 0. This can

Figure 2.6 Fixed points for a 2-neuron network. **Table 2.1** $[u_1, u_2]$ when ..

varied between 0 and 1.

β	$[u_1, u_2]$
0.1	[1.000, 1.000]
0.7	[0.9359, 0.9359]
0.998	[0.8446, 0.8446]

be readily seen from the experimental results given in Table 2.1 for the 2-neuron network where, the threshold function used was $g(u_i) = \frac{1}{2}(\tanh(\frac{u_i}{\beta}) + 1)$.

In general, the energy function which represents the state space for a continuous model is a Lyapunov function [Chetayav 1961] for the equations in (2.7) and is given by :

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j + \sum_i \left[\frac{1}{R_i} \int_0^{s_i} g^{-1}(s_i) ds \right] - \sum_i \epsilon_i s_i \quad (2.10)$$

Theorem 2.2 (T-SYM) *The energy function given in equation (2.10) is a strictly decreasing function of time and, for a symmetric T matrix, settles in a local minimum. (Note that diagonal elements of the T matrix need not be zero).*

Proof:

(A sketch of the proof appears in [Hopfield 1984]). Recall that the inputs to the neurons are determined by the solution to the system of differential equations :

$$C_i \frac{du_i}{dt} = \sum_{j=1}^N t_{ij} s_j - \frac{u_i}{R_i} + \epsilon_i$$

The derivative for the energy function (2.10) with respect to time is:

$$\begin{aligned} E &= -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j + \sum_i \left[\frac{1}{R_i} \int_0^{s_i} g^{-1}(s_i) ds \right] - \sum_i \epsilon_i s_i \\ \frac{dE}{dt} &= -\sum_i \frac{ds_i}{dt} \left(\sum_{j=1}^N t_{ij} s_j - \frac{u_i}{R_i} + \epsilon_i \right) \\ &= -\sum_i \frac{ds_i}{dt} \cdot C_i \frac{du_i}{dt} \quad (\text{by equation (2.6)}) \\ &= -\sum_i \frac{ds_i}{dt} \cdot C_i \cdot g^{-1}'(s_i) \cdot \frac{ds_i}{dt} \quad (\text{since } u_i = g^{-1}(s_i)) \\ &= -\sum_i C_i \cdot g^{-1}'(s_i) \cdot \left(\frac{ds_i}{dt} \right)^2 \end{aligned}$$

Each term in the above summation is non-negative since C_i is always positive and $g^{-1}(s_i)$ is a monotonically increasing function (therefore, $g^{-1}'(s_i) \geq 0$). Therefore, $\frac{dE}{dt}$ is always less than or equal to 0. $\square$

Analog Hopfield Nets are not just theoretical models of biological neurons but are also attractive architectures from a practical perspective. They have hardware realizations in VLSI (CCD and NMOS) and fiber optics technologies [Mead 1989, Sage 1987, Sage *et al.* 1986, Simpson 1989, Sivilotti *et al.* 1986]. The architectural simplicity of analog networks permits the construction of circuits with very large numbers of neurons. We shall now discuss the relation between the stable states of the discrete Hopfield model and the analog Hopfield model.

2.3 Relation Between Stable States of the Two Models

We have two models, namely, the discrete stochastic model, where the neurons have only two states and state updates are performed asynchronously and the continuous, deterministic model in which the neurons have a graded response. For a given T matrix, is there a correspondence between the stable states of the two models? For simplicity of understanding, let us consider the case in which the thresholds and the external inputs are both 0. The discrete system operates in a state space defined by the *corners* of an N -dimensional hypercube $[0, 1]^N$ and has the following underlying energy function for a symmetric T matrix with zero diagonals:

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j \quad (2.11)$$

and the continuous system moves in a state space which includes the *interior* of the N -dimensional hypercube $[0, 1]^N$ with the energy function defined by:

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j + \sum_i \left[\frac{1}{R_i} \int_0^{s_i} g^{-1}(s_i) ds \right] \quad (2.12)$$

For a continuous system, with a well-behaved T matrix, the maxima and minima of the first term in equation (2.12) lie at the *corners* of the N -dimensional hypercube. If the T matrix is positive or negative definite, it is possible that the extrema lie in the interior of the hypercube. This is usually not the case for matrices which store information in CAM applications or in optimization problems [Hopfield 1984]. In

the discrete case, the search for extrema of the first term of equation (2.11) takes place only in the corners, hence for a given T matrix, the extrema are exactly the same in both the stochastic and deterministic cases.

To understand the second term's contribution, we need to look at the sigmoid function $g(u_i)$:

$$g(u_i) = \frac{1}{2} \left\{ \tanh\left(\frac{u_i}{\beta}\right) + 1 \right\}$$

where, $0 < \beta < 1$, is the scaling factor which changes the steepness of the curve. Recall that a β close to 0 represents a system with a very high gain and the sigmoid curve is almost step-like and a β close to 1 corresponds to a low gain with a flat sigmoid curve (Figure 2.7). For ease of interpretation of output from a neural network, we are interested only in a high-gain case because we have an output vector in which the terms are either close to 1 or 0. Interpretation of intermediate values becomes more difficult. Therefore, in a high-gain case, the second term of the energy equation (2.12), becomes negligible, thus making the maxima and minima of the energy in equation (2.12) the same those of energy in equation (2.11).

Hence, the stable points of the high-gain, continuous, deterministic system correspond to the stable points of the discrete, stochastic system. In the case of a lower gain (i.e., a less steep output function), the continuous system will have fewer stable points than the discrete system (for the same T matrix), but the stable states will still correspond to some existing stable state in the stochastic system. This simple correspondence is possible because of the quadratic form of the interaction between the different neurons in the energy function [Hopfield 1984]. There are other complicated energy functions which have been occasionally used to solve constraint satisfaction problems but the stable states fail to have a correspondence to the stable states in a discrete two-state model [Kirkpatrick *et al.* 1983, Geman and Geman 1984]. The main advantage of the analog model over the discrete one is that the analog network is (in general) guaranteed to settle in a stable state, whereas, the discrete network may be trapped in cycles and never converge.

Figure 2.7 High and low gain sigmoidal responses.

(a) High Gain Response

(b) Low Gain Response

2.4 Combinatorial Optimization Using Analog Hopfield Neural Networks

The basic idea in most algorithms for optimization is to move through the space of possible solutions and progress in a direction that tends to minimize or decrease the value of the cost function associated with the problem. The cost function is chosen in such a way that 'low costs' represent good solutions in that problem domain. The aim is to assign values to each of the variables in such a way that all constraints for the problem are satisfied and the cost is minimized. In [Hopfield and Tank 1985], the authors propose a mapping of problems having high combinatorial complexity (e.g., the class of NP-complete and NP-hard problems) onto such networks as described in the earlier sections. The idea is to map the problem onto a neural network in such a way that the network configurations (i.e., the different states of the network) correspond to the possible solutions for the problem.

In order to map a problem onto a neural network, the following steps are required:

1. We need to choose a representation scheme whereby the outputs from the neurons can be directly related to a solution of the problem at hand.
2. Choose an energy function E , appropriate to the problem so that at different configurations representing the possible solutions to the problem, this energy E is proportional to the cost function of the problem. Hence, minimization of E will correspond to the "best" solution of the problem that was mapped.
3. The next important step is to derive the weights t_{ij} and also the external inputs c_i from the energy function designed in step 2 above.

A variety of combinatorial problems have been mapped onto Artificial Neural Networks, some of which are the following (for a comprehensive list of problems see [Simpson 1989][pp.56-59]):

- Graph Colouring and Graph Manipulation [Dahl 1987],
[Ramanujam and Sadayappan 1988]

- Scheduling and Resource allocation [Tagliarini and Page 1988], [Van den Bout and Miller 1988], [Gulati and Iyengar 1987]
- Solving equations and optimizing functions [Takeda and Goodman 1986], [Jeffrey and Rosner 1986]
- Load Balancing and Programming Parallel Computers [Barhen *et al.* 1987], [Simmes 1987]
- Various signal and image processing applications [Winters 1988], [Koch 1987], [Zhao and Mendel 1987], [Provence 1987]

In demonstrating the power of their neural network for solving complex optimization problems, Hopfield & Tank used the well known NP-hard optimization problem, namely, the Travelling Salesman Problem (TSP) [Hopfield and Tank 1985, Hopfield and Tank 1986b]. We discuss their solution in Chapter 6 and also present a survey of other solutions to the TSP. In the chapters to follow, we present the mappings on to the Analog Hopfield Network for a few computational geometry problems which are known or conjectured to be NP-complete.

Chapter 3

Minimum Weight Triangulation of a Planar Point Set

Triangulation of point sets or polygons is one of the most studied problems in Computational Geometry. It is a fundamental problem with a multitude of applications in areas like, CAD, Robotics, Pattern Recognition, GIS, etc., (see e.g., [Preparata and Shamos 1985]). In addition to the above applications, triangulation is an important tool for solving other geometric problems. Solutions to problems like planar point location [Kirkpatrick 1983], polyhedron intersection, finding the shortest path in a polygon [Guibas *et al.* 1987], data structuring [Levcopoulos *et al.* 1989] and guard placement in a simple polygon [Avis and Toussaint 1981, Fisk 1978, O'Rourke 1987], to name a few, require the surface to be pre-triangulated. In this chapter, we study this problem by giving a formal definition and then, present a neural network solution.

Let $S = \{s_1, s_2, \dots, s_n\}$ be a set of n distinct points in the plane in general position. These points s_i are called **vertices**. Let $E = \{e_1, e_2, \dots, e_N\}$ be the set of all possible line segments between the vertices of S , where $N = n(n-1)/2$. The line segments e_i are called **edges**. We present a few definitions.

Definition 3.1 *Two edges e_1 and e_2 ($e_1 \neq e_2$) properly intersect if they intersect at a point which is not an endpoint of e_1 or e_2 .*

(Note that for ease of presentation we will omit the word **properly** from our discussions).

Definition 3.2 *A triangulation of a planar point set S is a planar embedded straight-line graph $G = (S, E)$ with the vertex set S and a maximal set E of non-intersecting edges.*

In other words, any additional edge between the vertices in S intersects at least one edge of E . Thus, every region of G interior to the convex hull of S is a triangle. In order to define the minimum weight triangulation of a point set, consider a weight function assigning a positive weight to each of the edges $e_i \in E$. In practice, the weight assigned to an edge is often the Euclidean distance between its endpoints.

Definition 3.3 *The weight of a triangulation is defined to be the sum of the weights of all its edges. A minimum weight triangulation of S is a triangulation of S which has the minimum weight among all the possible triangulations of S .*

The minimum weight triangulation problem was originally studied by Duppe & Gottchalk [Duppe and Gottschalk 1970], and later by Shamos & Hoey [Shamos and Hoey 1975]. Of course, the weight of a triangulation is one measure by which the “quality” of a triangulation can be assessed. There are other criteria that constitute a “good” triangulation. For example, in applications like finite-element methods, finding good triangulations involves either maximizing or minimizing the angles of the generated triangles. These and other methods for triangulations are discussed in [Babuska and Aziz 1976, Bramble and Zlamal 1970, Bern *et al.* 1992].

3.1 Some Sequential Algorithms

It is interesting to observe that for any n there is a point set S of size n , for which the number of possible triangulations is large and is given by the Catalan number $C(n) = \Omega(4^n/n^{3/2})$. Hence, finding the minimum weight triangulation by the brute-force method of exhaustive search will yield an exponential time algorithm, and is

therefore a poor strategy for determining the minimum weight triangulation. It is an open problem whether finding the minimum weight triangulation for point sets in the plane is NP-complete [Garey and Johnson 1979]. A different version of the problem is the *triangulation existence* problem, stated as follows: Given a set of vertices V , and a subset $L \subset E$, does there exist a subset $T \subset L$, such that T is a triangulation of V ? Lloyd proved the NP-completeness of this triangulation existence problem [Lloyd 1977]. Even though finding the minimum weight triangulation of a point set may be intractable, it is interesting to note that polynomial algorithms to triangulate a polygonal domain exist! [Klincsek 1980].

Once a problem is known or conjectured to belong to the class of NP-complete problems, we have to rely on heuristics to give a good approximation to the actual solution. Since the triangulation problem is *most probably* NP-hard, approximate solutions to finding the minimum weight triangulation of a finite point set will have to be employed. The well-known heuristics for the triangulation problem are the classical **greedy** method of triangulation [Duppe and Gottschalk 1970, Lingas 1986, Lingas 1987] and the Delaunay triangulation [Shamos and Hoey 1975], both of which are very good approximations in general, but there exist point sets for which they behave poorly [Manacher and Zobrist 1979]. The neural network solution which we present aims to perform very well, always staying very close to the optimal solution for all point sets. Before we present our neural network solution, we present very briefly the sequential heuristics.

3.1.1 Greedy Triangulation

This method employs the well-known *greedy* principle, where an edge of the triangulation is added one at a time, always choosing the edge with the shortest length first. An edge is added to the triangulation only if it does not intersect all the edges already present. The algorithm terminates when the required number of edges in the triangulation have been found. It is known that the number of edges in a triangulation is $3n - k - 3$, where, k is the number of edges on the convex hull of the

point set and n , is the size of the point set. The algorithm is outlined in Algorithm Greedy below:

Algorithm Greedy

Input: A set of points $S = \{s_1, s_2, \dots, s_n\}$.

Output: A Triangulation of the point set S given by set of edges in T .

Begin

Step 1: Let E be the set of all edges with endpoints in S .

Step 2: Sort all the edges in E according to their edge lengths.

Step 3: $T \leftarrow \emptyset$.

Step 4: While $|T| \neq 3n - k - 3$ do:

Step 4a: Let w be an edge of least weight in E .

Step 4b: For each edge $e_i \in T$ check if w intersects e_i .

If yes, go to Step 4c, otherwise repeat process at Step 4a

Step 4c: $T \leftarrow T \cup \{w\}$

End

Algorithm 3.1: Algorithm Greedy

The claim that the greedy method of triangulation produces a minimum weight triangulation of a point set was disproved by means of a counter example [Lloyd 1977]. Figure 3.1 shows one such example where the greedy method does not produce the minimum weight triangulation.

3.1.2 Delaunay Triangulation

An alternate method of triangulating a set of points is by constructing the Voronoi diagram for the point set and then constructing its straight-line dual [Shamos and Hoey 1975]. The Voronoi diagram for a point set is the partition of the plane into regions, where each region is the locus of points (x, y) closer to a point of S than to any other point of S . These Voronoi regions are convex polygonal regions. The straight-line dual of the Voronoi diagram is the graph obtained by adding a straight-line segment between each pair of points of S whose Voronoi regions share

Figure 3.1 A counter example to the "greedy" solution.

an edge. (See [Preparata and Shamos 1985] for a wealth of information on Voronoi diagrams and their properties).

The claim by Shamos & Hoey that the Delaunay triangulation produce a minimum weight triangulation was once again disproved [Lloyd 1977]. In Figure 3.2 we give a counter example where the Delaunay triangulation does not produce the minimum weight triangulation.

3.1.3 Random Triangulation

Another method of triangulation is to randomly permute all the edges in set E and select edges one after another until we have the desired $3n - k - 3$ edges, which form a triangulation of the point set. This algorithm generates a *valid* triangulation, but is far from being a minimum weight triangulation. The random triangulation algorithm differs from the Greedy Algorithm, in Step 2, where instead of sorting the edge set, a random permutation of the edges is generated. The rest of the algorithm proceeds in a manner identical to the Greedy Algorithm.

Figure 3.2 A counter example to the Delaunay triangulation.

3.2 Our Neural Network Solution

We now describe the construction of an analog neural network for solving the minimum weight triangulation problem. We describe the network in terms of the initial state, the external inputs and the interconnection weights and then prove that every stable state of the network corresponds to a *valid* triangulation of the given point set. We also show that minimizing the weight of the triangulation is equivalent to minimizing the energy of the network. We conclude this section by presenting the experimental results which demonstrate the performance of the network.

3.2.1 The Analog Network Design and Analysis

Our analog triangulation network, called $TN(S)$, for the minimum weight triangulation problem is an *Analog Hopfield Networks* with $N = \frac{n(n-1)}{2}$ neurons. Each edge connecting two points of S is assigned to a unique neuron i , and the edge assigned to neuron i is referred to as edge e_i . In an equilibrium state as reached by the network, an edge e_i is called **selected** if the corresponding neuron i is selected. The set of all such selected edges is the **output** produced by the network. It will be shown that the output of the network is always a (valid) triangulation of the point set, and that minimizing the energy of the network is equivalent to minimizing the weight of the output triangulation.

In a preprocessing phase, we compute the weights l_i of all edges e_i , together with the maximum weight l_{max} . Furthermore, we determine for each pair of edges whether or not they intersect. We now define the network by setting up the interconnection weight matrix T , the vector of all external inputs (or bias) denoted by ϵ , and the vector of the initial states of all the N neurons. The bounds for the constants B , C_1 , C_2 used for the elements of T and the bias ϵ , respectively, will be derived in the Lemmas to follow. The initial state for each neuron is set to be 0.5 plus a random noise, which is given by the constant r . Since we wish to have the amplifier outputs determined by a very steep sigmoid curve, β is set to be a very small positive value $\ll 1$, thereby ensuring that the network is operating in the high gain limit. This

will ensure that in an equilibrium state, the state vector $\underline{s}$ converges to a corner of the N -dimensional hypercube $[0, 1]$, where each s_i is either contained in the interval $[0, \gamma]$ or $[1 - \gamma, 1]$ for some small positive constant $\gamma \ll \frac{1}{2}$. In order to define the network, we first select six constants $\beta > 0, \gamma > 0, r > 0, B > 0, C_1 > 0, C_2 > 0$ with the following properties (to be established in Lemma 3.1 to Lemma 3.3):

$$\beta \ll 1 \ (\beta \rightarrow 0) \quad (3.1)$$

$$\gamma \ll 1/2 \quad (3.2)$$

$$r \ll 0.5 \quad (3.3)$$

$$B \geq (C_1 + \Delta\beta)/(1 - \gamma) \quad (3.4)$$

$$C_1 > C_2 + 2\Delta\beta \quad (3.5)$$

$$C_2 \gg 1 \quad (3.6)$$

We now define the T -matrix as follows:

$$t_{ij} = -BX_{ij}(1 - \delta_{ij}) \quad (3.7)$$

where, δ_{ij} is the Kronecker symbol and X_{ij} is defined as:

$$X_{ij} = \begin{cases} 1 & \text{if the edges } e_i \text{ and } e_j \text{ intersect properly} \\ 0 & \text{otherwise.} \end{cases} \quad (3.8)$$

For each neuron i , we set the external input or *bias* ϵ_i to:

$$\epsilon_i = C_1 - C_2 \left(\frac{l_i}{l_{max}} \right). \quad (3.9)$$

The initial state of the system is set to:

$$s_i = 0.5 + random \quad (3.10)$$

where, *random* is a random number with the property:

$$-r \leq random \leq r. \quad (3.11)$$

We observe that the T matrix is symmetric, thereby meeting the requirement **T-SYM** (described in Chapter 2) needed for a Hopfield network to converge. Thus, it

follows that our triangulation network $TN(S)$ will reach an equilibrium state. Since equation (3.1) ensures that we operate the network in the high gain limit, every equilibrium state has the property that every s_i converges to either 0 or 1, and the energy E of the network is a local minimum. In the remainder of this section, we discuss the dynamic behaviour of the network with respect to our goal of computing a minimum weight triangulation of S .

