NEW NEUROCOMPUTATIONAL APPROACHES FOR ESTIMATING ROAD TRAVEL DISTANCES AND FOR SOLVING THE EUCLIDEAN TRAVELING SALESMAN PROBLEM

by

Mustafa Necati Aras

BS. in M.E., Boğaziçi University, 1993

BS. in I.E., Boğaziçi University, 1993

MS. in I.E., Boğaziçi University, 1994

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor
of
Philosophy

Boğaziçi University

1999

ACKNOWLEDGMENTS

My Ph.D. study began five years ago in 1994. Since then, many drastic and dramatic changes occurred in my life. Without the help and moral support of three people this study would not have been finished.

First, I would like to thank my thesis advisor, Kuban Altınel not only for his guidance with the thesis but also for his continuous understanding and friendship during the times I feel exhausted. A demanding study such as Ph.D. could not be more fun. I consider myself very lucky to work with a partner like him.

I would like to express my deepest gratitude to my mother who supports and gives without expectation.

Mehmet Purazer, the best friend of mine, who was, and I am sure that will always be with me when I need help, deserves a special "thank you". He shared the most of my good and bad times.

I wish also to thank to the members of my thesis committee, Associate Professor Ethem Alpaydın, Associate Professor Ümit Bilge, Associate Professor Selçuk Karabatı, and Professor Süleyman Özekici for their valuable suggestions and comments.

Finally, I appreciate the contributions of John Oommen during all the phases of this study. Our brain storming sessions and discussions were so efficient and beneficial for me.

I gratefully acknowledge that the research of this thesis has been partly supported by Boğaziçi Research Fund Grants 97H301D and 98A301D and by an equipment donation by Hewlett-Packard.

ABSTRACT

Neural networks are among the most rapidly developing new scientific tools. There are numerous publications reporting their success in estimation and optimization. This work concentrates on both of these aspects and applies neural networks for solving two problems from operations research.

One of the problems is the distance estimation problem, which mainly deals with the estimation of the length of the shortest road connecting two points on the earth surface. First, multilayer perceptrons have been adopted. Then, a neural clustering strategy which uses the principle of vector quantization has been utilized prior to the estimation. The results are superior than those reported in the literature.

The other problem is the well-known Euclidean traveling salesman problem. It tries to determine the shortest tour passing through the cities of a given set by visiting each city exactly once. A new adaptive scheme has been developed in order to solve this problem. The new approach incorporates explicit statistical information obtained from the city coordinates into the adaptation mechanism of Kohonen's self-organizing map. Results obtained for different problems are better than the previous ones.

The new approach is then adapted to the solution of the Euclidean Hamiltonian path problem whose combination with the decompositon philosophy resulted in a highly efficient all-neural Euclidean traveling salesman problem algorithm.

ÖZET

Sinir ağları en hızlı gelişen yeni bilimsel yaklaşımlar arasındadır. Sinir ağlarının kestirim ve optimizasyon alanlarında gösterdiği başarıyı aktaran birçok yayın bulunmaktadır. Bu çalışma her iki alana da odaklanmakta ve yöneylem araştırmasında yer alan iki problemin çözümünde sinir ağlarının kullanımını içermektedir.

Problemlerden bir tanesi yeryüzünde bulunan iki noktayı birbirine bağlayan en kısa yolun uzunluğunu kestirmeyi amaçlayan uzaklık kestirim problemidir. Problemin çözümü için ilk olarak çok katmanlı algaçlar kullanılmıştır. Daha sonra, kestirim öncesi vektör niceleme prensibini kullanan bir sinir ağı kümelendirme stratejisinden yararlanılmıştır. Sonuçlar yazında yer alanlardan çok daha iyidir.

Diğer problem iyi bilinen Öklidyen gezgin satıcı problemidir. Bu problem bir kümedeki şehirlerin tümünden geçen ve her bir şehire yalnızca bir kere uğrayan en kısa turu bulmaya çalışır. Problemi çözmek için yeni bir adaptif yaklaşım geliştirilmiştir. Bu yeni yaklaşım, şehir koordinatlarından elde edilen istatistiksel bilginin Kohonen'in kendini düzenleyen haritasında kullanılan adaptasyon mekanizmasında faydalanılmasını sağlamıştır. Farklı problemler için elde edilen sonuçlar önceki sonuçlardan daha iyidir.

Yeni yaklaşım daha sonra Öklidyen Hamiltoniyen yol probleminin çözümüne adapte edilmiş ve bunun, asıl problemin parçalanıp çözülmesi felsefesiyle biraraya getirilmesi, tamamıyla sinir ağı tabanlı oldukça etkin bir Öklidyen gezgin satıcı problemi algoritmasının elde edilmesine yol açmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

This work focuses on two different problems: the distance estimation problem and the Euclidean traveling salesman problem. The distance estimation problem tries to answer the following question: Given two points on the earth surface, what is the length of the shortest road connecting these points? When a look-up table is not ready at hand, the actual distance between any pair of points has to be estimated. Distribution problems, transportation problems, the traveling salesman problem, and the vehicle routing problem are the main areas where accurate distance estimations become necessary as their formulations include the actual distance information. The necessity of distance estimation rises also from facility location problems which aim at the determination of the best position of a point with respect to a criterion. This criterion usually involves the minimization of a distance-based objective function where distance information is provided by means of a distance estimator. The traveling salesman problem is concerned with finding the shortest possible tour through a set of points (cities or nodes) so that each point is visited exactly once. The traveling salesman problem that belongs to the class of combinatorial optimization problems is *NP*-hard [1], i.e., any algorithm devised to find an optimal solution (tour) for an instance of this problem must have a worst-case running time that grows faster than any polynomial (under the assumption which is still believed to be correct that $P \neq NP$). This, in turn, increases the value of heuristics that find near-optimal tours in a reasonable amount of time.

What brings these two unrelated problems together in this thesis is that artificial neural networks may be employed to solve both of them. Despite the diversity of the existing neural network models, there are many common features in their architecture. A neural network is composed of a number of layers which consist of neurons (also called processing units). There are at least two layers in all types of neural networks which are the input layer and output layer. Depending on the network's architecture there may be hidden layers of neurons. There exist connections from the neurons in the input layer to the neurons in the hidden layer or output layer. If there are hidden layers, then there exist also connections from the neurons in the hidden layers to the neurons in the output layer. Each neuron in a hidden layer or output layer computes a weighted sum of its inputs and transforms this

value according to a real-valued continuous function usually called an activation function. Each connection is associated with a weight and learning in neural networks occurs through adjusting the weights according to a learning scheme.

One type of artificial neural networks is called multilayer perceptron which is a feedforward neural network trained with backpropagation learning paradigm. In Chapter 3 the distance estimation problem is solved by multilayer perceptrons and the solution is compared with that obtained by solving the unconstrained optimization problem formulated for the parameters of the suitable distance functions used for estimating the distances.

Competitive neural networks are neural networks that consist of only input and output neurons (no hidden layers). The aim of such networks is to cluster or categorize the input data. For a given input pattern (vector) the output neurons compete for being the one fire, i.e., to win the competition. As a result, all the input vectors are assigned to classes or clusters, which are not known in advance, but the network discovers by itself the correlations of the input data. In this case we talk about unsupervised learning meaning that the input data vectors presented to the network are unlabeled. At the end of the process, each cluster is associated with a distinct output neuron. In other words, each output neuron gets activated only for a subset of input vectors. Competitive neural networks trained with unsupervised learning can be used for data encoding or compression through vector quantization.

Learning vector quantization is closely related to the vector quantization with the unique difference that the training of the network occurs via supervised learning. Namely, the input data vectors are labeled a priori, and when they are presented to the network, the output of the network is compared with the desired response. Weights of the connections are updated so that the output of the neural network gives the desired response as close as possible. This paradigm is used in Chapter 4 to cluster the input data of the distance estimation problem at hand, that is the cities and towns of Türkiye that make up the training data. The clusters are then utilized to estimate the actual distance between any two locations in Türkiye. Such a preprocessing of input data has been beneficial for the quality of the estimates.

Chapter 5 again solves the distance estimation problem with preclustering the cities, but this time accomplishes this using discrete learning vector quantization. In discrete learning vector quantization the neurons only assume discretized values and the updates of the neurons are made in a discretized space. In other words, there is a two-dimensional grid to move the neurons over.

Kohonen's self-organizing map (SOM) is also a competitive neural network trained with unsupervised learning. This model aims at generating mappings from higher to lower dimensional spaces, so that the relationships among the inputs are reflected as faithfully as possible in the outputs (topological mapping). As a consequence, topological relationships are established among the output neurons and those relationships are respected when the weights are updated. This is achieved by updating not only the winning neuron but also the neighbors of this neuron. The neighboring neurons are updated in the same direction as the winning neuron, but with decreasing intensity. The self-organizing map model has become an attractive alternative for solving various problems which traditionally have been the domain of conventional statistical and operations research techniques. These problems include, for example, optimization problems such as the traveling salesman problem [2], vehicle routing problem [3], scheduling problems [4, 5], function approximation [6], and part family/machine grouping in group technology [7]. Chapter 7 explains in detail the self-organizing map approach of Kohonen which is the most widely investigated and reported neural network paradigm in the literature and its application to the Euclidean traveling salesman problem (TSP). In the TSP context, a set of two-dimensional coordinates, locating the cities in the Euclidean space, must be mapped into a set of neurons located on a ring, so as to preserve the neighborhood relationships among the cities.

In Chapter 8, a new model, called Kohonen Network Incorporating Explicit Statistic (KNIES), is introduced. KNIES includes all the advantages of the self-organizing maps and additionally uses the statistical information existing in the input data (the cities of the TSP) in the evolution of the map by incorporating this information in the update equations of the neurons. We observe that the solutions obtained with KNIES are better than those provided by other techniques based on the SOM.

The new approach may also be extended to solve the Euclidean Hamiltonian path problem which tries to determine the shortest path that begins at a starting point, ends at a terminal point, and visits each point exactly once. This approach is the only reported neural network solution to the Hamiltonian path problem and discussed fully in Chapter 9, where the quality of the solutions are determined by statistical techniques.

The objective of the last chapter, which is Chapter 9, is to solve large traveling salesman problems by decomposing them into subproblems. This is done by first dividing the cities of the original problem into clusters by vector quantization. The cluster centers are then used to determine the order of the global tour through the clusters. In other words, the global tour gives the order in which the clusters are to be visited. This, at the same time provides the entering and leaving cities in each cluster. The remaining task is to find out the Hamiltonian paths between the entering and leaving cities for the clusters. For that purpose we employ the approach presented in Chapter 8.

2. DISTANCE ESTIMATION

2.1. Problem Definition

The *actual distance* between any two points on the earth surface can be defined as the length of the shortest road connecting them. The length of the road between two points on the earth surface is usually longer than the Euclidean distance (straight-line or crow-flight distance) because of geographical barriers like mountains, rivers, bays, etc. Since it is often not feasible to measure the *actual distances* for all pairs of points, it is a common practice to use distance estimators. Then, how to choose a good estimator so that accurate distance approximations are obtained, becomes an important issue. The estimator should be constructed in such a way that it considers the effect of natural barriers on the road distance between any given pair of points within a geographical region.

A good estimation of actual distances is critical in many applications. Problems such as the distribution problem, the transportation problem, the transshipment problem, the traveling salesman problem, and the vehicle routing problem assume the knowledge of actual distances in their formulations. Therefore, the existence of an estimator enables the reproduction of missing distance values whenever the coordinates of points are available. For example, in their simulation study to determine the number of fire-stations in İstanbul, Erkut and Polat multiply the Euclidean distance by an inflation factor, which they call the *road coefficient*, in order to estimate the actual distance between a fire-station and the fire area [8].

The necessity of distance estimators rise also from Location Theory which mainly deals with the determination of the best possible location within a region of a point with respect to a given criterion [9, 10]. This criterion usually involves the minimization of a distance-based objective function where the distance information is supplied by means of a distance function, which is a parameterized function of certain location data such as the

coordinates of the existing points. Therefore, the values of the parameters of the distance functions have to be known in order to formulate the problem.

Other application areas of distance functions are truck scheduling models and Geographical Information Systems (GIS). The two most widely used truck dispatching software packages in North America use distance functions either completely or to augment network data inputs [11, 12]. The great advantage of using a distance function rather than attempting to build a distance data bank is the speed of installation and absolute coverage of a geographical area by a distance function compared to the limited set of distances available from a data set and the high cost and large amount of time required to prepare a distance data file.

The problem of distance estimation can be defined formally as follows: Given two points P_1 and P_2 on the Cartesian plane with coordinates $\mathbf{x}_1 = (x_{11}, x_{12})^T$ and $\mathbf{x}_2 = (x_{21}, x_{22})^T$, respectively, the aim is to build an estimator $\Phi(\mathbf{x}_1, \mathbf{x}_2 | \boldsymbol{\theta})$ for the actual distance between P_1 and P_2. Let $\pi_i = \langle P_1^i, P_2^i \rangle$ be the ith pair of points, and let r^i be the actual distance between P_1^i and P_2^i. The set of all pairs and the corresponding distances is given by $\mathcal{S}$ as follows:

$$\mathcal{S} = \left\{ \left(\mathbf{x}_1^i, \mathbf{x}_2^i, r^i \right) : 1 \leq i \leq n \right\} \tag{2.1}$$

where, $n = \binom{N}{2} = N(N-1)/2$ is the number of all possible pairs formed by using N points. $\boldsymbol{\theta}$ is a vector of parameters estimated using $\mathcal{S}$ with respect to the following goodness-of-fit criterion:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{i=1}^{n} \delta \left(\Phi \left(\mathbf{x}_1^i, \mathbf{x}_2^i | \boldsymbol{\theta} \right), r^i \right) \right\}. \tag{2.2}$$

$\delta(\cdot)$ is called the difference or error measure. One possibility, originally proposed by Love and Morris [13], is the absolute value of the deviation:

$$\delta\left(\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right), r^i\right) = \left|\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right) - r^i\right|. \tag{2.3}$$

According to this criterion, a distance function must estimate greater actual distances relatively more accurately than shorter distances. For example, a 50 per cent error in estimating a 10 kilometer actual distance, which is 5 km., is relatively unimportant when compared to a 50 per cent error in estimating a 500 kilometer actual distance, which is 250 km., although the errors are proportionately the same. This is a drawback if more emphasis is given to proportional deviations than to absolute deviations. Another error measure, proposed again by Love and Morris [13], is obtained by normalizing the pairwise estimation errors by dividing with the square root of the actual distance between them, i.e.,

$$\delta\left(\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right), r^i\right) = \left(\frac{\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right) - r^i}{\sqrt{r^i}}\right)^2. \tag{2.4}$$

Although both criteria provide ample insight in their own right, the latter one is superior not only because it gives importance to proportional errors but also because of the following three reasons. First, most of the experimental results in the literature use the second criterion, e.g., [13–18] and hence serve as an excellent benchmark. Furthermore, it has important statistical properties which leads to statistical tests for comparing the accuracy of distance functions under certain normality and independence assumptions, and thus the results obtained can be statistically justified [19, 20]. Finally, it is a continuous and differentiable function of the parameter vector which enables the use of gradient descent minimization strategies important in various domains including neural network learning.

Although both criteria are very insightful in their own direction it might be of great interest to know the values of the parameters which minimize the maximum possible estimation error as well. The following two criteria are introduced to serve this purpose.

The first one of these two conservative criteria aims at minimizing the maximum absolute estimation error:

$$\delta\left(\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right), r^i\right) = \max_{1\le i\le n}\left\{\left|\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right) - r^i\right|\right\}. \tag{2.5}$$

This measure may be of special importance particularly when distances for emergency services are to be estimated. The importance comes from the fact that the maximum distance traveled by an ambulance or fire engine should be minimized and thus the parameters of the adopted distance estimator should estimate the length of the longest journey better.

The second criterion combines the conservatism with the intuition underlying the measure given by Equation (2.4) and minimizes the maximum of the normalized absolute deviations:

$$\delta\left(\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right), r^i\right) = \max_{1\le i\le n}\left\{\frac{\left|\Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right) - r^i\right|}{\sqrt{r^i}}\right\}. \tag{2.6}$$

The standard approach for distance estimation uses estimators that are parameterized functions of certain "easy-to-obtain" pieces of information, namely the coordinates of the points. This approach has been widely used ever since the first work by Love and Morris [13] since it provides simple analytical closed-form expressions of the coordinates once the values of the parameters have been determined. The next section gives an overview of the available distance functions and elaborates more on those which are used in this work.

2.2. Parametric Distance Functions

There is a large number of distance functions available to estimate actual distances between two points in a geographical region. Before classifying all these distance functions, it will be appropriate to give the formal definition of a distance function in general.

A distance function in $\Re^n$ is a real-valued function, $d(\mathbf{x},\mathbf{y}) : \Re^n \rightarrow \Re$, and is defined by the following properties:

(i)	nonnegativity:	$d(\mathbf{x},\mathbf{y}) \geq 0 \quad \forall\, \mathbf{x},\mathbf{y} \in \Re^n$;
(ii)	symmetry:	$d(\mathbf{x},\mathbf{y}) = d(\mathbf{y},\mathbf{x}) \quad \forall\, \mathbf{x},\mathbf{y} \in \Re^n$;
(iii)	identity:	$d(\mathbf{x},\mathbf{y}) = 0$ if and only if $\mathbf{x} = \mathbf{y}$;
(iv)	triangle inequality:	$d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{z}) + d(\mathbf{z},\mathbf{y}) \quad \forall\, \mathbf{x},\mathbf{y},\mathbf{z}.$

It is closely related to norms. A norm in $\Re^n$ is a real-valued function, $\|\cdot\| : \Re^n \rightarrow \Re$, having the following properties:

(i)	positivity:	$\|\mathbf{v}\| > 0 \quad \forall\, \mathbf{v} \neq \mathbf{0}$ where $\mathbf{v} = (v_1, \ldots, v_n)^T \in \Re^n$;
(ii)	symmetry:	$\|\mathbf{v}\| = \|-\mathbf{v}\|$;
(iii)	identity:	$\|\mathbf{v}\| = 0$ if and only if $\mathbf{v} = \mathbf{0}$;
(iv)	homogeneity:	$\|\alpha\mathbf{v}\| = \|\alpha\| \, \|\mathbf{v}\| \quad \forall\, \alpha \in \Re$;
(v)	triangle inequality:	$\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$.

Besides it can be shown easily that positive multiple of a norm and the sum of two norms are also norms. As it can be noticed easily, any norm is a distance function. Consequently, norms form the basis of many distance functions found in the literature.

Consider the situation in which a user is given a set of N points or nodes (cities and towns), $\mathcal{G} = \{P_i : 1 \leq i \leq N\}$, located in a two-dimensional "physical" space (e.g., on earth

surface). The actual road distance between each pair of these nodes can be modeled as an unknown arbitrary distance function Ψ. By arbitrary, it is implied that the set of inter-node distances dictated by Ψ may or may not satisfy all the rigorous properties of a well-defined distance function (e.g., the triangular inequality may be violated). However, to keep the informal concepts of a distance function valid, the following requirements are imposed to Ψ. First of all, $\Psi(\mathbf{x}_i, \mathbf{x}_i)$ must be zero, and $\Psi(\mathbf{x}_i, \mathbf{x}_j)$ must be symmetric. Furthermore, let $\mathbf{x}_i$, $\mathbf{x}_j$, $\mathbf{x}_m$, and $\mathbf{x}_n$ be the coordinates of any four nodes P_i, P_j, P_m, and P_n in $\mathcal{G}$. Then, informally speaking, if the pairs (P_i, P_j) and (P_m, P_n) are "close" to each other in the physical world, the respective arbitrary distances between (P_i, P_m) and (P_j, P_n) must be correspondingly of similar magnitude. These concepts are formalized below.

Definition. A function Ψ is defined to be a valid arbitrary distance function if for every P_i, P_j, P_m, and P_n in $\mathcal{G}$, the following is satisfied:

1. $\Psi(\mathbf{x}_i, \mathbf{x}_i) = 0$,
2. $\Psi(\mathbf{x}_i, \mathbf{x}_j) = \Psi(\mathbf{x}_j, \mathbf{x}_i)$,
3. For every $\delta > 0$ there exists an $\epsilon > 0$ such that
   $\|\mathbf{x}_i - \mathbf{x}_j\| < \delta$ and $\|\mathbf{x}_m - \mathbf{x}_n\| < \delta \Longrightarrow |\Psi(\mathbf{x}_i, \mathbf{x}_m) - \Psi(\mathbf{x}_j, \mathbf{x}_n)| < \epsilon$

In order to estimate the arbitrary distance between any two points (the actual distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the data set $\mathcal{G}$) a number of distance functions may be used which are parameterized functions of the planar coordinates of the two points. These functions can be classified in three major groups with respect to the type of coordinates they use. The members of the first group use spherical coordinates for the purpose of introducing the spherical effect of the earth surface into distance estimation [13, 20]. Although this idea provides certain additional accuracy, the contribution has been experimentally reported to be minor by Love and Morris [13]. The second group consists of functions which use polar coordinates [21, 22]. The motivation is based on the observation that the roads in historically older cities are not usually planned according to a rectangular grid structure and consequently, distances are often better approximated by a ring-radial measure. This approach seems to be very accurate especially for a spider's web-like road network structure.

Three different distance functions may be put under this category: radial, circumferential, and circum-radial metric (see [19, 23] for detailed information). The third group contains some simple functions of the Cartesian coordinates of the points. These are mostly norms or norm-based functions and will be introduced in detail below.

The Euclidean or straight-line distance is the most widely used and known distance function found in the literature:

$$d_1(\mathbf{x}_1, \mathbf{x}_2) = \ell_2(\mathbf{x}_1 - \mathbf{x}_2) = \left(|x_{11} - x_{21}|^2 + |x_{12} - x_{22}|^2\right)^{1/2}. \tag{2.7}$$

Here $\mathbf{x}_1 = (x_{11}, x_{12})^T$ and $\mathbf{x}_2 = (x_{21}, x_{22})^T$ denote the coordinates of the points and $d(\mathbf{x}_1, \mathbf{x}_2)$ is the Euclidean distance between them; this is simply the ℓ_2-norm of coordinate differences.

One reason to consider Euclidean distance is that it is the one people are most familiar with. Another important reason, however, is that Euclidean distance is the shortest distance between any two points on the plane and gives a tight lower bound on the actual road distance between these points. Perhaps it would seem that there are few applications for models using straight-line distances. Road travel between a pair of cities is seldom along a completely straight path. However, a good approximation of the average total distance between several pairs of cities in a region can often be made by using a weighted straight-line distance measure defined with the following equality:

$$d_2(\mathbf{x}_1, \mathbf{x}_2) = k\,\ell_2(\mathbf{x}_1 - \mathbf{x}_2) = k\left(|x_{11} - x_{21}|^2 + |x_{12} - x_{22}|^2\right)^{1/2}. \tag{2.8}$$

Since a positive multiple of a norm is also a norm, $d_2(\mathbf{x}_1, \mathbf{x}_2)$, known as the weighted Euclidean distance, is a norm. The parameter k is an inflation or correction factor that helps to account for hills, bends and other forms of "noise" in the transportation network. For example, Krarup and Pruzan [24] report that in a region with a fairly developed road network,

the average ratio of actual road distance to Euclidean distance between points varies between 1.15 and 1.25. But, studies have indicated that such corrected distances may in fact be rather inaccurate in describing the actual distances to be traveled, and therefore several researchers such as Love and Morris [13, 20] and Berens and Körling [25, 26] have attempted to find more accurate distance functions which are also computationally tractable.

As another example for the inappropriateness of the Euclidean distance consider the case of movement on plant and warehouse floors that are arranged into rectangular bays. Movement is allowed only along aisles between these bays. In fact, particularly in industrial contexts when there is often a rectilinear aisle network, the movement between any two points is a straight-line only when the points lie on the same aisle. Hence, another possibility is to consider the *rectilinear distances* given below when working with a rectilinear city-street network or plant floor network.

$$d_3(\mathbf{x}_1, \mathbf{x}_2) = \ell_1(\mathbf{x}_1 - \mathbf{x}_2) = (|x_{11} - x_{21}| + |x_{12} - x_{22}|) \tag{2.9}$$

Rectilinear or rectangular distance, which is simply the ℓ_1-norm of coordinate differences, is also known as Manhattan distance because of the recognition that the street network on the island of Manhattan in New York is a rectilinear one. The *weighted rectilinear distance* can be defined similar to the weighted Euclidean distance as follows:

$$d_4(\mathbf{x}_1, \mathbf{x}_2) = k\,\ell_1(\mathbf{x}_1 - \mathbf{x}_2) = k\,(|x_{11} - x_{21}| + |x_{12} - x_{22}|)\,. \tag{2.10}$$

The parameter k is again there to include the effect of hills, valleys or other noise sources in the road network.

Another distance function which is also a function of the absolute differences of the coordinates is based on the ℓ_∞-norm or *Tchebycheff norm*:

$$d_5(\mathbf{x}_1, \mathbf{x}_2) = \ell_\infty(\mathbf{x}_1 - \mathbf{x}_2) = \max\{|x_{11} - x_{21}|, |x_{12} - x_{22}|\}. \tag{2.11}$$

The ℓ_p-norm of coordinate differences which is given with the following equality is called the ℓ_p-distance. It is a distance function which is widely used in continuous location theory.

$$d_6(\mathbf{x}_1, \mathbf{x}_2) = \ell_p(\mathbf{x}_1 - \mathbf{x}_2) = (|x_{11} - x_{21}|^p + |x_{12} - x_{22}|^p)^{1/p} \quad p \geq 1 \tag{2.12}$$

Clearly, rectilinear, Euclidean, and Tchebycheff distances are special cases of the ℓ_p-distance. In particular, one can obtain the rectilinear distance when $p = 1$, the Euclidean distance when $p = 2$, and the Tchebycheff distance when $p = \infty$. It is worth mentioning that $\ell_p(\cdot)$ is not a norm for $p < 1$ since triangle inequality property is violated. It is reasonable to expect more accurate estimations of actual distances when ℓ_p-distance is used instead of its special cases, since the existence of p as a parameter increases the flexibility to model actual distances.

The ℓ_p-distance has also a weighted version, called the *weighted ℓ_p-distance*. Since it has two parameters, it provides an even more flexible function of the coordinates to model actual distances.

$$d_7(\mathbf{x}_1, \mathbf{x}_2) = k\,\ell_p(\mathbf{x}_1 - \mathbf{x}_2) = k\,(|x_{11} - x_{21}|^p + |x_{12} - x_{22}|^p)^{1/p} \quad p \geq 1 \tag{2.13}$$

It is possible to make the weighted ℓ_p-distance more flexible in order to provide a more accurate estimator by introducing a new parameter s in addition to the parameters k and p. This distance function, which is not a norm because of the violation of the homogeneity

requirement, — property (iv) of norms — has the following form:

$$d_8(\mathbf{x}_1, \mathbf{x}_2) = k\,[\ell_p(\mathbf{x}_1 - \mathbf{x}_2)]^{p/s} = k\left(|x_{11} - x_{21}|^p + |x_{12} - x_{22}|^p\right)^{1/s}. \qquad (2.14)$$

There are also distance measures which do not fit completely in any of the above mentioned three groups. They can be included in the last category, but they are not always simple functions of the coordinates. All of them are based on the idea that a travel has two major components; rectilinear and Euclidean, and the actual distance between any pair of points can be modeled as their non-negative linear combination. Ward and Wendell [18] initiate this hybrid idea by suggesting the weighted one-infinity norm. This norm is a positive linear combination of the rectilinear (ℓ_1) and Tchebycheff (ℓ_∞) norms. Letting α_1 and α_2 be nonnegative numbers, not both of which are zero, i.e., $\alpha_1 + \alpha_2 > 0$, the weighted one-infinity norm is defined as:

$$d_9(\mathbf{x}_1, \mathbf{x}_2) = \ell_{1,\infty}(\mathbf{x}_1 - \mathbf{x}_2) = \alpha_1 \ell_1(\mathbf{x}_1 - \mathbf{x}_2) + \alpha_2 \sqrt{2}\ell_\infty(\mathbf{x}_1 - \mathbf{x}_2). \qquad (2.15)$$

It is to be noted that when (α_1, α_2)=$(1, 0)$, the rectilinear norm and when (α_1, α_2)=$\left(0, 1/\sqrt{2}\right)$, the Tchebycheff norm is obtained. It can be shown that the factor $\sqrt{2}$ models the Euclidean part of the travel between $\mathbf{x}_1$ and $\mathbf{x}_2$ [19]. It is observed that the accuracy of this distance function is relatively close to the accuracy of the weighted ℓ_p-distance $d_7(\mathbf{x}_1, \mathbf{x}_2)$ based on the data set of Love and Morris [13].

The idea that a travel has a Euclidean and a rectilinear component can be regarded as decomposing the travel into two distinct moves, which are either parallel to one of the coordinate axes or parallel to the main diagonals. Hence, it is possible to obtain more accurate estimations if additional travel directions are also considered. Ward and Wendell use this rationale and generalize the one-infinity norm to obtain the family of block norms in which the accuracy of the approximation depends on possible travel directions [29]. The

block norm is characterized as follows:

$$d_{10}(\mathbf{x}_1, \mathbf{x}_2) = \ell_{block}(\mathbf{x}_1 - \mathbf{x}_2) = \min\left\{\sum_{g=1}^{r} |\beta_g| : \mathbf{x}_1 - \mathbf{x}_2 = \sum_{g=1}^{r} \beta_g \mathbf{b}_g\right\}. \quad (2.16)$$

In this definition, $\mathbf{b}_g$'s are unit vectors which define the permissible directions of travel in the network. If one imagines that there is a dense highway grid with roads in each of the direction $\{\mathbf{b}_g : g = \pm 1, \ldots, \pm r\}$ at all points, then a trip from the origin to a destination must be made along roads in only those directions and $|\beta_g|$ is the distance traveled on roads parallel to $\mathbf{b}_g$. Ward and Wendell report that the approximations obtained by the weighted ℓ_p-norm are more accurate than those obtained by a two-parameter block norm, which is actually the weighted one-infinity norm, and the accuracy of the weighted ℓ_p- norm is slightly worse than the one of eight-parameter block norm. A further study by Love and Walker [17] shows that the weighted ℓ_p-norm is generally more accurate than a block norm with five parameters. Moreover, increasing the number of parameters to eight or more in a block norm does not guarantee that a more accurate distance function than the weighted ℓ_p-norm will be obtained. Block norms play an important role in location models because they lead to linear programming problems for certain objective functions, such as the minimax distance function. However, the size of the linear program (the number of variables and constraints) can easily become very large with the problem size (the number of locations).

Another hybrid distance function is due to Brimberg and Love [28]. It is called the weighted one-two norm since the rectilinear and Euclidean elements of the travel are represented respectively by the weighted ℓ_1 and ℓ_2 norms:

$$d_{11}(\mathbf{x}_1, \mathbf{x}_2) = \ell_{1,2}(\mathbf{x}_1 - \mathbf{x}_2) = \beta_1 \ell_1(\mathbf{x}_1 - \mathbf{x}_2) + \beta_2 \ell_2(\mathbf{x}_1 - \mathbf{x}_2). \quad (2.17)$$

β_1 and β_2 are nonnegative constants not both of which are zero. Letting $\alpha_1 = \beta_1/(\beta_1 + \beta_2)$ and $\alpha_2 = \beta_2/(\beta_1 + \beta_2)$, where $0 \leq \alpha_1, \alpha_2 \leq 1$:

$$d_{11}(\mathbf{x}_1, \mathbf{x}_2) = (\beta_1 + \beta_2)\left[\alpha_1 \ell_1(\mathbf{x}_1 - \mathbf{x}_2) + \alpha_2 \ell_2(\mathbf{x}_1 - \mathbf{x}_2)\right]. \tag{2.18}$$

The parameter α_1 gives the proportion of the route which is rectilinear, while the remaining proportion, $\alpha_2 = 1 - \alpha_1$ is Euclidean. The parameter $(\beta_1 + \beta_2)$ takes on the role of an inflation factor. The authors suggest its use to approximate $d_7(\mathbf{x}_1, \mathbf{x}_2)$ in estimating distances. The parameters can be calculated easily by simple linear regression [27].

The effect of local characteristics on the actual distances can be introduced into estimations by rotating the original coordinate axes by an angle γ [20]. Love and Morris used this idea before employing the weighted ℓ_1-distance, $d_4(\mathbf{x}_1, \mathbf{x}_2)$. This approach results in the following two-parameter distance function:

$$d_4(\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2) = k\left(|\hat{x}_{11} - \hat{x}_{21}| + |\hat{x}_{12} - \hat{x}_{22}|\right). \tag{2.19}$$

Here, $\hat{x}_{11}$, $\hat{x}_{12}$, $\hat{x}_{21}$, and $\hat{x}_{22}$ are the coordinates in the rotated system and they can be calculated with the following formula when γ is given :

$$\begin{pmatrix} \hat{x}_{11} & \hat{x}_{12} \\ \hat{x}_{21} & \hat{x}_{22} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \end{pmatrix} \begin{pmatrix} \cos\gamma & -\sin\gamma \\ \sin\gamma & \cos\gamma \end{pmatrix}. \tag{2.20}$$

Later on axes rotation has been applied to two more general distance functions: the weighted ℓ_p-distance, $d_7(\mathbf{x}_1, \mathbf{x}_2)$ [16, 17], and the weighted one-two norm $d_{11}(\mathbf{x}_1, \mathbf{x}_2)$ [27,28].

TABLE 2.2.1. Suitable distance functions for Türkiye.

Estimator	Distance Function	Parameters (θ)
$\Phi_1(\mathbf{x}_1,\mathbf{x}_2)$	$d_4(\mathbf{x}_1,\mathbf{x}_2) = k(\lvert x_{11}-x_{21}\rvert + \lvert x_{12}-x_{22}\rvert)$	k
$\Phi_2(\mathbf{x}_1,\mathbf{x}_2)$	$d_2(\mathbf{x}_1,\mathbf{x}_2) = k\left(\lvert x_{11}-x_{21}\rvert^2 + \lvert x_{12}-x_{22}\rvert^2\right)^{1/2}$	k
$\Phi_3(\mathbf{x}_1,\mathbf{x}_2)$	$d_7(\mathbf{x}_1,\mathbf{x}_2) = k(\lvert x_{11}-x_{21}\rvert^p + \lvert x_{12}-x_{22}\rvert^p)^{1/p}$	k,p
$\Phi_4(\mathbf{x}_1,\mathbf{x}_2)$	$d_8(\mathbf{x}_1,\mathbf{x}_2) = k(\lvert x_{11}-x_{21}\rvert^p + \lvert x_{12}-x_{22}\rvert^p)^{1/s}$	k,p,s
$\Phi_5(\mathbf{x}_1,\mathbf{x}_2)$	$d_{11}(\mathbf{x}_1,\mathbf{x}_2) = \beta_1(\lvert x_{11}-x_{21}\rvert + \lvert x_{12}-x_{22}\rvert) + \beta_2\left(\lvert x_{11}-x_{21}\rvert^2 + \lvert x_{12}-x_{22}\rvert^2\right)^{1/2}$	β_1,β_2

Prior to deciding which distance functions to use for an application domain (which is the road network of Türkiye in this case) it may be appropriate to eliminate some of the distance functions beforehand by judging them with the structural properties of the application. First of all, the road structure in Türkiye has been developed arbitrarily rather than rectilinearly or ring-radially. This arbitrariness makes the identification of a fixed pattern for possible travel directions within the country impossible; this is crucial for the use of block norms. Besides, the area is too small to require the consideration of the earth's roundness in estimating actual distances. Being convinced by these observations, it is rational to concentrate on functions $d_2(\mathbf{x}_1,\mathbf{x}_2)$, $d_7(\mathbf{x}_1,\mathbf{x}_2)$, $d_8(\mathbf{x}_1,\mathbf{x}_2)$, $d_{11}(\mathbf{x}_1,\mathbf{x}_2)$ and compute the best possible values of the parameters k, p, s, β_1, and β_2. In spite of this fact, the value of k for $d_4(\mathbf{x}_1,\mathbf{x}_2)$ is also computed, since it has been heavily considered in the related literature [9, 10]. These distance function are listed in Table 2.2.1.

Indeed, in the literature these are the most important ones, because of their wide usage in location and distribution problems [9]. Determining the parameters of these distance functions will be covered in the subsequent section.

2.3. Estimation of Parameters

The parameters k, p, s, β_1 and β_2 of the distance functions listed in Table 2.2.1 constitute the parameter vector θ, and are estimated over a sample set to provide good

approximations (see Section 2.1) and to encode geographical characteristics of the region where they are used. The calculation of the parameters with respect to any of the goodness-of-fit criterion introduced in Section 2.1 requires minimizing an error function similar to that of Equation (2.2). Since the normalized error function given in Equation (2.4) is adopted, the minimization problems are continuous in the parameters. Karush-Kuhn-Tucker first order conditions have analytical solutions when the distance functions $\Phi_1(\mathbf{x}_1^i, \mathbf{x}_2^i)$, $\Phi_2(\mathbf{x}_1^i, \mathbf{x}_2^i)$ and $\Phi_5(\mathbf{x}_1^i, \mathbf{x}_2^i)$ are used and the values of k, β_1, and β_2 which minimize the error measure given with Equation (2.4) can be easily obtained by using the following equalities:

For $\Phi_1(\mathbf{x}_1^i, \mathbf{x}_2^i)$:

$$k = \frac{\sum_{i=1}^{n} \left(|x_{11}^i - x_{21}^i| + |x_{12}^i - x_{22}^i| \right)}{\sum_{i=1}^{n} \left(|x_{11}^i - x_{21}^i| + |x_{12}^i - x_{22}^i| \right)^2 / r^i} \tag{2.21}$$

for $\Phi_2(\mathbf{x}_1^i, \mathbf{x}_2^i)$:

$$k = \frac{\sum_{i=1}^{n} \left(|x_{11}^i - x_{21}^i|^2 + |x_{12}^i - x_{22}^i|^2 \right)^{1/2}}{\sum_{i=1}^{n} \left(|x_{11}^i - x_{21}^i|^2 + |x_{12}^i - x_{22}^i|^2 \right) / r^i} \tag{2.22}$$

and for $\Phi_5(\mathbf{x}_1^i, \mathbf{x}_2^i)$:

$$\beta_1 = \frac{a_{13}a_{22} - a_{12}a_{23}}{a_{21}^2 - a_{22}a_{11}}, \quad \beta_2 = \frac{a_{21}a_{13} - a_{23}a_{11}}{a_{21}^2 - a_{22}a_{11}} \tag{2.23}$$

where

$$
\begin{aligned}
a_{11} &= \sum_{i=1}^{n} \frac{\left(|x_{11}^i - x_{21}^i| + |x_{12}^i - x_{22}^i|\right)^2}{r^i} \\
a_{12} &= a_{21} \\
&= \sum_{i=1}^{n} \frac{\left(|x_{11}^i - x_{21}^i| + |x_{12}^i - x_{22}^i|\right)^2 \cdot \left(|x_{11}^i - x_{21}^i|^2 + |x_{12}^i - x_{22}^i|^2\right)^{1/2}}{r^i} \\
a_{22} &= \sum_{i=1}^{n} \frac{\left(|x_{11}^i - x_{21}^i|^2 + |x_{12}^i - x_{22}^i|^2\right)}{r^i} \\
a_{13} &= \sum_{i=1}^{n} \left(|x_{11}^i - x_{21}^i| + |x_{12}^i - x_{22}^i|\right) \\
a_{23} &= \sum_{i=1}^{n} \left(|x_{11}^i - x_{21}^i|^2 + |x_{12}^i - x_{22}^i|^2\right)^{1/2}
\end{aligned}
\tag{2.24}
$$

However, the calculations of the parameters k and p for $\Phi_3\left(\mathbf{x}_1^i, \mathbf{x}_2^i\right)$, and k, p and s for $\Phi_4\left(\mathbf{x}_1^i, \mathbf{x}_2^i\right)$ are slightly more complicated. They require the solution of the following unconstrained optimization problems:

$$
\min_{k,p} \sum_{i=1}^{n} \left(\frac{k\left(|x_{11}^i - x_{21}^i|^p + |x_{12}^i - x_{22}^i|^p\right)^{1/p} - r^i}{\sqrt{r^i}} \right)^2 \tag{2.25}
$$

$$
\min_{k,p,s} \sum_{i=1}^{n} \left(\frac{k\left(|x_{11}^i - x_{21}^i|^p + |x_{12}^i - x_{22}^i|^p\right)^{1/s} - r^i}{\sqrt{r^i}} \right)^2 \tag{2.26}
$$

Although they are quite simple with respect to the number of variables (which is two and three, respectively), the number of nonlinearities induced by these problems can be very large depending on the size of the pairs of points within the data set. These minimization problems can be handled by using any known nonlinear optimization package, such as MINOS 5.1 [30].

There is a large literature on the determination of the parameters and on the comparison of the goodness-of-fit of the distance functions. Astonishingly enough, some of the conclusions drawn in these papers are conflicting [13,15,20,25–27,31]. For all practical purposes, the function chosen to estimate actual road distances should be as accurate as possible. Love and Morris [13, 20] carry out an empirical study in which the best fitting parameter values of k, p, and s of $\Phi_1(\mathbf{x}_1, \mathbf{x}_2)$, $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$, $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$, and $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ are obtained for sets of data from urban and rural regions of the United States and compare these distance functions with respect to the accuracy they provide. The conclusion of this study is the superiority of $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ over the other three. The second best approximating function turns out to be $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$.

Berens and Körling [25] examine the road network of the former Federal Republic of Germany (FRG), and conclude that the accuracy provided by the weighted Euclidean distance $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ is sufficient and does not justify the extra computational effort necessary for calculating the parameters k, and p of the weighted ℓ_p-distance, $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$. However, in a further study over the largest 25 cities of FRG, Love and Morris [31] produce different results which demonstrate that the accuracy of the weighted ℓ_p-distance, $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$, is remarkably higher than the accuracy provided by the weighted Euclidean distance $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$. A further study by Berens [15] supports the earlier findings of Berens and Körling [25] for FRG. However, this latter study includes 11 other countries and produces mixed results. The relative improvement obtained by using $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$ with respect to $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ varied from 11.27 per cent in the United State to zero per cent in Austria. Finally, Berens and Körling [26], in their last comment state that if the accuracy is of primary interest, then the empirical distance functions should be tailored to the region under consideration. Currently, there is no single general distance function that provides the same accuracy all over the world.

The rotation of the coordinate axes may improve the estimation accuracy too. In a study of 18 geographical areas, Brimberg, Love and Walker [16] find improvements as high as 64 per cent using the weighted ℓ_p-distance, $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$ with axis rotation.

3. ARTIFICIAL NEURAL NETWORKS

3.1. Preliminary

Artificial neural networks (Neural Networks in short) and neural computing in the wide sense are among todays most rapidly developing scientific disciplines. Neural networks are parallel computational models that consist mainly of interconnected adaptive processing units. These networks are considered fine-grained parallel implementation of nonlinear dynamic and static systems. A neural network is an abstract simulation of real nervous system that contains a collection of processing units or processing elements or neurons that communicate with each other via axon connections. Such a model resembles the axons and dendrites of the nervous system. Because of its self-organizing and adaptive nature, the model provides a new parallel and distributed paradigm that has the potential to be more robust and user-friendly than traditional schemes.

The study of neural networks is an attempt to simulate and understand biological processes in an intriguing manner. Today we are witnessing the dawn of a new revolution in technology that will rebuild the infrastructure of many approaches to solve cybernetics, information, and system engineering problems among others. It is of interest to define alternative computational paradigms that attempt to mimic to brain's operation in several ways. Neural networks are an alternative approach to the traditional von Neumann programming schemes.

Interest in neural networks has increased in recent years due partly to some significant breakthroughs in research in architecture, learning/training algorithms and operational characteristics. Advances in computer hardware technology that make neural networks implementation faster and more efficient have also contributed to the progress in research and development in neural networks. Much of the drive has arisen because of the numerous successes achieved in demonstrating the ability of neural networks to deliver elegant and powerful solutions especially in the fields of learning, pattern recognition, optimization and

classification, which have proved to be difficult and computationally intensive for traditional von Neumann computing schemes. Due to its interdisciplinary nature, encompassing computing, biology, neuropsychology, physics, engineering, biomedicine, communications, pattern recognition, and image processing etc., the field of neural networks attracts a number of researchers and developers from a broad range of backgrounds. Today, there are many different paradigms and applications of neural networks, reflecting research and development groups.

Neural networks are viable computational models for a wide variety of applications including pattern recognition, speech recognition, and synthesis, image compression, adaptive interfaces between human and machines, clustering, forecasting and prediction, diagnosis, function approximation, nonlinear system modeling and control, optimization, routing in parallel computer systems and high-speed networks, and associative memory. The field of neural networks links a number of closely related areas that include parallel and distributed computing, connectionism, and neural computing. These areas are brought together with the common theme of attempting to exhibit the computing method witnessed in the study of biological neural systems.

In neural networks, the element that corresponds to a biological unit is called a processing element or a processing unit or a neuron. Each neuron combines its input paths by adding up the weighted sum of all inputs. The output of a neuron is the signal that is generated by applying the combined inputs to an appropriate transfer or activation function. (see Figure 3.1.1). A suitable activation function can be chosen by trial-and-error method from among several commonly used functions such as tanh, sigmoid, sink or linear.

Learning takes place in the form of adjustment of weights connecting the inputs to the neuron. In neural networks there are various ways in which the neurons are interconnected as groups called *layers*.

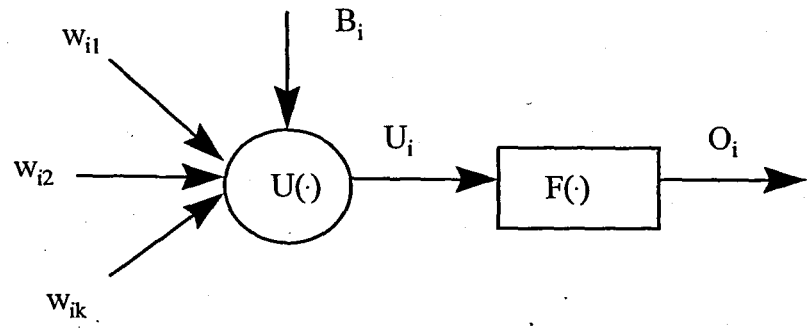


FIGURE 3.1.1. A processing element or neuron.

The behavior and performance of the neural network depends mainly on the interaction between the neurons in these layers. There are three types of layers: input, hidden and output layers. Two layers of neurons communicate with each other via the weight connections between them. Four types of weight connections can be identified:

(i) feedforward
(ii) feedback
(iii) lateral
(iv) time-delayed.

In the feedforward type of connection, data from neurons of lower layers are propagated forward to neurons of upper layers through feedforward connection networks. In the feedback type, data are brought from neurons of upper layer back to neurons of lower layer. In networks that contain lateral connections among the neurons in the output layer, it is permitted that neurons in this layer interact with each other. An example of this type of neural networks is winner-take-all circuits in which an output neuron may send an inhibitory or excitory signal to the other output neurons to which it is connected. Topological ordering relationships can be preserved in these networks. These networks will be discussed in detail in the forthcoming chapters where neural network based solution approaches to the Euclidean traveling salesman problem are introduced. Finally, time-delayed connections are included in a network in order to provide temporal dynamic models. There exists a variety of training (learning) rules that determine how, when, and by what magnitudes are the weights on all types of connections updated.

A fundamental feature of neural networks is their adaptive nature, where learning by examples replaces programming in solving problems. This important feature makes such computational paradigms very appealing in the application domain where one has little or incomplete understanding of the problem to be solved, however, there exist some training data. The parallel and distributed architecture feature of neural networks allows for fast computational solution when these networks are implemented on parallel and/or distributed computer systems.

One aspect of neural networks is the use of simple processing elements which are essentially approximate models of the neurons in the brain. It is estimated that the brain contains over 100 billion (10^{11}) neurons of different types and (10^{14}) synapses in the human nervous system. Recent studies in the brain have found that there are more than 1000 synapses on the input/output of each neuron. A neuron is the fundamental cellular unit of the nervous system and, in particular, the brain. The "artificial neuron" is the basic building block unit of any neural network. Each neuron can be regarded as a simple processing element that receives and combines signals from many other neurons through input structures called "dendrites". For a combined input signal having values greater than a certain threshold the firing of a neuron takes place resulting in an output signal that is transmitted along a cell component called "axon". The axon of a neuron splits up and connects to dendrites of other neurons through the "synapse". The strength of the signal transmitted across a synaps is called synaptic efficiency which is modified as the brain learns.

Neural networks can be classified into different categories based on the selected criterion. Based on the learning (training) method (algorithm), neural networks can be divided into supervised, unsupervised and reinforcement learning types.

Supervised learning refers to the design of a classifier in the case that the underlying class of available samples is known. In supervised learning, each input pattern received from the environment is associated with a specific desired target pattern. The weights are usually synthesized gradually, and at each step of the learning process they are updated in order to minimize the error between the network's output and a corresponding desired target.

In the unsupervised learning case, it is necessary to classify data into a number of groups without the aid of a training set. The goal here is to separate the data into M classes. The idea is to optimize some criterion or performance metric/measure defined in terms of the output activity of the units in the network. The weights and the outputs in this case are usually expected to converge to representations that capture the statistical regulatories of the input data. In order to accomplish this, a clustering criterion is defined which assigns a numerical value to each possible assignment of samples to clusters. It is too costly in general to simply evaluate the criterion for each possible assignment, therefore a method must be used to find an optimal assignment.

The third class is the reinforcement training (learning) algorithm which is between supervised and unsupervised learning. Reinforcement learning involves updating the network's weights in response to an "evaluative" teacher who basically tells whether the answer is correct or not. It involves rules that may be built within the framework of a stochastic search mechanism that attempts to maximize the probability of positive external reinforcement for a given training set.

From estimation point of view, neural networks can be classified into estimating and non-estimating families. The estimating neural networks use the Parzen estimators for estimating the probability density function for a given data set. The other family is made up of non-estimating neural networks which can not estimate the probability density function automatically from the data set.

The application domain of neural networks may be grouped in the following main categories:

(i) classification, clustering, diagnosis and association;
(ii) optimization;
(iii) regression/generalization;
(iv) pattern completion.

3.1.1. Classification, Clustering, Diagnosis and Association

In this paradigm, input static patterns or temporal signals are to be recognized or classified. A classifier should be trained so that when a slightly distorted version of a stimulus is presented, it can still be correctly recognized. The network should have a good noise impunity capability which is critical for some applications such as hollographic and retrieval applications. An example is given in Figure 3.1.2.

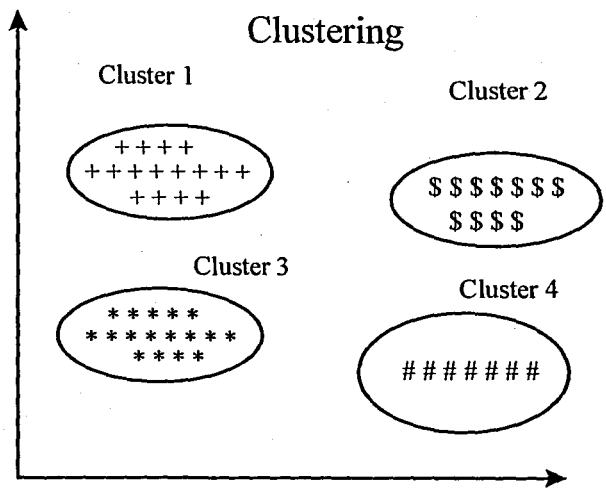


FIGURE 3.1.2. A classification example.

3.1.2. Optimization

Neural networks are very appealing for solving optimization problems which involve finding a global optimum of a function (see Figure 3.1.3). The determination of the synaptic weights is relatively easy once the energy function, $f(x)$, is found. The cost function is easy to find for some applications, however, in other applications, it has to be derived from a given cost criterion and some constraints related to the problem at hand. One of the main issues related to the optimization problems is the possibility of obtaining a solution converging to a local optimum instead of global optimum. Among the techniques that are proposed to tackle this problem are the simulated annealing and mean-field annealing [32,34].

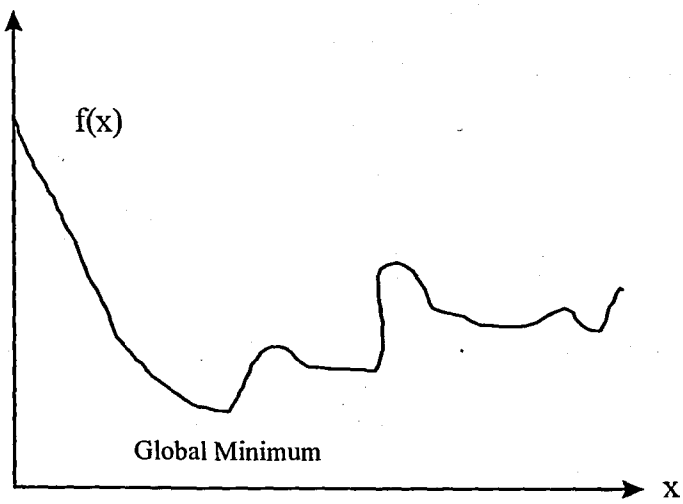


FIGURE 3.1.3. Global minimum of the function $f(x)$.

3.1.3. Regression and Generalization

Regression problem has been studied extensively. Linear and nonlinear regression provide a smooth and robust curve fitting to the training patterns. Usually, the system is trained based on the supervised training scheme using a large data set. A neural network is considered successful if it can closely approximate the teacher values for the trained data set and can provide smooth interpolation for the untrained data set. Generalization is used to yield a correct response to an input stimulus to which it has not been trained. The system must induce the salient feature of the input and detect the regularity. This regularity discovery is vital for many applications. It enables the system to function efficiently throughout the entire data set (space), although it has been trained only by limited portion of the entire data set (space). Figure 3.1.4 shows an example. It is important to note that each neural network model tends to impose its all prejudice in how to generalize from a finite set training data. A good example that shows the difficulty with generalization is the parity-check problem where the backpropagation model consistently generalizes into a wrong result.

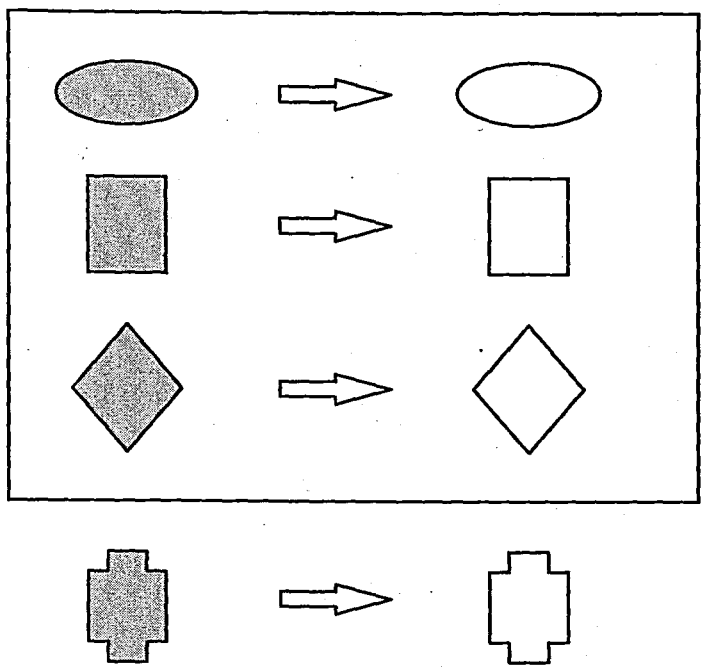


FIGURE 3.1.4. A generalization example.