Let $\underline{s} = [s_1, s_2, \dots, s_N]^t$ be an equilibrium state. Since $s_i = g(u_i)$, for all i , and $u_i = g^{-1}(s_i)$, for all i , an equilibrium state $\underline{s} = \underline{0}$ also implies $\underline{u} = \underline{0}$. Therefore, it follows from equation (2.7) that in an equilibrium state,

$$u_i = \sum_j s_j \cdot t_{ij} + \epsilon_i, \quad \text{for all } i. \quad (3.12)$$

We establish the correctness of the neural network solution by proving the following three lemmas.

Lemma 3.1 *Let ϵ_i and ϵ_k be two edges that are selected in an equilibrium state $\underline{s}$ of $TN(S)$. Then, ϵ_i and ϵ_k do not intersect properly.*

Proof:

The proof is by contradiction. Assume that there is a stable state $\underline{s}$, where two selected edges ϵ_i and ϵ_k , $i \neq k$, are intersecting properly, i.e., $X_{ik} = 1$. Since neurons i and k are selected, both s_i and s_k converge to 1; hence, $s_i \geq 1 - \gamma$ and $s_k \geq 1 - \gamma$. From (3.12), we obtain:

$$\begin{aligned} u_i &= \sum_j s_j \cdot t_{ij} + \epsilon_i \\ &= -Bs_k - \sum_{j \neq k} BX_{ij} \cdot (1 - \delta_{ij}) \cdot s_j + \epsilon_i \\ &\quad \text{(from equation (3.7) since } t_{ik} = -B) \\ &\leq -Bs_k + \epsilon_i \\ &\leq -B(1 - \gamma) + \epsilon_i \\ &\leq -(C_1 + \Delta\beta) + (C_1 - C_2 \frac{l_i}{l_{max}}) \quad \text{(from equations (3.4) and (3.9))} \\ &\leq -\Delta\beta. \end{aligned}$$

Thus, $s_i = g(u_i) \leq g(-\Delta\beta)$, resulting in the neuron i being unselected. This contradicts the assumption that neuron i is selected. Hence, if two edges are selected to be present in an output triangulation, they do not intersect properly, which satisfies the requirements for a (valid) triangulation. $\square$

We go on to show that the neural triangulation network produces the maximal set of non-intersecting edges, thus satisfying the second requirement for a triangulation.

Lemma 3.2 *Let $\underline{g}$ be an equilibrium state of $TN(S)$, and consider an arbitrary edge e_i . If all edges properly intersecting edge e_i are unselected, then edge e_i is selected.*

Proof:

Consider an equilibrium state $\underline{g}$ and the particular neuron i representing edge e_i . Assume that all edges which are properly intersecting e_i are unselected.

An edge e_j either intersects e_i or does not intersect e_i . In the former case, $t_{ij} = -B$ and the latter case, $t_{ij} = 0$ (since $X_{ij} = 0$). Since all edges which properly intersect e_i are unselected,

$$\sum_j t_{ij}s_j = 0. \quad (3.13)$$

We obtain from equation (3.12),

$$\begin{aligned} u_i &= \sum_j t_{ij}s_j + \epsilon_i && \text{(from equation (3.12))} \\ &= \epsilon_i && \text{(from equation (3.13))} \\ &\geq C_1 - C_2 \frac{l_i}{l_{max}} && \text{(from equation (3.9)).} \end{aligned}$$

Since $\frac{l_i}{l_{max}} \leq 1$, it follows that $u_i \geq C_1 - C_2 \geq \Delta\beta$ (from equation (3.5)). Thus, $s_i = g(u_i) \geq g(\Delta\beta)$, which implies that neuron i is indeed selected, when all other edges intersecting the edge e_i are unselected. $\square$

In the following lemma we show that minimizing the weight of the triangulation is equivalent to minimizing the energy of the network. Let S be a point set with k edges on its convex hull.

Lemma 3.3 *Let $\underline{s}$ be an equilibrium state for $TN(S)$, then, in a high gain case, the energy E converges to $D_1 + D_2 \cdot (\sum_{i \text{ selected}} l_i)$ where D_1, D_2 are constants satisfying $D_1 = C_1(3n - k - 3)$ and $D_2 = \frac{C_2}{l_{max}} > 0$.*

Proof:

The energy $E = E(\underline{s})$ of the network is given by:

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j + \sum_i \left[\frac{1}{R_i} \int_0^{s_i} g^{-1}(s_i) ds \right] - \sum_i \epsilon_i s_i \quad (3.14)$$

From Lemma 3.1 and Lemma 3.2 it follows that the first term converges to zero. For the high gain case, i.e., when $\beta \rightarrow 0$, the second term also converges to zero [Hopfield 1984]. Hence,

$$\begin{aligned} E &= -\sum_i \epsilon_i s_i \\ &= -\sum_i (C_1 - C_2 \frac{l_i}{l_{max}}) s_i \quad (\text{from equation (3.9)}) \\ &= -C_1 \sum_i s_i + \frac{C_2}{l_{max}} \sum_i l_i s_i \end{aligned}$$

Lemma 3.1 and Lemma 3.2 imply that the selected set of edges is a maximal set of non-intersecting edges, i.e., a valid triangulation of a point set. It is well known that there are $3n - k - 3$ edges, where k is the number of edges on the convex hull of the given point set. Therefore, we obtain

$$E = D_1 + D_2 \left(\sum_{i \text{ selected}} l_i \right) \quad (\text{from equation (3.6)})$$

where, $D_1 = -C_1(3n - k - 3)$ and $D_2 = \frac{C_2}{l_{max}} > 0$. □

Hence, summarizing the above Lemmas, we have the following theorem.

Theorem 3.1 *$TN(S)$ will always converge to a stable state. In the high gain case, a stable state of $TN(S)$ represents a triangulation of the point set S . Minimizing the weight of the reported triangulation is equivalent to minimizing the value of the network's energy function.*

3.2.2 Experimental Results

An analog network always migrates towards a state that minimizes its energy. An unavoidable property of an analog Hopfield Net (and analog networks in general) is that the network's energy might stabilize in a local minimum which is not necessarily the global minimum. Due to the high dimensionality of the problem, the current literature considers an analytical evaluation infeasible. Thus, it is necessary to verify experimentally the quality of the results produced by such a network.

We have implemented an Analog Hopfield Net simulator running on a SUN SPARC workstation as well as a parallelized version running on a Transputer Network. The simulator is essentially a numerical integrator for the system of differential equations (2.7). The simple Euler method [Gear 1971] is used to numerically integrate the system of equations and is given as follows:

$$\begin{aligned}\dot{\underline{u}} &= -\underline{u} + T\underline{s} + \underline{\epsilon} \\ &= \lim_{\delta t \rightarrow 0} \{\underline{u}(t + \delta t) - \underline{u}(t)/\delta t\} \\ \underline{u}(t + \delta t) &\approx \underline{u}(t) + \delta t [-\underline{u}(t) + T \cdot g(\underline{u}(t)) + \underline{\epsilon}] \\ &\approx (1 - \delta t) \cdot \underline{u}(t) + \delta t [T \cdot g(\underline{u}(t)) + \underline{\epsilon}]\end{aligned}$$

where, δt is the time step.

An additional front-end program converts a point set into an analog network according to equations (3.1)-(3.11), and interprets the stable state of the network back into a set of line segments. We selected the following constants: $\Delta = 19$, $\beta = 0.1$, $\gamma = 10^{-8}$, $r = 0.01$, $\delta t = 0.001$, $\beta = 8600$, $C_1 = 8500$, $C_2 = 8000$. It is easy to see that these constants are consistent with equations (3.1) to (3.6). The variable β determines how sharply the sigmoid function rises. It is important to set β small enough to ensure that the network operates in the high gain limit. (For all our experiments $\beta = 0.1$ was sufficient. Note that β may not be equal to 0, because then the sigmoid function degenerates to a non-differentiable step function, in which case the network may not converge at all). From Lemma 3.3 it follows that the energy of the system is proportional to $C_2(\sum_{i \text{ selected}} l_i)$. Hence, C_2 determines

the shape of the energy function and should be set to a large value to create steep valleys.

Figure 3.3 Network $TN(S)$ converging to a Minimum Weight Triangulation. (a) A sample Point Set and its Minimum Weight Triangulation. (b) Assignment of Possible Edges to Neurons. (c) State Vector $\underline{s}$ as a Function of Time, from its Initial State to a Final Stable State.

It is instructive to follow the behaviour of our triangulation network on a particular example as illustrated in Figure 3.3. Figure 3.3a shows a sample point set S and its minimum weight triangulation. For our network $TN(S)$, a neuron i is associated with every pair of points (i.e. a possible edge of a minimum weight triangulation). The assignment of neuron numbers to vertex pairs is shown in Figure 3.3b. The dynamic behaviour of $TN(S)$, from its initial state to a final stable state, is shown in Figure 3.3c. For every neuron i , the value of s_i is shown as a function of time. All neurons start at a random value s_i close to 0.5, and in the final stable state each neuron has a value s_i close to either 0 or 1. The state s_1 of neuron 1, for example,

first drops from around 0.5 to approximately 0, and then in time steps 8 and 10 it rises to a value close to 1. The final states of all neurons represent the minimum length triangulation in Figure 3.3a.

Figure 3.4 The Triangulation network compared to various sequential heuristics: Average Weight of the Triangulation produced by our Triangulation Circuit (W_{TN}), Average Weight of the Minimum Weight Triangulation (W_{OPT}), Average Weight of a Greedy Triangulation (W_{GT}), Average Weight of a Delaunay Triangulation (W_{DT}) and Average Weight of a Random Triangulation (W_{RT}).

Statistical results obtained by executing circuit $TN(S)$ repeatedly on different random point sets of size n are shown in Figure 3.4. We compare the average weight of the triangulation produced by our triangulation circuit (W_{TN}) with the average weight of the actual minimum weight triangulation (W_{OPT}), the average weight of the greedy triangulation (W_{GT}), the average weight of the Delaunay triangulation (W_{DT}) and the average weight of a random triangulation (W_{RT}). The values shown have a variance of about 9%. The main result of the data displayed in Figure 3.4 is that, for our tests, the difference between the optimum weight and the weight of the triangulation produced by our circuit is less than 2.2% (with variance < 5.4%). Similarly, the difference between W_{TN} and the weight of the triangulation produced

by the greedy algorithm (W_{GT}) is less than 2.3% (with variance $< 2.5\%$). Due to the exponential complexity of the deterministic algorithm, computing the exact minimum weight W_{OPT} for $n \geq 30$ turned out to be impossible, even with a parallel algorithm running on an Intel iPSC/860 hypercube. The fact that for $n \geq 30$ the W_{TN} value increases only very slowly leads us to conjecture that it will stay very close to the optimum weight. In this context, another interesting observation is that for $n \geq 30$, the weight produced by our circuit is more than 50% better than a random triangulation, with steadily increasing difference.

3.2.3 Related Problems

In this section, we outline the design of analog neural networks for two closely related geometric problems. We give only a brief exposition of our results.

The analog network described in Section 3.2 can be modified to compute a **minimum weight triangulation for a simple polygon P with holes**. The holes may be polygonal or simply points. To our knowledge, no polynomial time algorithm is known for this problem. (This is in contrast to the case of polygons without holes for which a polynomial time algorithm exists [Klincsek 1980].) Consider the set S of vertices of P and use the network described in Section 3.2 with the following two modifications: Change equation (3.7), the T -matrix, to be

$$t_{ij} = -BX_{ij}Y_{ij}(1 - \delta_{ij}) \quad (3.15)$$

where,

$$X_{ij} = \begin{cases} 1 & \text{if the edges } e_i \text{ and } e_j \text{ intersect properly} \\ 0 & \text{otherwise} \end{cases} \quad (3.16)$$

$$Y_{ij} = \begin{cases} 0 & \text{if at least one of } e_i \text{ or } e_j \text{ is a boundary edge} \\ & \text{of } P, \text{ or intersects the exterior of } P \\ 1 & \text{otherwise.} \end{cases} \quad (3.17)$$

For the initial state, set $s_i = 0$ for all neurons i corresponding to an edge that lies (partially) outside P , and use equations (3.10) - (3.11) to set the initial state of all

other neurons. This implies that all the edges that are polygon edges and those that are candidate edges for a triangulation are active initially. It follows analogously to the analysis given in Section 3.2 that such a network produces a triangulation where all boundary edges of P are selected and all edges outside P are unselected. The energy of the network is proportional to the total weight of the selected edges (plus some constant).

Maximum length subset of non-intersecting line segments: Consider the problem of selecting from a given set S of n line segments a subset S' of non-intersecting line segments such that the total length of the selected line segments is maximized. The analog network consists of n neurons, each representing one line segment from the set S . All the network parameters are set in a manner identical to the triangulation network (described in Section 3.2). The elements of the weight matrix T and the bias vector are defined as follows:

$$t_{ij} = -BX_{ij}(1 - \delta_{ij})$$

where, δ_{ij} is the Kronecker symbol and X_{ij} is defined as:

$$X_{ij} = \begin{cases} 1 & \text{if line segment } s_i \text{ and } s_j \text{ intersect properly} \\ 0 & \text{otherwise} \end{cases}$$

$$c_i = C_1 + C_2 \cdot \left(\frac{l_i}{l_{max}} \right) \quad \text{where, } C_1 > C_2 + 2\Delta\beta; C_2 \gg 1$$

where, l_i is the length of the line segment represented by neuron i and l_{max} is the maximum of all such lengths. The analysis of this network is identical to the one given in Section 3.2. When the network reaches an equilibrium state, the energy of the network is proportional (with a negative scaling factor) to the total length of the selected line segments (plus a constant). The total energy is:

$$\begin{aligned} E &= - \sum_i c_i s_i \\ &= D_1 - \frac{C_2}{l_{max}} \left(\sum_{i \text{ selected}} l_i \right) \quad \text{where, } D_1 = -C_1(3n - k - 3) \end{aligned}$$

It is clear that minimizing this energy function is equivalent to maximizing $\sum l_i$.

Chapter 4

Point Congruence

Consider two distinct sets of points $P = \{p_1, p_2, \dots, p_n\}$ and $Q = \{q_1, q_2, \dots, q_n\}$ located in the 2-dimension plane. If P can be translated so as to coincide with Q , then, the sets P and Q are said to be congruent via translation. More formally,

Exact Congruence Problem: Given two distinct sets of points P and Q , does there exist an exact congruence between the two sets by means of a *translation*?

In one-dimension, the points in the plane reduce to numbers on the real line and the problem of exact congruence can be stated as follows: By means of an arithmetic operation, can be two sets of numbers be equal? By reducing the set equality problem of real numbers, Atkinson derived a lower bound of $\Omega(n \log n)$, where n is the cardinality of the set of numbers [Atkinson 1987]. Optimal algorithms for the two- and three-dimensional cases of the exact point congruence problem have been found [Atallah 1984, Atkinson 1987, Highnam 1986]. An algorithm for points in dimensions $d \geq 3$ is given in [Alt *et al.* 1988], where the authors not only consider the geometric transformation translation, but include other transformations such as rotation, reflection, and stretch.

Practical applications of such operations on point sets are: movement of an object from one position to another (in robotics), determining speed and direction of moving objects from pictures taken at different times and recognition of patterns and of written text. From a practical point of view, the problem of deciding exact congruence is not very realistic and numerically ill-conditioned due to inaccuracies

in physical measurements. Hence, the notion of *approximate congruence* has been introduced [Alt *et al.* 1988], where the congruence is within a certain tolerance of some $\varepsilon > 0$. We now give a formal definition for the problem as follows:

Approximate Congruence Problem with ε -tolerance: Given two point sets $\mathcal{A} = \{A_1, A_2, \dots, A_n\}$, and $\mathcal{B} = \{B_1, B_2, \dots, B_n\}$, where $|\mathcal{A}| = |\mathcal{B}| = n$, does there exist a congruence via a translation, which maps in a one-to-one fashion the points of $\mathcal{B}$ into the ε -neighbourhood of the points in $\mathcal{A}$?

We are interested in determining the smallest ε which allows to move every point $B_{b_i} \in \mathcal{B}$ into the ε -neighbourhood of some point $f(B_{b_i}) \in \mathcal{A}$, where $f : \mathcal{B} \rightarrow \mathcal{A}$ is a one-to-one known or unknown mapping, called a *labeling*. The current sequential time bounds for the two-dimensional optimization problem (where ε needs to be determined), with *unknown labeling* f , is $O(n^6 \log n)$ and for the decision problem (where ε is given), also with an *unknown labeling* f , is $O(n^6)$ [Alt *et al.* 1988]. [Heffernan and Schirra 1992] present approximate decision algorithms for the ε -congruence problem which runs in $O(n^{1.5})$ time for a transformation via translation and $O(n^{2.5})$ for rotational transformations. In the case of a known labeling f , both the decision and the optimization problems can be solved in linear time [Megiddo 1983].