3.1.4. Pattern Completion

In some classification problems, an implied task is the completion of information, that is, recovery of original data given only partial information. We differentiate between a static and temporal pattern completion problems. Markov models and time-delay dynamic networks are for temporal pattern completion while most traditional multilayer neural networks, Boltzmann machines, and Hopfield networks are for static pattern completion [33].

3.2. Multilayer Perceptrons

Layered feedforward networks were called perceptrons when first studied in detail by Rosenblatt and his colleagues [34]. The following figure shows two examples of perceptrons.

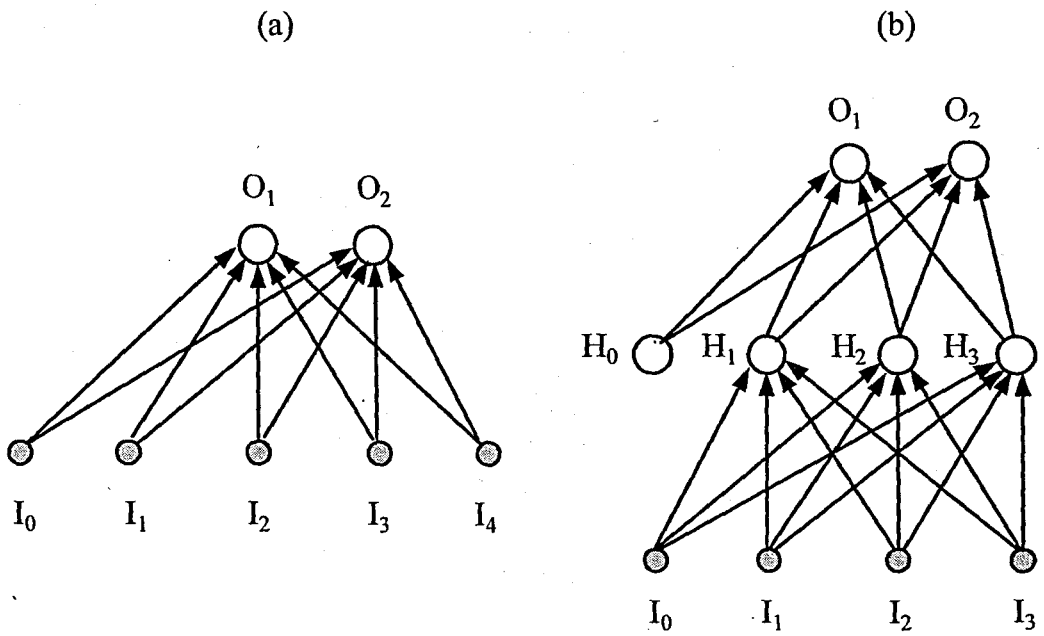


FIGURE 3.2.1. Perceptrons. (a) A simple perceptron with one layer. (b) A two-layer perceptron.

In a multilayer perceptron, input is fed to the input layer which propagates to the output layer through one or more layers of hidden units. The size of a network depends on the number of hidden units per layer and the number of layers. The number of hidden units is directly related to the network capability. There is an optimum number of hidden units that must be properly determined in order to have the best performance. The number of layers is usually counted as the number of weight layers instead of neuron layers. Weights between consecutive layers encode interdependencies or constraints between the values that the units take simultaneously. It has been shown that such a network with one hidden layer is a universal approximator, i.e., can approximate any function with the desired accuracy [35, 36]. A multilayer perceptron with one hidden layer of K hidden units operates as follows: d-dimensional input $\mathbf{u}$ is fed to the input layer I that also includes the bias unit. The "activation" propagates in the forward direction and the values of the hidden units H_h are computed as:

$$\begin{aligned} I_j &= u_j & j &= 1, \ldots, d, \quad I_0 = 1 \\ H_h &= f\left(\sum_{j=0}^{d} W_{hj} I_j\right) & h &= 1, \ldots, K \quad H_0 = 1 \end{aligned} \tag{3.1}$$

where W_{hj} is the weight of the connection from hidden unit h to input unit j. I_0 and H_0 are the bias units of the input layer and hidden layer, respectively.

$f(\cdot)$ is the activation function and is generally taken as the sigmoid which is given below:

$$f(z) = \frac{1}{1+e^{-z}} \tag{3.2}$$

Thus, the output of a hidden unit becomes

$$H_h = \frac{1}{1+\exp\left(-\sum_{j=1}^{k} W_{hj} I_j - W_{h0}\right)} \tag{3.3}$$

There are c output units and the value of each output unit O_o is computed similarly:

$$O_o = g\left(\sum_{h=0}^{K} T_{oh} H_h\right) \quad o = 1, \ldots, c \tag{3.4}$$

where T_{oh} is the weight of the connection from output unit o to hidden unit h.

In a nonlinear regression problem, which is the case with the distance estimation problem, there is only one output unit, i.e., $c = 1$, and $g(\cdot)$ is the identity function. In a classification (clustering) problem, there are as many output units as there classes and $g(\cdot)$ is also sigmoid.

The most popular training method for multilayer perceptrons is the back-propagation learning algorithm [37] which implements gradient descent over an error function. Denoting

E as the error function, which is the sum of all errors over the sample patterns in the data set, the following update equations are obtained for the weights in a multilayer perceptron with one hidden layer:

$$E = \sum_{i \in patterns} E^i \tag{3.5}$$

$$\Delta T^i_{oh} = -\eta \frac{\partial E^i}{\partial T_{oh}} \tag{3.6}$$

$$\Delta W^i_{hj} = -\eta \frac{\partial E^i}{\partial W_{hj}} = -\eta \frac{\partial E^i}{\partial H_h} \frac{\partial H_h}{\partial W_{hj}} \tag{3.7}$$

The latter equality follows from the chain rule which propagates the error back from the output layer towards the input layer. The values of the weights are given random values initially; frequently they are assigned values from the range $[-0.01, 0.01]$. With the learning equations given in (3.5) and (3.6), both the direction in which each weight needs to be changed and the magnitude of this change is computed for each pattern.

In *batch* learning, the changes over all patterns are accumulated until all the patterns in the training set are presented to the network. Such a complete pass over all the patterns in the training set is called an *epoch.* The change in the weights is made after each epoch. In *on-line* learning, the weights are updated immediately after each pattern is introduced to the network. The learning factor or step size of the gradient descent, η, should be chosen smaller if on-line learning is applied.

As the number of epochs increases, performance, i.e., the approximation quality, over the training set increases, but performance over the cross-validation set (test set) starts to decrease beyond a certain point. Learning should be stopped at this point in order to avoid this problem referred to as *overtraining*. A similar behavior happens when the number of hidden units is increased. Too many hidden units force the network to memorize the noise in the training set and thereby prevent the generalization ability of the network to the test set. Hence, it is possible to conclude that when a network is trained too long or when it has too many parameters, its generalization becomes poor. The performance measures of a multilayer perceptron is:

1. Success on the test set that measures how well the network generalizes,
2. Number of weights that measures the memory requirement,
3. Number of training epochs that measures the speed of learning.

Because of the nonlinearity the error function may have many local minima and gradient descent method may converge to one of these local minima. It is recommended that the network be trained several times starting from different initial weight values. Average and standard deviations should also be computed.

Gradient descent may converge very slowly. Whereas there are other techniques such as conjugate gradient method and second order methods for faster and better convergence, there exist also two frequently used techniques that improve the performance of gradient descent fairly. One of these techniques involves adding a momentum term to the update equations and will be reviewed as it is used in the distance estimation problem. At each weight update, successive ΔW_{hj} values may be so different that large oscillations may result. In order to decrease this effect the amount of the previous update is incorporated in the current update as if it plays the role of a momentum or inertia:

$$\Delta W_{hj}(t) = -\eta \frac{\partial E}{\partial \Delta W_{hj}} + \alpha \Delta W_{hj}(t-1). \tag{3.8}$$

t denotes the iteration or epoch index depending on whether batch or on-line learning is used. The value of α is usually taken between 0.5 and 1.0. This approach is particularly useful when on-line learning is employed.

In the application of distance estimation the input to the neural network is a vector function of the coordinates of the two points: $\mathbf{u}^i = \phi(\mathbf{x}_1^i, \mathbf{x}_2^i)$. Here $\mathbf{x}_1^i = (x_{11}^i, x_{12}^i)^T$ and $\mathbf{x}_2^i = (x_{21}^i, x_{22}^i)^T$ denote the coordinates of the ith pair of points in the training set. It should be recalled that there are n pairs of points in the training set which constitute the input patterns to the network. There is only one output unit (therefore the subscript of the weights connecting the output unit with the hidden units is dropped, i.e., $T_{1h} = T_h$, $h = 0, \ldots, H$) and the output of the neural network by a known transformation gives the desired distance:

$$y^i = \Phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i \,|\boldsymbol{\theta}\right) = \chi\left(F\left(\mathbf{u}^i \,|\mathbf{T}, \mathbf{W}\right)\right) \tag{3.9}$$

where

$$\begin{aligned} F\left(\mathbf{u}^i \,|\mathbf{T}, \mathbf{W}\right) &= \sum_{h=1}^{H} T_h H_h + T_0 \\ &= \sum_{h=1}^{H} \frac{T_h}{1 + \exp\left(\left(-\sum_{j=1}^{d} u_j W_{hj} - W_{h0}\right)\right)} + T_0 \end{aligned} \tag{3.10}$$

The parameter vector of the estimator, $\boldsymbol{\theta}$, corresponds to the weights of the neural network, i.e., $\mathbf{T}$ and $\mathbf{W}$. The problem of choosing good *coding* or *representation* functions, $\phi(\cdot)$ and $\chi(\cdot)$, is critical in the application of neural networks. For the difference or error measure

given in Equation (2.4) the update equations of the weights become:

$$E^i = \left(\frac{y^i - r^i}{\sqrt{r^i}}\right)^2 = \left(\frac{\left(\sum_{h=1}^{H} T_h H_h + T_0\right) - r^i}{\sqrt{r^i}}\right)^2 \tag{3.11}$$

$$\Delta T_h = -\eta \frac{\partial E^i}{\partial T_h} = -\eta \left(\frac{y^i - r^i}{r^i}\right) H_h \tag{3.12}$$

$$\Delta T_0 = -\eta \frac{\partial E^i}{\partial T_0} = -\eta \left(\frac{y^i - r^i}{r^i}\right) \tag{3.13}$$

$$\Delta W_{hj} = -\eta \frac{\partial E}{\partial H_h} \frac{\partial H_h}{\partial W_{hj}} = -\eta \left(\frac{y^i - r^i}{r^i}\right) T_h H_h \left(1 - H_h\right) u_j \tag{3.14}$$

$$\Delta W_{h0} = -\eta \frac{\partial E}{\partial H_h} \frac{\partial H_h}{\partial W_{h0}} = -\eta \left(\frac{y^i - r^i}{r^i}\right) T_h H_h \left(1 - H_h\right) \tag{3.15}$$

The last two equations follow since $g'(z) = g(z)(1 - g(z))$ when $g(\cdot)$ is sigmoid, namely:

$$\begin{aligned} g(z) &= \frac{1}{1 + e^{-z}} \\ \frac{\partial g(z)}{\partial z} &= \frac{e^{-z}}{\left(1 + e^{-z}\right)^2} = \frac{1}{1 + e^{-z}} \left(1 - \frac{1}{1 + e^{-z}}\right) \end{aligned} \tag{3.16}$$

3.3. Regression Neural Networks

The nonparametric density estimate of probability at a certain value is a weighted sum of the effects of sample points. Given a sample of n real d-dimensional $\mathbf{X}_i$ values, the estimate of probability at $\mathbf{x}$ is [38]:

$$\hat{p}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{X}_i}{h}\right) \tag{3.17}$$

K is the kernel density function and h is the window width of the kernel. This approach can be used for regression, i.e., for inference of a scalar y value for a given multi-dimensional vector $\mathbf{x}$ if we are given pairs of $(\mathbf{X}_i, Y_i)$. The kernel estimate of the joint probability density $f(\mathbf{x},y)$ is then written as:

$$\hat{f}(\mathbf{x},y) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{X}_i}{h}\right) \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y - Y_i)^2}{2\sigma^2}\right). \tag{3.18}$$

Using Bayes theorem:

$$\hat{f}(y,\mathbf{x}) = \frac{\hat{f}(\mathbf{x},y)}{f(\mathbf{x})} = \frac{\hat{f}(\mathbf{x},y)}{\int \hat{f}(\mathbf{x},y)\, dy} \tag{3.19}$$

Our estimate of y is the expected value:

$$E[y|\mathbf{x}] = \frac{\int y f(\mathbf{x},y)\, dy}{\int \hat{f}(\mathbf{x},y)\, dy} \tag{3.20}$$

Replacing the joint density by its kernel estimate, the Nadaraya–Watson estimator is obtained:

$$\hat{y}(\mathbf{x}) = \frac{\sum_i Y_i K\left(\frac{\mathbf{x}-\mathbf{X}_i}{h}\right)}{\sum_i K\left(\frac{\mathbf{x}-\mathbf{X}_i}{h}\right)} \tag{3.21}$$

Because of its attractive analytical properties [38], the kernel function is generally taken as the Gaussian, called a *Parzen window* [39]:

$$K(\mathbf{u}) = \left(\frac{1}{\sqrt{2\pi}}\right)^d \exp\left(-\frac{\|\mathbf{u}\|}{2}\right) \tag{3.22}$$

This then leads to the following equation for the estimate $\hat{y}(\mathbf{x})$:

$$\hat{y}(\mathbf{x}) = \frac{\sum_{i=1}^{n} Y_i \exp\left(-\frac{\|\mathbf{x}-\mathbf{X}_i\|^2}{2h^2}\right)}{\sum_{i=1}^{n} \exp\left(-\frac{\|\mathbf{x}-\mathbf{X}_i\|^2}{2h^2}\right)} \tag{3.23}$$

The estimate $\hat{y}(\mathbf{x})$ is thus a weighted average of all the observed Y_i values where each weight is exponentially proportional to its Euclidean distance from $\mathbf{x}$. Note that a good choice of the window width h is critical. If it is large, then even distant neighbors affect the estimate at $\mathbf{x}$ leading to a very smooth estimate. In the extreme case when h goes to infinity, our estimate is the average of all Y_i, independent of the input. When h is small, only a few (if any) samples play a role, leading to a noisy estimate.

The "bias/variance dilemma" stated by Geman, Bienenstock, and Doursat [40], points out that when h is large, $\hat{y}(\mathbf{x})$ is an average of many samples and the variance contribution of our estimator is small since our estimator does not change from one training

set to another (of course assuming that they are all drawn from the same distribution). The bias is large in this case as the estimate is biased toward the population response. When h is small, there is small bias but the estimate is dependent on the particular training set used. Therefore the variance contribution is high. Thus choosing h implies a trade-off between bias (systematic error) and variance (random error). h is determined by trial and error, cross-validating on a separate test set. A more detailed analysis of the approach and its properties are beyond the scope of this thesis; the interested reader is referred to [41].

Back-propagation rule is generally criticized on the grounds that learning takes many epochs over the training set and also that the performance depends on a good choice of H, the number of hidden units. The method just explained belongs to the class of *memory-based methods* where approximation is directly done from a table of stored values. These methods require only one epoch over the training set, i.e., to read in the table. Here, $\boldsymbol{\theta}$, the parameter vector, includes the whole training set and the window width h. Thus learning is fast but the memory requirement is large. There are also methods to choose a subset of the training set without decreasing performance; see [42] for a review.

As also noted by Specht who names this a *generalized regression neural network* [43], this approach can also be written in the framework of Equation (3.10). Here H is equal to table size, i.e., the number of training patterns with $\mathbf{W}_h = \mathbf{u}^h$, $T_h = y^h$, the desired output for $\mathbf{u}^h$ and we have:

$$g_h(\mathbf{W}_h, \mathbf{u}) = \frac{\exp\left(-\frac{\|\mathbf{W}_h - \mathbf{u}\|^2}{2h^2}\right)}{\sum_{i=1}^{H} \exp\left(-\frac{\|\mathbf{W}_i - \mathbf{u}\|^2}{2h^2}\right)} \tag{3.24}$$

The problem of finding good input and output codings apply also here.

3.4. Combining Multiple Estimators

There are many choices that should be made before an estimator can be used in an application. These include the model, method of estimating parameters, i.e., learning rule, training sample, input representation, error function minimized, etc. Each choice made adds one additional sort of bias which may not be appropriate. The general approach is to try alternatives and choose the one that best performs on a test sample, distinct from the sample used for training, i.e., cross-validation. In this section it will be argued that it is better not to discard the alternatives but combine all to improve performance.

There are two ways in which multiple estimators can be combined: One is *voting* and the other is *stacking*. Note that regardless of the way the models are combined, there is no reason to expect improvement in performance through having multiple models if they are quite similar. One can only get fault tolerance through redundancy if the models fail under different circumstances. In neural networks, choosing parameters randomly like the initial weight values and hyperparameters like the network architecture or by having different learning rules, one guarantees this to a certain extent. Basically when we have one model that has a certain success, we want to add another model that succeeds best for inputs on which the first one fails; we do not care about the new model's *overall* performance.

As seen in Figure 3.4.1, there are c estimators, $\Phi_j(x_1, x_2|\theta_j)$, each one trained separately. Then during estimation, each one gives an output and these outputs are combined by a *combiner* system to determine the final output r. Two types of combiners are presented here: The *voting* system which computes a weighted sum where weights are fixed and the *stacking* approach where the combiner is also an estimator that is trained.

In this application the following three estimators are combined—parametric, multilayer perceptron, and the regression neural network—using both types of combining. They are sufficiently different in the application of distance estimation.

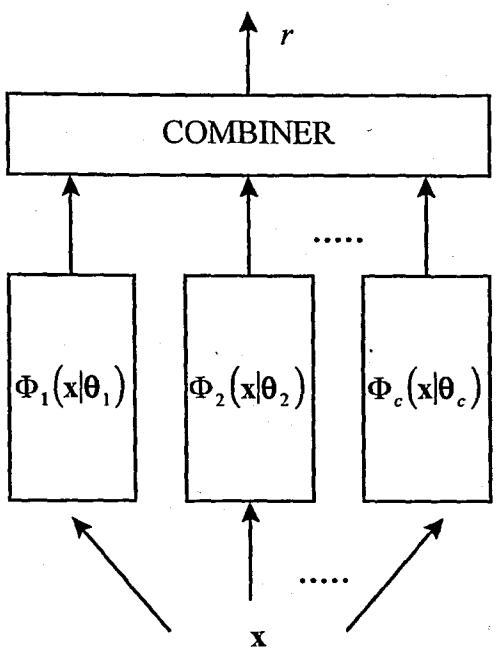


FIGURE 3.4.1. Block diagram of combining multiple estimators.

3.4.1. Voting

In voting with c voters, each estimator $\Phi_j(\mathbf{x}_1, \mathbf{x}_2 | \theta_j)$ gives an output for the input $(\mathbf{x}_1, \mathbf{x}_2)$. The final output, r, is a weighted sum [44]:

$$r = \sum_{j=1}^{c} \Phi_j(\mathbf{x}_1, \mathbf{x}_2 | \theta_j)\, \omega_j \tag{3.25}$$

where $\sum_{j=1}^{c} \omega_j = 1$. Unless there is a prior reason to favor one voter over another, weights are taken as equal: $\omega_j = 1/c$. If we look at the variance of r [45,46]:

$$\begin{aligned} \mathrm{Var}(r) &= \mathrm{Var}\left(\frac{1}{c}\sum_{j=1}^{c} d_j\right) \\ &= \frac{1}{c^2}\left[\sum_j \mathrm{Var}(\Phi_j) + 2\sum_j \sum_{k \neq j} \mathrm{Cov}(\Phi_j, \Phi_k)\right] \end{aligned} \tag{3.26}$$

If the models are statistically independent, the second term cancels and variance (and mean square error) decreases with increasing c. However variance can further be decreased by choosing mutually "orthogonal" methods [47]. For example, instead of having two independent models, one may prefer having two negatively correlated models.

It may be possible to have an a priori preference of one estimator over another. In that case higher weights may be assigned to these estimators in voting. For example estimators may be assessed over a cross-validation set and their performances may be used to determine the weights so as to assign larger weights to more successful voters. Using Occam's razor, it may be a priori assumed that complex models with many free parameters tend to overfit on a small sample. Thus one may have the tendency to give them less weight unless they are more successful than simpler models, i.e., their additional complexity is justified [44]. These latter approaches have not been tested in this work.

3.4.2. Stacking

Combination of multiple estimators can also be considered as an estimation problem. Wolpert who initiated the idea calls it *stacked generalization* [47]. Breiman converted it to statistical language as *stacked regression* [48]. Here we have two levels of estimators. First on level 0, L_0, are our usual estimators chosen carefully to be as different as possible while maintaining individually acceptable levels of performances. On top of these, on level 1, L_1, is another estimator (the combiner) whose inputs are the outputs of the first level estimators.

L_0 estimators are trained separately on the same training sample. Then they are applied to a different set on which they make estimations. The L_1 estimator takes those estimations as input and maps them to the correct value. Thus L_0 and L_1 estimators are trained on separate training sets. This is because the performances of the L_0 estimators on the data that they are trained on can be very different from their behavior on the data that they are *not* trained on; this is exactly what L_1 should know about and be trained on. This is generally done by dividing the training set into two and using one half for training each

level. When the training set is small, one can use leave-k-out (jackknife) to avoid losing precious data. Significant improvement in success has been achieved using this approach on the classification task of protein secondary structure prediction [49].

3.5. Implementation Results

3.5.1. Database

Two distinct samples have been collected by pairing respectively 28 and 23 cities and towns of Türkiye for training and test sets. The first set is used for estimating the parameters while the second one is utilized to assess the performance of these parameters. The training and test data sets contain planar coordinates and intercity distances for 28 and 23 cities, respectively, which make $28 \times 27/2 = 378$ and $23 \times 22/2 = 253$ data pairs. The third dimension is ignored since previous empirical studies have shown that the effect of elevation in the accuracy of the estimators in Türkiye is almost null [14]. The values reported in the following subsections are average error per pair in kilometers on both the training and the test sets. Recall that the error per pair is measured by the normalized error measure given in Equation (2.4).

3.5.2. Distance Functions

As it is mentioned in Section 2.2, the distance functions given in Table 2.2.1 have been employed to estimate actual road distances in Türkiye. The optimization problems formulated by using the normalized and squared error measure (Equation 2.4) with the distance functions $\Phi_1(\mathbf{x}_1, \mathbf{x}_2)$, $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ and $\Phi_5(\mathbf{x}_1, \mathbf{x}_2)$ have analytical solutions. The minimization of the unconstrained optimization problems which are obtained when $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$ and $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ employed are carried out by routines that perform the simplex search method. The results are given in Table 3.5.1. Observe that the rectilinear distance metric has the worst performance. Meanwhile the accuracy of $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ is the highest.

TABLE 3.5.1. Average error per pair using the five parametric distance functions.

Estimator	Parameters	Average Training Error	Average Test Error
$\Phi_1(x_1, x_2)$	$k = 1.073$	8.55	12.49
$\Phi_2(x_1, x_2)$	$k = 1.310$	4.54	9.05
$\Phi_3(x_1, x_2)$	$k = 1.262$ $p = 1.603$	3.92	8.48
$\Phi_4(x_1, x_2)$	$k = 1.688$ $p = 1.686$ $s = 1.761$	3.61	8.10
$\Phi_5(x_1, x_2)$	$\beta_1 = 0.280$ $\beta_2 = 0.971$	3.98	8.49

These facts definitely support the previous inference on the arbitrariness of the road structure in Türkiye.

The parameters of the distance functions should be customized according to the geographical characteristics of the region they are to be used; this has been agreed as a consequence of many empirical studies previously mentioned. In other words, a distance function must be flexible enough to introduce the effect of regional geography on distances into the estimations. Therefore the effect of rotation is also studied for all the distance functions. Although rotation requires the addition of γ as a new parameter to the original parameter vector $\boldsymbol{\theta}$, which is shown in the second column of Table 2.2.1, the method that is used to obtain Table 3.5.1 can be adopted easily. Once γ is fixed, namely axes are rotated for a certain angle, then Equations (2.21), (2.23) and (2.24) can be used to compute parameters k, β_1, and β_2 of the estimators $\Phi_1(x_1, x_2)$ and $\Phi_5(x_1, x_2)$ and the unconstrained optimization problems given with Equations (2.25) and (2.26) can be solved in order to compute the parameters k, p, and s of $\Phi_3(x_1, x_2)$ and $\Phi_4(x_1, x_2)$. Note that $\Phi_2(x_1, x_2)$ is invariant under rotation, and therefore it is not considered in this part of the study. The results are given in Table 3.5.2. They support the findings of Table 3.5.1. The consideration of γ as a new parameter increases the accuracy. The rectilinear distance function has again the worst performance and the accuracy of $\Phi_4(x_1, x_2)$, which is closely followed by the accuracies of $\Phi_3(x_1, x_2)$ and $\Phi_5(x_1, x_2)$, is still the highest.

TABLE 3.5.2. Average error per pair with rotation.

Estimator	Parameters	Average Training Error	Average Test Error
$\Phi_1(\mathbf{x}_1, \mathbf{x}_2)$	$\gamma = 5°$ $k = 1.069$	8.28	12.90
$\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$	$\gamma = 48°$ $k = 1.395$ $p = 2.740$	3.82	8.45
$\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$	$\gamma = 50°$ $k = 1.807$ $p = 1.548$ $s = 1.761$	3.52	8.09
$\Phi_5(\mathbf{x}_1, \mathbf{x}_2)$	$\gamma = 4°$ $k_1 = 0.290$ $k_2 = 0.958$	3.94	8.56

3.5.3. Multilayer Perceptrons

The multilayer perceptron used in this study has one hidden layer and it is trained by the back-propagation learning rule. After several trials, the best input representation is determined as the coordinates of the two points and the Euclidean distance between these points as a hint:

$$\mathbf{u}^i = \phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i\right) = \left(x_{11}, x_{12}, x_{21}, x_{22}, \|\mathbf{x}_1 - \mathbf{x}_2\|\right)^T \tag{3.27}$$

In terms of output representation, it is found out that learning the ratio of the actual distance to the Euclidean distance is better than learning the actual distance itself. This ratio is called the *directional bias* by Brimberg and Love [28] and Brimberg and Wesolowsky [50]:

$$y = \chi(r) = \frac{r}{\|\mathbf{x}_1 - \mathbf{x}_2\|} \tag{3.28}$$

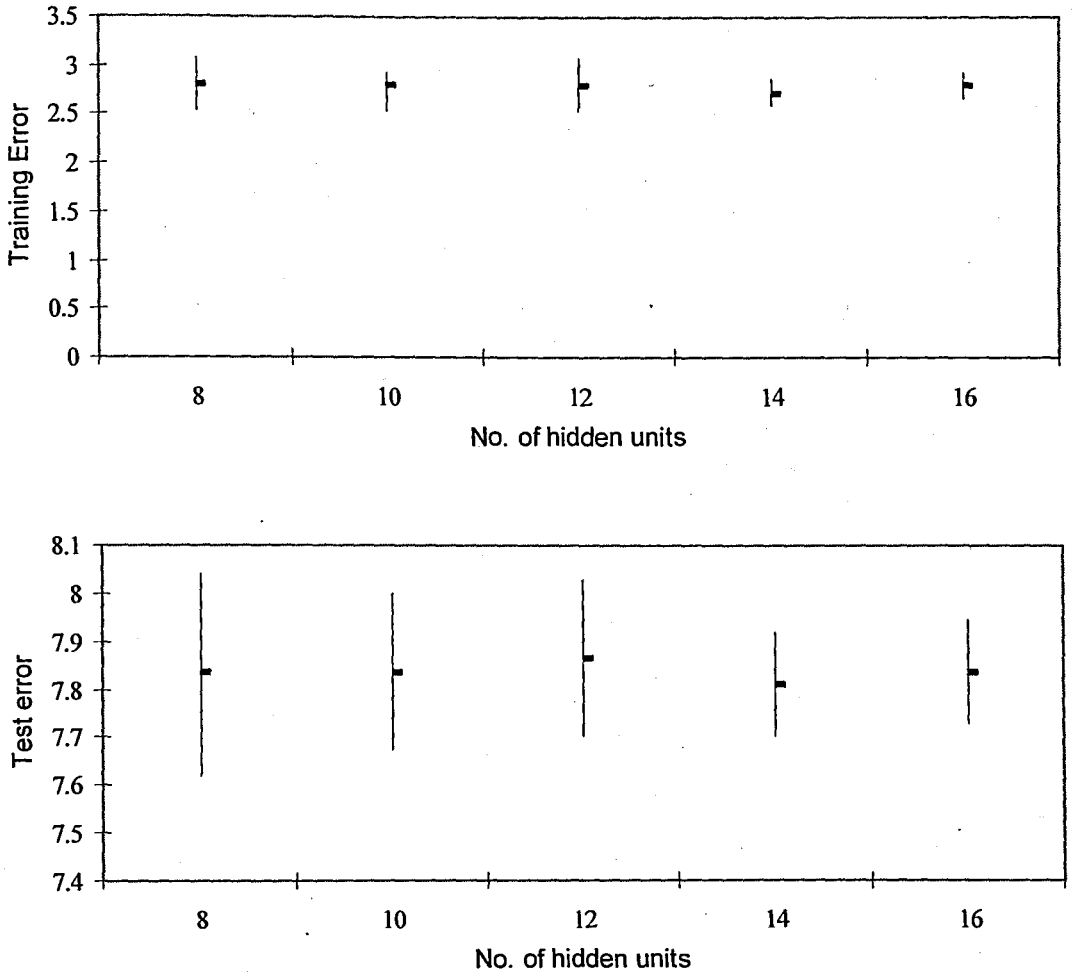


FIGURE 3.5.1. The errors as a function of the number of hidden units.