4.1 Our Neural Network Solution

In this section, we present a neural network solution to the general optimization version of the approximate point congruence problem in d -dimensions: unknown labeling and unknown ε . We describe the construction of an analog neural network to solve the problem for an arbitrary dimension d . The network is specified by the initial state, the value of the external inputs and the strength of the interconnections. We also establish that such a network does indeed output a one-to-one mapping of all the points in set $\mathcal{B}$ to some point in set $\mathcal{A}$.

There are $N = n \times n$ neurons in the network, each of which represents the mapping of a point in set $\mathcal{B}$ to some point in $\mathcal{A}$. These neurons are referred to by integers in the range $[1, n^2]$. The n^2 neurons can be viewed as being arranged in an $n \times n$ matrix. The row and column numbers associated with a neuron i essentially represent the indices of the points from $\mathcal{A}$ and $\mathcal{B}$, which are mapped via the labeling f , if this neuron is selected. Neuron i is treated as an ordered pair, i.e., $i = \langle a_i, b_i \rangle$. In other words, neuron i maps the b_i^{th} point in set $\mathcal{B}$ to the a_i^{th} point in set $\mathcal{A}$. An example of one such mapping is shown in Figure 4.1, where the sets $\mathcal{A}$ and $\mathcal{B}$ have 5 points each. The resulting mapping of Figure 4.1, represented by a $\rightarrow$, is the following: $a_1 \rightarrow b_1$; $a_2 \rightarrow b_4$; $a_3 \rightarrow b_3$; $a_4 \rightarrow b_2$; $a_5 \rightarrow b_5$.

Figure 4.1 A possible mapping between sets $\mathcal{A}$ and $\mathcal{B}$.

set $\mathcal{A}$	set $\mathcal{B}$				
	b_1	b_2	b_3	b_4	b_5
a_1	1	0	0	0	0
a_2	0	0	0	1	0
a_3	0	0	1	0	0
a_4	0	1	0	0	0
a_5	0	0	0	0	1

4.1.1 Initialization of Network Parameters

The preprocessing work involves the calculation of the distance between the mapping of every pair of points, one from set $\mathcal{A}$ and one from set $\mathcal{B}$. In other words, the following quantity is calculated for every pair of neurons $\langle i, j \rangle$, for all $1 \leq i, j \leq n^2$,

$$dist_{i,j} = \|(A_{a_i} - B_{b_i}) - (A_{a_j} - B_{b_j})\|$$

We also find the maximum of such distances $dist_{i,j}$ in M . Each neuron represents a mapping and hence an associated translation vector. Thus, for example, the translation vector associated with neuron i is the vector $A_{a_i} - B_{b_i}$. Our aim is to select neurons which translate by (*nearly*) identical amounts. Thus, intuitively,

the energy of neural network is proportional to the sum of the magnitude of the difference between all pairs of translation vectors. We are now ready to define the terms of the interconnection weight matrix T .

$$t_{ij} = t'_{ij} + t''_{ij}$$

where,

$$t'_{ij} = -C(1 - \delta_{ij}) \cdot share_{ij} \quad (4.1)$$

$$t''_{ij} = -D(1 - \delta_{ij}) \cdot value_{ij} \quad (4.2)$$

and,

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (4.3)$$

$$share_{ij} = \begin{cases} 1 & \text{if } a_i = a_j \text{ or } b_i = b_j \\ 0 & \text{otherwise} \end{cases} \quad (4.4)$$

$$value_{ij} = \frac{dist_{ij}^k}{M} \leq 1 \quad (4.5)$$

$$M = \max_{i,j} \{dist_{ij}\}. \quad (4.6)$$

Select constants $C, D, \beta > 0$, and $\omega > 0$., such that they satisfy the following constraints:

$$C \geq \frac{1}{(1 - \gamma)}(\Delta\beta + \omega) \quad (4.7)$$

$$D > 0 \quad (4.8)$$

$$\beta \ll 0 \quad (4.9)$$

$$\omega \geq 3(\Delta\beta + Dn). \quad (4.10)$$

The bounds for the constants C , and ω stated in (4.7), (4.10) will be derived later, so that in equilibrium, the neurons will represent a one-to-one mapping of the points in the sets $\mathcal{A}$ and $\mathcal{B}$. For each neuron i , we set the external input or *bias* ϵ_i :

$$\epsilon_i = \omega, \quad \text{for all } i.$$

The initial state of the system is set to:

$$s_i = 0.5 + random \quad (4.11)$$

where, *random* is a random number with the property:

$$-r \leq random \leq r, \quad r \ll 0.5.$$

Observe that with the T -matrix as defined above, we have satisfied the requirement **T-SYM** which guarantees that the network will converge to a local minimum. Using the matrix of n^2 neurons, we would like one point of $\mathcal{B}$ to be mapped onto one and only one point in $\mathcal{A}$ and every point of $\mathcal{B}$ to be mapped to some point in $\mathcal{A}$. This implies that exactly n neurons have to be selected. The one-to-one mapping can be easily visualized as precisely one neuron being selected in each row and each column.

We now have the following lemmas establishing the correctness of the network in finding the best mapping that exists between the set of points $\mathcal{A}$ and $\mathcal{B}$.

Lemma 4.1 *No two neurons in the same row or in the same column are selected.*

Proof:

We prove the result by contradiction. We know that in equilibrium, $\underline{u} = \mathbf{T}\underline{s} + \underline{c}$. Let us assume that two neurons i and k , in the same row/column are selected; hence, $s_i, s_k \geq 1 - \gamma$.

$$\begin{aligned} \text{Since } t_{ii} = 0, u_i &= t_{ik}s_k + \sum_{j \neq k} t_{ij}s_j + c_i \\ &= (t'_{ik}s_k + t''_{ik}s_k) + \sum_{j \neq k} t_{ij}s_j + c_i \\ &= (-C \cdot s_k - D \cdot value_{ik} \cdot s_k) + \sum_{j \neq k} t_{ij}s_j + c_i \\ &\leq -C(1 - \gamma) - D \cdot value_{ik}(1 - \gamma) \end{aligned} \quad (4.12)$$

$$\begin{aligned} &\quad - \sum_{j \neq k} (C \cdot share_{ij} + D \cdot value_{ij})(1 - \delta_{ij})s_j + c_i \\ &\leq -C(1 - \gamma) + c_i. \end{aligned} \quad (4.13)$$

So far we have, $u_i \leq -C(1 - \gamma) + c_i$. To obtain a contradiction, we need $u_i \leq -\Delta\beta$, which will make neuron i unselected. For this, we derive bounds for the constant C

in the next few steps. Let $\epsilon_i = \omega$. The right hand side of (4.13) should be $\leq -\Delta\beta$, thereby making neuron i to be unselected. That is,

$$\begin{aligned} -C'(1-\gamma) + \omega &\leq -\Delta\beta \\ C' &\geq \frac{1}{(1-\gamma)}(\Delta\beta + \omega) \end{aligned}$$

Therefore, if $C' \geq \frac{1}{(1-\gamma)}(\Delta\beta + \omega)$, neuron i will be unselected, which is a contradiction to the hypothesis that i was selected. Hence, as a by-product of this lemma, we have derived the bounds for constant C' . We have just proved that no two neurons in the same row or column can be selected simultaneously. This implies that there can be at most one neuron selected in any row or column. $\square$

Corollary 4.1 *At most n neurons, in total, are selected simultaneously.*

So far, the results we have derived show that **at most** n neurons can be selected from the grid of n^2 neurons. For a neural network stable state to correspond to a proper mapping, we must ensure that **exactly** n neurons are selected. To establish the correctness of the neural network solution, we state and prove a technical result.

Lemma 4.2 *If an entire column c of neurons were not selected, then every row would contain a selected neuron.*

Proof:

Assume that there exists a row r which has all the neurons unselected. So, we have an entire row and column (r and c , respectively) unselected. Consider neuron i , which is at the intersection of row r and column c , as shown in Figure 4.2.

At equilibrium, since $\dot{\underline{u}} = 0$,

$$\begin{aligned} u_i &= \sum_j t_{ij}s_j + \epsilon_i \\ &= \sum_j (t'_{ij} + t''_{ij})s_j + \epsilon_i \end{aligned}$$

Figure 4.2 Row r and Column c of *selected* neurons.

Recall that $t'_{ij} = -C(1 - \delta_{ij}) \cdot share_{ij}$. Therefore, $t'_{ij} = 0$ whenever j refers to one of the neurons *not* in row r or column c (because $share_{ij} = 0$). For all j , $j \neq i$ in row r , or column c , $share_{ij} = 1$ and $\delta_{ij} = 1$. Therefore,

$$u_i = \sum_j t'_{ij} s_j + \sum_j t''_{ij} s_j + \epsilon_i \quad (4.14)$$

Let us look at the two summation terms in (4.14) independently

$$\sum_j t'_{ij} s_j \geq -C \cdot 2(n-1) \cdot \gamma$$

(Note: There are $2(n-1)$ interconnections between neurons in row r and column c . Moreover, the neurons in row r and column c are unselected, therefore, $s_j \leq \gamma$).

$$\begin{aligned} \sum_j t''_{ij} s_j &= - \sum_j D s_j (1 - \delta_{ij}) \cdot value_{ij} \\ &= - \sum_{j \neq i} D s_j \cdot value_{ij} \\ &= -D \left\{ \sum_{j \in r \cup c} value_{ij} \cdot s_j \right\} - D \left\{ \sum_{j \notin r \cup c} value_{ij} \cdot s_j \right\} \end{aligned}$$

By Lemma 4.1, we know that **at most** n neurons are selected (meaning, $s_j \geq (1-\gamma)$) and **at least** $n^2 - n$ neurons are unselected, that is, $s_j \leq \gamma$. Therefore,

$$\sum_j t''_{ij} s_j \geq -D \cdot n \cdot 1 - D(n^2 - n)\gamma.$$

Hence, summarizing the above discussion, we have,

$$u_i \geq -2C(n-1)\gamma - D\{n(n-1)\gamma + n\} + \epsilon_i. \quad (4.15)$$

In order to arrive at a contradiction to the hypothesis, we would like to select the constants in such a manner that $u_i \geq \Delta\beta$ (thereby selecting neuron i). We choose $C \geq \frac{1}{1-\gamma}(\Delta\beta + \epsilon_i)$, as was derived in Lemma 4.1. Let $\epsilon_i = \omega$. If $\gamma \ll \frac{1}{2}$, $C = 2(\Delta\beta + \omega)$. Let D be the free variable. We have to choose ω appropriately so that $u_i \geq \Delta\beta$. That is, for (4.15), we have to satisfy the following condition:

$$\omega \geq 2C(n-1)\gamma + D\{n(n-1)\gamma + n\} + \Delta\beta \quad (4.16)$$

Choose $\gamma = \frac{1}{8(n-1)}$ which makes γ sufficiently small, i.e., $\gamma \ll \frac{1}{2}$. Substituting for γ and C in (4.16), we have the following bounds for ω :

$$\begin{aligned} \omega &\geq \frac{1}{4}C + D\left(\frac{n}{8} + n\right) + \Delta\beta \\ &\geq \frac{1}{2}(\Delta\beta + \omega) + \frac{9}{8}Dn + \Delta\beta \\ &\geq 3(\Delta\beta + Dn). \end{aligned}$$

So, we have chosen constants C and ω such that $u_i \geq \Delta\beta$, which means that $s_i \geq (1-\gamma)$, implying that neuron i is indeed selected. This is a contradiction to our assumption that row r has all its neurons unselected! $\square$

Hence, we have the following theorem:

Theorem 4.1 *In every row or column, exactly one neuron is selected. Thus, the neural network produces a one-to-one mapping between the sets $\mathcal{A}$ and $\mathcal{B}$.*

Proof:

Lemma 4.1 permits no two neurons to be selected in any row or any column. Assume, as in Lemma 4.2, that an entire column c of neurons is unselected. Then, by Lemma 4.2, every row contains a selected neuron. Since, one column is completely unselected, then, by the pigeon-hole principle, there exists a column with *two* selected neurons. This is again a contradiction to Lemma 4.1, which disallows *two* neurons to be selected in any row or column.

Hence we have proved that we have a one-to-one mapping and that all n points of $\mathcal{B}$ are mapped to some point in $\mathcal{A}$. $\square$

In the following lemma we show that minimizing the maximum distance between translation vectors is equivalent to minimizing the energy of the network.

Lemma 4.3 *In the high gain case, the total energy is proportional to the sum of the k^{th} powers of the magnitudes of the differences between all pairs of selected translation vectors (plus a constant).*

Proof:

The energy of the network is given by:

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j + \sum_i \left[\frac{1}{R_i} \int_0^{s_i} g^{-1}(s_i) ds \right] - \sum_i c_i s_i \quad (4.17)$$

For the high gain case, i.e., when $\beta \rightarrow 0$, the second term converges to zero [Hopfield 1984], hence,

$$\begin{aligned} E &= -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j - \sum_i c_i s_i \\ &= -\frac{1}{2} \sum_i \sum_j t''_{ij} s_i s_j - N\omega \\ &= \frac{1}{2} D \sum_{i \text{ selected}} \sum_{j \text{ selected}} \text{value}_{ij} - N\omega \\ &= \frac{D}{2M} \sum_{i,j \text{ selected}} \text{dist}_{ij}^k - N\omega \end{aligned}$$

An optimal mapping minimizes $\max_{ij} \{\text{dist}_{ij}\}$. Since it is not possible to directly map such a function into an energy function of the format specified in equation

(4.17), we minimize the quantity $\sum_{i,j} \text{value}_{i,j} = \frac{1}{M} \sum_{i,j} \text{dist}_{i,j}^k$, which approximates $\max_{i,j} \{\text{dist}_{i,j}\}$, for large k and where, i, j represent only *selected* neurons. The error between the correct value and the approximation by our energy function is not significant, as long as the correct set of neurons are selected (i.e., as long as we get a correct one-to-one mapping). For our experiments, the exponent $k = 2$ was sufficient for sets of size $n \leq 30$. $\square$

4.2 Experimental Results

The network was tested using the analog neural network simulator described in Section 3.2.2. The points in sets $\mathcal{A}$ and $\mathcal{B}$ were generated such that a mapping was known. The values for the various constants were selected as follows: $\Delta = 19$, $\beta = 0.1$, $\gamma = 10^{-8}$, $r = 0.01$, and $D = 800$. These values are consistent with the equations specified in Section 4.1.1. The next step was to calculate the weight matrix, the input bias and initial state using equations (4.2)-(4.11). Figure 4.3 shows an example of a one-to-one mapping produced by the analog neural network

Table 4.1 summarizes the results of running the program for different sizes of the sets $\mathcal{A}$ and $\mathcal{B}$. The number of times the network exactly identified the given mapping between the points is shown in the column "optimal mapping" and the number of times it failed to do is shown in the column "non-optimal mapping". It shows that our network reports only very few non-optimal results (at most 4 out of 25). Even in those cases, the reported mapping differs only very slightly from the optimal answer (usually just one pair of points are "flipped").

Figure 4.3 An example of a one-to-one mapping identified by the neural network. (a) The point sets $\mathcal{A}$ and $\mathcal{B}$ (b) Mapping of points from $\mathcal{A}$ to $\mathcal{B}$

Table 4.1 Summary of simulation results. n represents the size of the point sets and k represents the exponent in $dist_{ij}^k$. Columns 4 and 5 show the number of times the given mapping was identified exactly and the number of times a non-optimal mapping was found, respectively.

n	k	samples	optimal mapping	non-optimal mapping
5	1	50	50	0
	2	50	50	0
10	1	25	23	2
	2	25	25	0
12	1	25	25	0
	2	25	25	0
15	1	25	21	4
	2	25	25	0
20	1	10	9	1
	2	10	10	0
25	1	10	9	1
	2	10	10	0
30	1	5	5	0
	2	5	5	0

Chapter 5

Geometric k -clustering problem

Let S be a set of n points in the plane. A partitioning of S into k disjoint sets $C_1, C_2, \dots, C_k$ is called a k -clustering, where the individual sets C_i are called its *clusters*. Clustering is a fundamental problem that is encountered in many applications [Jain and Dubes 1988]. In cluster analysis, the points represent the properties of real-world objects, and the goal is to collect “similar” objects into the same cluster and to put objects which are very “different” into different clusters. In pattern recognition, the idea is to partition the given data points into several clusters such that each cluster represents a distinct “class” of objects.

In forming these clusters, two approaches could be taken. One is that the number of clusters k , is known and the other is where k is unknown. The latter one is a far more difficult problem as there are approximately $k^n/k!$ possible ways of partitioning the n points into k clusters [Feller 1959] and this results in an exponential time algorithm to exhaustively search for the best solution.

The goal of the system is to form clusters so as to satisfy some criteria. Let there be a weight function assigned to a set of points in the plane. Typically, the weight function would be the diameter of the set, the radius of the point set, the perimeter or the area of the convex hull. Many such criteria of clustering have been extensively studied by Brückner [Brückner 1977]. Most of the variations of the clustering problem are known to be NP-complete [Supowit 1981, Megiddo and Tamir 1982, Johnson 1982, Brückner 1977, Shamos 1976]. Supowit has shown that if the weight

function is the diameter of the set of points, this problem is NP-complete for an unknown k [Supowit 1981]. We present a neural network solution to the geometric k -clustering problem, which is described as follows.

5.1 Problem Definition

Given a set of n points S and a positive integer $k > 0$, partition the set into a family $\mathcal{C} = \{C_1, C_2, \dots, C_k\}$ of k clusters (disjoint subsets of S), such that the following constraints are met.

1. $C_r \cap C_s = \emptyset$, for all $r \neq s$
2. $\bigcup_{r=1}^k C_r = S$
3. The maximum diameter of $C_r \in \mathcal{C}$ is to be minimized.