This may be thought of as extending $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ in the parametric case in that, instead of computing one constant global k factor, it is as if the neural network computes a continuous function $k(\mathbf{x})$ by which it scales the Euclidean distance. Note that it is possible to take advantage of the a priori knowledge that the distances are symmetric, namely, $\Psi(\mathbf{x}_1, \mathbf{x}_2) = \Psi(\mathbf{x}_2, \mathbf{x}_1)$. As a consequence of this the training set can be doubled. This can be done by adding data triples $(\mathbf{x}_2, \mathbf{x}_1, \Psi(\mathbf{x}_2, \mathbf{x}_1))$ to the triples $(\mathbf{x}_1, \mathbf{x}_2, \Psi(\mathbf{x}_1, \mathbf{x}_2))$ in the training set. In Figure 3.5.1 average errors and standard deviations over the training and test sets are given over 10 independent runs as a function of the number of hidden units.

A momentum term is used while the parameter η is updated dynamically for faster convergence. Training stops when no further improvements are made. Networks require

around a hundred epochs for convergence after which slight improvements (less than 5 per cent) are achieved if learning is continued.

3.5.4. Regression Neural Networks

By playing with the alternatives, it is found out that the following input representation is the best:

$$\mathbf{u}^i = \phi\left(\mathbf{x}_1^i, \mathbf{x}_2^i\right) = \left(|x_{11} - x_{21}|, |x_{12} - x_{22}|, \|\mathbf{x}_1 - \mathbf{x}_2\|\right)^T \tag{3.29}$$

Kernel-based estimation methods suffer from the "curse of dimensionality" [39] because the neighborhood information is lost when the dimensionality of the space increases. Thus decreasing the input dimensions from five to three helps. The output is the desired distance:

$$y = \chi(r) = r \tag{3.30}$$

Results achieved are given in Figure 3.5.2 as a function of the window width, h. Note that there is a large dependence on h. Overall, the success is slightly inferior to that of the multilayer perceptron.

3.5.5. Combining Estimators

Taking into account the fact that combining multiple estimators is a better idea when estimators are as different as possible, the following three estimators are selected for combination:

1. Parameterized distance function, $\Phi_4\left(\mathbf{x}_1, \mathbf{x}_2\right)$ with three parameters k, p, and s.
2. Multilayer perceptron with 12 hidden units.

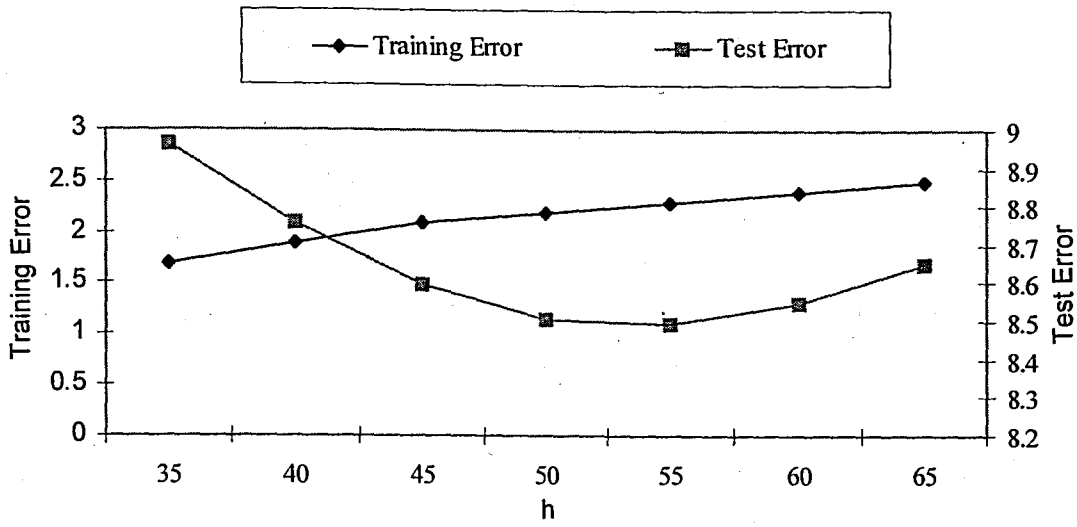


FIGURE 3.5.2. The errors as a function of the window width, h.

3. Regression neural network with $h = 54$.

The votes are taken as equal. Results achieved are given in Table 3.5.3. Note that the result of voting is better than the result of all voters. This indicates that different estimators perform better under different circumstances.

While applying stacking again three estimators are used as our L_0 estimators:

1. Parameterized distance function, $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ with the parameter k.
2. Multilayer perceptron with 12 hidden units.
3. Regression neural network with $h = 54$.

The combiner estimator L_1 is a multilayer perceptron with 2 hidden units. The training set is divided into two parts performing "leave-$n/2$-out"; T_0 and T_1. All three L_0 estimators are trained using T_0 only and saved their predictions on T_1 as R_1. Then the estimators are trained on T_1 and their predictions are saved on T_0 as R_0. The combiner network L_1 is trained subsequently on R_0 and R_1. After training the combiner network, all three estimators are trained on the complete training set. Stacking is superior to voting as is

TABLE 3.5.3. Average error of three estimators and the result of voting.

Estimator	Average Training Error	Average Test Error
Parametric model	3.61	8.10
Multilayer perceptron	2.63	7.91
Regression neural network	2.38	8.49
Voting	2.26	7.63

also shown in the study. Values reported are average errors on both the training and test sets (see Table 3.5.4).

3.6. Conclusions

The approaches can be compared based on the three criteria of accuracy, memory requirement, and time.

3.6.1. Accuracy

It seems that the multilayer perceptron can be a better estimator than the parameterized distance functions in terms of estimation accuracy. This is because in the latter models, the parameters are fixed over the whole space. For example when $\Phi_2(x_1, x_2)$ is used, k is constant over the whole country but it is known that the eastern part is more mountainous than the western part and thus k there is higher than k of the west as a consequence of the extra curvature caused by these natural obstacles. The multilayer perceptron may be seen as

TABLE 3.5.4. Average error achieved through voting vs. stacking.

Combiner	Training Error	Test Error
Voting	2.26	7.63
Stacking	3.81	7.41

approximating an input dependent, continuous function $k(\mathbf{x})$. Being nonparametric, it does this without assuming any a priori form. An approach in distance function based estimation is to divide the space into multiple regions and make separate estimations in separate regions; this is a piecewise constant approximation of the $k(\mathbf{x})$ [19].

Though the regression neural network and the parameterized distance function do not perform as well as the multilayer perceptron, when they are combined, their accuracy is higher. This indicates that on cases where the perceptron fails, the others make good guesses. The stacking approach performs better than voting.

3.6.2. Memory Requirement

Out of the three estimators implemented, the regression neural network has the highest memory requirement by storing $4n + 1$ parameters: 4 values for each pair for a training set of n pairs and the parameter h, here $4 \times 378 + 1 = 1513$. When H is the number of hidden units, the multilayer perceptron requires $6H$ (5 inputs and one bias) values for $\mathbf{W}$ and $H+1$ for $\mathbf{T}$, thus total of $7H+1$ weights, here 85. Memory requirement increases when one has multiple models. In the stacking approach, there are also parameters of the combiner network. However, the parameterized estimators considered in this work respectively have 1, 1, 2, and 3 parameters in addition to the rotating angle γ. Therefore, they use memory very efficiently.

3.6.3. Learning Time

The regression neural network is trained in one epoch. Three of the first parameterized distance functions has an analytical solution where we compute k, k_1, and k_2 by making one pass over the training set. The computational effort spent for training the multilayer perceptron is considerably higher than that is required to find the parameters of the estimators $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$ and $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ since many trials should be made with the multilayer perceptron to find out the good input and output representations and the number of hidden

units. To train the stacking network, when one divides the set into two, the estimators are trained twice. Ideally, "leave-one-out" where at each step, the training set is divided into two parts of size one and $n-1$ is better for bias reduction but it requires n trainings. This is the approach Wolpert originally proposed for stacked generalization [47]. In short the effort required to compute the parameters of distance functions is very small comparing to the training effort of neural networks.

It is generally accepted that using this or that learning technique by itself is not sufficient to train a neural network appropriately. A good knowledge of the application and the learning methods are necessary to guide the estimator to learn the important and ignore the irrelevant. As shown in the preceding sections, different methods may benefit from different input and output representations. Perhaps the quality of the training sample is the most important factor that affects the quality of estimation. For example, a large enough and representative sample is required for the approximation not to be biased by the idiosyncrasies of the particular sample used.

Viewing the results achieved, we can say that nonparametric approaches can be considered better than parameterized estimators in terms of estimation accuracy. This is not just due to having more parameters. The fact that both training *and* test errors is lower implies that the nonparametric methods are able to extract some underlying structure which cannot be captured completely by the parametric approaches. The nonparametric method does not simply memorize the training set but generalizes accurately to unseen patterns in the test set. We believe that this is due to the fact that the nonparametric estimator can adjust itself to follow the change in the geographical characteristics of a region as opposed to the parameterized distance function estimating over the whole country. On the other hand, the advantage of the parameterized methods is that because they have a small number of parameters, they require much less memory and smaller training samples, making them suitable for applications where memory and learning time is limited.

Parametric distance estimators are often incorporated into the objective functions of many optimization models, e.g. the objective function of continuous space facility

location problems. This is also possible for neural network estimators. The optimization of the objective function calculated via a neural network can, in principle, be undertaken by standard optimization algorithms. Analytic formulae for the gradient and Hessian of the function on the network are available, for example for the case of multilayer perceptrons. The major drawback of these functions compared to distance functions, is their being nonconvex, which is not the case for the parametric estimators we study in this work—at least under certain restrictions on the parameters k, p, s, β_1, and β_2 [13,20]. Convexity with respect to the coordinates is an important property because the effort necessary to spend on the solution of the optimization models decreases drastically if the objective function to be minimized is a convex function and the solution space is a convex set.

The estimator determined via a neural network is very suitable for parallel computations, which may provide considerable advantages especially when many different optimization calculations have to be undertaken.

In short there is not one method that is significantly superior to others in all four respects of accuracy, memory requirement, learning time, and the advantages it provides when incorporated into objective functions of optimization models. Thus when choosing a particular estimator, all these aspects should be taken into account according to the particular implementation constraints and not only accuracy as it is frequently done in the literature. Finally, it is recommended that multiple estimators be used and combined to get more accurate estimators.

4. CLUSTERING AND VECTOR QUANTIZATION

4.1. Competitive Learning and Vector Quantization

As mentioned in the previous chapter, one application area of neural networks is clustering. This is accomplished through competitive learning. A number of output units each representing a cluster compete for being fired. Therefore, the output units in these networks are called winner-take-all units. The requirement is that similar input data vectors should fire the same output unit. In other words, the output units tend to fire for different domains of the input variables. Such networks can be used for data encoding or data compression where an input data vector is replaced by the index of the output unit that it fires.

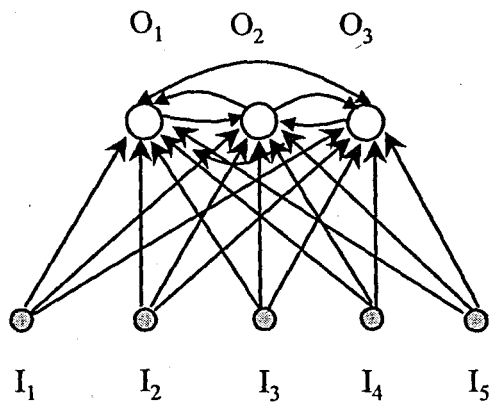


FIGURE 4.1.1. A competitive learning network.

The structure of a competitive neural network is given in Figure 4.1.1. Each output unit i, computes the distance between the input vector $\mathbf{x}$, and its weight vector $\mathbf{w}_i$ and the one with the closest distance gets fired while other output units remain inactive. Namely,

$$O_c = \begin{cases} 1 & \text{if } d(\mathbf{w}_c, \mathbf{x}) = \min_i \{d(\mathbf{w}_i, \mathbf{x})\} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

where c is the winner unit. The distance measure $d(\cdot)$ is usually taken as the Euclidean distance, i.e., $d(\mathbf{w}_i, \mathbf{x}) = \|\mathbf{w}_i - \mathbf{x}\|$. The winner unit O_c may also be selected as the one with the largest net input, i.e., by computing the inner (scalar or dot) product of the current input vector $\mathbf{x}$, and the weight vector $\mathbf{w}_i$:

$$O_c = \begin{cases} 1 & \text{if } \mathbf{w}_c \cdot \mathbf{x} = \max_i \{\mathbf{w}_i \cdot \mathbf{x}\} = \max_i \left\{ \sum_j w_{ij} x_j \right\} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

If the weights for each unit are normalized (e.g., $|\mathbf{w}_i| = 1$ for all i), then both measures become equivalent meaning that the winner is the unit with normalized weight vector $\mathbf{w}$ closest to the input vector $\mathbf{x}$.

The error of such a reconstruction is equal to the difference between input and output units over all inputs:

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{2} \sum_p \sum_i m_i(\mathbf{x}^p) \|\mathbf{w}_i - \mathbf{x}^p\|^2 \\ &= \frac{1}{2} \sum_p \sum_i m_i(\mathbf{x}^p) \sum_j \left(w_{ij} - x_j^p\right)^2 \end{aligned} \tag{4.3}$$

where $m_i(\mathbf{x}^p) = 1$ if $\mathbf{w}_i$ is the winner for input pattern $\mathbf{x}^p$:

$$m_i(\mathbf{x}^p) = \begin{cases} 1 & \text{if } \|\mathbf{w}_i - \mathbf{x}^p\| = \min_k \{\|\mathbf{w}_k - \mathbf{x}^p\|\} \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

Using gradient descent as the learning algorithm we obtain:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial W_{ij}} = \eta m_i \left(\mathbf{x}^p\right) \left(x_j^p - w_{ij}\right). \tag{4.5}$$

The foundational ideas motivating Vector Quantization (VQ) are classical concepts that have been applied in the estimation of probability density functions. Traditionally, (in the realms of both statistical analysis and statistical pattern recognition) distributions have been represented either parametrically or nonparametrically. In the former, the user generally assumes the form of the distribution function and the parameters of the function are learned using the available data points. In pattern recognition (classification), these estimated distributions are subsequently utilized to generate the discriminant hyperspheres (or hyperellipsoids) whence the classification is achieved.

In nonparametric methods, the practitioner assumes that the data must be processed in its entirety (and not just by using a functional form to represent the data). The corresponding pattern recognition (classification) algorithms which result are generally of the nearest neighbor (or k-nearest neighbor) philosophy and are thus computationally expensive. The comparison of these two perspectives is found in standard pattern recognition textbooks [39,51], and bounds on the classification error rate of nonparametric strategies (as compared to the optimal Bayesian parametric strategies) have also been derived.

The concept of VQ can be perceived as a compromise between the above two schools of thought. Rather than representing the entire data in a compressed form using only the estimates (and in the estimate domain), VQ opts to represent the data in the actual feature space. However, as opposed to the nonparametric methods which use all the data, VQ produces an approximation to a continuous probability density function $p\left(\mathbf{x}\right)$ of the input variable $\mathbf{x}$ using a finite number of prototype or reference or *codebook* vectors $\{\boldsymbol{\mu}_i : i = 1, \ldots, n\}$ which define the clusters or classes. In other words, a d-dimensional vector is mapped onto another real-valued d-dimensional vector $\boldsymbol{\mu}$. Once these codebook vectors are determined, the approximation of $\mathbf{x}$ involves finding the closest reference vector

μ_c to $\mathbf{x}$ using the Euclidean distance. This divides up the input space into a Voronoi tesselation (see Figure 4.1.2). The question that remains is finding the codebook vectors.

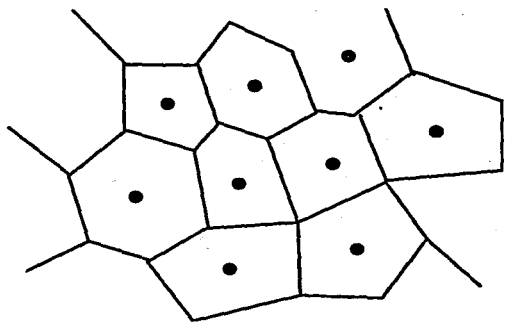


FIGURE 4.1.2. Voronoi tesselation.

One kind of optimal placement of the μ_i's minimizes the expected rth power of the reconstruction error where the index c of the best-matching codebook vector ("winner") is a function of the input vector $\mathbf{x}$:

$$c = \underset{i}{\operatorname{argmin}} \{\|\mathbf{x} - \mu_i\|\} \tag{4.6}$$

No closed-form solution exists for the optimal placement of the μ_i's and iterative approximation procedures have to be used. When the square-error criterion is used ($r = 2$), it can be shown that the codebook vectors converge to the optimal values asymptotically upon the application of the following iterative scheme which is simply the steepest-descent optimization of E:

$$\begin{aligned} \mu_c(t+1) &= \mu_c(t) + \alpha(t)(\mathbf{x}(t) - \mu_c(t)) \\ \mu_i(t+1) &= \mu_i(t) \quad i \neq c \end{aligned} \tag{4.7}$$

Here, $\mu_c = \mu_c(t)$ is the closest codebook vector to $\mathbf{x} = \mathbf{x}(t)$ when the Euclidean distance is used where t is the discrete-time index. $\alpha(t)$ assumes monotonically decreasing scalar

values with $0 < \alpha(t) < 1$. It should be noted that this procedure produces exactly the same sequence of codebook vectors as the competitive learning.

The dissimilarity between $\mathbf{x}$ and μ_i's may not be measured by the Euclidean distance. Therefore, when a general distance function $d(\mathbf{x}, \mu_i)$ is used, the "winner" μ_c is identified such that

$$c = \underset{i}{\operatorname{argmin}} \{d(\mathbf{x}, \mu_i)\} \tag{4.8}$$

For this general distance function an update rule should be devised such that $d(\mathbf{x}, \mu_c)$ decreases monotonically, i.e., the following should be satisfied:

$$\nabla_{\mu_i}^T d(\mathbf{x}, \mu_i) \cdot \delta\mu_i < 0 \tag{4.9}$$

4.2. Learning Vector Quantization

If the input vectors are to be classified into a finite number of categories (classes or clusters), then several codebook vectors are usually made to represent each class where their identity within the classes is not critical. Merely decisions made at the class borders become important. The aim is then to locate the codebook vectors such that they directly define near-optimal class boundaries among the classes. The learning algorithms that accomplish this are called Learning Vector Quantization (LVQ). In other words, LVQ attempts to design codebook vectors which can discriminate the classes instead of minimizing an error functional such as mean-square error (the case $r = 2$). There are three types of Learning Vector Quantization each of which is described below.

4.2.1. Type One Learning Vector Quantization (LVQ1)

When several codebook vectors μ_i are assigned to each class and each of them is labeled with the corresponding class index, the class regions in the x-space are defined by the nearest-neighbor comparison of x with μ_i. Then, the label of the closest (in terms of the chosen distance function) μ_i determines the classification of x.

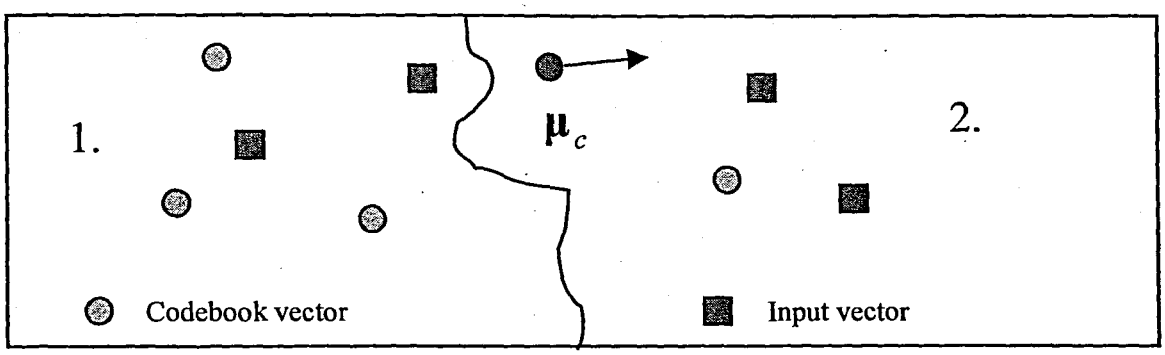


FIGURE 4.2.1. Movement of the codebook vector in LVQ1 as per Equation (4.10).

The optimal placement of the μ_i are found by updating the μ_i according to the following algorithm where the initial values for them may be set by a number of methods such as Vector Quantization or randomly.

$$\begin{aligned} \mu_c(t+1) &= \mu_c(t) + \alpha(t)[\mathbf{x}(t) - \mu_c(t)] && \text{if x is classified correctly} \\ \mu_c(t+1) &= \mu_c(t) - \alpha(t)[\mathbf{x}(t) - \mu_c(t)] && \text{if x is classified incorrectly} \\ \mu_i(t+1) &= \mu_i(t) && i \neq c \end{aligned} \tag{4.10}$$

The rationale of the algorithm is to pull codebook vectors away from the class boundaries so that the borders are identified more accurately (see Figure 4.2.1). Furthermore, x is said to be correctly classified if the class to which x belongs is the same as the label of the closest codebook vector. $\alpha(t)$ is defined as before (see Equation 4.7). Since this is a fine-tuning method, the initial value of $\alpha(t)$ should be fairly small, e.g., 0.01 or 0.02, and this must be decreased to zero in, say, 100,000 steps.

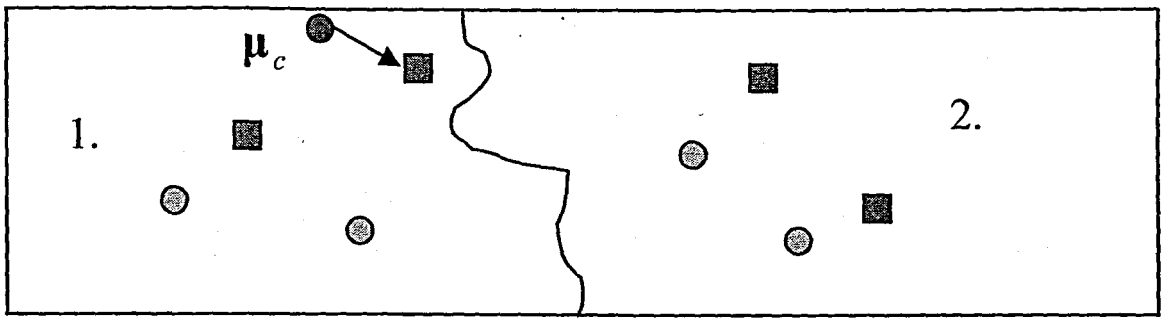


FIGURE 4.2.2. Movement of the codebook vector in LVQ1 as per Equation (4.10).

4.2.2. Type Two Learning Vector Quantization (LVQ2)

LVQ1 can be modified in the following manner. Assume that two codebook vectors μ_i and μ_j that belong to different classes and are closest neighbors (the nearest codebook to μ_i is μ_j and vice versa) are initially in a wrong position. The discrimination boundary (which is incorrect because of the wrong positions of the two codebooks) is always defined as the midplane of μ_i and μ_j. A symmetric window of nonzero width is defined around the midplane and updates to μ_i and μ_j are only made if x falls within this window on the wrong side of the midplane. LVQ2 algorithms is as follows:

$$\begin{aligned} \mu_j(t+1) &= \mu_j + \alpha(t)\left[\mathbf{x}(t) - \mu_j(t)\right] && \text{if } \mu_i \text{ is the nearest codebook,} \\ \mu_i(t+1) &= \mu_i(t) - \alpha(t)\left[\mathbf{x}(t) - \mu_i(t)\right] && \text{but } \mathbf{x} \in C_j \neq C_i \text{ where } \mu_j \text{ is} \\ & && \text{the second nearest codebook} \\ \mu_k(t+1) &= \mu_k(t) && \text{otherwise} \end{aligned} \tag{4.11}$$

This is illustrated in Figure 4.2.3.

The optimal width ω of the window must be determined experimentally. When the number of training samples is small, selecting a width of 10 or 20 per cent of the difference between μ_i and μ_j seems to be appropriate.

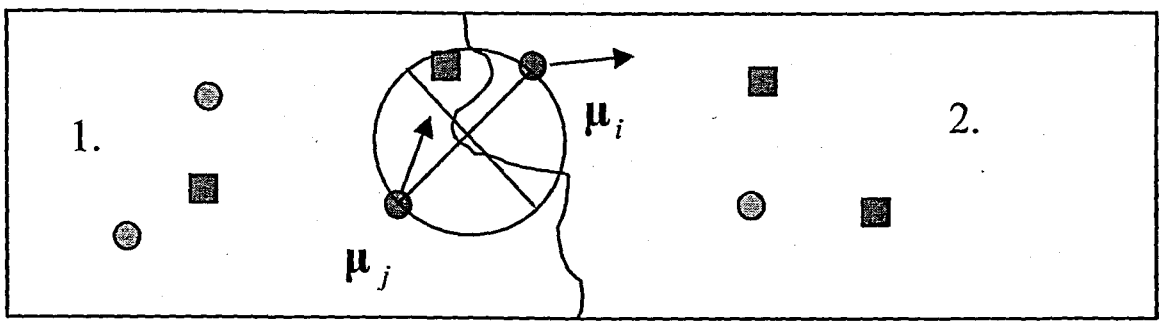


FIGURE 4.2.3. Movement of the codebook vector in LVQ2 as per Equation (4.11).

4.2.3. Type Three Learning Vector Quantization (LVQ3)

The LVQ2 algorithm is based on the idea that the decision boundaries are shifted towards the Bayesian limits. However, no attention was paid to the location of μ_i as the process evolves over time. There are two different kinds of problem associated with LVQ2. The first problem is that the distance between μ_i and μ_j, i.e., $\|\mu_i - \mu_j\|$ decreases continuously since the amount of updates are proportional to the difference between x and μ_i or x and μ_j while the update on μ_j (correct class) is larger in magnitude that on μ_i (wrong class). In order to overcome this problem, all the training vectors are accepted from the window with the condition that one of the μ_i and μ_j must belong to the correct class and the other to the incorrect one. The second problem is due to the possibility that another asymptotic equilibrium of μ_i may come out upon the application of Equation (4.11) which is not optimal anymore. Therefore the update formulas of LVQ3 include corrections which ensure that the μ_i roughly approximate the class distributions (see Figure 4.2.4).

$$\begin{aligned}
&\mu_j(t+1) = \mu_j + \alpha(t)\left[\mathrm{x}(t) - \mu_j(t)\right] && \text{if } \mu_i \text{ is the nearest codebook,}\\
&\mu_i(t+1) = \mu_i(t) - \alpha(t)\left[\mathrm{x}(t) - \mu_i(t)\right] && \text{but } \mathrm{x} \in C_j \neq C_i \text{where } \mu_j \text{ is}\\
& && \text{the second nearest codebook}\\
&\mu_k(t+1) = \mu_k(t) + \epsilon\alpha(t)\left[\mathrm{x}(t) - \mu_k(t)\right] && \text{for } k \in \{i,j\} \text{ and if x, } \mu_i,\\
& && \text{and } \mu_j \text{ belong to the same}\\
& && \text{class } j
\end{aligned} \tag{4.12}$$

Experimentally, it has been found that values for ϵ range between 0.1 and 0.5. The optimal value of ϵ depends on the size of the window and it decreases as the width of the window gets narrower.

It should be noted that only one codebook vector is updated at LVQ1 whereas two codebook vectors are modified at LVQ2 and LVQ3. The details of LVQ can be found in [52] and in an excellent survey by Kohonen [53].

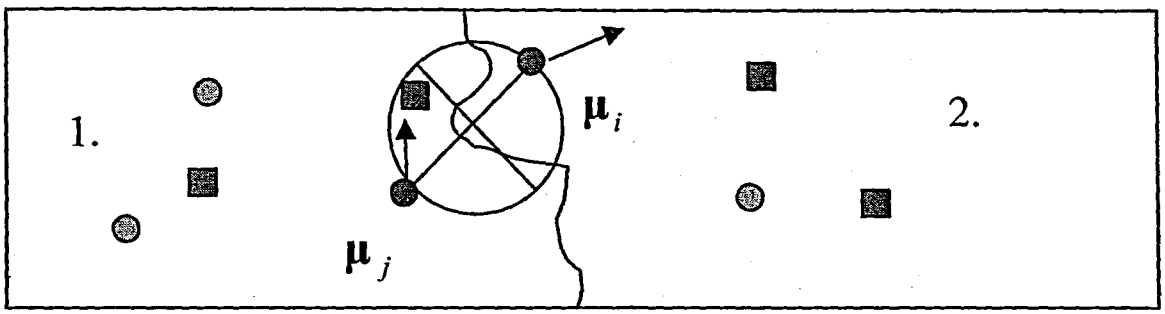


FIGURE 4.2.4. Movement of the codebook vector in LVQ3 as per Equation (4.12).

4.3. LVQ and Distance Estimation

The principles of the LVQ lend themselves naturally to the domain of arbitrary distance function estimation. Since the aim in this problem is to estimate the actual road distances between cities and towns in a geographical region, it is possible to approximate the unknown distance function by an adaptive multiregional approach. This is equivalent to approximating the unknown function by a "patchwork" (lattice) of intraregional and interregional explicit subfunctions. In all of the early works subregions are selected *a priori* based on subjective judgements and not subsequently modified [19, 54]. However, in this method, although the subregions are selected in an a priori manner (whether randomly or otherwise), the a posteriori partitions are obtained adaptively using an LVQ method. This imposes an implicit discriminant mapping on the domain. Subsequently, the arbitrary distance function is subclassified as a set of intraset and interset distance functions each of

them being characterized using their own respective parameters. The training sites and their corresponding available coordinates and interdistances are then used to train the intra and interset parameters.

The algorithm is divided into three main steps or phases. The first phase is referred to as the *Intraregional Polarizing* phase where the information in each region (cities and towns) is compressed by using codebook vectors. In a multi-class problem the codebook vectors for each region are subsequently migrated so as to ensure that they adequately represent their own regions and furthermore distinguish between the other regions. This phase, which is referred to as the *Interregional Polarizing* phase, also implicitly learns the discriminant function to be used in a subsequent classification module. Note that these discriminant functions are of a nearest neighbor philosophy, except that the nearest neighbors are drawn from the set of codebook vectors (as opposed to the entire set of training samples). They thus drastically reduce the computational burden incurred in the testing of traditional nonparametric methods.

4.3.1. Intraregional Polarizing

It is assumed that we are to estimate the actual road distance r between any two points or nodes P_j and P_m in the set of points $\mathcal{G}$. Let $\Psi(\mathbf{x}_j, \mathbf{x}_m)$ be the function which gives the actual distance r where $\mathbf{x}_j$ and $\mathbf{x}_m$ are the two-dimensional coordinates of P_j and P_m, respectively. Another assumption is that we are given the training set $\mathcal{L}$ which is a subset of $\mathcal{G}$ and the internode distances for the nodes in $\mathcal{L}$ (i.e., $\{\Psi(P_j, P_m) : P_j, P_m \in \mathcal{L}\}$).

The basic hypothesis in distance estimation using a multiregional approach is that $\mathcal{G}$ can be partitioned into a set of smaller regions as a result of which intraregional and interregional approximates of Ψ can be obtained. Thus, in the training phase we partition $\mathcal{L}$ into R subsets, $C_k = \{P_{k,i} : 1 \leq i \leq N_k\}$ $(1 \leq k \leq R)$ each containing N_k points ($P_{k,i}$ denotes the ith point in the kth region).

The primary aim is to represent each C_k by M representative points $\{Q_{k,j} : 1 \leq j \leq M\}$ $(M \ll N_k)$. Although strictly speaking, the set C_k can be represented by M_k points (where M_k increases with N_k), in the interest of simplicity, it is assumed that the number of representative codebook vectors for all the classes is the same. The set of codebook vectors $\{Q_{k,j} : 1 \leq j \leq M\}$ are initially assigned random positions within or close to their respective regions. In the intraregional polarizing the algorithm is repeatedly presented with a node $P_{k,i}$ from C_k. The closest codebook vector, $Q_{k,j}$, to $P_{k,i}$ is determined and this vector is moved in the direction of the data point $P_{k,i}$. Indeed, this is achieved by rendering the new $Q_{k,j}$ to be a convex combination of the current $Q_{k,j}$ and the data point $P_{k,i}$. More explicitly, the updating algorithm is as follows:

$$Q_{k,j}(t+1) = \begin{cases} (1-\alpha(t))\, Q_{k,j}(t) + \alpha(t)\, P_{k,i} & \text{if } Q_{k,j} \text{ is the closest codebook} \\ & \text{vector to the data point } P_{k,i} \\ Q_{k,j}(t) & \text{otherwise} \end{cases} \tag{4.13}$$

where t is the discretized (synchronized) time index. Note that this is the traditional LVQ strategy [52,53,55–57] except that the radius of the "bubble" of interest used by Kohonen is restricted to be unity. The reasons for this are two-fold:

1. Since we are attempting to represent the nodes in C_k by a set of representative codebook vectors, the *topological ordering* of these codebook vectors is absolutely irrelevant. This, in turn, makes the algorithm computationally extremely inexpensive, because, at each time it is only needed to locate the nearest codebook vector and do not have to find all the codebook vectors within the bubble of activity.

2. In a typical application, the number of codebook vectors must be kept extremely small. This is because, we want to partition $\mathcal{G}$ into R subregions and thus, effectively, we are attempting to approximate the function Ψ using $(M \times R + 1) \cdot (M \times R)/2$ "patched" functions. If each of these functions has 3 parameters, the number of parameters to be

estimated becomes prohibitively large. Thus, when there are four regions ($R = 4$) and each region contains three codebook vectors ($M = 3$), the number of parameters is 234. If each region is represented by four codebook vectors, the number of parameters involved becomes 408. Indeed, considering the extremely small values of M encountered in this application domain, ($M = 3$ per region) rendering the radius of the bubble of activity to be unity is far from unreasonable. Furthermore, as mentioned above, it only increases the rate of convergence of the scheme.

In the update equation above, $\alpha(t)$ is decremented linearly from unity for the initial learning phase and then switched to small values of α which decrease linearly from 0.2 for the fine-tuning phase. This is as recommended in the literature [52, 53].

4.3.2. Interregional Polarizing

After the individual regions have been represented by M codebook vectors using the above migration strategy, the codebook vectors are tested using $\mathcal{L}$ to see whether they adequately classify the points within their respective subregions. To achieve this, an algorithm analogous, in principle, to the LVQ3 algorithm suggested by Kohonen [53] is adopted. Every data point in the training set, $\mathcal{L}$ (not just in the individual clusters, C_k), is tested against the set of $\{Q_{k,j} : 1 \leq j \leq M_k\}$ to see whether their nearest codebook vector falls within their partition. Thus, unlike the previous phase, where the codebook vectors found their respective places by learning only from the location of the data points within their own respective classes, in this phase, these representative vectors are migrated so that they polarize away from the data points of the other competing clusters. The principle by which this is done is as follows:

Suppose that a point $P \in C_k$ is examined. Also assume that the two closest codebook vectors to P (among all the codebook vectors) are Q_a and Q_b. If both Q_a and Q_b do not belong to the cluster C_k, clearly, the information content in P (with respect to Q_a and Q_b) is misleading, and so it is futile to migrate Q_a and Q_b using this information. However, if both

of them are intended to represent C_k, clearly, the information in P can be used to achieve an even finer tuning to their locations. Thus, in this scenario, both Q_a and Q_b are moved marginally from their current locations along the line towards P. The final scenario is the case when one of them, Q_a (Q_b), correctly belongs to C_k, and the other, Q_b (Q_a), belongs to a different partition. In this case, the information in P can be used to achieve an even finer tuning to their locations by migrating Q_a (Q_b) marginally from its current location along the hyperline towards P, and migrating the other codebook vector Q_b (Q_a) marginally from its current locations along the line away from P.

Since we do not want the "straggler" points (the points which are misclassified, but which probably would not have been correctly classified even by an optimal classifier) to completely dictate (and thus, disturb) the polarizing, this migration is invoked only if the node P lies within a pre-specified window of interest, W. This restriction has also been recommended in the literature [52, 53], and typically, this window, W, is a hypersphere centered at the bisector between the codebook vectors Q_a and Q_b. Also, as recommended in the literature, the polarizing of both Q_a and Q_b (when both of them correctly classify P) is made to be of much smaller magnitude than in the scenario when either of them misclassifies it. These steps are formally given below:

If Q_a and Q_b are the two closest codebook representative vectors to a given point $P \in C_k$,

$$
\begin{aligned}
Q_a(t+1) &= (1-\epsilon\alpha)\,Q_a(t) + \epsilon\alpha P && \text{if } Q_a, Q_b \in C_k;\ P \in W \\
Q_b(t+1) &= (1-\epsilon\alpha)\,Q_b(t) + \epsilon\alpha P && \text{if } Q_a, Q_b \in C_k;\ P \in W \\
Q_a(t+1) &= (1-\alpha)\,Q_a(t) + \alpha P && \text{if } Q_a \in C_k;\ Q_b \in C_j \neq C_k;\ P \in W \\
Q_b(t+1) &= (1+\alpha)\,Q_a(t) - \alpha P && \text{if } Q_a \in C_k;\ Q_b \in C_j \neq C_k;\ P \in W \\
Q_a(t+1) &= (1+\alpha)\,Q_a(t) - \alpha P && \text{if } Q_a \in C_j \neq C_k;\ Q_b \in C_k;\ P \in W \\
Q_b(t+1) &= (1-\alpha)\,Q_a(t) + \alpha P && \text{if } Q_a \in C_j \neq C_k;\ Q_b \in C_k;\ P \in W \\
Q_a(t+1) &= Q_a(t) && \text{if } P \notin W \\
Q_b(t+1) &= Q_b(t) && \text{if } P \notin W
\end{aligned}
\tag{4.14}
$$

In the update equations above, α is maintained at a constant value of 0.1 (as opposed to varying it as recommended in [52, 53]), and ϵ is set equal to be 0.25. The value of the diameter of W is one-hundredth of the distance between Q_a and Q_b.

It should be observed that after the intraregional and interregional polarizing, the representative codebook vectors impose a set of piecewise linear boundaries which assign the original nodes, $\mathcal{L}$, into potentially slightly different regions than that which was initially assigned. Thus, although the initial demarcation boundaries may have been incorrectly assigned, the sequence of polarizing operations tends to re-allocate them. The effect of this boundary re-allocation will be discussed in greater detail in the section describing the experimental results.

After the training points have been re-allocated and the set of codebook vectors for each cluster has been fixed, the patchwork of functions approximating the arbitrary distance function Ψ is now learned. This can be done using either an independent optimizing strategy or a LVQ scheme. It is now demonstrated how these are achieved.

4.3.3. Parameter Learning Using LVQ

After the individual regions have been represented by the various codebook vectors (using the above migration strategies), it is possible to estimate Ψ. The basic assumption in this phase, is that Ψ can be approximated by a patchwork (or lattice) of intraregional and interregional functions. In this phase, these respective approximating functions are learned using the codebook vectors and the given distances $\{\Psi(P_i, P_j) : P_i, P_j \in \mathcal{L}\}$.

Let P_i and P_j be any two nodes in $\mathcal{G}$. Obviously, if only the points in $\mathcal{G}$ are of interest, Ψ can be approximated (indeed, exactly represented) in terms of all the internode Euclidean

norms as follows:

$$\Psi(P_i, P_j) = k_{i,j} \|P_i - P_j\| \tag{4.15}$$

Since Ψ is symmetric, only roughly half of these coefficients will have to be estimated. Clearly, such a representation defeats the fundamental purpose of a distance estimation strategy, for it would necessitate the learning of all the $\{k_{i,j} : 1 \leq i \leq j \leq N\}$ coefficients. The intention is to approximate the actual distance between P_i and P_j by hypothesizing that the constants $\{k_{i,j}\}$ are dependent on the representative codebook vectors. Thus, rather than specify $\Psi(P_i, P_j)$ using Equation (4.15) above, it is assumed that $\Psi(P_i, P_j)$ can be reasonably approximated by locating the closest codebook vectors for P_i and P_j and evaluating a simple function between these respective points. Thus, $\Psi(P_i, P_j)$ is approximated as follows:

$$\Psi(P_i, P_j) \cong k_{a,b} \|P_i - P_j\| \tag{4.16}$$

where, closest codebook to P_i and P_j are Q_a and Q_b respectively. The problem that remains to be solved is now determining the set of parameterizing constants $\{k_{a,b} : 1 \leq a \leq b \leq R \times M\}$.

There are at least three distinct schemes for evaluating the above set of parameterizing coefficients, $\{k_{a,b}\}$ using the training set, $\mathcal{L}$ and their corresponding true distances. The first is by a simple averaging strategy. For every pair of nodes in the training set, a cumulative sum of the ratio of their true distance to their Euclidean distance is maintained. This ratio is called the *directional bias* by Brimberg and Love [28] and Brimberg and Wesolowsky [50]. This sum is associated with the pair of closest codebook vectors. The cumulative sum divided by the number of pairs represented by these codebook vectors yields the average value of $k_{a,b}$ for the codebook vectors Q_a and Q_b.

An alternate strategy to obtain the set of parameterizing coefficients is to perform a VQ learning algorithm in the space involving the coefficients themselves. Let us suppose that we have a current value of $k_{a,b}$. When a new pair of points in $\mathcal{L}$ is examined, if the closest codebook vectors are Q_a and Q_b respectively, the updated value of $k_{a,b}$ is obtained by moving the current value towards the value estimated using just this set of points. This update is performed along the line joining the points. This is formally described below:

Input: The set of codebook vectors, the training set $\mathcal{L}$ and the distances $\Psi(P_i, P_j)$ for all $P_i, P_j \in \mathcal{L}$.

Output: The set of parameterizing coefficients, $\{k_{a,b} : 1 \leq a \leq b \leq R \times M\}$

Begin

For $a = b$ to $R \times M$ **Do**

$\lambda_{a,b} = 1$

$k_{a,b} = 0$

End For

Repeat until satisfied

Get any distinct pair of points $P_i, P_j \in \mathcal{L}$

If closest codebook vectors to P_i and P_j are Q_a and Q_b respectively then

$k_{a,b} = (1 - \lambda_{a,b})\, k_{a,b} + \lambda_{a,b}\, (\Psi(P_i, P_j) \,/\, \|P_i - P_j\|)$

Decrease $\lambda_{a,b}$

End If

EndRepeat

End

It is easy to see that if $\lambda_{a,b}$ decreases inversely with the number of samples encountered (which have Q_a and Q_b as the codebook vectors) the above VQ strategy converges exactly to the average value of $k_{a,b}$ computed earlier. Any other updating method for $\lambda_{a,b}$ would converge to an alternate (hopefully, closer-to-optimal) value of the $k_{a,b}$. In all the experiments, an inverse decreasing function for $k_{a,b}$ is used. Indeed, in this setting, the results tend to show that, (as opposed to the speech recognition example discussed in [53]) we now have a scenario when an all-neural approach ([53], pp. 82) is recommendable.

The third approach involves explicit optimization. Here, each internode distance is specified by a function which is completely defined by the closest codebook vectors, and whose functional form is one of those types tabulated in Table 4.4.1. The question of estimation is now reduced to one of optimization as has been done in the literature [13, 20, 58]. Here, Kohonen's recommendation of using a traditional scheme subsequent to a neural strategy has proven to be superior.

4.4. Implementation Results

4.4.1. Database

Two distinct samples have been collected by pairing respectively 80 and 23 cities and towns of Türkiye for training and test sets. The first set is used for estimating the parameters and the second one to assess their performances. The training and test data sets contain planar coordinates and intercity distances for 80 and 23 cities respectively which make 3160 and 253 data pairs. The third dimension is ignored since previous empirical studies have shown that the effect of elevation in the accuracy of the estimators in Türkiye is almost null [14]. The reported values in the following subsections are average error per pair in kilometers on both the training and the test sets. Recall that the error per pair is measured by the normalized error measure given in Equation (2.4).

4.4.2. Distance Functions

As it has been pointed out before, some distance functions are eliminated because of the road network structure of Türkiye. The distance functions which are used in constructing the estimators $\Phi_1(\mathbf{x}_1, \mathbf{x}_2)$, $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$, $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$, and $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ of Table 2.2.1 are $d_4(\mathbf{x}_1, \mathbf{x}_2)$, $d_2(\mathbf{x}_1, \mathbf{x}_2)$, $d_7(\mathbf{x}_1, \mathbf{x}_2)$, and $d_8(\mathbf{x}_1, \mathbf{x}_2)$, respectively. The calculation of the parameters are obtained by solving the problems given in Equations (2.21), (2.22), (2.25), and (2.26). These problems are solved analytically for $\Phi_1(\mathbf{x}_1, \mathbf{x}_2)$ and $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$. One

TABLE 4.4.1. Average error per pair.

Estimator	Parameters	Training Error	Test Error
$\Phi_1(x_1, x_2)$	$k = 1.082$	8.02	12.40
$\Phi_2(x_1, x_2)$	$k = 1.320$	3.65	8.41
$\Phi_3(x_1, x_2)$	$k = 1.286$ $p = 1.688$	3.36	8.94
$\Phi_4(x_1, x_2)$	$k = 1.600$ $p = 1.769$ $s = 1.829$	3.18	8.14

of the numerical optimization techniques—simplex search method—has been employed for problems associated with $\Phi_3(x_1, x_2)$ and $\Phi_4(x_1, x_2)$. The results are given in Table 4.4.1.

4.4.3. Learning Vector Quantization

To demonstrate the power of the LVQ strategy numerous experiments were done involving initial random and non-random partitions of Türkiye. These experimental results demonstrate the strengths of the scheme. In all the implementations of the algorithms Türkiye is first divided into various geographical subregions. The initial partitioning into regions was done either randomly or fairly arbitrarily (in a non-random manner) so as to ensure that each subregion has an equal number of towns or cities. In each case a plot of the Turkish towns used for training is shown in the respective figures where the latitude and longitude of the bounding rectangle are 36°N, 26°E, 42°N and 45°E. In each of the figures the towns themselves are marked with an '×'. Note that the actual map of Türkiye has not been superimposed in the figure to avoid cluttering it. The twelve squares, '□' represent the final positions of the codebook vectors. Consider Figure 4.4.1.

In this case the initial partitioning has four subregions and is achieved "manually" but in an arbitrary manner. The subregions divides the country into four rectangles each containing 20 towns in the training set $\mathcal{L}$. To demonstrate the power of the strategy, an LVQ strategy is used with only three codebook vectors in each region which are initialized to be on the border of their representative regions. During the intraregional polarizing phase the

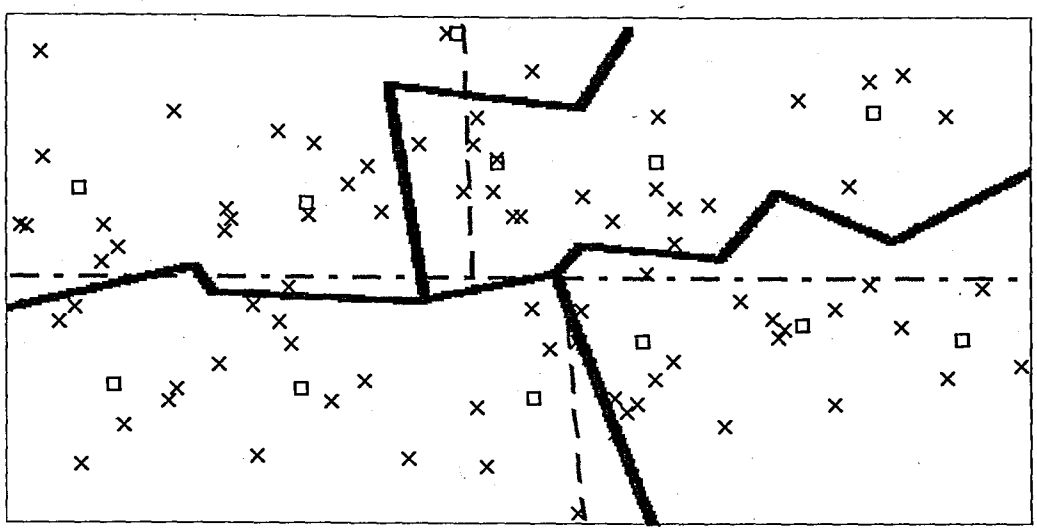


FIGURE 4.4.1. The initial partitions are generated manually.

LVQ algorithm is invoked with a value of α which starts at unity and decreases linearly to 0.9 in 1,000 iterations. As expected, most of the learning is accomplished in this phase. Thereafter, in the fine tuning phase the value of α was drastically switched to 0.2 and decreases linearly to attain to 0.1 in 2,000 time steps. In the interregional polarizing phase, the value of α is maintained at 0.1. The constant for the migration of codebook vectors from the same class, ϵ is maintained at 0.25 and done for 2000 iterations (i.e., 25 cycles of all the 80 training sites) Indeed, the entire convergence for both these phases takes only a matter of a couple of seconds. As mentioned earlier, the final partitioning (after the codebook vectors converge) is fully determined by the discriminant function implicitly created by the bisectors of the lines joining the codebook vectors. This partitioning is adaptively learned and is shown in Figure 4.4.1 in bold lines. After the intra and interregional learning the constants for the underlying patchwork functions are estimated using the training sites and their corresponding recorded distances. The estimates computed are of two sorts. First of all, to show the power of the multi-regional approach, it is assumed that the distances within each region and the distances between the regions are each characterized by a single functional form. Thus, since $\mathcal{G}$ is partitioned into four sets, a functional form of the type $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ involves 10 constants. These constants are estimated by both a simple averaging scheme and an LVQ method as explained in Section 4.3. When the explicit form of each intra and interregional function is of the types $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$ and $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ (where the parameters to be estimated are $\{k, p\}$ and $\{k, p, s\}$), the total number of parameters to be estimated becomes

TABLE 4.4.2. Average error in the case of single functional form for each subregion.

Estimator	Training Error	Test Error
$\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ and simple average	3.34	7.83
$\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ and optimization	3.12	8.05
$\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$ and optimization	2.93	7.80
$\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ and optimization	2.77	7.60

20 and 30, respectively. In the latter two cases, the optimization is done independently, and this is typically more time consuming because it involves invoking separate optimization procedures as mentioned previously. The results which are obtained are quite remarkable and are tabulated in Table 4.4.2, where the average training and testing errors are recorded. For example, when the functional form is assumed to be of type $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$, the average error obtained by averaging k (which is exactly the error obtained by an LVQ algorithm in the k-space) is 7.83. This decreases to 7.80 and 7.60 for the cases when the parameters are $\{k, p\}$ and $\{k, p, s\}$ respectively.

The full power of the multi-regional approach is clearly displayed if the distance function is patched using a separate subfunction for each codebook vector and between each codebook vector. In this case, since $\mathcal{G}$ is partitioned into four sets with three codebook vectors in each region, 78 explicit subfunctions are involved when $\Phi_2(\mathbf{x}_1, \mathbf{x}_2)$ is used as the distance function. When the form of each intra and interregional function are of the types $\Phi_3(\mathbf{x}_1, \mathbf{x}_2)$ and $\Phi_4(\mathbf{x}_1, \mathbf{x}_2)$ (where the parameters to be estimated are $\{k, p\}$ and $\{k, p, s\}$), the total number of parameters to be estimated becomes 156 and 234, respectively. Again, as in the above, the latter two cases involve an independent nonlinear optimization. The results obtained are given in Table 4.4.3.

In the most conservative case, the testing error is only 7.69, and in the case when the functions are characterized by $\{k, p, s\}$ the test error goes as low as 7.12. Note that the most time-consuming phase of the learning is the optimization stage. But since this is done only once (during the training phase) this is by no means excessive and may be regarded as preprocessing computation. Subsequently, in the testing phase, the estimation

TABLE 4.4.3. Average error in the case of a separate subfunction for each codebook vector.