[Kangar-Parsi and Kangar-Parsi 1989] report a solution to an alternate formulation of this problem based on the *Analog Hopfield* Neural Networks. Their clustering criterion is the following: minimize the sum of squares of the distances of the points from their respective cluster centroids (i.e., the sum of the "within cluster variances"). A matrix of $k \times n$ neurons is used, where, the points contained in a certain cluster are represented by a row of neurons. The energy function which incorporates the constraints of the problem is:

$$E = \frac{A}{2} \sum_i \sum_p \sum_{q \neq p} V_{pi} V_{qi} \quad (5.1)$$

$$+ \frac{B}{2} \sum_i \left(\sum_p V_{pi} - 1 \right)^2 \quad (5.2)$$

$$+ \frac{C}{2} \sum_p \sum_i R_{pi} V_{pi}^2 \quad (5.3)$$

$$- \sum_p \sum_i I_{pi} V_{pi} + \frac{1}{r} \sum_p \sum_i \int^{V_{pi}} g^{-1}(z) dz$$

The indices p and q refer to the clusters and run from 1 to k . The indices i and j refer to the points and are in the range $[1, n]$. The constraint that each

point must belong to one and only one cluster is enforced by the first two terms given in (5.1) and (5.2). The third term given by (5.3) represents the sum of the within cluster variances, where R_{pi} is the square of the distance of point i from the centroid of cluster p . The last two terms represent the bias and the integral terms from the original Lyapunov energy function. This energy function for the clustering problem was tested on 128 samples. Each sample contained a set of 128 points divided among 5 clusters with within-cluster Gaussian distributions. When the clusters were well-defined, the neural network was able to identify the optimum clusters. When the points were uniformly distributed in the unit square, no valid solutions were reported.

We present a clustering solution for an alternate clustering criterion and prove that in a stable state, the network is able to produce a clustering satisfying the constraints of the problem. We also derive bounds for the constants used in defining the interconnection weights and the external bias.

5.2 Our Neural Network Solution

There are $N = k \times n$ neurons, where each neuron $i = \langle c_i, s_i \rangle$, $i = 1 \dots N$, represents a point $s_i \in S$ and cluster $c_i \in C$ and neuron i is selected if point s_i is in cluster c_i . The above conditions (1) and (2) ensure that every point belongs to one and only one cluster. We picture the neurons as being arranged in a $k \times n$ matrix, with one row for each cluster and one column for each point. To satisfy the first condition, we have to ensure that there is no more than one neuron selected in the same column. The third condition specifies the clustering criteria whereby the maximum diameter among all the clusters is minimized. For example, Table 5.1 shows the output from an 8×3 neural network, where 8 points are grouped into 3 clusters.

The neural network solution to this problem is approached in a manner similar to the point-congruence problem described in Chapter 4.

Table 5.1 An example for grouping 8 points into 3 clusters.

cluster	points							
	1	2	3	4	5	6	7	8
1	0	1	1	0	0	1	0	0
2	1	0	0	0	1	0	0	0
3	0	0	0	1	0	0	1	1

Select constants $\beta > 0$, $\omega > 0$, $C > 0$ and $D > 0$, such that they satisfy the following constraints:

$$\beta \leq 1 \quad (5.4)$$

$$\omega \geq 2(\Delta\beta + Dn) \quad (5.5)$$

$$C \geq \frac{1}{(1-\gamma)}(\Delta\beta + \omega) \quad (5.6)$$

$$D > 0 \quad (5.7)$$

The bounds for the constants C and ω stated in (5.6), (5.5) will be derived later which will ensure that in equilibrium, the neurons will represent feasible solutions. For each neuron i , we set the external input or *bias* e_i to:

$$e_i = \omega, \quad \text{for all } i. \quad (5.8)$$

The initial state of the system is set to:

$$s_i = 0.5 + random \quad (5.9)$$

where, *random* is a random number with the property:

$$-r \leq random \leq r \quad \text{with } r \leq 0.5.$$

The elements of the interconnection weight matrix T are chosen as given below:

$$t_{ij} = t'_{ij} + t''_{ij} \quad (5.10)$$

where,

$$t'_{ij} = -C(1 - \delta_{ij}) \cdot share_{ij}$$

$$t''_{ij} = -D(1 - \delta_{ij}) \cdot value_{ij}$$

where,

$$\begin{aligned} share_{ij} &= \begin{cases} 1 & \text{if } s_i = s_j \\ 0 & \text{otherwise} \end{cases} \\ value_{ij} &= \begin{cases} 0 & \text{if } c_i \neq c_j \\ \left[\frac{d(s_i, s_j)}{M} \right]^l & \text{otherwise, } (l \geq 1) \end{cases} \end{aligned} \quad (5.11)$$

$$M = \max_{ij} \{d(s_i, s_j)\} \quad (5.12)$$

and, $d(s_i, s_j)$ = Euclidean distance of the two points represented by neurons i and j .

Observe that with the T -matrix as defined above, we have satisfied the requirement **T-SYM** which guarantees the network to stabilize to a local minimum. It remains to be shown that we indeed have a proper partition of set S into k clusters. That is, we need to establish that exactly n neurons are selected when the network stabilizes into equilibrium. The following lemmas establish this correctness.

Lemma 5.1 *No two neurons in the same column are selected*

Proof:

The proof is identical to Lemma 4.1, except that it involves only the columns. We prove the result by contradiction. We know that in equilibrium, $\underline{u} = \mathbf{T}\underline{s} + \underline{c}$. Let us assume that two neurons i and k , in the same column are selected; hence, $s_i, s_k \geq 1 - \gamma$.

Since $t_{ii} = 0$,

$$\begin{aligned} u_i &= t_{ik}s_k + \sum_{j \neq k} t_{ij}s_j + c_i \\ &= (t'_{ik}s_k + t''_{ik}s_k) + \sum_{j \neq k} t_{ij}s_j + c_i \\ &= (-C \cdot s_k - D \cdot value_{ik} \cdot s_k) + \sum_{j \neq k} t_{ij}s_j + c_i \\ &\leq -C(1 - \gamma) - D \cdot value_{ik}(1 - \gamma) + \end{aligned} \quad (5.13)$$

$$\begin{aligned} &\sum_{j \neq k} (C \cdot share_{ij} + D \cdot value_{ij})(1 - \delta_{ij})s_j + c_i \\ &\leq -C(1 - \gamma) + c_i. \end{aligned} \quad (5.14)$$

So far we have, $u_i \leq -C(1 - \gamma) + \epsilon_i$. To obtain a contradiction, we need $u_i \leq -\Delta\beta$, which will make neuron i unselected. For this, we derive bounds for the constant C in the next few steps. Let $\epsilon_i = \omega$. The right hand side of (5.14) should be $\leq -\Delta\beta$, thereby making neuron i to be unselected. That is,

$$\begin{aligned} -C(1 - \gamma) + \omega &\leq -\Delta\beta \\ C &\geq \frac{1}{(1 - \gamma)}(\Delta\beta + \omega) \end{aligned}$$

Therefore, if $C \geq \frac{1}{(1 - \gamma)}(\Delta\beta + \omega)$, neuron i will be unselected, which is a contradiction to the hypothesis that i was selected. As a by-product of this lemma, we have derived a bound for constant C . We have just proved that no two neurons in the same column can be selected simultaneously. This implies that there can be at most one neuron selected in any column. $\square$

Corollary 5.1 *At most n neurons, in total, are selected simultaneously.*

In the following lemma we show that exactly n neurons are selected, a result that is required to prove that exactly one neuron is selected in every column.

Lemma 5.2 *Each column contains exactly one selected neuron.*

Proof:

Assume that there exists an entire column c of unselected neurons, as shown in Figure 5.1. Consider any neuron i in this column.

At equilibrium, since $\underline{\mathbf{u}} = 0$,

$$\begin{aligned} u_i &= \sum_j t_{ij}s_j + \epsilon_i \\ &= \sum_j (t'_{ij} + t''_{ij})s_j + \epsilon_i \\ &= \sum_j t'_{ij}s_j + \sum_j t''_{ij}s_j + \epsilon_i \end{aligned}$$

Since all the neurons in column c are unselected, $s_j \leq \gamma$. Moreover, there are $k - 1$ interconnections among these k neurons in column c . We simplify the two summations terms independently.

$$\sum_j t'_{ij}s_j \leq -C \cdot k \cdot \gamma$$

Figure 5.1 An *unselected* column c .

$$\begin{aligned}
 \sum_j t''_{ij} s_j &= - \sum_j D s_j (1 - \delta_{ij}) \cdot \text{value}_{ij} \\
 &= - \sum_{j \neq i} D s_j \cdot \text{value}_{ij} \\
 &= -D \left\{ \sum_{j \in \text{column } c} \text{value}_{ij} \cdot s_j \right\} = D \left\{ \sum_{j \notin \text{column } c} \text{value}_{ij} \cdot s_j \right\}
 \end{aligned}$$

By Lemma 5.1, we know that **at most** n neurons are selected (meaning, $s_j \leq (1 - \gamma)$) and **at least** $n(k-1)$ neurons are unselected, that is, $n(k-1)$ s_j 's are $\leq \gamma$. Therefore,

$$\begin{aligned}
 \sum_j t''_{ij} s_j &\geq -D \cdot 1 \cdot (n-1) = D \cdot n(k-1) \cdot \gamma \\
 &\geq -D \cdot n + D(nk - n)\gamma.
 \end{aligned}$$

Hence, summarizing the above discussion, we have,

$$a_i \geq -C'k\gamma + D\{(nk - n)\gamma + n\} + c_i. \quad (5.15)$$

In order to arrive at a contradiction to the hypothesis, we would like to select the constants in such a manner that $a_i > \Delta \beta$ (thereby selecting neuron i). We choose $C' \geq \frac{1}{1-\gamma}(\Delta \beta + \omega)$, (where $\omega = c_i$, as was derived in Lemma 5.1). Let $\epsilon_i = \omega$.

If $\gamma \ll \frac{1}{2}$, $C = 2(\Delta\beta + \omega)$. Let D be the free variable. We have to choose ω appropriately so that $u_i \geq \Delta\beta$. That is, for (5.15), we have to satisfy the following condition:

$$\begin{aligned}\omega &\geq Ck\gamma + D\{(nk - n)\gamma + n\} + \Delta\beta \\ \omega &> Ck\gamma + D\{nk\gamma + n\} + \Delta\beta\end{aligned}$$

Choose $\gamma = \frac{1}{8k}$, which makes it sufficiently small, i.e., $\gamma \ll \frac{1}{2}$. Now,

$$\begin{aligned}\omega &> \frac{1}{8}C + D\left(\frac{1}{8}n + n\right) + \Delta\beta \\ &> \frac{1}{4}(\Delta\beta + \omega) + \frac{9}{8}Dn + \Delta\beta \\ \frac{3}{4}\omega &> \frac{5}{4}\Delta\beta + \frac{9}{8}Dn \\ \omega &> 2(\Delta\beta + Dn)\end{aligned}$$

So, we have chosen constants C and ω such that $u_i \geq \Delta\beta$, which means that $s_i \geq (1 - \gamma)$, implying that neuron i is indeed selected. This is a contradiction to our assumption that column c has all its neurons unselected! $\square$

Hence, we have the following theorem:

Theorem 5.1 *In every column exactly one neuron is selected. Thus the neural network produces a proper k -clustering of the points in set S .*

Proof:

Lemma 5.1 permits no two neurons to be selected in a column. What remains to be shown is that it is **not** possible for **all** neurons in a column to be **unselected**. Assume, as in Lemma 5.2, that an entire column c of neurons is unselected. Then, by Lemma 5.2, every column contains a selected neuron. Since, one column is completely unselected, then, by the pigeon-hole principle, there exists a column with *two* selected neurons. This is again a contradiction to Lemma 5.1, which disallows *two* neurons to be selected in any column. Therefore, exactly one neuron in every column is selected. Hence the neural network partitions the n points into k clusters so that all the points are in some cluster and no point is member of more than one cluster. $\square$

In the following lemma we show that the clustering criterion is met, that is, the maximum cluster diameter is minimized.

Lemma 5.3 *In the high gain case, the total energy is proportional to the sum of the l^{th} power of the distances between all pairs of points selected to be in the same cluster (plus a constant).*

Proof:

In the high gain case, i.e., when $\beta \rightarrow 0$, we have the following energy function [Hopfield 1984]:

$$\begin{aligned} E &= -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j - \sum_i c_i s_i \\ &= -\frac{1}{2} \sum_i \sum_j t''_{ij} s_i s_j - N\omega \\ &= \frac{1}{2} D \sum_{i,j \text{ selected}} \text{value}_{ij} - N\omega \\ &= \frac{D}{2M^l} \sum_{i,j \text{ selected}} d(s_i, s_j)^l - N\omega \end{aligned}$$

An optimal mapping minimizes the maximum diameter of all the k clusters. As is it not possible to directly specify such a function in terms of our energy function, we wish to minimize an approximation of the maximum diameter as given by $\sum_{i,j} \left[\frac{d(s_i, s_j)}{M} \right]^l$, where, $M = \max_{i,j} \{d(s_i, s_j)\}$ and s_i, s_j represent points selected to be in the same cluster. In choosing such an approximation function, the error that arises (between the exact and approximated functions) is not significant, as long as the network produces a correct clustering of the given n points into k groups. $\square$

5.3 Experimental Results

The performance of our neural network was tested using the neural network simulator described in Section 3.2.2. For a known clustering, the criteria was to check if the network was able to identify the clusters properly. The input point sets were generated as follows. Given k , the number of clusters, into which n points have to be

classified, the k cluster centers were randomly generated so as to coincide with the intersection points of a grid where the adjacent grid lines are unit distance apart. In generating the n points to be clustered, each of the points is randomly assigned to one of k clusters in such a way that it lies within a circle of radius 0.23 centered at the chosen cluster center. This choice of the radius ensures that points within a given cluster are nearer to one another than to any other point from a neighbouring cluster. Hence, we have successfully generated input samples for which a clustering is known.

The neural network parameters were set to be the following: $\Delta = 19$, $\beta = 0.03$, $\gamma = 10^{-5}$, $\tau = 0.01$, $\delta t = 0.01$ and $D = 800$, which are consistent with the constraints in equations (5.4) to (5.7). The exponent i in equation (5.11) was chosen to be equal to 2 for $n = 5$ and 10 and equal to 3 for n greater than 10. Using these initial parameters, the initial state of the network, the weight matrix and the external bias were calculated according to equations (5.8) to (5.11). Several samples for each value of n were generated and simulations of the network were performed for various sizes of k . Figure 5.2 shows a typical k -clustering produced by the analog neural network.

In Table 5.2 we report the number of times the neural network was able to identify the input clusters exactly. When the network converged to a solution, it was found that this happened always in less than 10,000 iterations. Hence when it did not converge to a stable state in 50,000 iterations, it is said to be *timed out*. In cases where the optimal clustering was not found, a different, non-optimal clustering of the given points was generated. In all the cases, a *valid* clustering is generated. An additional constraint which is not specified in Section 5.1 is that each cluster should contain at least one point. In terms of our neural network, this translates to the constraint that every row contains *at least* a single selected neuron. As this constraint is difficult to specify and enforce, we merely report the number of times this phenomenon occurred.

Table 5.2 Performance of the Neural Net based clustering. Each row represents a set of experiments. We report the number of samples tried, the number of clusters and the exponent l in equation (5.11). The columns "Optimal" and "Non Optimal" report the number of times the neural network produced optimal and non optimal clusterings, respectively.

n	samples	k	l	Optimal	Non-Optimal	$< k$ clusters	Timeout
5	50	2	2	45	2	2	1
10	50	2	2	38	12	0	0
	50	3	2	21	13	4	12
15	50	2	2	38	12	0	0
	50	3	2	20	22	0	8
	50	4	3	10	13	3	21
20	25	2	3	19	6	0	0
	25	3	3	12	5	3	5
	25	4	3	0	12	6	7
	25	5	3	0	8	4	13
25	25	2	2	18	7	0	0
	25	3	3	7	8	1	9
	25	4	3	0	11	5	9
	25	5	3	0	11	1	13
30	25	2	3	15	10	0	0
	25	3	3	5	15	3	2
	25	4	3	2	14	0	9
	25	5	3	0	7	1	17
35	25	2	3	11	14	0	0
	25	3	3	5	11	1	5
	25	4	3	1	13	1	10
	25	5	3	1	4	1	19

Figure 5.2 An example of grouping 35 points into 5 clusters which were exactly identified by the neural network. The co-ordinates are scaled by 100.

Chapter 6

The Travelling Salesman Problem

The Travelling Salesman Problem can be easily stated as follows: “A salesman is required to visit in some sequence, each of n cities, such that no city is visited more than once and that the total distance travelled is minimized”. More formally,

Input: A set of n cities $C = \{c_1, c_2, \dots, c_n\}$ and the distances $d(c_i, c_j)$ for each pair of cities $c_i, c_j \in C$.

Problem: Find a tour i.e., a permutation $\pi = \langle c_{\pi(1)}, c_{\pi(2)}, \dots, c_{\pi(n)} \rangle$ of the cities in C , minimizing the *length of the tour* defined as:

$$D_\pi(C) = \left[\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) \right] + d(c_{\pi(n)}, c_{\pi(1)}).$$

Typical distance measures between two cities c_i, c_j are: Euclidean distance, travel time in hours, fuel consumption, highway tolls, length of the roads connecting c_i and c_j .

In this thesis, we shall be studying a restricted instance of this general problem. In our case, the cities are points on the two-dimensional plane, i.e., the city c_i has the co-ordinates (x_i, y_i) . The Euclidean distance between a pair of cities c_i and c_j is defined to be $\sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$. When the distance function $d(c_i, c_j)$ is defined to be the Euclidean distance, then we have an instance of the Euclidean Travelling Salesman Problem or the Geometric Travelling Salesman Problem. This problem is known to be NP-complete [Papadimitriou 1977]. There are no known polynomial

time algorithms which produce the minimum length tour [Garey and Johnson 1979]. The formulation of the Travelling Salesman Problem (TSP) dates back to 1931-32 and since then it has resisted the efforts of many researchers to find an efficient algorithm yielding the optimal tour. The best known deterministic solution to this problem is due to [Held and Karp 1962] and has a complexity of $O(n^2 2^n)$, where n is the number of cities. There are numerous algorithms in the literature which strive to produce *approximate* solutions. Some of the well known heuristics are due to [Lin and Kernighan 1973] and [Bentley 1990].