Estimator	Training Error	Test Error
$\Phi_2(x_1, x_2)$ and simple average	2.42	7.69
$\Phi_2(x_1, x_2)$ and optimization	2.36	7.90
$\Phi_3(x_1, x_2)$ and optimization	2.11	7.48
$\Phi_4(x_1, x_2)$ and optimization	1.93	7.12

of the distance between any two points merely involves computing the Euclidean distance between them and invoking computation of the functional form associated with their nearest codebook vectors. From Figure 4.4.1, it can be observed that the final regional boundaries do not differ "significantly" from the original "arbitrary" ones. The difference between the two sets of boundaries would have been a lot more accentuated if the initial boundaries had been more randomly generated. To demonstrate this results are reported for a case when the initial quadrilaterals are randomly generated. To achieve this the bounding rectangle of Türkiye is divided into four "random" quadrilaterals. This is accomplished by generating a random point on each of the four edges of the bounding rectangle. The lines joining the points on the opposite sides of the bounding rectangle are now used to constitute the four quadrilaterals. As in the previous case the initial random partitions are shown in Figure 4.4.2. In each subregion the number of codebook vectors was three. An LVQ strategy with only these codebook vectors in each region was invoked with values of α, ϵ and the number of cycles being as in the above experiment. As in Figure 4.4.1, the final partitioning (after the codebook vectors converged) was fully determined by the bisector discriminant functions and is also shown in Figure 4.4.2 in bold lines. The power of the method is clear because even though a random partitioning is used, the final partitioning yields reasonably good results. Indeed, after the intra and interregional learning, the constants for the underlying patchwork functions are estimated using the training sites and their corresponding recorded distances as in the above case. In the interest of brevity only the error is reported when the explicit form of each intra and interregional function is of the type $\Phi_4(x_1, x_2)$ when the distance function is "patched" using a separate subfunction for each codebook vector and between each codebook vector. As in the previous case these characterizing constants are computed by separate optimization procedures. In this case the training error turns out to be 1.787 and the testing error becomes as low as 7.189—which is far superior to all the previously

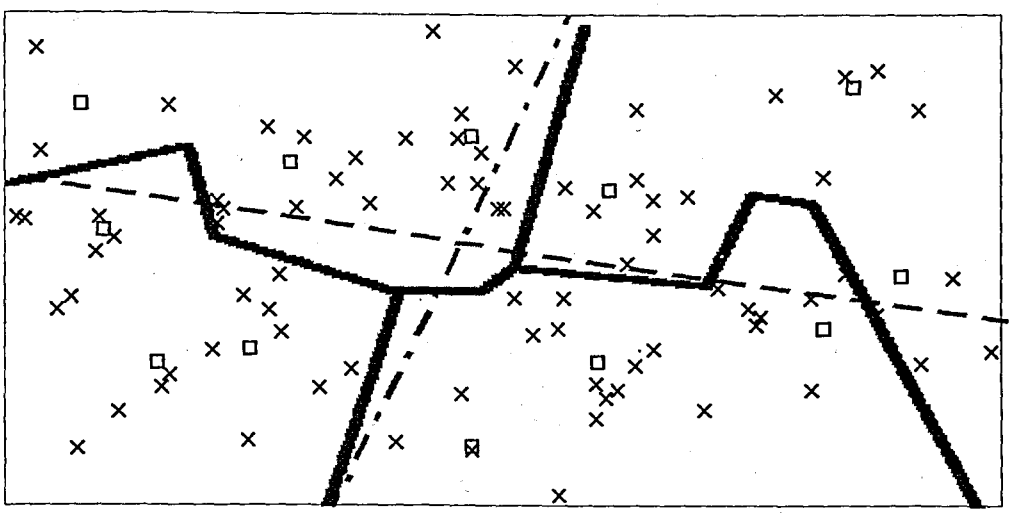


FIGURE 4.4.2. The initial partitions are generated randomly.

reported methods. Note that in the nonrandom case cited above the corresponding errors are 1.93 and 7.12, respectively.Although it is known that the initial boundaries influence the final ones, the way by which they influence them is still unknown. In general, although the results for the random case are so promising, it may be argued that it is disadvantageous to partition the cities in a completely random way, because it would defeat the very purpose of partitioning which attempts to take advantage of the geographical proximity between cities in the various subpartitions.

From the results of the above tables it is also clear that the error decreases as more parameters are used to approximate the distance function. Indeed, this is not surprising because, in the limiting case, one could (conceptually, at least) consider all the points to be singleton set partitions. Clearly, the designer must now strike a happy medium between the use of many subpartitions and number of parameters used in the computation.

To study the effect of the number of partitions a scenario is considered in which a design decision requires 12 codebook vectors. Apart from the experiments already conducted one additional scenario is also considered where the cities are subpartitioned into two regions with six codebook vectors per region.

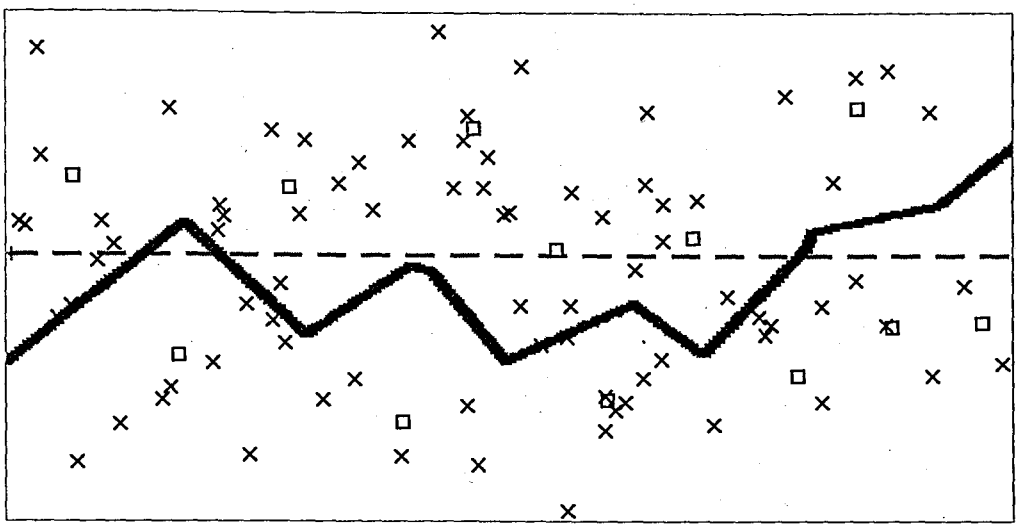


FIGURE 4.4.3. The initial partitions are generated manually.

As in the previous cases the initial partitions are shown in Figure 4.4.3. Notice that the bounding rectangle is now divided into two horizontal subpartitions with six codebook vectors in each subregion. It is clear that the philosophy of the partitions influences the results. Two vertical subpartitions with six codebook vectors in each did not perform so well. As in the above case the LVQ strategy with these codebook vectors in each region is invoked with values of α, ϵ and the number of cycles just as in the above experiments. The final partitioning (after the codebook vectors converged) is shown in Figure 4.4.3 in bold lines. Again, after the intra and interregional learning are done, the constants for the underlying patchwork functions are estimated using the training sites and their corresponding recorded distances. Again, when the explicit form of each intra and interregional function is of the type $\Phi_4(x_1, x_2)$, and when the distance function is patched using a separate subfunction for each codebook vector and between each codebook vector the results obtained are very good. After invoking the nonlinear optimization procedures to compute the characterizing constants, the training and test errors are evaluated. In this case the training error is 2.014 and the test error is as low as 7.173, which is again far superior to all the previously reported methods.

To demonstrate the effect of explicitly using "intelligent" information in the initial partitioning another experiment is conducted in which the initial subpartitions are not

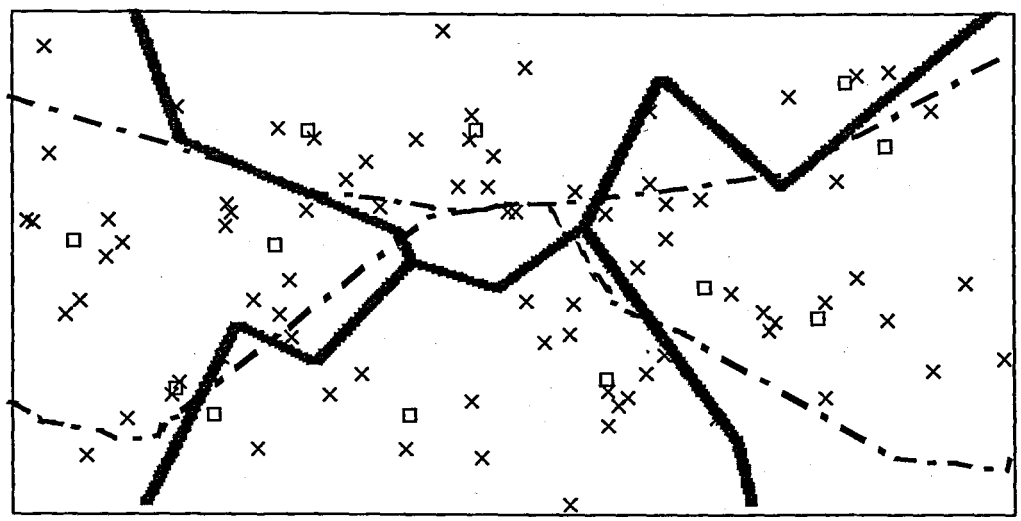


FIGURE 4.4.4. The initial partitions are generated manually using intelligent geographical information for the initial boundaries.

quadrilaterals but manually constructed as in Figure 4.4.4. In other words, it is manually attempted to take full advantage of the geographical proximity between cities in the various initial subpartitions. As in the previous cases the final partitioning (after the codebook vectors converged) is shown in bold lines. Again, after the intra and interregional learning are done, the constants for the underlying patchwork functions are estimated using the training sites and their corresponding recorded distances. Here the case is considered when the explicit form of each intra and interregional function is of the type $\Phi_4(x_1, x_2)$, and when the distance function is patched using separate subfunctions. As in the above cases the results obtained are extremely good. Indeed, as can be expected, they are the best results so far. In this case the training error is 1.941 and the test error is only 6.972. Clearly, it is very advantageous to explicitly use geographical proximity information in the initial partitioning.

4.5. Conclusions

In this chapter the problem of estimating arbitrary distance functions is studied using the concepts of learning vector quantization. This problem has received much attention in Logistics and Location Analysis. The assumptions made on the arbitrary distance function, Ψ are quite relaxed : The set of internode distances dictated by Ψ may or may not satisfy

all the rigorous properties of a well-defined metric. Furthermore, the triangular inequality may also be violated. However, to keep the informal concepts of a distance measure valid, the requirement that Ψ is loosely related to the Euclidean norm is imposed. Hence, if P_i, P_j, P_m, and P_n, are any four nodes, and if the pairs (P_i, P_j) and (P_m, P_n) are close to each other, the respective arbitrary distances between (P_i, P_m) and (P_j, P_n) must be correspondingly of similar magnitude. Also, it is assumed that the explicit form of this distance function is both unknown and uncomputable. Unlike traditional operations research methods, which use parametric distance estimators, learning vector quantization principles are utilized to first adaptively polarize the nodes into subregions. Subsequently, the parameters characterizing the subregions are learned using by a variety of methods including a distinct learning vector quantization strategy in the (meta) parameter-domain.

The salient feature of this chapter is that it explains a novel method for estimating distances which is the "adaptive" version of the multi-regional approach to the distance estimation. The regions are learned adaptively using discriminant functions derived implicitly from the codebook vectors. Subsequent to the partitioning, the actual parameters of the intraregional and interregional functions can be obtained either by optimization (in a non-all-neural approach, for example when k, p and s are parameters) or by using a learning vector quantization algorithm in this parameter space itself. This is also novel, because the problem lends itself to both philosophies of learning.

5. DISCRETIZED LEARNING VECTOR QUANTIZATION

5.1. Motivation

There are currently many vastly different areas of research involving adaptive learning. Two of them are neural networks and learning automata. The competitive neural network of Kohonen which uses the principles of Learning Vector Quantization [52] has been proposed as a fundamental model for neural computing. It has also been used extensively in hundreds of applications. Conversely, the field of learning automata has demonstrated the power of working in a discretized space [59–62] when interacting with a random environment. In this chapter a method is developed by which the general philosophies of these families can be incorporated so as to yield enhanced algorithms for distance estimation problem.

The study of the families of Learning Automata (LA) was developed by Tsetlin [63]. His intention was to model biological learning using a stochastic finite state machine interacting with a random environment. The LA selects an action from a finite set of possible actions. Feedback from the environment tells the LA whether the chosen action was rewarded or penalized. The LA uses this information to decide which action to take next, and the cycle repeats itself. Learning automata and their applications have been reviewed by Lakshmivarahan [64], and by Narendra and Thathachar [65]. Learning automata are useful whenever complete knowledge about a stochastic environment is unknown, expensive to obtain or impossible to quantify. Thus they have found applications in various fields including game playing [65], pattern recognition [65], and object partitioning [66]. Learning automata are also useful when the characteristics of the environment with which they interact change during operation, and are thus useful in priority assignments in a queuing system, and the routing of telephone calls [65].

In the previous chapter it was demonstrated that the principles of LVQ could naturally and powerfully be utilized to solve the arbitrary distance estimation problem. The proposed

solution was a sequence of pattern recognition and polarizing modules governed by the laws of LVQ. In this chapter it is shown that by merging the learning principles of two *families* of adaptive algorithms an enhanced superior learning algorithm can be achieved. Indeed, this is done by having the LVQ operate in a discretized space, as will be clarified below. The new strategy is referred to as Discretized Learning Vector Quantization (DLVQ).

5.2. Learning Automata, and Discretized Vector Quantization

5.2.1. Learning Automata

For variable structure stochastic automata (VSSA) the learning process is generalized so that the state transition probabilities and the action selecting probabilities evolve with time [65]. The automaton is simplified in the sense that each state now corresponds uniquely to a particular action. This means that while in state s_i the automaton always picks one action α_i from a finite set $\mathcal{A}$ that consists of r actions. Thus VSSA has the following components: the set of inputs (one of which serves as the input to the automaton at any time instant), the set of actions (or output from the automaton), and a learning algorithm T. The learning algorithm operates on a probability vector $\mathbf{P}(t)$ whose ith component $p_i(t)$ is the probability that the automaton will select action α_i at time t, with the components summing to unity. Indeed, if $\mathcal{B}$ is the set of inputs and $\mathcal{A}$ the set of actions, the learning algorithm is completely defined by a function T such that $T(\mathbf{P}(t), \mathcal{A}(t), \mathcal{B}(t)) = \mathbf{P}(t+1)$. Many varieties of absorbing and ergodic VSSA have been documented [64, 65]. In both cases they can be made to converge to the optimal action with a probability as close to unity as desired.

5.2.2. Discretized Learning Automata

The beauty of a discrete learning algorithm is that it does not ignore the limitations of practical implementations; on the contrary this limitation is used to its advantage. VSSA evolved from fixed structure stochastic automata as an attempt to simplify the analysis of the automata's properties [64, 65]. However, VSSA have a limitation. Implicit in the definition

of VSSA is the fact that the probability of choosing an action can be any real number in the interval [0,1]. Rendering this probability space discrete is a general approach for improving VSSA [59–62]; this is implemented by restricting the probability of choosing an action to only finitely many values from the interval [0,1]. Consequently, probability changes are made not continuously but in jumps. In a sense, the discrete VSSA represent a hybrid of a fixed structure automaton and VSSA. Discrete automata consist of finite sets like Finite Structure Stochastic Automata, but they are VSSA because they are characterized by a probability vector which evolves with time. Discrete algorithms are linear if the probability values are equally spaced in the interval [0,1]; otherwise, they are called nonlinear [59]. Existing literature [59, 60, 62] uses the term "discretized" in front of the name of a learning automaton to indicate the discrete version of a continuous VSSA. The history of discretized automata (which ignore and use estimates) and the various reported families and their asymptotic properties are catalogued in [60, 62]. Probably the biggest limitation of learning automata is their slow rate of convergence [64, 65]. By limiting the number of assumptions that learning automata have about the environment, they are a general approach for machine learning. However, this also means that there are fewer properties that can be used to speed up the rate of convergence. Originally the intent of introducing discrete learning automata was to increase the rate of convergence and to eliminate the assumption that the random number generator could generate real numbers with arbitrary precision [59, 60, 62]. Once the optimal action has been determined, and the probability of selecting that action is close to unity, discrete automata increase this probability directly, rather than approach the value unity asymptotically. Indeed, by making the probability space discrete, a minimum step size is obtained. If the automaton is close to an end state, the minimum step size forces it to this state with just a few more favorable responses.

The central issue from a theoretical point of view is that the properties of a Markov process can change if the probability of choosing an action is restricted to a finite subset of [0,1]. For example, a continuous space will have recurrent states, but a finite space will only have positive recurrent states [65]. As well, discrete Markov processes have properties that are not true for general Markov processes. Round off error will cause the automaton that approaches its endpoint asymptotically to artificially reach its end point [59, 60, 62]. Also, the proofs of convergence in continuous spaces may not be applicable

to a finite state machine. This point is demonstrated by the fact that, the existing proofs of convergence for discrete algorithms are significantly different from the proofs of their continuous counterparts (compare [59, 60, 62] to the methods used in [64, 65]).

Another benefit of discretizing the probability of choosing an action is that it reduces the requirements on the system's random number generators. This is important since VSSA use a random number generator in its implementation [64, 65]. In theory, it is assumed that any real value in [0,1] can be obtained from the machine; in practice, only a finite number of these values are available.

The final benefit is an issue related with implementation and representation. Discrete versions lead, quite naturally, to the use of integers for keeping track of how many multiples of the resolution parameter the action probabilities are. While the above consideration frequently increases the rate of convergence measured in terms of the number of iterations, a discrete algorithm also has the benefit of reducing the time measured in terms of the clock cycles that a microprocessor would take to do each iteration of the task. It also reduces the amount of memory needed. Typically addition is quicker than multiplication on a digital computer, and the amount of memory used for a floating point number is usually more than that required for an integer. In the schemes that have been discretized so far, whereas the continuous versions update their probability vectors via multiplication, the discretized counterparts achieve this with addition and subtraction. Thus, in terms of both time and space, discrete algorithms seem to be superior.

5.2.3. Analog to Digital Conversion

The entire concept of discretization can be perceived as a way by which the output of the machine is constrained to take a value which is on a finite grid whose resolution is J_i on the ith axis. Effectively, this means that we resort to "Analog-to-Digital" (*A_To_D*) converters each of which has as its input a real value, and which yields as its output an integer value in $[0, ..., J_i]$ which is the discretized "grid" coordinate of the point concerned

in the respective direction. Thus, central to the process of discretization is the function "*A_To_D*" which achieves this. The function has parameters which are the size of the grids on the axes, say, $GridSize$. The function itself has as its input a point P on the Cartesian plane with coordinates $P = (x_1, x_2)^T$. It yields as its output the discretized version of the point, $P^d = (x_1^d, x_2^d)^T$ where the components are integers and given by:

$$\begin{aligned} x_1^d &= Round(x_1/GridSize) \\ x_2^d &= Round(x_2/GridSize) \end{aligned} \tag{5.1}$$

The following sections explain how the intra and interregional polarizing concepts of LVQ are extended to the discretized domain using the analog-to-digital conversion described above.

5.2.4. Intraregional Polarizing

The concepts described in Section 4.3.1 can be extended to a discretized philosophy. To discretize things, in the training phase, all the points in the training set $\mathcal{L}$ are projected onto the grid by repeatedly invoking *A_To_D* on them. Thus, we have R subsets of *discretized* training partitions C_k^d:

$$C_k^d = \{A\_To\_D\,(P_{k,i}) : 1 \leq i \leq N_k\} \quad (1 \leq k \leq R) \tag{5.2}$$

Here $P_{k,i}$ denotes the ith point in the kth region. The aim is again to represent each C_k by M codebook vectors which are also discretized:

$$\{Q_{k,j}^d : 1 \leq j \leq M\} \quad M \ll N_k \tag{5.3}$$

The set of codebook vectors $Q^d_{k,j}$ are initially assigned random positions within or close to their respective regions, but they are constrained to be on the grid themselves by invoking *A_To_D* to their random real representations. Thus, in the intraregional polarizing the algorithm is repeatedly presented with a node $P^d_{k,i}$ from C^d_k. The closest codebook vector to $P^d_{k,i}$, $Q^d_{k,j}$, is determined and this vector is moved in the direction of this data point. Since the space that is worked on is a discretized space, it should be clarified what by the "closest" codebook vector is meant. Indeed, the more fundamental question is one of determining how distances will be measured in this space [53]. Since the primary intention in working in a discretized space is to work with integers (and to minimize real computations) the distance used in this case is what is called the "Discretized Euclidean" (E^d) which approximates the distance between pixels points if traversed along the pixel directions in a two-dimensional pixel array. Let us now consider how we can move from pixel P^d_1 to P^d_2. If $P^d_1 = (x^d_{11}, x^d_{12})^T$ and $P^d_2 = (x^d_{21}, x^d_{22})^T$ are two discretized points on the grid, then the number of pixels to be traversed in the two directions is the difference of their respective coordinates given by $Diff_1$ and $Diff_2$ as:

$$\begin{aligned} Diff_1 &= \left|x^d_{11} - x^d_{21}\right| \\ Diff_2 &= \left|x^d_{12} - x^d_{22}\right| \end{aligned} \tag{5.4}$$

The minimum number of diagonal pixel traversal which has to be done is given by $DiagMoves$, where

$$DiagMoves = \min\{Diff_1, Diff_2\} \tag{5.5}$$

Once these diagonal moves have been achieved, the linear moves to be done to go from P^d_1 to P^d_2 is given by $LinearMoves$, where

$$LinearMoves = Max\{Diff_1 - DiagMoves\,,\; Diff_2 - DiagMoves\} \tag{5.6}$$

Thus, the discretized Euclidean distance (E^d) between P_1^d and P_2^d is:

$$E^d(P_1^d, P_2^d) = \sqrt{2} \cdot DiagMoves + LinearMoves. \tag{5.7}$$

This is illustrated in Figure 5.2.1.

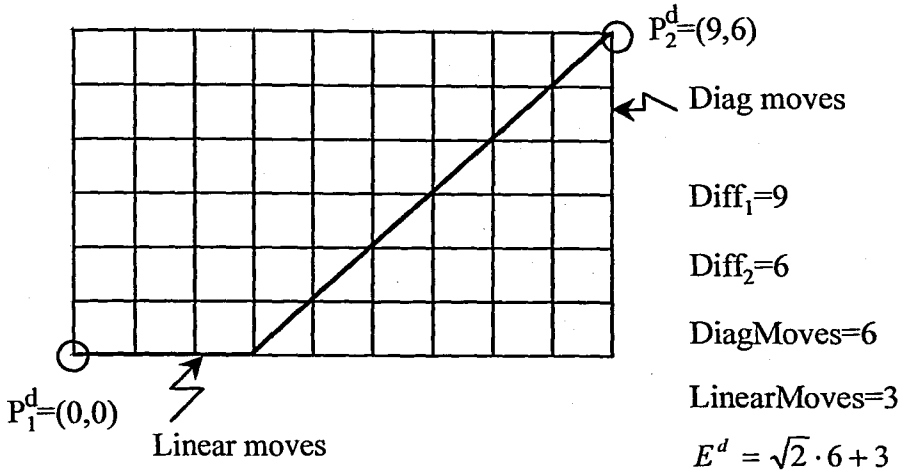


FIGURE 5.2.1. The discretized Euclidean distance.

It is clear that the computation of the discretized Euclidean distance (E^d) requires *only five* integer additions (subtractions), two comparisons and a single multiplication. Since the distance function is a linear combination of the distances traversed in the individual dimensions, minimizing in each direction would minimize the overall distance. Thus, arguing as in [53], the codebook vectors can be updated by setting $Q_{k,j}^d$ to be a convex combination of the current $Q_{k,j}^d$ and the discretized data point $P_{k,i}^d$ and projecting back onto the discretized space after computation. Consequently, the new updating algorithm is as

follows:

$$Q_{k,j}^d(t+1) = \begin{cases} A\_To\_D & \text{if } Q_{k,j}^d \text{ is the closest point} \\ \left((1-\alpha(t))Q_{k,j}^d(t) + \alpha(t)\, P_{k,i}^d\right) & \text{to the data point } P_{k,i}^d \\ Q_{k,j}^d(t) & \text{otherwise} \end{cases} \tag{5.8}$$

Note that this can be seen to be the discretized version of the traditional LVQ strategy. In the above equation, α is decremented linearly from unity for the initial learning phase and then switched to small values which decrease linearly from 0.2 for the fine-tuning phase. This is as recommended in the literature [52, 53] and has been justified in the continuous domain [67]. The next step is interregional polarizing.

5.2.5. Interregional Polarizing

After the individual regions have been represented by a subset of M codebook vectors using the above migration strategy, the codebook vectors are tested using $\mathcal{L}$ to see whether they adequately classify the points within their respective subregions. The steps performed in this phase are formally given in Section 4.4 for the continuous case. In the discretized world the modifications are done by performing all the migrations on the grid with the additional constraint that "distances" and "nearest neighbors" are evaluated using the discretized Euclidean distance $E^d(\cdot)$. Thus the modified interregional polarizing equations that are the counterparts of the equations given for the continuous case (Equation 4.14 on page 63) become as follows. If Q_a^d and Q_b^d are the two closest codebook vectors to a given point $P^d \in C_k^d$:

$$
\begin{aligned}
Q_a^d(t+1) &= (1-\epsilon\alpha)\,Q_a^d(t) + \epsilon\alpha P^d && \text{if } Q_a^d, Q_b^d \in C_k^d;\ P^d \in W \\
Q_b^d(t+1) &= (1-\epsilon\alpha)\,Q_b^d(t) + \epsilon\alpha P^d && \text{if } Q_a^d, Q_b^d \in C_k^d;\ P^d \in W \\
Q_a^d(t+1) &= (1-\alpha)\,Q_a^d(t) + \alpha P^d && \text{if } Q_a^d \in C_k^d;\ Q_b^d \in C_j^d \neq C_k^d;\ P^d \in W \\
Q_b^d(t+1) &= (1+\alpha)\,Q_b^d(t) - \alpha P^d && \text{if } Q_a^d \in C_k^d;\ Q_b^d \in C_j^d \neq C_k^d;\ P^d \in W \\
Q_a^d(t+1) &= (1+\alpha)\,Q_a^d(t) - \alpha P^d && \text{if } Q_a^d \in C_j^d \neq C_k^d;\ Q_b^d \in C_k^d;\ P^d \in W \\
Q_b(t+1) &= (1-\alpha)\,Q_b^d(t) + \alpha P^d && \text{if } Q_a^d \in C_j^d \neq C_k^d;\ Q_b^d \in C_k^d;\ P^d \in W \\
Q_a^d(t+1) &= Q_a^d(t) && \text{if } P^d \notin W \\
Q_b^d(t+1) &= Q_b^d(t) && \text{if } P^d \notin W
\end{aligned}
\tag{5.9}
$$

In the equation above, α is maintained at a constant value of 0.1 (as opposed to varying it as recommended in [52,53]), and ϵ is kept to be 0.25. Also, the window W defined above is set to be a circle of diameter 1/100 of the distance between $Q_a^d(t)$ and $Q_b^d(t)$. As in the continuous case, the reader should observe that after the intraregional polarizing and the interregional polarizing, the representative codebook vectors impose a set of piecewise linear boundaries which assign the original nodes, $\mathcal{L}$, into potentially slightly different regions than that which is initially assigned. Thus, although the initial demarcation boundaries may have been incorrectly assigned, the sequence of polarizing operations tends to re-allocate them. The effect of this boundary re-allocation will be discussed in greater detail in the section describing our experimental results. After the training points have been allocated and the set of codebook vectors for each cluster has been learned, the patchwork of functions approximating the arbitrary distance function Ψ is now learned. This can be done using either an independent optimizing strategy or an LVQ scheme. We shall now demonstrate how these are achieved.

5.2.6. Parameter Learning Using VQ

After the individual regions have been represented by the various *discretized* codebook vectors (using the above migration strategies), the actual distances may be estimated. The ideas that are used for finding the parameters in the continuous case are

exactly the same in the discrete case. The only difference is that the codebook vectors lie on a discretized space. The algorithm computing the parameter k for the distance function $\Phi_2(P_a, P_b)$ becomes:

Input: The set of codebook vectors, the training set $\mathcal{L}$ and the distances $\Psi(P_i, P_j)$ for all $P_i, P_j \in \mathcal{L}$.