Though the problem has been posed in a "travel setting", namely, a salesman wishing to minimize his total travel distance, various other applications of this problem exist, which are quite different. Some of the well known applications for this problem are:

Vehicle Routing: Determining for a fleet of vehicles, which customers should be served by which vehicles and in which order should each vehicle visits its customers.

Computer Wiring: A set of pins (on an electronic component) must be connected by wires in such a way that the total wire length used is minimized.

Job Sequencing: n jobs have to be sequenced on a single machine. The machine must be in a certain state S_i in order to do job i . A certain amount of time is required to change the state of the machine between jobs. The task is to complete all of the jobs, in any order, but in the shortest possible total time.

Cutting Wallpaper: We are to cut n sheets of wallpaper from a very long roll. The wallpaper has a pattern that repeats itself at unit intervals. Sheet i starts at position s_i and ends at position f_i , with respect to the starting point of the pattern. The task is to minimize the total intersheet waste rounded up to the nearest pattern unit.

Rural School Bus Routing: Given a network of roads (not all of them connected) and a fleet of school buses, the problem is to transport students waiting

at bus stops using the minimum number of buses while adhering to the restrictions laid down by the School Board.

Numerous generalizations and variations of the TSP exist in the field of combinatorial optimization. A comprehensive coverage of several aspects of TSP can be found in the excellent book entitled *The Travelling Salesman Problem: A Guided tour of Combinatorial Optimization* [Lawler *et al.* 1985].

6.1 Hopfield's Neural Network Solution

Hopfield and Tank proposed an analog neural network for solving the TSP [Hopfield and Tank 1985]. In their solution, each city is represented by a row of n neurons, implying a network of size $N = n^2$ neurons. The relative position of a city in the tour is given by the column number. To be a valid tour, one and only one such neuron in each row and column must be set to 1 and all others must be 0. For example, if the neuron from row 2 and column 5 is selected, (i.e., has a value = 1), then city 2 is in the fifth position in the final tour. Let A, B, C, D be 4 cities to be visited with the distances given as $d_{AB}, d_{AC}, \dots, d_{CD}$. Figure 6.1 shows one possible tour of the four cities — they are visited in the sequence A, C, D, B .

Figure 6.1 A possible TSP solution for 4 cities.

City	Position			
	1	2	3	4
A	1	0	0	0
B	0	0	0	1
C	0	1	0	0
D	0	0	1	0

The constraints for the TSP are the following:

1. Visit all cities
2. Visit each city only once.

3. Produce a tour of shortest distance.

Considering that we are operating in a high-gain limit, we have to now design an energy function that will satisfy the following:

- the value of the energy function will be low only for those solutions that produce a single 1 in each row and column.
- the energy function designed must favour only solutions having short paths.

The mapping of the TSP onto a network of N neurons is done as follows. Let neuron V_{Xj} represent city X being visited in position j of the tour. If $V_{Xj} = 0$, then, city X is *not* visited in position j and if $V_{Xj} = 1$, then city X is visited in position j . We will therefore have a matrix of neurons V_{Xj} . The energy function designed to satisfy the constraints of the problem is discussed below. Let A, B, C and D be some constants.

1. The condition that each row contains no more than a single 1 is specified by

$$E_1 = \frac{1}{2}A \left(\sum_X \sum_i \sum_{j \neq i} V_{Xi} V_{Xj} \right) \quad (6.1)$$

2. The condition that each column contains no more than a single 1 is realized by minimizing

$$E_2 = \frac{1}{2}B \left(\sum_i \sum_X \sum_{j \neq k} V_{Xi} V_{Xj} \right). \quad (6.2)$$

3. Guaranteeing that there are exactly n 1's in the whole matrix is achieved by

$$E_3 = \frac{1}{2}C \left(\sum_X \sum_i V_{Xi} - n \right)^2. \quad (6.3)$$

4. The requirement that the solution favour only valid tours with short paths is fulfilled by

$$E_4 = \frac{1}{2}D \left(\sum_X \sum_{j \neq k} \sum_i d_{Xj} V_{Xi} (V_{Yj+1} + V_{Yj-1}) \right). \quad (6.4)$$

The total energy function for the TSP problem is given by $E = E_1 + E_2 + E_3 + E_4$ and it represents the length of a particular tour and the shortest tour will be the minimum of this function. The next step is to determine the elements in the weight matrix T . This is done by algebraically simplifying the designed energy function into a form similar to the global energy function given by (2.2) (with zero thresholds) and then reading off the weights. The following are the weights as presented in [Hopfield and Tank 1985]:

$$\begin{aligned}
 t_{X_i, Y_j} = & -A\delta_{XY}(1 - \delta_{ii}) && \text{(row inhibitions)} \\
 & -B\delta_{ij}(1 - \delta_{YY}) && \text{(column inhibitions)} \\
 & -C && \text{(ensures exactly } n \text{ 1's in total)} \\
 & -Dd_{XY}(\delta_{j,j+1} + \delta_{j,j-1}) && \text{(the distance term)}
 \end{aligned} \tag{6.5}$$

where, δ_{ij} is the Kronecker symbol.

6.2 Discussion of Hopfield's TSP Solution

Hopfield & Tank report that sufficiently large values for the constants A , B , and C will represent valid tours, while a large value for the constant D will ensure that a short tour will be found. It has also been noted that convergence of Hopfield's solution to the TSP is highly dependent on these coefficients. The important point is that a systematic way to determine the appropriate values for these constants is missing. Moreover, proofs for the correctness of their neural network solution to the TSP are also not given. That is, there is no guarantee that the neural network produces correct solutions. In fact, for a TSP with 10 cities and with the constants $A = B = 500$, $C = 200$, $D = 500$, given by Hopfield & Tank, only an 80% success rate was reported [Hopfield and Tank 1985]. That is, only 16 out of 20 trials converged to valid tours and about 50% of the solutions produced one of the 2 possible shortest paths! What we desire is, of course, to obtain the global minimum, as it gives the optimum solution to the problem. However, in cases when the minimum was not reported, they are not even sure that the solution is a "valid" one!

2 OF/DE 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

Many researchers have tried to reproduce Hopfield's results but with little success. Perhaps the most acclaimed critical analysis paper on the Hopfield solution to TSP is by Wilson & Pawley [Wilson and Pawley 1988]. Their experiments on the TSP (for $n = 10$) using Hopfield's energy function yielded only 8% valid tours, 48% of the tours were invalid tours while 44% of the attempts did not converge in 1000 iterations (which were termed as being "timed out"). But it was observed that when a valid tour was found, the quality of the tour was very good [Kamgar-Parsi and Kamgar-Parsi 1989]. Wilson & Pawley suggested several modifications so as to generate valid and better quality tours. Among those are changes to the parameter values, varying input currents to the neurons, having a different starting (initial) state and also fixing some city positions in the tour. For instance, fixing the position of two cities led to a remarkable difference in the percentage of valid tours generated (42% valid tours), while a different starting state yielded only 18% valid tours. It was concluded that though the method introduced by Hopfield & Tank was of primary importance, it lacked a lot of detailed justification [Wilson and Pawley 1988].

Since Hopfield & Tank do not suggest a systematic method to choose the constants A, B, C and D in their energy formulation, [Davis and Ansari 1987] and [Hegde *et al.* 1988] provide an experimental analysis on the choice of these parameters. Specifically, Davis & Ansari present an experimental sensitivity analysis of the network (i.e., the ability of the network to produce a valid tour) when the input bias and the constant D (associated with the distance term) are varied. Hegde *et al.* present a cookbook approach to setting the TSP parameters. They provide some empirical relationships among the constants in the energy function and also present an experimental understanding of why the Hopfield TSP energy function is not a useful tool for TSP computations as the number of cities increases.

[Aiyer *et al.* 1990] and [Rong and Ze-min 1991] present a theoretical investigation into the performance of the Hopfield model. The original non-linear sigmoid output function $g(u_i) = \frac{1}{2}(1 + \tanh(\frac{u_i}{\beta}))$ is replaced by a piece-wise linear function $g(u_i) = 0.5 + \frac{1}{4\beta}u_i$. It is shown that this new g function will not alter the dynamics

of the network because the resulting system of differential equations will be very similar to the system given in equation (2.7). Moreover, it allows the evaluation of the eigenvalues of the interconnection weight matrix T and how these eigenvalues relate to the constants in the Hopfield's TSP energy function.

[Lee and Sheu 1988] give an understanding on the local minima of Hopfield's energy function in general. [Shirazi and Yih 1989] present a rigorous matrix-based analysis of the Hopfield model as well. They show that the results obtained by applying the Hopfield approach to most NP-complete problems in general is very unsatisfactory, because (i) the idea of mapping constraints in the optimization problem to weights on the interconnections in the neural networks will become invalid when the number of constraints is greater than the number of interconnections. In such a situation, several constraints must be encoded on to one weight and this may not guarantee a valid solution. (ii) The energy function is only guaranteed to settle in a state lower than the initial state and this equilibrium state need not correspond to the minimum of the cost function of the original optimization problem.

[Yue and Fu 1990] analyze Hopfield's approach by bringing out the relationship between optimization problems and that of associative memory. The Hopfield network was designed to function as an associative memory using an information storage algorithm [Hopfield 1982, Hopfield 1984] to locate the associated patterns at the local minima of a well defined energy function. The TSP is a direct application of an associative memory. The TSP is transformed into a memory association problem by means of an energy function which incorporates the constraints of the problem. Yue & Fu claim that the Hopfield's TSP network contains too much information as $n!$ patterns are stored with only n^2 neurons causing an overload in the network and hence its inability to produce valid tours consistently.

Many researchers worked on methods for forcing the Hopfield network to yield a permutation matrix as the solution for the TSP. Even though the network scales as $O(n^2)$ in problem size, it is known that this model performs well on non-Euclidean or even non-metric TSPs (i.e., problem instances in which the distance measure fails to satisfy the triangle equality). Numerous alternate neural network models,

e.g., the Elastic Net [Durbin and Willshaw 1987], require only a linear number of processing units, but are not as robust (ability to produce good solutions) for the non-Euclidean and the non-metric TSPs [Kahng 1989].

We will now present a survey of the work done to improve the Hopfield solution to the TSP. This will be followed by a brief discussion of the alternate neural network models, where again, the TSP has been used to demonstrate the capabilities of the proposed models.

6.3 Modifications to the Hopfield and Tank TSP formulations

Most of the modifications done to Hopfield's TSP solution can be categorized as

1. Those that did not change the original TSP energy function proposed by Hopfield & Tank, but analyzed the solution carefully and derived bounds for the constants.
2. Those that presented an alternate energy function to improve the incidence of valid tours.
3. Those that combined the "goodness" of Hopfield's solution with other methods, like simulated annealing [Kirkpatrick *et al.* 1983].

We shall now briefly present some results that have been classified as belonging to each of the three categories discussed above, in sequence.

[Abe 1991, Abe 1989] analyzes the convergence characteristics of the Hopfield model in general. Firstly, he establishes the condition for the original Hopfield model to converge to a vertex of an n -dimensional hypercube. Based on this result, a method to analyze and suppress the spurious states (those that result in infeasible solutions) in the network is presented. The direct result of this method is to place bounds on some of the parameters used in the Hopfield's energy function for the TSP given by equations (6.1) to (6.4). 100% feasible solutions are reported in experiments

when the constants A, B, C were chosen to be > 3.33 . [Clement *et al.* 1988] claim that an alternative normalization of the parameters will force the occurrence of valid solutions. They suggest for example, that the parameter D in Hopfield's energy function be normalized to $D' = D/\text{hypotenuse}$, where *hypotenuse* is the diagonal of the bounding box of all the city locations.

Tagliarini & Page reformulate Hopfield's TSP energy function to the following [Tagliarini and Page 1987, Tagliarini and Page 1988]:

$$E = \frac{1}{2} \sum_i \sum_j t_{ij} v_i v_j - \sum_i v_i I_i$$

where, $t_{ij} = -M$, an arbitrary real positive number, for all $i, j \in S$, a subset of neurons in the Hopfield model. $I_i = Mk - 1$, which enforces " k out of n " constraints and guarantees that exactly k neurons will be active at equilibrium. An understanding of the constant D , which represents the scaling factor for the tour distance is achieved by means of experiments.

Brandt *et al.* give an alternative formulation for the TSP [Brandt *et al.* 1988]. Their modified TSP energy function is given by

$$E = \frac{D}{2} \sum_{ij} \sum_{kl} r_{ijkl} x_{ij} x_{kl} + \frac{A}{2} \sum_i \left(\sum_j x_{ij} - 1 \right)^2 + \frac{B}{2} \sum_j \left(\sum_i x_{ij} - 1 \right)^2$$

where, d_{ik} is the distance between city i and city k and

$$r_{ijkl} = \begin{cases} 0 & \text{if } |j - l| \neq 1 \bmod n \\ d_{ik} & \text{otherwise.} \end{cases}$$

The first term represents the total tour length and the second and the third terms enforce the constraints for a permutation matrix. Experimental results suggest that valid tours can be consistently obtained as compared to the original Hopfield's solution, but it is not proven to be always the case.

[Szu 1988] takes a totally different approach to overcome the problems encountered by Wilson & Pawley. He claims that the Hopfield's TSP formulation incorporates only the *necessary* condition, which means that the total number of neurons selected in the whole matrix of n^2 neurons must be n . The constraints

in equations (6.1)–(6.4) are not *sufficient* to guarantee a valid solution, because it is sometimes possible for a whole column of neurons to be selected and the rest remaining unselected¹. It only ensures that n neurons are selected in total. The remedy suggested is by means of the third and fourth terms of the following energy function, which enforce the sufficient condition.

$$\begin{aligned}
 E = & A \sum_X \sum_i \sum_{j \neq i} v_{Xi} v_{Xj} + B \sum_i \sum_X \sum_{X \neq Y} v_{Xi} v_{Yi} + \\
 & C \sum_X \left(\sum_i v_{Xi} - 1 \right)^2 + C \sum_i \left(\sum_X v_{Xi} - 1 \right)^2 + \\
 & D \sum_X \sum_{Y \neq X} \sum_i d_{XY} v_{Xi} (v_{Y,i+1} + v_{Y,i-1}).
 \end{aligned}$$

The constants A, B, C are always set equal to 1. The model is based on neurons with binary output (and not the usual g function) and analog input. The results presented seem to indicate only a 9.1% incidence of valid tours for 10 cities [Kahng 1989].

Kahng proposes heuristic modifications to the Hopfield's TSP solution [Kahng 1989]. Experimental results presented for various geometric heuristics (convex-hull based) and combinatorial geometry and discrete optimization heuristics (forbidden edge, plane cutting) yield valid tours about 92% of the times. Their implementation for the original Hopfield solution resulted in valid tours only 15% of the times. [Neotzel and Graziano 1988] present an alternate formulation for the TSP based on the Hopfield model. They claim that the original TSP solution has a $2n$ -fold tour degeneracy¹ and they present methods to remove degeneracies. For example, their penalty function forces city A to occupy the first position in the tour and discourages all other cities from occupying this position. [Gunn and Weidlich 1989] have also tried to reproduce Hopfield's TSP results but without success. They present a "derivative" of the Hopfield model, where heuristics are used to evaluate parameter values. They also claim good results through experimentation.

¹Each tour can be specified in an anti-clockwise or clockwise direction with any one of the n cities in the starting position, resulting in $\frac{2n}{n}$ distinct tours.

Van den Bout & Miller propose a revised TSP energy function specified by [Van den Bout and Miller 1988]:

$$E = \frac{B}{2} \sum_i \sum_X \sum_{Y \neq X} v_{X_i} v_{Y_i} + \frac{1}{2} \sum_X \sum_{Y \neq X} \sum_i d_{XY} v_{X_i} (v_{Y_{i+1}} + v_{Y_{i-1}}).$$

While the first term discourages two cities from occupying the same position in the final tour, the second term is for the total distance. They suggest that $B = 4\sqrt{n}$ will guarantee no more than once city in any tour position. This energy function does not explicitly handle the constraint that each city should exist at exactly one position in the tour. A simulated annealing algorithm used in conjunction with this energy function ensures this condition. Their experimental results always report valid tours and the average tour length was 12% longer than those produced by pure simulated annealing. [de Carvalho and Barbosa 1989, de Carvalho and Barbosa 1990] also propose a similar combination solution where simulated annealing techniques are built into the Hopfield model.

Some of the solutions to the TSP cannot be strictly categorized, but nonetheless, they use some form of the Hopfield model to improve on the quality of the original TSP energy formulation. In Hopfield's model, a neuron V_{X_i} denotes city X as occupying position i in the tour. [Joppe *et al.* 1990] present an alternate neural network where a neuron V_{XY} indicates whether or not cities X and Y are adjacent to the tour. They utilize two layers of networks, where the first (of size $O(n^2)$) determines which cities are adjacent to each other and the second layer (also consisting of $O(n^2)$ neurons) detects and eliminates closed subtours. They claim faster convergence during computer simulations and that larger instances of the TSP can be solved readily. They report experiments on a 22-city tour for 50 runs which produce 38 optimal tours, no solution was found in 3 instances and the rest 9 runs are unaccounted for.

A new *competitive-Hopfield* neural network, a separate network, is designed to function together with the original Hopfield network. The competitive net is used to realize the constraint functions of the optimization problem while the Hopfield net is programmed with a reduced set of constraint functions [Lee and Sheu 1988, Lee and Sheu 1990]. The experimental results reported are that valid tours were always generated with a reasonable amount of speed up.

As it is believed that the Hopfield TSP solution becomes unsuitable for large problem sizes [Hegde *et al.* 1988] and [Foo and Szu 1989] introduce a Divide and Conquer neural network to solve large-scale optimization problems. They present the concept of a divide-and-conquer network, where one layer of neurons is used for partitioning the problem and the second layer is used for combining results. Their ad-hoc cut-and-splice technique is used for solving the TSP. Experimental results show improvements over the modified Hopfield TSP solution proposed in [Szu 1988].

It is also interesting to note that parallel execution models were developed to implement Hopfield's TSP solution in parallel [Bagherzadeh *et al.* 1987]. Also, variations of the original TSP are solutions for the Dynamic TSP (where the cost changes with respect to the time of travel) and the Multiple TSP. These variants have been modeled using the Hopfield neural network in [Shinozawa *et al.* 1991] and [Wacholder *et al.* 1988] respectively.

6.4 Other Neural Network Models

Having presented a brief survey of the attempts at correcting Hopfield and Tank's solution to the TSP, we now present a list of alternate neural network models on which the TSP has been the first problem used as a test.

1. The *Elastic Net* uses an iterative procedure to solve the TSP. The tour is initially a small circle centered at the centroid of the cities. This closed tour is gradually elongated non-uniformly until it eventually passes through all the cities to form a tour [Durbin and Willshaw 1987, Durbin *et al.* 1989, Burr 1988].
2. An *iterated descent* or hill climbing algorithm for use in conjunction with hard wired neural networks results in faster convergence to the energy minimum [Baum 1986].

3. A data guided *Lexisearch Algorithm* for the TSP, where, a pre-processing of the distances of the given cities improves the efficiency of the search for the optimal tour [Pandit and Srinivas 1991].
4. A *Deterministic Annealing and Constrained Clustering* algorithm using the Elastic Net to solve TSP [Rose *et al.* 1991].
5. *FLEXIMAP* is a self-organizing neural network based on Kohonen's Adaptive neurons for solving the TSP with $O(n)$ time and space complexity. Their experiments produce tours which are within 10% of the optimum (which was a nearest-neighbour 2-OPT algorithm) [Fritzke and Wilkie 1991].
6. A new artificial neuron model called the *Gaussian Machine* incorporates the principles of both the Hopfield Network and the Boltzmann Machine. They propose an alternate energy function for the TSP by which the probability of finding an optimal tour is 22.2% compared to 11.8% for the original Hopfield solution [Akiyama *et al.* 1989].
7. A matrix based neural network architecture called *The Matrix Column Rotation Architecture (MCRotA)* and *The Matrix Column Reversal Architecture (MCRevA)* with only $O(\log n)$ interconnections for a TSP of size n [Lister 1990].
8. An *Adaptive* neural network algorithm for solving the TSP using n^2 neurons (similar to Hopfield's solution) and an energy function to encode problem constraints is shown to improve the tour given by 2-OPT by at least 1.1% [Xu and Tsai 1990].
9. A fast *Parallel Simulated Annealing* algorithm to solve the TSP, which can be applied not only to SIMD machines with linear interconnections but also to the HYPERCUBE class of computers without linear interconnections [Jeong and Kim 1990].

10. A new approach of *encoding logical constraints* into the neural network cost functions yielded valid solutions nearly always [Thomae and den Bout 1990].
11. A *Training* algorithm similar to Kohonen's training algorithm incorporating ideas from Elastic Net to solve the TSP produced results superior to the ones produced by pure simulated annealing [Matsuyama 1990].
12. The non-Euclidean TSP is solved heuristically using a new *Continuous State Boltzmann Machine*. Their experiments produce valid tours 75% of the times [Gutzmann 1987].
13. Using an $O(n \log n)$ number of neurons, where the state of the neurons varies between +1 and -1 (inclusive) and all the neurons update their states simultaneously, a *concurrent neural network algorithm* to solve the TSP is proposed. Experiments on an NCUBE multiprocessor yield valid tours 93% of the times for a 32-city problem [Toomarian 1988].
14. A neural network with $O(n^3)$ connections compared to $O(n^4)$ connections in the Hopfield TSP solution is also known to perform well [Park and Jeong 1990].

6.5 Our Neural Network Solution to the TSP

In the following section, we will present "another" neural network solution to the TSP. In contrast to the surveyed previous approaches, our solution has the advantage that we can **prove** that the network will **always** produce a feasible solution, and the final energy (at equilibrium) is proportional to the length of the tour. As it turns out, our methods developed for solving geometric problems (particularly, the ones for point congruence) are also helpful for improving Hopfield's TSP solution.

The neural network consists of a matrix of $N = n \times n$ neurons. Each city is represented by a row of n neurons. The position of each city in the final tour is given by the column number. Each neuron $i = \langle \text{city}_i, \text{pos}_i \rangle$, $i = 1 \dots N$, represents

a particular city as being in a particular position in the final tour, i.e., $city_i$ = the row number of neuron i and pos_i = the column number of neuron i . We specify the network by means of an initial state for these N neurons, the value of the external bias and also the strength of the interconnections between these neurons. In a preprocessing phase, the distance between every pair of cities is calculated and the maximum of such distance is found. The terms of the interconnection weight matrix T are defined to be

$$t_{ij} = t'_{ij} + t''_{ij}$$

where,

$$t'_{ij} = -C(1 - \delta_{ij}) \cdot share_{ij} \quad (6.6)$$

$$t''_{ij} = -D(1 - \delta_{ij}) \cdot value_{ij} \quad (6.7)$$

where,

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (6.8)$$

$$share_{ij} = \begin{cases} 1 & \text{if } city_i = city_j \text{ or } pos_i = pos_j \\ 0 & \text{otherwise} \end{cases} \quad (6.9)$$

$$value_{ij} = \begin{cases} \frac{dist(city_i, city_j)}{M} & \text{if } (pos_i, pos_j) \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \quad (6.10)$$

$$M = \max_{k,l} \{dist(k,l)\} \quad 0 < k, l \leq n. \quad (6.11)$$

Select constants $\beta > 0$, $\omega > 0$, $C > 0$ and $D > 0$, such that they satisfy the following constraints:

$$\beta \ll 1 \quad (6.12)$$

$$D > 0 \quad (6.13)$$

$$C \geq \frac{1}{(1-\gamma)}(\Delta\beta + \omega) \quad (6.14)$$

$$\omega \geq 3(\Delta\beta + Dn). \quad (6.15)$$

The bounds for the constants C , and ω stated in equations (6.14) and (6.15) will be derived later. For each neuron i , we set the external input or *bias* ϵ_i :

$$\epsilon_i = \omega, \quad \text{for all } i.$$

The initial state of the system is set to:

$$s_i = 0.5 + \text{random} \quad (6.16)$$

where, *random* is a random number with the property:

$$-r \leq \text{random} \leq r, \quad r \ll 0.5.$$

Observe that with the T -matrix as defined above, we have satisfied the requirement **T-SYM** which guarantees the network to converge to a local minimum. We would now have to show that the constraints for the TSP are satisfied using the interconnections stated above. That is, we have to establish that we indeed get a permutation matrix and also that the energy of the system is proportional to the total distance of the tour covered. We have the following lemmas which establish the correctness of our network. Note that the following proofs are analogous to Lemma 4.1 to Lemma 4.3 in Chapter 4.

Lemma 6.1 *No two neurons in the same row or in the same column are selected.*

Proof:

(The proof is identical to Lemma 4.1). We prove the result by contradiction. We know that in equilibrium, $\underline{u} = \mathbf{T}\underline{s} + \underline{\epsilon}$. Let us assume that two neurons i and k , in the same row/column are selected; hence, $s_i, s_k \geq 1 - \gamma$.

Since $t_{ii} = 0$,

$$\begin{aligned} u_i &= t_{ik}s_k + \sum_{j \neq k} t_{ij}s_j + \epsilon_i \\ &= (t'_{ik}s_k + t''_{ik}s_k) + \sum_{j \neq k} t_{ij}s_j + \epsilon_i \\ &= (-C \cdot s_k - D \cdot \text{value}_{ik} \cdot s_k) + \sum_{j \neq k} t_{ij}s_j + \epsilon_i \\ &\leq -C(1 - \gamma) + \epsilon_i. \end{aligned} \quad (6.17)$$

Substituting $C = \frac{1}{(1-\gamma)}(\Delta\beta + \omega)$, we get $u_i \leq \Delta\beta$. This means that neuron i will be unselected, which is a contradiction to the hypothesis that i was selected. Hence we have proved that no two neurons in the same row or column can be selected simultaneously. This implies that there can be at most one neuron selected in any row or column. $\square$

Corollary 6.1 *At most n neurons, in total, are selected simultaneously.*

So far, the results we have derived show that **at most** n neurons can be selected from the grid of n^2 neurons. For a neural network stable state to correspond to a valid tour, we must ensure that **exactly** n neurons are selected. To establish the correctness of the solution, we state and prove a technical result.

Lemma 6.2 *If an entire column c of neurons were not selected, then every row would contain a selected neuron.*

Proof:

Assume that there exists a row r which has all the neurons unselected. So, we have an entire row and column (r and c , respectively) unselected. Consider neuron i , which is at the intersection of row r and column c . At equilibrium, since $\dot{u}_i = 0$,

$$\begin{aligned} u_i &= \sum_j t_{ij}s_j + \epsilon_i \\ &= \sum_j (t'_{ij} + t''_{ij})s_j + \epsilon_i \end{aligned}$$

Proceeding in a manner identical to the proof in Lemma 6.2, we have the following inequality.

$$u_i \geq -2C(n-1)\gamma - D\{n(n-1)\gamma + n\} + \epsilon_i. \quad (6.18)$$

Choose $\gamma = \frac{1}{8(n-1)}$ (which makes γ sufficiently small, i.e., $\gamma \ll \frac{1}{2}$), $\omega = 3(\Delta\beta + Dn)$ and $C = 2(\Delta\beta + \omega)$. This will result in $u_i \geq \Delta\beta$, which means that $s_i \geq (1-\gamma)$, implying that neuron i is indeed selected. This is a contradiction to our assumption that row r has all its neurons unselected! $\square$

Hence, we have the following theorem:

Theorem 6.1 *In every row or column, exactly one neuron is selected. Thus, the neural network always produces a valid tour for the travelling salesman.*

Proof:

Lemma 6.1 permits no two neurons to be selected in any row or any column. Assume, as in Lemma 6.2, that an entire column c of neurons is unselected. Then, by Lemma 6.2, every row contains a selected neuron. Since, one column is completely unselected, then, by the pigeon-hole principle, there exists a column with *two* selected neurons. This is again a contradiction to Lemma 6.1, which disallows *two* neurons to be selected in any row or column.

Hence we have shown that the salesman's tour covers all the cities and each city is visited only once. $\square$

In the following lemma we show that minimizing the energy of the network is equivalent to minimizing the total distance covered.

Lemma 6.3 *In the high gain case, the total energy is proportional to the length of the selected tour (plus a constant).*

Proof:

The energy of the network is given by:

$$E = -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j + \sum_i \left[\frac{1}{R_i} \int_0^{s_i} g^{-1}(s_i) ds \right] - \sum_i c_i s_i \quad (6.19)$$

For the high gain case, i.e., when $\beta \rightarrow 0$, the second term converges to zero [Hopfield 1984], hence,

$$\begin{aligned} E &= -\frac{1}{2} \sum_i \sum_j t_{ij} s_i s_j - \sum_i c_i s_i \\ &= -\frac{1}{2} \sum_i \sum_j t''_{ij} s_i s_j - N\omega \\ &= \frac{1}{2} D \sum_{i \text{ selected}} \sum_{j \text{ selected}} \text{value}_{ij} - N\omega \\ &= \frac{D}{2M} \sum_{i \text{ selected}} \sum_{j \text{ selected}} \text{dist}(\text{city}_i, \text{city}_j) - N\omega \end{aligned}$$

$\square$

Summarizing, we have presented a new TSP solution which guarantees to always report a valid tour. The optimal tour is obtained when the energy reaches a global minimum; in cases of a local minimum of the energy function, we get *near* optimal tours. To our knowledge, none of the previous solutions (see our literature survey) have such properties.

6.6 Experimental Results

Experiments were conducted to compare the performance of our neural network solution to the TSP with that of Hopfield & Tank's solution [Hopfield and Tank 1985]. Our solution was also compared to that obtained by the well known sequential heuristics, namely, the 2-OPT algorithm [Cres 1958, Syslo *et al.* 1983] and the variable k -OPT algorithm due to Lin & Kernighan [Lin and Kernighan 1973]. In the case of both the sequential methods, the initial tour was randomly chosen. We now describe the simulation of the two neural network solutions to the TSP, i.e., the one due to Hopfield & Tank (HT) and ours.

An instance of the problem is described by a set of co-ordinates for the n cities. For implementing the solution described in Section 6.5, the values for the constants were selected as follows: $\Delta = 19$, $\beta = 0.03$, $\gamma = 10^{-6}$, $r = 0.01$, $\delta t = 0.003$ and $D = 2500$. These constants are consistent with the constraints specified in equations (6.12) and (6.13). The low value of β ensures that we are operating in a high-gain case. From the co-ordinates of the cities, the weight matrix, the external bias and the initial state were calculated using equations (6.7) to (6.16). Once all the parameters for the neural network were known, we employed the simulator to numerically integrate the system of differential equations (2.7). The simulation is carried out until the neural network *converged* or until the prescribed number of iterations is exceeded *without convergence*. The latter condition is termed as *time-out*.

For implementing the HT solution, the constants A, B, C, D in equations (6.1) to (6.4) were given the values $A = B = D = 500$ and $C = 200$. These are

exactly the values reported by Hopfield & Tank in their experimental results section [Hopfield and Tank 1985]. β was set to 0.02 and ϵ , or the excitation bias, was equal to $n \times C$ (where n = number of cities). Although Hopfield & Tank used an initial setting of neurons such that $\sum_i s_i = n$, this choice did not work very well in our experiments. We obtained better results when the initial state for each of the neurons was set to 0. As in the previous case, an upper limit on the number of iterations was used to declare a "time-out".

Our experiments were conducted for various values of the parameter n which represents the number of cities. The co-ordinates of the cities were uniformly distributed on a 100×100 square. For each value of n several samples were generated. For each sample, the tours obtained by the four methods (our solution, HT solution, 2-OPT and Lin-Kernighan heuristics) were recorded. In all the experiments, our neural network *always* produced a valid tour. A time-out was observed only in 1 sample involving 20 cities. The HT network produced a valid tour in only a very small fraction of the experiments (1 out of 45 samples for $n = 10$). It was observed that although HT network produced invalid tours very frequently, when it did produce a valid tour, the quality of the tour was very good. In Figure 6.2 to Figure 6.4 we show the tours produced by our network, the HT network and the Lin-Kernighan heuristic, respectively, for a specific sample involving 10 cities.

Figure 6.2 Tour of 10 cities produced by our neural network.

Length = 470.49

Figure 6.3 Tour of 10 cities produced by the HT neural network.

Length = 401.64

Figure 6.4 Tour of 10 cities produced by Lin-Kernighan heuristic.

Length = 286.55

Figure 6.5 Tour of 20 cities produced by our neural network.

Length = 831.54

Figure 6.6 A Random tour of 20 cities.

Length = 1134.27

A comparison of the tour produced by our neural network to that of a random tour is given in Figure 6.5 and Figure 6.6. Since the HT network did not produce valid tours a significant number of times, the mean length of the tour is not reported for this algorithm.

Figure 6.7 shows the mean length of the tours produced by our network, the sequential heuristics and the random tour as a function of the city size n . These means were obtained from various random samples for the same size n . A comparison of the performance of our neural network to that of Hopfield & Tank's is presented in Table 6.1. It can be clearly seen that the performance of our network is far superior to that of HT solution.

In Table 6.2 we tabulate the mean percentage difference between the length of the tours produced by our network and the sequential heuristics. Note that a positive (negative) percentage indicates that our neural network is better (worse) than the heuristic it is being compared to. It is clear that the sequential heuristics outperform both the neural network solutions. Our solution yields tours which are significantly

Figure 6.7 Mean tour length given by our neural network (*NN*) in comparison with (a) Lin-Kernighan algorithm (*LK*) (b) 2-OPT algorithm (2 - *OPT*) (c) A Random tour (*RAND*), for various sizes of n .

Table 6.1 Performance of our neural network based TSP compared to Hopfield & Tank's Solution.

n	Samples	Our Net		HT		
		Valid	Time-Out	Valid	Invalid	Time-Out
5	35	35	0	2	14	19
8	35	35	0	2	22	11
10	30	30	0	0	14	16
12	25	25	0	0	13	12
15	25	25	0	0	8	17
18	25	20	0	0	0	20
20	15	14	1	0	1	14
25	10	10	0	0	2	8

shorter than a random tour, but are longer than the minimum tour lengths reported by 2-OPT and k -OPT algorithms. The advantage of our solution is that we *never* obtain invalid tours. The quality of the tour seems to suffer probably because of rigid syntactic constraints.

Table 6.2 Mean percentage difference between the tour lengths produced by our neural network and the other sequential heuristics.

% diff. Length of our tour Vs.				
n	Samples	Lin-Kern	2-OPT	Random
5	35	-7.83	-7.83	16.9742
8	35	-27.585	-27.291	16.8763
10	30	-34.211	-33.975	16.2202
12	25	-42.738	-42.335	17.0091
15	25	-47.415	-46.137	20.6687
18	25	-51.074	-50.034	19.1154
20	15	-53.153	-51.501	27.2896
25	10	-62.473	-61.521	11.5707

Chapter 7

Conclusions

7.1 Summary of the Thesis

The pioneering work by Hopfield [Hopfield 1984] led many researchers to explore the usefulness of *Analog Hopfield Neural Networks* to solving problems which are either known to be NP-hard or have a high combinatorial complexity. Since the *Analog Hopfield Neural Networks* naturally minimize a quantity termed as the *energy*, they have been employed in solving problems which involve optimization. The network is designed in such a way that its stable states can be interpreted as solutions to the problem posed. Some of the problems that were tackled using this approach include graph colouring, list matching, scheduling and image processing.

In this thesis, we have studied the applications of analog networks to solve Computational Geometry problems. The set of problems chosen are either known (or conjectured) to be NP-hard or have a very high sequential complexity. The problem of triangulating a planar point set is conjectured to be NP-hard, whereas, partitioning a set of points into k clusters and the Euclidean TSP are known to be NP-complete. The approximate point congruence problem has a sequential time complexity of $O(n^6 \log n)$.

Most of the existing literature on applications of neural networks to optimization problems can be described as follows. The first step in the process is to formulate the objective function (energy) as a quadratic form. Once the energy function

is designed, the weight matrix can be easily deduced. The advantage with this approach is that the *global* minimum of the analog network exactly yields the optimal solution to the optimization problem. Whereas, if the network stabilizes to a local minimum, the correctness of the solution cannot be guaranteed. As a result, the network produces very good solutions if its output is syntactically correct, but may fail to do so in many situations.