Output: The set of parameterizing coefficients, $\{k_{a,b} : 1 \leq a \leq b \leq R \times M\}$

Begin

For $a = b$ to $R \times M$ **Do**

$\lambda_{a,b} = 1$

$k_{a,b} = 0$

End For

Repeat until satisfied

Get any distinct pair of points $P_i, P_j \in \mathcal{L}$

If closest codebook vectors to P_i^d and P_j^d are Q_a^d and Q_b^d respectively then

$k_{a,b} = (1 - \lambda_{a,b})\, k_{a,b} + \lambda_{a,b}\, (\Psi(P_i, P_j) / \|P_i - P_j\|)$

Decrease $\lambda_{a,b}$

End If

End Repeat

End

5.3. Implementation Results

5.3.1. Database

Two distinct samples have been collected by pairing respectively 80 and 23 cities and towns of Türkiye for training and test sets. The first set is used for estimating the parameters and the second one to assess their performances. The training and test data sets contain planar coordinates and intercity distances for 80 and 23 cities respectively which make 3160 and 253 data pairs. The third dimension is ignored since previous empirical studies have shown

that the effect of elevation in the accuracy of the estimators in Türkiye is almost null [14]. The values reported in the following subsections are average error per pair in kilometers on both the training and the test sets. Recall that the error per pair is measured by the normalized error measure given in Equation (2.4).

5.3.2. Discrete Vector Quantization and The Self-Organizing Map

The very first step involved in implementing the discrete LVQ-algorithm is to build a grid structure with a specific cell length, which actually indirectly specifies the resolution. One should note that as the cell length decreases the resolution increases resulting in a finer grid structure and vice versa. In the experimental setting this is achieved as follows. In all the experiments reported in earlier publications involving Türkiye [14, 67, 68], the original map of Türkiye is enclosed within a bounding rectangle defined between the latitudes and longitudes 36 ° N, 26 ° E, 42 ° N and 45 ° E respectively.

The coordinates of the cities in the database are obtained by plotting the national boundary, the cities and towns, and the grid on a large map in which Ankara, the capital, is placed as the origin. Thus the coordinate axes for the bounding rectangle has the equations $x = -545.9$, $y = -430.1$, $x = 1013.2$, and $y = 243.1$ respectively. Consequently, the data points representing the cities have coordinate values within these ranges, and Mardin for example has coordinates $(666.4, -244.8)$. Similarly, Sorgun has the coordinates $(193.8, -5.1)$ and the actual road distance between Mardin and Sorgun was known to be 703.5 kms. Since the data points *do not* all lie on a grid, discretizing must be effected in the process of the computation. This is quite easily done by multiplying every coordinate by a factor, called the *magnifying factor*, specified by ϱ. The new coordinates are then rounded off to the closest integer. Note that by multiplying the coordinates by ϱ and rounding, the integers are mapped equivalently to the model proposed earlier except for a simple translation. Indeed, in any actual implementation, this translation need not be done, since the computations are not effected whether we work in the range $[0, \ldots, J]$ or in the range $[-a, \ldots, b]$.

Observe that, what is lost (from going from the continuous world to the discretized) is the fractional portions of the coordinates, but what is gained is that these fractional portions are magnified by ϱ and then rounded off to the closest integer. Thus when the resolution parameter ϱ is 4, Mardin and Sorgun have the *discretized* coordinates $(2666, -979)$ and $(775, -20)$ respectively. However it should be emphasized that although the coordinates of the points are discretized, and thus have their magnitudes magnified, the *actual* distances between the corresponding cities are still the real world travel distances, and are thus unchanged.

The results of experiments involving initial random and non-random partitions for Türkiye highlight the characteristic features of the scheme. Also, in the interest of comparing the current discretized work with its continuous counterpart, the initial partitions are exactly the same as in the continuous case. In each of the figures, the towns themselves are marked with an '$\times$'. Note that the actual map of Türkiye has not been superimposed in the figure to avoid cluttering it. The twelve squares, '$\square$' represent the final positions of the codebook vectors.

Consider Figure 5.3.1. In this case the initial partitioning has four subregions and is achieved "manually" but in an arbitrary manner. The subregions divide the country into four rectangles and shown in the figure, each containing 20 towns in the training set $\mathcal{L}$. To demonstrate the power of the strategy, a DLVQ strategy is used with only three codebook vectors in each region which are initialized to be on the border of their representative regions. During the intraregional polarizing phase the DLVQ algorithm is invoked with a value of α which starts at unity and decreases linearly to 0.9 in 1,000 iterations. As expected, most of the learning is accomplished in this phase. Thereafter, in the fine tuning phase the value of α is drastically switched to 0.2 and decreases linearly to attain to 0.1 in 2,000 time steps. In the interregional polarizing phase, the value of α is maintained to 0.1. The constant for the migration of codebook vectors from the same class, ϵ is maintained at 0.25 and done for 2000 iterations (i.e., 25 cycles of all the 80 training sites). Indeed, the entire convergence for both these phases takes only a matter of a couple of seconds. The DLVQ is implemented for various values of the magnifying factor ϱ ranging from 1 to 32 in powers of 2. Thus, the

coarsest resolution with $\varrho = 1$ corresponds to the case when the original data are rounded off, and as explained earlier, larger values of ϱ resulted in data points which are rounded off after multiplying the coordinates by ϱ.

It is generally observed that for small values of ϱ, both the intra and interregional polarizing affects the codebook vectors. But as the value of ϱ increases (typically more than 4) most of the polarizing is effected by the intraregional polarizing and the interregional polarizing merely verifies the locations of the final codebook vectors without invoking any additional changes. In each case, the final partitioning (after the codebook vectors converge) is fully determined by the discriminant function implicitly created by the bisectors of the lines joining the codebook vectors. This partitioning is adaptively learned and is shown in Figure 5.3.1 in bold lines for the case when $\varrho = 8$. Observe the power of the adaptive regional partitioning scheme.

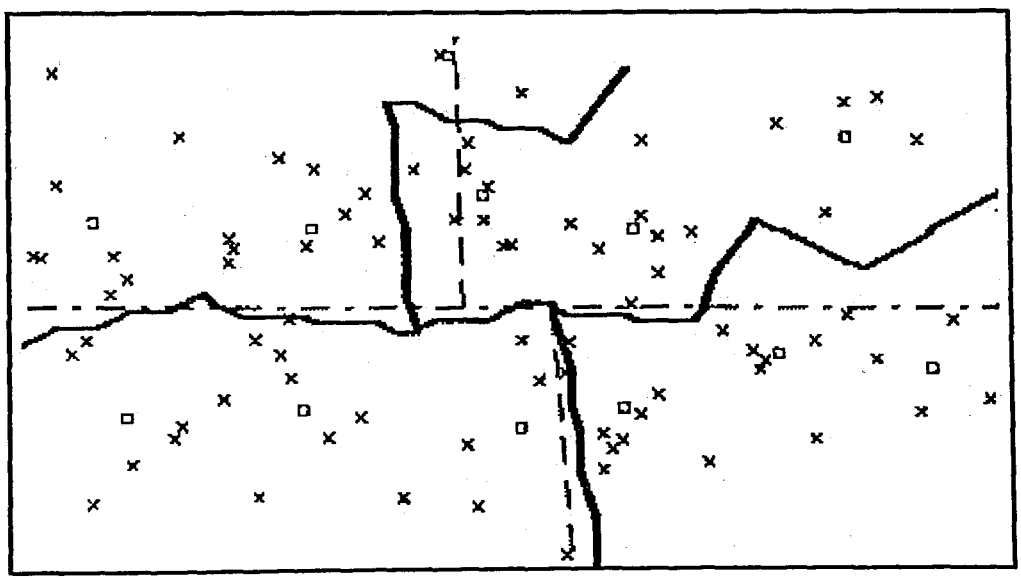


FIGURE 5.3.1. The original and final boundaries of the four sub-partitions of Türkiye. The initial partitions are generated manually.

After the intra and interregional learning, the constants for the underlying patchwork functions are estimated using the *true coordinates* of the training sites and their corresponding recorded distances. The estimates computed are of two sorts. First of all, to

TABLE 5.3.1. Comparison of the average error for the single functional form for each subregion.

Estimator	Continuous Training Error	Continuous Test Error	Discrete Training Error ($\varrho = 8$)	Discrete Test Error ($\varrho = 8$)
$\Phi_2(x_1, x_2)$ and simple average	3.34	7.83	3.363	7.836
$\Phi_2(x_1, x_2)$ and optimization	3.12	8.05	3.141	8.046
$\Phi_3(x_1, x_2)$ and optimization	2.93	7.80	2.926	7.731
$\Phi_4(x_1, x_2)$ and optimization	2.77	7.60	2.766	7.523

show the power of the multi-regional approach, it is assumed that the distances within each region and the distances between the regions are each characterized by a single functional form. Thus, since $\mathcal{G}$ is partitioned into four sets, a functional form of the type $\Phi_2(P_a, P_b)$ involves 10 constants. These constants are estimated by both a simple averaging scheme and an LVQ method as explained in Section 4.3. When the explicit form of each intra and interregional function is of the types $\Phi_3(P_a, P_b)$ and $\Phi_4(P_a, P_b)$ (where the parameters to be estimated are $\{k, p\}$ and $\{k, p, s\}$), the total number of parameters to be estimated becomes 20 and 30, respectively. In the latter two cases, the optimization is done independently, and this is typically more time consuming because it involves invoking separate nonlinear optimization procedures. The results obtained are quite remarkable and are tabulated in Table 5.3.1, where the average training and testing errors are recorded for the case when $\varrho = 8$.

For example, when the functional form is assumed to be of type $\Phi_2(P_a, P_b)$, the average error obtained by averaging k (which is exactly the error obtained by an LVQ algorithm in the k-space) is 7.836. This decreases to 7.731 and 7.523 for the cases when the parameters are $\{k, p\}$ and $\{k, p, s\}$ respectively. The corresponding results for averaging in the k-space using the continuous LVQ solution is 7.83, which decreases to 7.80 and 7.60 for the cases when the parameters are $\{k, p\}$ and $\{k, p, s\}$ respectively. The results for both the algorithms are tabulated in Table 5.3.1.

The full power of the multiregional approach is clearly displayed if the distance function is patched using a separate subfunction for each codebook vector and between each

TABLE 5.3.2. Comparison of the average error for a separate subfunction for each codebook vector.

Estimator	Continuous Training Error	Continuous Test Error	Discrete Training Error ($\varrho = 8$)	Discrete Test Error ($\varrho = 8$)
$\Phi_2(x_1, x_2)$ and simple average	2.42	7.69	2.433	7.622
$\Phi_2(x_1, x_2)$ and optimization	2.36	7.90	2.371	7.793
$\Phi_3(x_1, x_2)$ and optimization	2.11	7.48	2.091	7.325
$\Phi_4(x_1, x_2)$ and optimization	1.93	7.12	1.917	7.077

codebook vector. In this case, since $\mathcal{G}$ is partitioned into four sets with three codebook vectors in each region, 78 explicit subfunctions are involved for $\Phi_2(P_a, P_b)$. When the forms of each intra and interregional function are of the types $\Phi_3(P_a, P_b)$ and $\Phi_4(P_a, P_b)$ (where the parameters to be estimated are $\{k, p\}$ and $\{k, p, s\}$), the total number of parameters to be estimated is 156 and 234 respectively. Again, as in the above, the latter two cases involves an independent nonlinear optimization. The results obtained are given in Table 5.3.2.

In the most conservative case, the testing error is only 7.622 and in the case when the functions are characterized by $\{k, p, s\}$ the test error goes as low as 7.077. This should be compared with the results for the continuous LVQ scheme where the most conservative case (obtained by averaging in the k-space) yields a testing error of 7.69, and in the case when the functions are characterized by $\{k, p, s\}$ the test error becomes 7.12. The power of the scheme is clear — it is generally observed that for this value of the magnifying parameter ϱ, the discretized scheme performs uniformly better than the continuous scheme which has been the most accurate scheme reported in the literature.

Observe too that like the continuous scheme the most time consuming phase of the learning is the optimization stage. But since this is done only once (during the training phase) the work done is well worth its while. But unlike the continuous scheme, all the learning is performed using only simple integer computations without even evaluating the Euclidean norm between points. Thus, from the point of view of both speed and accuracy, the current scheme seems to be the most superior scheme currently available. Subsequently,

in the testing phase, the estimation of the distance between any two points merely involves computing the Discretized Euclidean distance between them and invoking the computation of the functional form associated with their nearest codebook vectors.

A word regarding the variation of the accuracy with the magnifying parameter is not out of place. Generally speaking, the accuracy is comparable to the other reported schemes (other than the continuous LVQ scheme) for small values of ϱ. This accuracy increases remarkably with the magnification as ϱ increases from 2 to 8 and then tends to stabilize thereafter. This implies that the effect of magnification and discretization can be used profitably only till a certain limit. Beyond this limit, magnifying and rounding yields no (incremental) marginal advantage. The variation of the test error as a function of ϱ (drawn on a logarithmic scale) is shown in Figure 5.3.2 for the case when the estimators are characterized by $\{k, p, s\}$. Notice that this error starts at the value of 7.907 when $\varrho = 1$, and decreases to the value of 7.526 for $\varrho = 2$. It further decreases to the value of 7.077 for $\varrho = 4$, stays at this value till $\varrho = 256$. It is observed that this performance is typical.

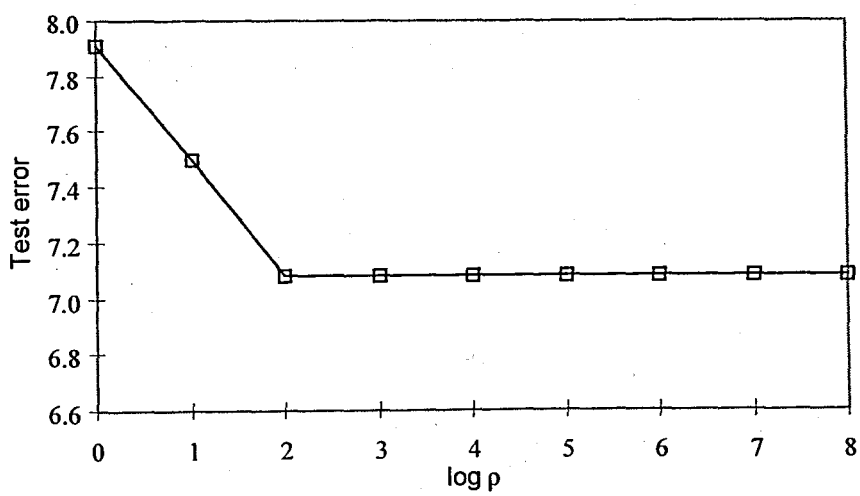


FIGURE 5.3.2. Variation of the test error with log ϱ where ϱ is the magnification factor.

From Figure 5.3.1, it can be observed that the final regional boundaries do not differ "significantly" from the original "arbitrary" ones. The difference between the two sets of boundaries would have been a lot more accentuated if the initial boundaries had been more

randomly generated. To demonstrate this, results are also reported for a case when the initial quadrilaterals are randomly generated. The bounding rectangle of Türkiye is divided into four "random" quadrilaterals. This is achieved by generating a random point on each of the four edges of the bounding rectangle. The lines joining the points on the opposite sides of the bounding rectangle are now used to constitute the four quadrilaterals. As in the previous case the initial random partitions are shown in Figure 5.3.3. In each subregion the number of codebook vectors is three. A DLVQ strategy with $\varrho = 8$ with only these codebook vectors in each region is invoked with values of α, ϵ and the number of cycles being as in the above experiment. As in Figure 5.3.1, the final partitioning (after the codebook vectors converge) is fully determined by the bisector discriminant functions and is also shown in Figure 5.3.3 in bold lines. The power of the method is clear because even though a random partitioning is used, the final partitioning yields reasonably good results. Indeed, after the intra and interregional learning, the constants for the underlying patchwork functions are estimated using the training sites and their corresponding recorded distances as in the above case. Only the error when the explicit form of each intra and interregional function is of the type $\Phi_4(P_a, P_b)$ is reported. As in the previous case these characterizing constants are computed by an optimization procedure. In this case the training error is 1.856 and the testing error was as low as 6.987 which is far superior to all the previously reported methods. Note that in the random case cited for the continuous VQ algorithm, the corresponding errors are 1.787 and 7.189 respectively. The way by which the initial boundaries influence the final ones is still unknown in both the continuous and discretized scenarios.

In general, although these results for the random case are so promising, it is disadvantageous to partition the cities in a completely random way, because it would defeat the very purpose of partitioning—which attempts to take advantage of the geographical proximity between cities in the various subpartitions.

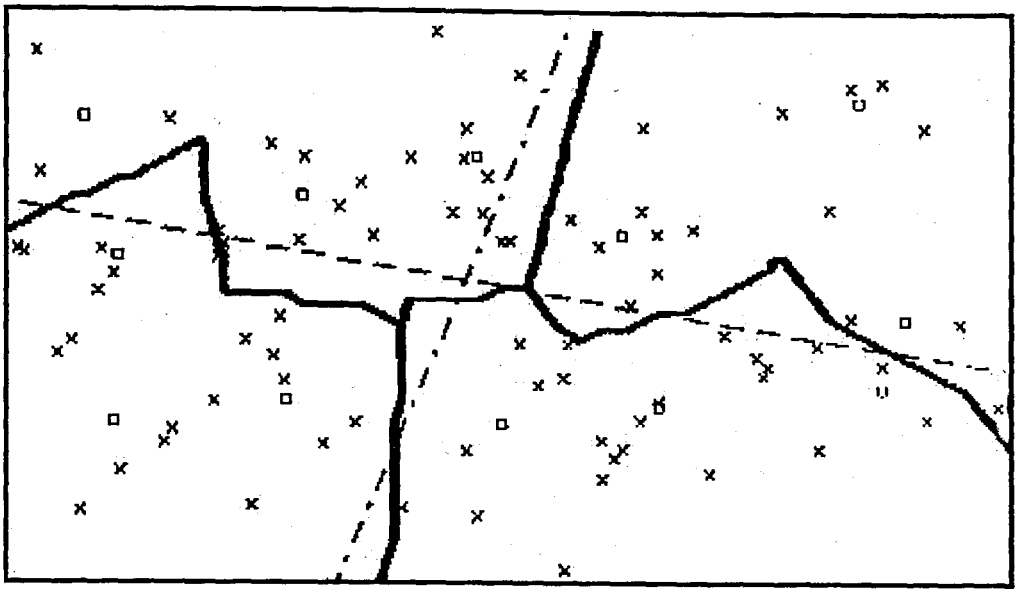


FIGURE 5.3.3. The original and final boundaries of the four sub-partitions of Türkiye. The initial partitions are generated manually.

5.4. Conclusions

In this chapter the problem of estimating arbitrary distance functions on a grid is studied. To achieve this the learning concepts involving two vastly different areas of adaptive learning namely, neural networks and learning automata are utilized. A method is developed by which the general philosophies of Learning Vector Quantization and discretized automata learning can be incorporated to yield Discretized Learning Vector Quantization. The algorithms have been rigorously tested for the actual road-travel distances involving cities and towns in Türkiye. The results are among the best results currently available from any *single or hybrid* strategy and are often superior even to the case when continuous LVQ was used for the polarizing [67]. The results of Alpaydın et al. [58] show how a combination of learning strategies can be used to yield superior results by incorporating stacking and voting principles. Clearly, such principles can be used subsequent to the current results to yield even smaller estimation errors.

The regions are learned adaptively using discriminant functions derived implicitly from the codebook vectors. In this process the algorithms perform simple integer

manipulations and are thus extremely fast. Subsequent to the partitioning, the actual parameters of the intraregional and interregional functions can be obtained either by optimization (in a non-all-neural approach, for example when k, p and s are parameters) or by using an LVQ algorithm in this parameter space itself. This is also novel, because the problem lends itself to many distinct philosophies of learning.

6. SELF-ORGANIZING MAPS AND THE KNIES

6.1. Biological Connection to the Self-organizing Maps

The prototype or codebook vectors $\mu_i(t)$ used in the vector quantization (see Section 4.1) actually develop into a set of *feature-sensitive detectors* as the update mechanism given with Equation (4.7) is applied. Feature-sensitive cells are also known to be common in the brain.

In vector quantization all the codebook vectors act independently. Therefore the order in which they are assigned to the different domains of input signals x is more or less haphazard, strongly depending on the initial values of the $\mu_i(0)$. In fact, in 1973 von der Malsburg [69] had already published a computer simulation in which he demonstrated local ordering of feature-sensitive cells such that in small subsets of cells roughly corresponding to the so-called columns of the cortex, the cells were tuned more closely than were more remote cells. Later, Amari [70] formulated and analyzed the corresponding system of differential equations and related these differential equations to spatially continuous two-dimensional media. Such continuous layers interacted in the lateral direction and the arrangement was called a nerve field. These studies are important theoretically since they involve a self-organizing tendency. However, the ordering power was weak as nerve-field type equations only describe the ordering tendency as a marginal effect. In spite of numerous attempts, no "maps" of practical importance could be produced; ordering was either restricted to a one-dimensional case, or confined to small parcelled areas of the network.

It became later apparent that system equations have to involve much stronger, idealized self-organizing effects. Furthermore, the organizing effect has to be maximized in every possible way before useful global maps can be created. In 1981, Kohonen experimented with various architectures and system equations and found a process description that seemed generally to produce globally well-organized maps [71, 72]. Neurophysiological studies have demonstrated the existence of various kinds of maps (e.g.,

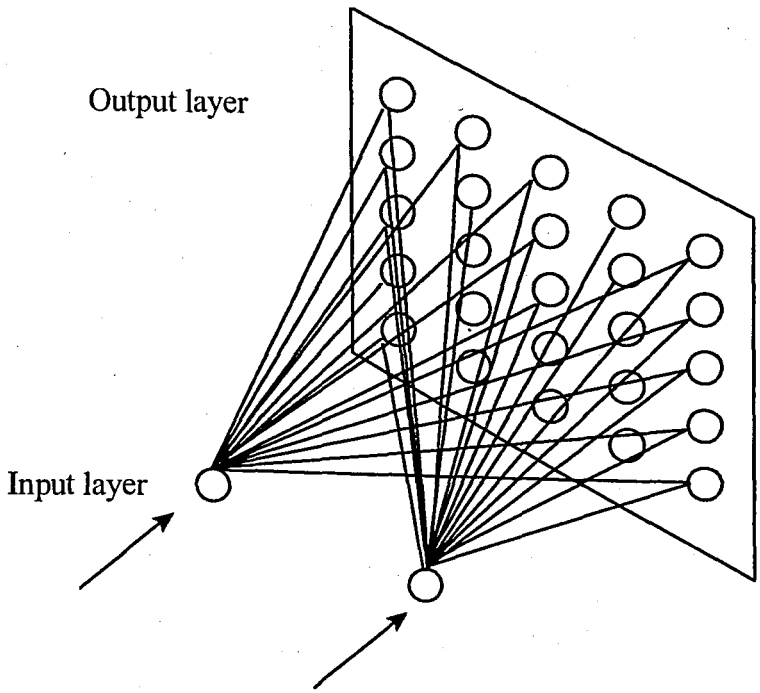


FIGURE 6.1.1. Typical architecture of the SOM.

tonotopic map, somatotopic map, etc.) in the brain of higher animals [53]. These maps serve as centers of activities such as speech, vision, hearing, and motor functions in specific areas of the cortex [73]. Different regions (maps) in the brain, therefore, seem to be dedicated to specific tasks.

The typical architecture of the self-organizing map (SOM) network includes a sheetlike, planar layer of neurons (codebook vectors called neurons in the neural network terminology); i.e., the neurons are organized into a two-dimensional and fully lateral connected topology (see Figure 6.1.1). The neurons adapt (learn) in such a way that the topological relationships in the input pattern space are preserved. Hence, these maps are sometimes called self-organizing feature maps or topology preserving maps. Apparently, physically close neurons will respond to naturally similar input patterns. Accordingly, each neuron, or a collection of neurons, becomes sensitive for a particular type of input signals. This means that different artificial neurons become feature sensitive detectors, which are known to be common in the mammalian brain.

The close relationship of artificial self-organizing maps to the biological nervous system has given the Kohonen Self-organizing Map Network the potential to model a variety of real world problems.

6.2. The SOM Algorithm

The SOM algorithm follows a competitive learning technique with lateral excitation. As in the vector quantization, when an input vector X is given, the closest codebook vector or neuron is the winner. Unlike the vector quantization, however, not only the winner but also its neighboring neurons are updated. Namely, the winner and a group of neurons in the neighborhood of the winner are moved towards the current input. Thus, in addition to their positions in the input space defined by their weight vectors, they also have a position among other neurons. In an self-organizing map neurons are organized linearly in one dimension or as a grid in two dimensions and neighbors of a neuron is defined in one or two dimensional space. Consequently, SOM reduces the dimensionality of the input data and provides a graphical organization of pattern relationships which can be visualized and analyzed. This is the biggest advantage of the SOM over traditional classifiers including the vector quantization. The formal definition of the SOM follows.

There are two layers in the SOM network; the input layer and the output or competitive layer which are fully interconnected (no hidden layers as is the case in multi-layer perceptrons). The output layer includes the output units $\{O_j : j = 1, \ldots, M\}$ which are organized in a linear or planar lattice. The input layer is an array of n neurons that act as linear "gates" for the ith input vector $\mathbf{X}_i = (X_{i1}, X_{i2}, \ldots, X_{in})$. The weights from the second layer neurons to the first layer neurons are denoted as $\{Y_{jk}(t) : j = 1, \ldots, M;\ k = 1, \ldots, n\}$; i.e., the weight from output neuron O_j to input neuron X_{ik}. For the sake of simplicity, the weight vector $\mathbf{Y}_j(t) = (Y_{j1}(t), Y_{j2}(t), \ldots, Y_{jn}(t))$ between the output neuron O_j and the input vector $\mathbf{X}_i$ is used to denote the jth neuron.

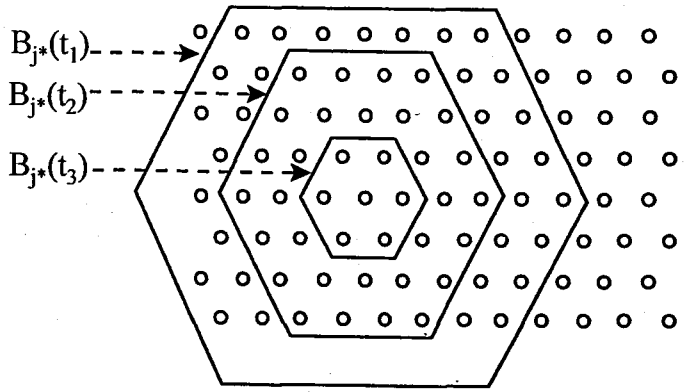


FIGURE 6.2.1. Examples of topological neighborhood $B_{j^*}(t)$, where $t_1 < t_2 < t_3$.

The SOM algorithm computes a dissimilarity or distance measure (mostly Euclidean distance) between the input vector $\mathbf{X}_i$ and each neuron $\mathbf{Y}_j(t)$. The neuron $\mathbf{Y}_{j^*}(t)$ with the shortest Euclidean distance is chosen as the winner, i.e.,

$$j^* = \underset{j}{\operatorname{argmin}} \{\|\mathbf{X}_i - \mathbf{Y}_j(t)\|\} \tag{6.1}$$

which is then allowed to move even closer to the input vector $\mathbf{X}_i$. Close neighbors of the winner neuron are also allowed to participate in the weight updates. For this reason a neighborhood set $B_{j^*}(t)$ is defined which is centered around the winner neuron $\mathbf{Y}_{j^*}(t)$. The width or radius of $B_{j^*}(t)$ is time-variable; actually it has been experimentally shown that for good global ordering it is advantageous to let $B_{j^*}(t)$ be very wide in the beginning and shrink monotonically with time (Figure 6.2.1).