We have taken a slightly different approach. We have placed great emphasis on producing feasible solutions always. We endeavoured to start from the perspective of the interconnection weight between two neurons and relate this to the problem being solved. The weights assigned by us have two components, namely, (i) the syntax or the constraints part and (ii) the objective function related component. The first of these is the basis for enforcing *valid* solutions for all the problems studied. The merits of our approach is that we have been able to prove the correctness of the solutions yielded by the neural network. As a by product of these proofs, we have derived bounds for the various constants that were largely chosen so far on an *ad-hoc* basis. Our experiments have been done with a view to judge the quality of solutions given by the neural network.

It is noted however that a synthesis of these two approaches may be valuable and may perform better than any one approach in isolation. In this modified approach, we proceed with the traditional method in the initial stages of the neural network evolution, and switch to the weights which guarantee correct solutions. This hybrid approach may have the advantage of utilizing the traditional method to better explore the energy terrain in the initial phases, and then enforcing the syntax of the solution by switching to the alternate set of weights.

7.2 Unanswered Questions

We found that the neural network produced excellent triangulations of point sets, whereas the TSP tours produced are not competitive to the sequential heuristics. More study is needed to understand this behaviour. That is, it is worthwhile to study

the problem characteristics which make them specifically suitable or unsuitable for neural network implementations. Probably the first characteristic of problems suitable for neural networks is that their objective function must be a *summation* of terms. If a more complex function (for e.g., minmax of some quantity) is *approximated* by summations (as was done in point congruence and geometric clustering), the results are not very attractive.

The few other problems that we investigated but did not present in this thesis are:

- Disk cover problem [Lee and Preparata 1984]: Given a set of n points and several discs of radius r , the problem is to cover the n points with the minimum number of such discs.
- k -median problem [Megiddo and Supowit 1984]: Given a set S of n points and an integer k , the problem is to select a subset K of size k from S such that

$$\sum_{p \in K} \min_{q \in (S-K)} d(p, q)$$

is minimized, where $d(x, y)$ denotes the distance between points x and y .

- Minimum Length Pseudo-triangulation [Johnson 1982]: Given a set S of n points, the problem is to triangulate the convex hull of S in such a way that each point in S is covered by one of the edges introduced and the total length of the edges is minimized.

These problems do not admit to easy solutions based on the method we have introduced in this thesis. Perhaps a completely new way of thinking about these problems must be discovered, or perhaps, a better computational model (say, a hybrid model of Hopfield networks and simulated annealing) can be developed which is better suited to a larger class of problems.

References

- [Abe, 1989] S. Abe. Theories on the Hopfield neural networks. In *IEEE Int. 7 Conference on Neural Networks*, volume 1, pages 557–564, 1989.
- [Abe, 1991] S. Abe. Global convergence and suppression of spurious states of the Hopfield neural networks. In *Int. 7 Joint Conference on Neural Networks*, pages 935–940, 1991.
- [Aiyer *et al.*, 1990] S. V. B. Aiyer, M. Niranjan, and F. Fallside. A theoretical investigation into the performance of the Hopfield model. *IEEE Trans. on Neural Networks*, 1(2), 1990.
- [Akiyama *et al.*, 1989] Y. Akiyama, A. Yamashita, and M. Kajiura. Combinatorial optimization with Gaussian machines. In *IEEE INNS Int. 7 Joint Conference on Neural Networks*, volume 1, pages 533–540, 1989.
- [Akl and Lyons, 1993] S. G. Akl and K. A. Lyons. *Parallel Computational Geometry*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- [Akl and Toussaint, 1978] S. G. Akl and G. T. Toussaint. A fast convex hull algorithm. *Information Processing Letters*, 7:219–222, 1978.
- [Akl, 1989] S. G. Akl. *The Design and Analysis of Parallel Algorithms*. Prentice Hall, Englewood Cliffs, New Jersey, 1989.
- [Alt *et al.*, 1988] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity, and symmetries of geometric objects. *Discrete and Computational Geometry*, 3:237–256, 1988.

- [Atallah and Tsay, 1989] M. J. Atallah and J. J. Tsay. On the parallel-decomposability of geometric problems. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 104–113, 1989.
- [Atallah et al., 1989] M. J. Atallah, R. Cole, and M. T. Goodrich. Cascading divide-and-conquer: a technique for designing parallel algorithms. *SIAM J. Computing*, 18(3):499–532, 1989.
- [Atallah, 1984] M. J. Atallah. Checking similarity of planar figures. *Int. J. Computing and Information Science*, 13:279–290, 1984.
- [Atkinson, 1987] M. D. Atkinson. An optimal algorithm for geometrical congruence. *Journal of Algorithms*, 8:159–172, 1987.
- [Avis and Toussaint, 1981] D. Avis and G. T. Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE Trans. on Computers*, C-30(12):910–914, 1981.
- [Babuska and Aziz, 1976] I. Babuska and A. K. Aziz. On the angle condition in the finite element method. *SIAM Journal on Numerical Analysis*, 13(2):214–226, 1976.
- [Bagherzadeh et al., 1987] N. Bagherzadeh, T. Kerola, B. Leddy, and R. Brice. On parallel execution of the Travelling Salesman Problem on a neural network model. In *IEEE 1st Int. J Conference on Neural Networks*, volume 3, pages 317–324, 1987.
- [Barhen et al., 1987] J. Barhen, N. Toomarian, V. Protopopescu, and M. Clinard. Concurrent neuromorphic algorithms for optimization of the computational load of a hypercube supercomputer. In *IEEE First Int. J Conference on Neural Networks*, volume 4, pages 687–696, 1987.
- [Baum, 1986] E. B. Baum. Towards practical ‘neural’ computation for combinatorial optimization problems. In J. Denker, editor, *AIP Conference Proceedings 151: Neural Networks for Computing*, pages 53–58. American Institute of Physics, New York, 1986.

- [Bentley and Maurer, 1980] J. L. Bentley and H. A. Maurer. Efficient worst-case data structures for range searching. *Acta Informatica*, 13:155-168, 1980.
- [Bentley and Shamos, 1977] J. L. Bentley and M. I. Shamos. A problem in Multivariate Statistics: Algorithms, data structure and applications. In *Proc. 15th Allerton Conf. Commun., Contr., Comput.*, pages 193-201, 1977.
- [Bentley, 1975] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509-517, 1975.
- [Bentley, 1990] J. L. Bentley. Fast algorithms for Geometric Travelling Salesman Problems. Technical Report 151, AT&T Bell Labs, Murray Hill, NJ 07974, 1990.
- [Bern *et al.*, 1992] M. Bern, D. Dobkin, and D. Eppstein. Triangulating polygons without large angles. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 222-231. ACM Press, 1992.
- [Bramble and Zlamal, 1970] J. H. Bramble and M. Zlamal. Triangular elements in the finite element method. *Mathematics of Computation*, 24:809-820, 1970.
- [Brandt *et al.*, 1988] R.D. Brandt, Y. Wang, and A. Laub. Alternative networks for solving the Travelling Salesman Problem and the List Matching Problem. In *IEEE INNS Int. Joint Conference on Neural Networks*, volume 2, pages 333-340, 1988.
- [Brücker, 1977] P. Brücker. On the complexity of clustering problems. In B. Korte R. Henn and W. Oettli, editors, *Optimization and Operations Research*, pages 45-54. Springer-Verlag, New York/Berlin, 1977.
- [Burr, 1988] D. J. Burr. An improved Elastic Net method for the Travelling Salesman Problem. In *IEEE Int. Joint Conference on Neural Networks*, volume 1, pages 69-76, 1988.
- [Chazelle and Edelsbrunner, 1988] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. In *Proc. IEEE Symposium on Foundations of Computer Science*, number 29, pages 590-600, 1988.

- [Chazelle *et al.*, 1984] B. M. Chazelle, R. L. Drysdale III, and D. T. Lee. Computing the largest empty rectangle. In *Proc. Symp. Theoretic Aspects Comput. Sci.*, Paris, France, April 1984.
- [Chazelle, 1984] B. M. Chazelle. Intersecting is easier than sorting. In *Proc. 16th ACM Annu. Symp. Theory Comput.*, pages 125-131, 1984.
- [Chetayav, 1961] N. Chetayav. *The Stability of Motion*. Pergamon Press, New York, 1961.
- [Chow, 1980] A. L. Chow. *Parallel Algorithms for Geometric Problems*. PhD thesis, University of Illinois at Urbana-Champaign, 1980.
- [Chow, 1981] A. L. Chow. A parallel algorithm for determining convex hulls of sets of points in two dimensions. In *Proc. of 19th Annual Allerton Conference on Comm., Control and Computing*, pages 214-223, 1981.
- [Clement *et al.*, 1988] W. I. Clement, R. M. Inigo, and E. S. McVey. Synaptic strengths for neural simulations of the Travelling Salesman Problem. In *Proceedings of Applications of Artificial Intelligence SPEI927*, pages 373-380, 1988.
- [Cohen and Grossberg, 1983] M. Cohen and S. Grossberg. Absolute stability of global pattern formation and parallel memory storage by competitive neural networks. *IEEE Transactions on System, Man and Cybernetics*, 13:815-826, 1983.
- [Cole and Zajicek, 1990] R. Cole and O. Zajicek. An optimal parallel algorithm for building a data structure for planar point location. *Journal of Parallel and Distributed Computing*, 8:280-285, 1990.
- [Cröes, 1958] G. A. Cröes. A method for solving Travelling-Salesman Problems. *Operations Research*, 6:791-812, 1958.
- [Dahl, 1987] E. Dahl. Neural network algorithm for an NP-complete problem: Map and graph coloring. In *IEEE First Int'l Conference on Neural Networks*, volume 3, pages 113-120, 1987.

- [Davis and Ansari, 1987] G. W. Davis and A. Ansari. Sensitivity analysis of Hopfield neural net. In *IEEE 1st Int.'l Conference on Neural Networks*, volume 3, pages 721-734, 1987.
- [de Carvalho and Barbosa, 1989] L. A. V. de Carvalho and V. C. Barbosa. Towards a stochastic neural model for combinatorial optimization. In *IEEE INNS Int.'l Joint Conference on Neural Networks*, volume 2, page 587, 1989.
- [de Carvalho and Barbosa, 1990] L. A. V. de Carvalho and V. C. Barbosa. A TSP objective functions that ensures feasibility at stable points. In *Int.'l Conference on Neural Networks, Paris*, volume 2, pages 249-253, 1990.
- [Dehne and Rau-Chaplin, 1990] F. Dehne and A. Rau-Chaplin. Implementing data structures on a hypercube multiprocessor and applications in parallel computational geometry. *Journal of Parallel and Distributed Computing*, 8:367-375, 1990.
- [Dehne and Sack, 1988] F. Dehne and J. -R. Sack. A survey of parallel computational geometry algorithms. In *Proc. Int.'l Workshop on Parallel Processing by Cellular Automata and Arrays (PARCELLA)*, volume 342 of *Lecture Notes in Computer Science*, pages 73-88, Berlin, Germany, 1988. Springer Verlag. Invited Paper.
- [Dehne et al., 1986] F. Dehne, J. -R. Sack, and N. Santoro. Computing on a systolic screen: hulls, contours and applications. Technical Report SCS-TR-102, Carleton University, Ottawa, Canada, 1986.
- [Dehne, 1988] F. Dehne. Solving visibility and separability problems on a mesh-of-processors. *The Visual Computer*, 3:356-370, 1988.
- [Dickerson et al., 1992] M. T. Dickerson, R. L. S. Drysdale, and J. -R. Sack. Simple algorithms for enumerating interpoint distances and finding the k nearest neighbours. *Int.'l Journal on Computational Geometry and Applications*, August 1992. in press.

- [Dobkin and Lipton, 1976] D. P. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM J. Computing*, 5(2):181-186, June 1976.
- [Du and Hwang, 1993] Ding-Zhu Du and F. K. Hwang. *Computing in Euclidean Geometry*. World Scientific Publishing Co., 1993.
- [Duda and Hart, 1973] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Wiley & Sons, 1973.
- [Duppe and Gottschalk, 1970] R. Duppe and H. Gottschalk. Automatische Interpolation von Isolinien bei willkürlichen Stützpunkten. *Allgemeine Vermessungsnachrichten*, 77:423-426, 1970.
- [Durbin and Willshaw, 1987] R. Durbin and D. Willshaw. An analogue approach to the Travelling Salesman Problem using an Elastic Net Method. *Nature*, 326:689-691, 1987.
- [Durbin *et al.*, 1989] R. Durbin, R. Szeliski, and A. L. Yuille. An analysis of the Elastic Net approach to the Travelling Salesman Problem. *Neural Computation*, 1:348-358, 1989.
- [Edelsbrunner, 1987] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag, Berlin, 1987.
- [ElGindy and Goodrich, 1988] H. ElGindy and M. T. Goodrich. Parallel algorithms for shortest path problems in polygons. *The Visual Computer*, 3:371-378, 1988.
- [Essen and Maunsell, 1983] D. Van Essen and J. Maunsell. Hierarchical organization and functional streams in the visual cortex. *Trends in Neuro Science*, 6:370-375, 1983.
- [Evans and Stojmenovic, 1989] D. J. Evans and I. Stojmenovic. On parallel computation of Voronoi diagrams. *Parallel Computing*, 12:121-125, 1989.
- [Feller, 1959] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley, 2 edition, 1959.

- [Fisk, 1978] S. Fisk. A short proof of Chvatal's watchman theorem. *Journal of Combinatorial Theory, Ser. B*, 24(3):374, June 1978.
- [Foo and Szu, 1989] Y. P. Simon Foo and H. Szu. Solving large-scale optimization problems by divide and conquer neural networks. In *IEEE INNS Int.'l Joint Conference on Neural Networks*, volume 1, pages 507-509, 1989.
- [Fritzke and Wilkie, 1991] B. Fritzke and P. Wilkie. FLEXMAP - a neural network for the Travelling Salesman Problem with linear time and space complexity. In *Int.'l Joint Conference on Neural Networks*, pages 929-934, 1991.
- [Garey and Johnson, 1979] M. Garey and D. Johnson. *Computers and Intractability*. W.H. Freeman, New York, 1979.
- [Garey *et al.*, 1976] M. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete problems. In *Proc. 8th ACM Annual Symp. Theory Comput.*, pages 10-22, May 1976.
- [Gear, 1971] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, 1971.
- [Geman and Geman, 1984] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, PAMI-6(6):721-741, 1984.
- [Goodrich and Yap, 1991] M. T. Goodrich and C. K. Yap. What can be parallelized in computational geometry : A survey. 1991. Draft.
- [Goodrich *et al.*, 1989] M. T. Goodrich, C. O'Dunlaing, and C. K. Yap. Constructing the Voronoi diagram of a set of line segments in parallel. In F. Dehne, J. -R. Sack, and N. Santoro, editors, *Proc. of the 1989 Workshop on Algorithms and Data Structures, Wads'89*, number 382 in Lecture Notes in Computer Science, pages 12-23. Berlin, Germany, 1989. Springer Verlag.

- [Graham, 1972] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1:132-133, 1972.
- [Grossberg, 1982] S. Grossberg. *Studies of Mind and Brain: Neural Principles of Learning, Perception, Cognition and Motor Control*. Reidel Press, Boston, Mass., 1982.
- [Grossberg, 1986] S. Grossberg. Conditioning, expectancy, attention, and rhythm: The role of gated dipoles. In R. Chapman and R. Zayan, editors, *Relevance of Models and Theories in Ethology*, pages 109-123. Private I.E.C, Boston, Mass., 1986.
- [Guibas *et al.*, 1987] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. Tarjan. Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209-233, 1987.
- [Gulati and Iyengar, 1987] S. Gulati and S. Iyengar. Nonlinear neural networks for deterministic scheduling. In *IEEE First Int'l Conference on Neural Networks*, volume 4, pages 745-752, 1987.
- [Gunn and Weidlich, 1989] J.P. Gunn and R.B. Weidlich. A derivative of the Hopfield-Tank neural network model that reliably solves the Travelling Salesman Problem. In *IEEE INNS Int'l Joint Conference on Neural Networks*, volume 2, page 588, 1989.
- [Gutzmann, 1987] K. M. Gutzmann. Combinatorial optimization using a continuous state Boltzmann machine. In *IEEE 1st Int'l Conference on Neural Networks*, volume 3, pages 721-734, 1987.
- [Hebb, 1949] D.O. Hebb. *Organization of Behavior*. John Wiley & Sons, New York, 1949.
- [Heffernan and Schirra, 1992] P. J. Heffernan and S. Schirra. Approximate decision algorithms for point set congruence. In *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 93-101. ACM Press, 1992.

- [Hegde *et al.*, 1988] S. U. Hegde, J. L. Sweet, and W. B. Levy. Determination of parameters in a Hopfield/Tank computational network. In *IEEE 2nd. Int.'l Conference on Neural Networks*, volume 2, pages 291–298, 1988.
- [Held and Karp, 1962] M. Held and R. M. Karp. A dynamic programming approach to sequencing problems. *SIAM J. Appl. Math.*, 10:196–210, 1962.
- [Highnam, 1986] P. T. Highnam. Optimal algorithms for finding the symmetries of a planar point set. *Information Processing Letters*, 22:219–222, 1986.
- [Holey and Ibarra, 1990] J. A. Holey and O. H. Ibarra. Iterative algorithms for planar convex hull on mesh-connected arrays. In *The 1990 Int.'l Conference on Parallel Processing*, pages 102–109, 1990.
- [Hopfield and Tank, 1985] J. Hopfield and D. Tank. “Neural” computation of decisions in optimization problems. *Biological Cybernetics*, 52:141–152, 1985.
- [Hopfield and Tank, 1986a] J. Hopfield and D. Tank. Computing with neural circuits: A model. *Science*, 233:625–633, 1986.
- [Hopfield and Tank, 1986b] J. J. Hopfield and D. W. Tank. Collective computation with continuous variables. In E. Bienenstock, F. Fogelman Soulie, and G. Weisbuch, editors, *Disordered Systems and Biological Organization*, pages 155–170. Springer-Verlag, 1986.
- [Hopfield *et al.*, 1983] J. Hopfield, D. Feinstein, and R. Palmer. “Unlearning” has a stabilizing effect in collective memories. *Nature*, 304:158–159, 1983.
- [Hopfield, 1982] J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79:2554–2558, 1982.
- [Hopfield, 1984] J. Hopfield. Neurons with graded response have collective computational properties like those of two-state neurons. *Proceedings of the National Academy of Sciences*, 81:3088–3092, 1984.

- [Hopfield, 1986] J. Hopfield. Collective computation, content-addressable memory, and optimization problems. In Y. Abu-Mostafa, editor, *Complexity in Information Theory*. Springer-Verlag, New York, 1986.
- [Jain and Dubes, 1988] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.
- [Jarvis, 1973] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18-21, 1973.
- [Jeffrey and Rosner, 1986] W. Jeffrey and R. Rosner. Neural network processing as a tool for function optimization. In J. Denker, editor, *AIP Conference Proceedings 151: Neural Networks for Computing*, pages 241-246. American Institute of Physics, New York, 1986.
- [Jeong and Kim, 1990] C. S. Jeong and M. H. Kim. Fast parallel simulated annealing for Travelling Salesman Problem. In *IEEE INNS Int'l Joint Conference on Neural Networks*, volume 3, pages 947-953, 1990.
- [Jeong and Lee, 1990] C. S. Jeong and D. T. Lee. Parallel geometric algorithms on a mesh-connected computer. *Algorithmica*, 5(2):155-178, 1990.
- [Johnson, 1982] D. S. Johnson. The NP-completeness column: An ongoing guide. *Journal of Algorithms*, 3:182-195, 1982.
- [Joppe et al., 1990] A. Joppe, H. R. A. Cardon, and J. C. Bioch. A neural network for solving the Travelling Salesman Problem on the basis of city adjacency in the tour. In *IEEE INNS Int'l Joint Conference on Neural Networks*, volume 3, pages 961-964, 1990.
- [Kahng, 1989] A. B. Kahng. Travelling Salesman heuristics and embedding dimension in the Hopfield model. In *IEEE INNS Int'l Joint Conference on Neural Networks*, volume 1, pages 513-520, 1989.

- [Kamgar-Parsi and Kamgar-Parsi, 1987] B. Kamgar-Parsi and B. Kamgar-Parsi. An efficient model of neural networks for optimization. In *IEEE First Int. '1 Conference on Neural Networks*, volume 3, pages 785–790, 1987.
- [Kamgar-Parsi and Kamgar-Parsi, 1989] B. Kamgar-Parsi and B. Kamgar-Parsi. On problem solving with Hopfield neural networks. Technical Report CAR-TR-462, Center for Automation Research, University of Maryland, 1989.
- [Kariv and Hakimi, 1979a] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems: Part I : the centers. *SIAM J. Appl. Math.*, 37:513–538, 1979.
- [Kariv and Hakimi, 1979b] O. Kariv and S. L. Hakimi. An algorithmic approach to network location problems: Part II : p-medians. *SIAM J. Appl. Math.*, 37:539–560, 1979.
- [Kirkpatrick and Seidel, 1986] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM J. of Computing*, 15(1):287–299, 1986.
- [Kirkpatrick *et al.*, 1983] C. D. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.
- [Kirkpatrick, 1983] D. G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comput.*, 12(1):28–35, Feb. 1983.
- [Klincsek, 1980] G. T. Klincsek. Minimal triangulations of polygonal domain. *Annals of Discrete Mathematics* 9:121–123, 1980.
- [Koch, 1987] C. Koch. Analog neuronal networks for real-time vision systems. In *Workshop on Neural Network Devices and Applications*, pages 337–359, Jet Propulsion Laboratory, 1987.
- [Lawler *et al.*, 1985] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Travelling Salesman Problem*. John Wiley & Sons. New York, 1985.

- [Lee and Preparata, 1984] D. T. Lee and F. P. Preparata. Computational geometry—a survey. *IEEE Tran. on Computers*, C-33(12):1072–1101, Dec. 1984.
- [Lee and Sheu, 1988] B. W. Lee and B. J. Sheu. An investigation on local minima of Hopfield network for optimization circuits. In *IEEE Int. l Conference on Neural Networks*, volume 1, pages 45–51, 1988.
- [Lee and Sheu, 1990] B. W. Lee and B. J. Sheu. Combinatorial optimization using Competitive-Hopfield neural network. In *Int. l Joint Conference on Neural Networks-WASH DC*, volume 2, pages 627–630, 1990.
- [Lee and Wong, 1977] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9:23–29, 1977.
- [Levcopoulos *et al.*, 1989] C. Levcopoulos, A. Lingas, and J. -R. Sack. Nearly optimal heuristics for binary search trees with geometric generalization. *Theoretical Computer Science*, 66(2):181–203, 1989.
- [Lin and Kernighan, 1973] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the Travelling Salesman Problem. *Operations Research*, 21:498–516, 1973.
- [Lingas, 1986] A. Lingas. The Greedy and Delaunay triangulations are not bad in the average case. *Information Processing Letters*, 22:25–31, 1986.
- [Lingas, 1987] A. Lingas. The greedy triangulation heuristic for minimum weight triangulation of convex polygons approximates the optimum. Technical Report LITH-IDA-R86-11, Linköping University, Sweden, 1987. Research Report.
- [Lister, 1990] R. Lister. Segment reversal and the Travelling Salesman Problem. In *Int. l Joint Conference on Neural Networks-WASH DC*, volume 1, pages 424–427, 1990.

- [Lloyd, 1977] E. L. Lloyd. On triangulations of a set of points in the plane. In *Proceedings of the 18th IEEE Conference on Foundations of Computer Science*, pages 228–240, 1977.
- [Manacher and Zobrist, 1979] G. K. Manacher and A. L. Zobrist. Neither the Greedy nor the Delauney triangulation of a planar point set approximate the optimal triangulation. *Information Processing Letters*, 9:31–34, 1979.
- [Matsuyama, 1990] Y. Matsuyama. Competitive self-organization and combinatorial optimization: Applications to Travelling Salesman Problem. In *IEEE INNS Int. Joint Conference on Neural Networks*, volume 3, pages 819–824, 1990.
- [McCarrager and Chase, 1985] G. McCarrager and R. Chase. Quantification of untrastructural symmetry at molluscan chemical synapses. *J. Neurobiological*, 16:69–74, 1985.
- [McCulloch and Pitts, 1943] W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 7:115–133, 1943.
- [McCulloch, 1965] W. McCulloch. *Embodiments of Mind*. MIT Press, Cambridge, Mass., 1965.
- [Mead, 1989] C. Mead. *Analog VLSI and Neural Systems*. Addison-Wesley, Reading, Mass., 1989.
- [Megiddo, 1983] N. Megiddo. Linear-time algorithm for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12:759–776, 1983.
- [Megiddo and Supowit, 1984] N. Megiddo and K.J. Supowit. On the complexity of some common geometric location problems. *SIAM J. of Computing*, 13:182–196, 1984.
- [Megiddo and Tamir, 1982] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations Res. Lett.*, 1:194–197, 1982.

- [Mehlhorn, 1984] K. Mehlhorn. *Data Structures and Algorithms, Vol. 1-3*. Springer-Verlag, Berlin, 1984.
- [Miller and Stout, 1989] R. Miller and Q. F. Stout. Mesh computer algorithms for computational geometry. *IEEE Trans. on Computers*, 38(3):321-340, 1989.
- [Millman and Halkias, 1975] J. Millman and C. Halkias. *Integrated Electronics*. McGraw Hill, New York, 1975.
- [Minsky and Papert, 1969] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, 1969.
- [Neotzel and Graziano, 1988] A. S. Neotzel and M. J. Graziano. Global combinatorial optimization by neural networks. In *INNS Abstracts*, volume 1, page 119, 1988.
- [O'Rourke, 1987] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, 1987.
- [Pandit and Srinivas, 1991] S. N. N. Pandit and K. Srinivas. A lexisearch algorithm for Travelling Salesman Problem. In *Int.'l Joint Conference on Neural Networks*, pages 2521-2527, 1991.
- [Pao, 1989] Yoh-Han Pao. *Adaptive Pattern-Recognition and Neural Networks*. Addison-Wesley, Reading, Mass., 1989.
- [Papadimitriou, 1977] C. H. Papadimitriou. The Euclidean Travelling Salesman Problem is NP-complete. *Theoretical Comput. Science*, 4:237-244, 1977.
- [Papadimitriou, 1981] C. H. Papadimitriou. Worst-case and probabilistic analysis of a geometric location problem. *SIAM J. Comput.*, 10:542-557, 1981.
- [Park and Jeong, 1990] J. H. Park and H. Jeong. Solving the Travelling Salesman Problem using an effective Hopfield network. In *Int.'l Conference on Neural Networks, Paris*, volume 2, page 291, 1990.

- [Perkel and Bulloch, 1969] D. H. Perkel and T. H. Bulloch. Neural coding. In *Neuroscience Research Symposium Summ.*, volume 3, pages 405–527, 1969.
- [Preparata and Shamos, 1985] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.
- [Providence, 1987] J. Providence. Neural network implementation for maximum-likelihood sequence estimation of binary signals in gaussian noise. In *IEEE First Int'l Conference on Neural Networks*, volume 3, pages 703–714, 1987.
- [Ramanujam and Sadayappan, 1988] J. Ramanujam and P. Sadayappan. Optimization by neural networks. In *IEEE First Int'l Conference on Neural Networks*, volume 2, pages 325–332, 1988.
- [Reif and Sen, 1990] J. H. Reif and S. Sen. Randomized algorithms for binary search and load balancing on fixed connection networks with geometric applications. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, 1990.
- [Roger, 1964] C. A. Roger. *Packing and Covering*. Cambridge University Press, Cambridge, England, 1964.
- [Rong and Ze-min, 1991] L. Rong and L. Ze-min. Determination of the parameters in a modified Hopfield-Tank model on solving TSP. In *Int'l Joint Conference on Neural Networks*, pages 2455–2560, 1991.
- [Rose et al., 1991] K. Rose, E. Guremitz, and G. C. Fox. Deterministic annealing, constrained clustering and optimization. In *Int'l Joint Conference on Neural Networks*, pages 2515–2520, 1991.
- [Rosenblatt, 1957] F. Rosenblatt. The perceptron: A perceiving and recognizing automation. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.
- [Rosenblatt, 1958a] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

- [Rosenblatt, 1958b] F. Rosenblatt. The perceptron: A theory of statistical separability in cognitive systems. Technical Report VG-1196-G-1, Cornell Aeronautical Laboratory. 1958.
- [Rosenfeld, 1969] A. Rosenfeld. *Picture Processing by Computers*. Academic Press, New York, 1969.
- [Rumelhart *et al.*, 1986] R. Rumelhart, G. Hinton, and R. Williams. Learning representations by backpropagating errors. *Nature*, 323:533-536, 1986.
- [Sage *et al.*, 1986] J. Sage, K. Thompson, and R. Withers. An artificial neural network integrated circuit based on NMOS/CCD principles. In J. Denker, editor, *AIP Conference Proceedings 151: Neural Networks for Computing*, pages 381-385. American Institute of Physics, New York, 1986.
- [Sage, 1987] J. Sage. Artificial neural system implementations using CCD and NMOS technologies. In *Workshop on Artificial Neural Systems*, pages 72-84, Jet Propulsion Laboratory, 1987.
- [Shamos and Hoey, 1973] M. Shamos and D. Hoey. Closest point problems. In *Proceedings of the 16th IEEE Conference on Foundations of Computer Science*, pages 151-162, 1973.
- [Shamos and Hoey, 1976] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. 17th IEEE Annual Symposium on Foundations of Computer Science*, pages 208-215, Oct. 1976.
- [Shamos, 1975] M. I. Shamos. Geometric complexity. In *Proc. 7th ACM Annu. Symp. Theory Comput.*, pages 224-233, May 1975.
- [Shamos, 1976] M. I. Shamos. Geometry and Statistics: Problems at the interface. In J. F. Traub, editor, *Algorithms and Complexity: New Directions and Recent Results*, pages 251-280. Academic Press, Inc., New York, 1976.

- [Shinozawa *et al.*, 1991] K. Shinozawa, T. Uchiyama, and K. Shomohara. An approach for solving dynamic TSPs using neural networks. In *Int. 'l Joint Conference on Neural Networks*, pages 2450–2454, 1991.
- [Shirazi and Yih, 1989] B. Shirazi and S. Yih. Critical analysis of applying Hopfield neural net model to optimization problems. In *IEEE Int. 'l Conference on Systems, Man and Cybernetics*, volume 1, pages 210–215, 1989.
- [Simmes, 1987] S. Simmes. Programming parallel computers using energy minimization algorithms. In *IEEE First Int. 'l Conference on Neural Networks*, volume 2, pages 205–212, 1987.
- [Simpson, 1989] P.K. Simpson. *Artificial Neural Systems*. Pergamon Press, New York, 1989.
- [Sivilotti *et al.*, 1986] M. Sivilotti, M. Emerling, and C. Mead. VLSI architectures for implementation of neural networks. In J. Denker, editor, *AIP Conference Proceedings 151: Neural Networks for Computing*, pages 408–413. American Institute of Physics, New York, 1986.
- [Stojmenovic, 1988] I. Stojmenovic. Computational geometry on a hypercube. In *Proc. of the 1988 Int 'l Conference on Parallel Processing*, volume III of *Algorithms and Applications*, pages 100–103, August 1988.
- [Supowit, 1981] K. J. Supowit. Topics in computational geometry. Technical Report University of Illinois at Urbana-Champaign, UIUCDCS-R-81-1062, Ph.D. Dissertation, Dept. of Computer Science, 1981.
- [Syslo *et al.*, 1983] M. M. Syslo, N. Deo, and J. S. Kowalik. *Discrete Optimization Algorithms with Pascal Programs*. Prentice-Hall, Englewood Cliffs, N.J., 1983.
- [Szu, 1988] H. Szu. Fast TSP algorithm based on binary neuron output and analog neuron input using the zero-diagonal interconnect matrix and necessary and sufficient constraints of the permutation matrix. In *IEEE 2nd. Int. 'l Conference on Neural Networks*, volume 2, pages 259–265, 1988.

- [Tagliarini and Page, 1987] G. A. Tagliarini and E. W. Page. Solving constraint satisfaction problems with neural networks. In *IEEE 1st Int'l Conference on Neural Networks*, volume 3, pages 741-747, 1987.
- [Tagliarini and Page, 1988] G. Tagliarini and E. Page. A neural-network solution to the concentrator assignment problem. In D. Anderson, editor, *Proceedings of 1987 IEEE Conference on Neural Information Processing Systems - Natural and Synthetic*, pages 775-782. American Institute of Physics, New York, 1988.
- [Takeda and Goodman, 1986] M. Takeda and J. Goodman. Neural networks for computation: Number representations and programming complexity. *Applied Optics*, 25:3033-3046, 1986.
- [Tank and Hopfield, 1986] D. Tank and J. Hopfield. Simple "Neural" optimization networks: An A/D converter, signal decision circuit, and a linear programming circuit. *IEEE Transactions on Circuits and Systems*, 33:533-541, 1986.
- [Tank and Hopfield, 1987] D. Tank and J. Hopfield. Collective computation in neuronlike circuits. *Scientific American*, 257:104-114, 1987.
- [Thomae and den Bout, 1990] D. A. Thomae and D. E. Van den Bout. Encoding logical constraints into neural network cost function. In *IEEE INNS Int'l Joint Conference on Neural Networks*, volume 3, pages 863-868, 1990.
- [Toomarian, 1988] N. Toomarian. A concurrent neural network algorithm for the Travelling Salesman Problem. In *III Conf. on Hypercube Concurrent Computers and Applications, Pasadena*, pages 1483-1490, 1988.
- [Van den Bout and Miller, 1988] D. Van den Bout and T. Miller. A Travelling Salesman objective function that works. In *IEEE Int'l Conference on Neural Networks*, volume 2, pages 299-303, 1988.
- [Wacholder *et al.*, 1988] E. Wacholder, J. Han, and R. C. Mann. An extension of the Hopfield-Tank model for solution of the multiple Travelling Salesman Problem.

- In *IEEE 2nd. Int'l Conference on Neural Networks*, volume 2, pages 305–324, 1988.
- [Wang and Chen, 1990] B. F. Wang and G. H. Chen. Constant time algorithms for sorting and computing convex hulls. Technical report. Dept. of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan., 1990.
- [Wasserman, 1989] P. D. Wasserman. *Neural Computing: Theory and Practice*. Van Nostrand Reinhold, 1989.
- [Widrow and Hoff, 1960] B. Widrow and M. Hoff. Adaptive switching circuits. In *1960 WESCON Convention Record: Part 4*, pages 96–104, 1960.
- [Widrow, 1959] B. Widrow. Adaptive sampled-data systems – a statistical theory of adaptation. In *1959 WESCON Convention Record: Part 4*, pages 74–85, 1959.
- [Widrow, 1960] B. Widrow. An adaptive “adaline” neuron using chemical “memistors”. Technical Report 1553-2, Stanford Electronics Laboratory, 1960.
- [Wilson and Pawley, 1988] G. V. Wilson and G. S. Pawley. On the stability of the Travelling Salesman Problem algorithm of Hopfield and Tank. *Biological Cybernetics*, 58:63–70, 1988.
- [Winters, 1988] J. Winters. Super-resolution in air using neural networks. In *IEEE First Int'l Conference on Neural Networks*, volume 1, pages 609–616, 1988.
- [Xu and Tsai, 1990] X. Xu and W. T. Tsai. An adaptive neural algorithm for Travelling Salesman Problem. In *Int'l Joint Conference on Neural Networks-WASH DC*, volume 2, pages 716–719, 1990.
- [Yue and Fu, 1990] Tsai-Wen Yue and Li-Chen Fu. Ineffectiveness in solving combinatorial optimization problems using a Hopfield network. In *IEEE INNS Int'l Joint Conference on Neural Networks*, volume 3, pages 787–792, 1990.

- [Zhao and Mendel, 1987] Y. Zhao and J. Mendel. An artificial neural minimum-variance estimator. In *IEEE First Int. l Conference on Neural Networks*, volume 2, pages 499–506, 1987.

END

14-12-93

FIN