The explanation for this may be that a wide initial $B_{j^*}(t)$, corresponding to a coarse spatial resolution in the learning process, first induces a rough global order in the output neurons, after which narrowing the $B_{j^*}(t)$ improves the spatial resolution of the map while preserving the attained global order. It is also possible to end the process with $B_{j^*}(t) = \{j^*\}$ meaning that at the final iterations only the winner neuron is updated in which case the process is reduced to vector quantization. However, the "topological order" of the map have

to be formed prior to this phase. The updating process is as follows:

$$\mathbf{Y}_j(t+1) = \begin{cases} \mathbf{Y}_j(t) + \alpha(t)(\mathbf{X}_i - \mathbf{Y}_j(t)) & j \in B_{j^*}(t) \\ \mathbf{Y}_j(t) & \text{otherwise} \end{cases} \tag{6.2}$$

where $0 \leq \alpha(t) \leq 1$ is a linearly or monotonically decreasing scalar-valued function of time or iteration index.

It is possible to specify the neighborhood as a scalar kernel function $\Lambda(j, j^*)(t)$ as:

$$\mathbf{Y}_j(t+1) = \mathbf{Y}_j(t) + \Lambda(j, j^*)(t)(\mathbf{X}_i - \mathbf{Y}_j(t)) \tag{6.3}$$

where the neighborhood function $\Lambda(j, j^*)(t)$ is maximum when $j = j^*$ and decreases as the distance between neurons j and j^* increases. In one dimensional map the distance is simply $|j - j^*|$ (the difference between the neurons' indices) whereas in the two-dimensional map, it is computed either as $\|\mathbf{Y}_j - \mathbf{Y}_{j^*}\|$ (the Euclidean distance between the neurons j and j^*) or again as $|j - j^*|$. $\Lambda(j, j^*)(t)$ should also decrease in time as explained previously. A suitable form for $\Lambda(j, j^*)(t)$, which is at the same time the mostly adopted one is the Gaussian function given as:

$$\Lambda(j, j^*)(t) = \Lambda_0 \exp\left(\frac{-\|\mathbf{Y}_j(t) - \mathbf{Y}_{j^*}(t)\|^2}{\sigma^2}\right) \tag{6.4}$$

where $\Lambda_0 = \Lambda_0(t)$ and $\sigma = \sigma(t)$ are suitable decreasing functions of time.

As training (and learning) proceeds, the output neurons (weight vectors) slowly move towards separate groupings (clusters) of the input vectors, as shown in Figure 6.2.2. After training, the network converges so that each weight vector may coincide with the centroid

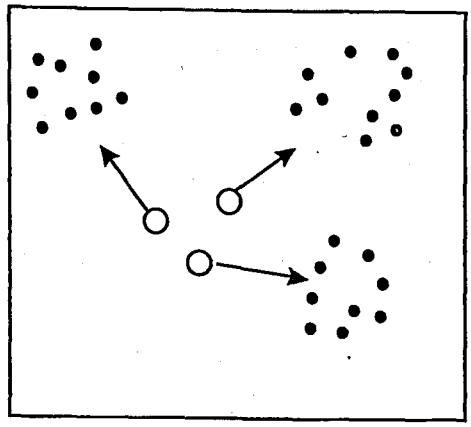


FIGURE 6.2.2. Neurons moving towards separate clusters.

of a particular cluster of vectors from the input data, as indicated by Figure 6.2.3. Thus, the output neurons become detectors of different classes or clusters.

The input vector $\mathbf{X}_i$ is usually a random variable with a density function $p(\mathbf{X})$, from which the successive values $\mathbf{X}_i$ are drawn. In real-world observations, such as speech recognition, the input $\mathbf{X}_i$ may be successive samples of the input observables in their natural order of occurrence.

The process may be started by choosing arbitrary, even random, initial values for $\mathbf{Y}_j(0)$ with the condition of being different. Since learning is a stochastic process, the final statistical accuracy of the mapping depends on the number of steps or iterations which should be large in number (Kohonen used 100,000 iterations in his experiments). Since the algorithm is computationally light, if only a small number of input samples are available, they must be presented to the network repeatedly.

The choice of the neighborhood $B_{j^*}(t)$ is important. If the neighborhood is too small to begin with, the map will not be ordered globally. Therefore, it is recommended that the process be started with a fairly large neighborhood. The radius of $B_{j^*}(0)$ may even be more than the radius of the network. After the proper ordering takes place, the radius of $B_{j^*}(t)$ can shrink linearly to one unit.

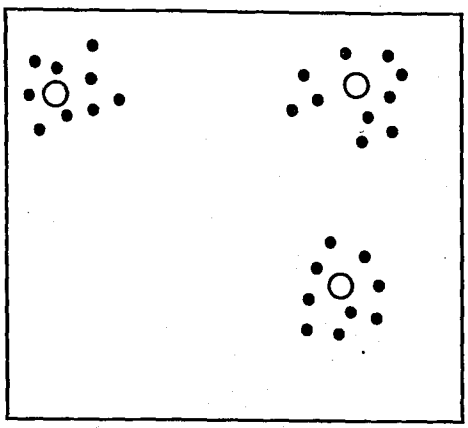


FIGURE 6.2.3. Neurons as cluster centers.

Of all the families of neural networks described in the literature, the Kohonen self-organizing neural network has been the most widely investigated one [52, 53]. This is not surprising, given its simplicity and the wide variety of problem areas to which it may be applied. In statistical pattern recognition it has been used in the recognition of Finnish and Japanese speech [74–76], sentence understanding [77], in classification of sea-ice [78] and even in the classification of insect courtship songs [79]. From a hardware point of view the SOM has been used in the design of algorithms, which at the lowest level can control the production of semi-conductor substrates [80,81], and at a higher level the synthesis of digital systems [82].

The beauty of the SOM is the fact that the individual neurons adaptively tend to learn the properties of the underlying distribution of the space in which they operate. Additionally, they also tend to learn their places topologically. This feature is particularly important for problems which involve two and three-dimensional physical spaces, and is indeed, principal motivation for the SOM being used in path planning and obstacle avoidance in robotics [83–87]. Although the SOM is described and implemented in such an elementary manner it has extremely elegant properties. A collection of these is found in [52, 88].

First of all Kohonen showed that the resulting map forms a Voronoi tessellation of a given input space which is a kind of vector quantization partitioning of the input space into nonoverlapping regions [52]. Furthermore, apart from these tessellations, the neurons also

ultimately collectively posses the stochastic properties of the underlying data points. Indeed, if the width of the neighborhood is fixed and $\alpha(t)$ is very small Ritter [89] showed that for the scalar case, the density of the neurons (weight vectors) $G(\mathbf{Y})$ is proportional to $F(\mathbf{X})^{\gamma}$ where $F(\mathbf{X})$ is the distribution of the input vectors, and

$$\gamma = \frac{(W+1)^2/3}{(W+1)^2+W^2}, \tag{6.5}$$

where $[-W, W]$ specifies the activation bubble.

Although the SOM is very powerful, there is a relatively vast amount of information that it ignores in the training process. This information, which can be informally perceived as the global information, is the statistical information resident in the data points (input vectors) in their entirety. Thus, although in the SOM the neurons asymptotically learn the distribution of the points statistically, the SOM does so only by virtue of the points themselves and not by utilizing the information resident in the overall set of points, such as their mean etc.

In the following section a new self-organizing map called The Kohonen Network Incorporating Explicit Statistics (KNIES) is presented. The primary difference between the SOM and the KNIES is the fact that every iteration in the training phase includes two distinct modules—the *attracting* module and the *dispersing* module. In the attracting module a subset of the neurons migrate towards the data point that has been presented to the network. This phase is essentially identical to the learning phase of the SOM. However, subsequent to this phase the rest of the neurons which have not been involved in the attracting module participate in a dispersing (repellent) migration. Indeed, these neurons now move away from their current positions in a manner that ensures that the global statistical properties of the data points are imitated by the neurons. Thus, although in the SOM the neurons *asymptotically* find their places both statistically and topologically, in the KNIES they collectively maintain their means to be the means of the data points which they represent.

6.3. The Kohonen Network Incorporating Explicit Statistics (KNIES)

6.3.1. Introduction

Although the overall effects of using the self-organizing map is that the neurons ultimately obey the distribution of the original points, there is unfortunately a lot of information that it does not use. Indeed, in the effort to preserve the "non-parametric" form of the underlying algorithm both VQ and SOM completely ignore the statistical information that is already collectively resident in the data points. For example, as soon as $\mathcal{P}$, the set of data points is known, all the associated statistics of the points such as the mean, covariance etc. can be computed quickly. Thus we can conceive of enhanced SOM methods which, apart from only using the information in the data point presented, uses this information in conjunction with these pieces of statistical information. In other words, at every iteration, we seek a strategy by which we will utilize both the local information present in the point presented and also the global information latent in the entire data set, $\mathcal{P}$. How this is achieved is described below.

Let us suppose that the mean of the set of points to be represented by the neurons in $\mathcal{P}$ is μ. The intention is that we maintain μ to be invariant for the entire training. The question which remains is merely one of determining how the neurons can be collectively updated so as to retain their mean to be also μ. To do this, as alluded to earlier, every training step is decomposed into two phases, namely the attracting and dispersing phases.

When the data point P_i is presented to the network, the closest neuron and all the points within the activity bubble (the neighborhood of the winner) are migrated using an equation identical to Equation (6.2). This constitutes the attracting module, and in this phase the neurons within the bubble use the local information present in the points as they appear individually and move towards them.

Observe that since the neurons are being partitioned as those which participate in the attracting phase and those which do not, in KNIES it is assumed that the activation bubble is maintained constant for a specified value of M. The question of decreasing the effect of the attraction as the neuron $j \in B_{j*}(t)$ is more distant from j^* must necessarily be achieved by using a kernel function. For the sake of simplicity the kernel is selected as the Gaussian kernel [39, 51] given by:

$$\Lambda(a,b) = K_g \cdot e^{\frac{-\|Y_a - Y_b\|^2}{\sigma^2}}, \tag{6.6}$$

where K_g is a normalizing constant, σ specifies the standard deviation of the kernel which is progressively decremented (for example, linearly [88, 90]) to get the effect of reducing the bubble of activity, and $\mathbf{Y}_a$ and $\mathbf{Y}_b$ are the coordinates of the points in question. Thus the attraction phase obeys:

$$\mathbf{Y}_j(t+1) = \begin{cases} \mathbf{Y}_j(t) + \Lambda(j, j^*)(\mathbf{X}_i - \mathbf{Y}_j(t)) & \text{if } j \in B_{j*}(t) \\ \mathbf{Y}_j(t) & \text{otherwise} \end{cases}, \tag{6.7}$$

where $\mathbf{X}_i$ is the input vector presented to the system and j^* is the winning neuron closest (using the Euclidean distance) to $\mathbf{X}_i$.

As a result of the migration of the neuron within the activation bubble it is observed that the mean of all the neurons moves away from the initially assigned value, μ. Thus, the dispersing phase now forces the neurons outside the activation bubble to migrate away from their current locations to render the mean of the neurons to continue to be invariant. This is done as follows. The total migration that has been effected as a result of the attraction module is the vector $\Delta\mathbf{Y}(t)$, where,

$$\Delta\mathbf{Y}(t) = \sum_{j \in B_{j*}(t)} [\mathbf{Y}_j(t+1) - \mathbf{Y}_j(t)]. \tag{6.8}$$

This change $\Delta \mathbf{Y}(t)$, is distributed among the neurons which did not participate in the attraction as:

$$\mathbf{Y}_j(t+1) = \mathbf{Y}_j(t) - \frac{1}{M(t)-|B_{j^*}(t)|}\Delta \mathbf{Y}(t), \quad \text{if } j \notin B_{j*}(t). \tag{6.9}$$

Notice that in (6.9) the change $\Delta \mathbf{Y}(t)$ can also be distributed among the neurons in a non-uniform manner—for example by using a second weighting kernel function for the dispersing neurons. However, in the interest of simplicity (since there is no such neuron as the loser "neuron") the change is distributed uniformly. It is easy to see that as a result of both (6.7) and (6.9) the attracted neurons move towards the point $\mathbf{X}_i$, and the net result of both the phases retains the mean of the neurons to be $\boldsymbol{\mu}$, as desired. This section is concluded with a formal description of the algorithm.

Algorithm KNIES

Input : The N data points.

Output : The M neurons.

Parameters : ω, the width determining the window $B_{j*}(t)$ and σ, the standard deviation of the Gaussian kernel decremented linearly as time progresses.

Method

Begin

Repeat

For every point $\mathbf{X}_i \in \mathcal{P}$ **Do**

$j^* \leftarrow \text{argmin}_j \|\mathbf{X}_i - \mathbf{Y}_j(t)\|$

$\Delta \mathbf{Y}(t) \leftarrow \mathbf{0}$

$|B_{j*}(t)| = \omega M(t)$

For all $j \in B_{j*}(t)$ **Do**

$\Delta \mathbf{Y}_j(t) \leftarrow \Lambda(j, j^*)(\mathbf{X}_i - \mathbf{Y}_j(t))$, where Λ is defined by (6.6)

$\mathbf{Y}_j(t) \leftarrow \mathbf{Y}_j(t) + \Delta \mathbf{Y}_j(t)$

$\Delta \mathbf{Y}(t) = \Delta \mathbf{Y}(t) + \Delta \mathbf{Y}_j(t)$

End For

For all $j \notin B_{j*}(t)$ **Do**

$$\mathbf{Y}_j(t) \leftarrow \mathbf{Y}_j(t) - \frac{1}{M(t) - |B_{j^*}(t)|}\Delta\mathbf{Y}(t)$$

End For

End For

Decrement σ

Until Satisfied

End Algorithm KNIES

6.3.2. The Limitations of KNIES

Although KNIES is philosophically superior to the traditional SOM, the additional effort used in maintaining the statistics is generally not useful for most problems. The reason for this is quite straightforward: the advantage that is gained is purely in terms of the rate of convergence of the neurons. Experimentally, for the problem of distance estimation (see chapter 4), it is observed that the convergence is about 20 per cent faster for clustering. However, if the points are cycled through the traditional SOM algorithm, we know that the neurons will asymptotically converge in distribution to a distribution that mimics the underlying distribution of $\mathcal{P}$. Thus by enhancing the rate of convergence nothing drastic is achieved (except in terms of the number of iterations), unless the quality of the final solution of the underlying problem itself depends on the intermediate distribution of the neurons. For most problems this is not the case, and so, it is not anticipated that the KNIES will be more effective in solving problems than the traditional SOM. However, there are some problems where the intermediate location of the neurons can cause a difference in the final solution and one such problem is the Euclidean traveling salesman problem (TSP). In this case, a poor transient distribution of the neurons in the initial phase can cause the algorithm to yield a final TSP solution that is inferior to that which a superior transient distribution would yield. This property is used in the next section to develop two new algorithms for the TSP.

6.4. The Traveling Salesman Problem

The Euclidean Traveling Salesman Problem (TSP) has been one of the oldest "hard nuts" of operations research and mathematical programming. It is known to be *NP*-complete [91]. Any algorithm devised to solve the TSP tries to answer the following question: Given a set of N nodes (cities or vertices) and distances for each pair of nodes, what is the shortest tour that visits each node exactly once?

There are many exact and heuristic algorithms in the literature for the TSP [92, 93]. The exact algorithms, which are computationally expensive, provide an optimal solution, while the heuristic algorithms do not guarantee an optimal solution most of the time, but they "quickly" find near-optimal solutions which are within a few percent of the optimum [94]. The heuristic algorithms can be divided into three broad categories, i.e., tour construction heuristics, tour improvement heuristics, and composite heuristics which make use of the first two kinds of heuristics. The Lin-Kernighan heuristic [94] which is able to find solutions within one per cent of the optimum belongs to the third category.

In addition to the classical heuristic algorithms of operations research, there have also been several approaches based on artificial neural networks which solve the TSP. These approaches are either based on the self-organizing map of Kohonen or minimize an appropriate energy function. An overview of these schemes is given in the next section. The next chapter includes the new solution method which is obtained by adapting the KNIES to tackle the TSP.

6.5. Neural Solutions to the TSP

The first successful attempt in solving the TSP was the work of Hopfield and Tank [95]. Actually much of the excitement for the neural network approaches to optimization is initiated by Hopfield and Tank. In their seminal work, Hopfield and Tank used the

continuous Hopfield networks [96] for solving combinatorial optimization problems (e.g., the traveling salesman problem [95]) and the linear programming problem [97]. The usual procedure of this approach to optimization begins with the formulation of an energy function based on the objective function and constraints of the optimization problem under study—in this case the TSP. Specifically, the energy function incorporates the objective function and constraints through functional transformation and numerical weighting. Functional transformation is usually used to convert the constraints to a penalty function to penalize the violation of constraints and numerical weighting adjusts the balance between the constraint satisfaction and objective minimization (or maximization). The way how the energy function is formulated plays an important role in the success of the procedure based on neural networks.

It is expected that the local minimum of the energy function corresponds to a good solution of the original problem. The Hopfield-Tank model basically performs a gradient descent search to find a local minimum of the energy function E. However, the model suffers from the fact that the feasibility is not guaranteed, namely, not all the minima of the energy function represent feasible solutions to the TSP. This and some other deficiencies of the model have led researchers to improve the original model [98]. Many remedies have been proposed and some improvements have been really made. For example, Brandt et al. used a different energy function to solve the TSP [99] while Cuykendall and Reese used a scaling technique [100]. As a conclusion, however, it can be stated that both the original model as well as its variants with many refinements provide solutions for TSPs which consist of 200 cities or less.

Besides the Hopfield-Tank model there is another neural network scheme for the TSP which is the elastic net approach proposed by Durbin and Willshaw [101] in 1987. It is fundamentally different from the approach of Hopfield and Tank. It is a geometrically inspired algorithm and since it is closely related to the self-organizing map the details of it are provided below. The elastic net algorithm starts with M neurons lying on an imaginary "elastic band" originally located at the center of the cities, where M is larger than the number of cities N (see Figure 6.5.1). The neurons are then moved in the Euclidean plane so that

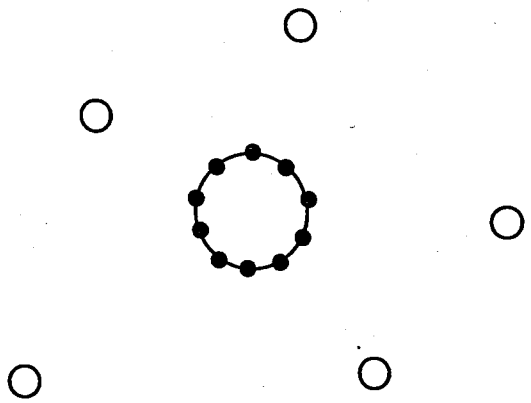


FIGURE 6.5.1. The initial elastic band.

the elastic band is gradually stretched until it passes sufficiently near each city to define a tour. During that process two forces apply: one for minimizing the length of the band, and the other one for minimizing the distance between the cities and the neurons on the band.

More formally, the neurons on the elastic band are updated according to the following equation:

$$\Delta \mathbf{Y}_j = -\alpha \sum_{i=1}^{N} w_{ij} \left(\mathbf{X}_i - \mathbf{Y}_j\right) + \beta K \left(\mathbf{Y}_{j+1} + \mathbf{Y}_{j-1} - 2\mathbf{Y}_j\right) \quad j = 1, \ldots, M \qquad (6.10)$$

where $\mathbf{X}_i$ represents the position of the ith city, $\mathbf{Y}_j$ represents the position of the jth neuron on the band, α and β are constant parameters, K is the scale parameter, and w_{ij} is a measure of the influence of city i on neuron j which is given as:

$$w_{ij} = \frac{e^{-\|\mathbf{X}_i - \mathbf{Y}_j\|^2/2K^2}}{\sum_k e^{-\|\mathbf{X}_i - \mathbf{Y}_k\|^2/2K^2}}. \qquad (6.11)$$

The α term drives the neurons on the band towards the cities, so that for each city i there is at least one neuron j within distance K. The β term is aimed at keeping neighboring neurons together so as to produce a short tour. The update equations can be expressed as the

derivative of an appropriate energy function E, namely

$$\Delta \mathbf{Y}_j = -K \frac{\partial E}{\partial \mathbf{Y}_j}, \tag{6.12}$$

where

$$E(\mathbf{Y}, K) = -\alpha K \sum_{i=1}^{N} \ln \sum_{j=1}^{M} e^{-\|\mathbf{X}_i - \mathbf{Y}_j\|^2/2K^2} + \frac{\beta}{2} \sum_{j=1}^{M} \|\mathbf{Y}_j - \mathbf{Y}_{j+1}\|^2. \tag{6.13}$$

Thus, the elastic net method finds a local minimum of the above energy function by performing a gradient descent in the two-dimensional Euclidean space. The first term of the energy function is minimized when every city is covered by a neuron on the elastic band, while the second term constitutes the length of the band, and hence the TSP tour length if the first term is negligible. This is achieved by starting the algorithm with a large value of the scale parameter K, and then gradually reducing it (like the temperature parameter in the simulated annealing). In the limit as $K \to 0$, a local minimum will eventually be reached.

Durbin and Willshaw applied this algorithm to the 30-city TSP used by Hopfield and Tank [95] and obtained in 1000 iterations the best known solution (the same tour as the one found by the Lin-Kernighan exchange heuristic [94]) which is 19 per cent better in terms of solution quality than the best solution of Hopfield and Tank. The variants of this algorithm may be found in [2].

Actually, Simic [102] has shown that both the Hopfield-Tank model and the elastic net approach can be derived from the principles of statistical mechanics. It should be reminded that while the Hopfield-Tank model can also be applied for the asymmetric TSPs (ATSP) the elastic net approach is applicable only for the Euclidean TSPs.

The Hopfield-Tank algorithm is quite slow and is computationally extremely expensive, even when the number of cities considered is small (less than 50). The elastic net [101] method is also quite slow, but Vakhutinsky and Golden [103] considered speeding up its convergence by using a hierarchical strategy. The idea of their algorithm is to subdivide the plane into smaller areas of square shape and replace the neurons in each square with their centroid or mean. As the algorithm proceeds, the squares are further divided so that finally all the cities are represented by only one city per square.

Of all the neural network methods that are available, the algorithms based on the self-organizing map of Kohonen are both the most accurate and the fastest. Some of these methods are reviewed here. A more detailed survey of other neural network schemes for solving the TSP can be found in [2, 104–106].

6.6. Pure Kohonen Network Solutions

The self-organizing map of Kohonen in its virgin form has been used to solve the TSP [107–109]. The idea that is used is essentially as follows. The network is fully specified by a fixed number of neurons, M, and the input to the network are the coordinates of the cities, say $\mathcal{P}$. These cities are mapped onto M neurons which are located on a ring, where the ordering on the ring represents the traversal of the cities. Observe that by virtue of the SOM, this mapping preserves the neighborhood relationships among the cities and also reflects this topological information.

The cities are first arranged in a random order and presented at each epoch in this order to the network. At every instant one city $\mathbf{X}_i$ from $\mathcal{P}$ is presented to the network. The neurons now compete and the closest one to $\mathbf{X}_i$, $\mathbf{Y}_{j*}(t)$, is determined. This vector and a group of neurons within its activation bubble, $B_{j*}(t)$, are now moved in the direction of $\mathbf{X}_i$ as per Equation (6.7). As mentioned earlier, the set $B_{j*}(t)$ can consist of a subset of neurons from the neighborhood of $\mathbf{Y}_{j*}(t)$. These neurons are given equal weights during the updates while those which are not contained in this subset have zero weights (not migrated at all).

Another possibility for $B_{j*}(t)$ is the inclusion of all neurons. In such situation neurons are given different weights by using a (Gaussian) kernel function. Experimentally, the former strategy has proven to be quite poor and so the use of a weighting function is generally chosen to be the scheme by which the neurons are drawn towards the presented city.

The Kohonen Network in its original form performs poorly for the TSP. First of all, the scheme is not guaranteed to converge. Furthermore, even when it does, the results obtained are quite suboptimal. If the variance of the Gaussian kernel is large the convergence is slower. If on the other hand the variance is made small, the algorithm may not converge (because in the limit the presented city effects only a single neuron) and even if it does, the quality of the tour is far from optimal. These drawbacks have motivated other researchers to consider improvements on the scheme and these are presented in the next subsections.

6.6.1. The Guilty Net

The main drawback of using the SOM for the TSP is the fact that the scheme changes the weights based only on the distance of the currently examined city to the various neurons on the ring. In [110], Burke et al. modified the above strategy in their "guilty net" algorithm, by introducing two modifications. The first modification involved incorporating a penalty term to the distance function for determining the winning neuron. The second modification involved specifying an alternate method for the neighborhood function which determined the strength by which the weights of the neighbor neurons are modified.

Just like in the pure Kohonen scheme, the number of neurons on the ring is fixed (Burke et al. recommend that it be equal to the number of cities) and remains the same throughout the algorithm [110]. The neurons on the ring are indexed from 1 to M, where $M = N$, and the distance between a neuron and its two immediate neighbors is equal to one. The guilty net tries to avoid the mapping of one or more cities on the same neuron on the ring by using the idea of "conscience" introduced by DeSieno [111]. This mechanism gives a chance to other neurons by inhibiting those which win too often. To achieve such an

inhibition, the winning neuron is selected according to the following formula:

$$j^* = \underset{j}{\operatorname{argmin}} \left\{ \|\mathbf{X}_i - \mathbf{Y}_j\| + \lambda \cdot \frac{win_j}{1 + \sum_k win_k} \right\}, \tag{6.14}$$

where win_j is the number of wins for neuron j, and λ is a penalty factor, a parameter which assumes real values. Notice now that the winner is not the one which is closest to $\mathbf{X}_i$—this distance must also take into consideration the number of times a particular neuron has won, and this combined effect tends to distribute the winning position among the others.

The neighborhood function in this method is defined as:

$$\Lambda(j, j^*) = \begin{cases} (1 - d_{jj^*}/L)^\beta & \text{if } d_{jj^*} < L \\ 0 & \text{otherwise} \end{cases} \tag{6.15}$$

where $d_{jj^*} = \min\{|j - j^*|, M - |j - j^*|\}$ and $L = N/2$. Notice that this distance function is identical to the Gaussian kernel except that, from a probabilistic point of view, it is more of a Geometric type—the indices of the neurons instead of their coordinates are used in determining the proximity of the neurons within a neighborhood.

It should be noted that $\Lambda(j, j^*)$ is equal to unity for the winning neuron j^*, and its value decreases as d_{jj^*} increases during one pass or cycle (the complete presentation of all cities). Furthermore, the updates of the weight vectors of the neighboring units are reduced over time by increasing the value of β at each cycle. This means that the neurons have a large neighborhood at the beginning of the procedure which gets smaller from cycle to cycle.

Reported computational results indicate that the guilty net approach performs better than the Hopfield-Tank model, but it is inferior to the elastic net method (on average 2.2 per cent) [2]. Our experience shows that the scheme is not too accurate, primarily

because the number of neurons is not allowed to increase or decrease (due to the absence of insertion/deletion operations) which other methods utilize [90]. When λ is small, convergence is not guaranteed. However, in contrast, when λ increases the quality of the tour deteriorates, because it is no longer a simple tour—the edges (which may be distant from each other) tend to cross each other. Experimentally, the scheme is inferior to the scheme due to Angéniol et al. presented in the next subsection [90].

In a more recent work, Burke [112, 113] has extended the guilty net to automate the separation of neurons in order to guarantee the convergence. This new network is called the "vigilant net" and uses a simpler measure inspired by the vigilance of the adaptive resonance theory. In adaptive resonance theory (ART), neuron j^* is selected as the winner upon presenting an input to the network if it satisfies two criteria: it must be the closest neuron to the input and must be sufficiently close to it. Namely,

$$O_{j^*} = \begin{cases} 1 & \text{if } \|\mathbf{X}_i - \mathbf{Y}_{j^*}\| = \min_j \|\mathbf{X}_i - \mathbf{Y}_j\| \text{ and } \|\mathbf{X}_i - \mathbf{Y}_{j^*}\| < \zeta \\ 0 & \text{otherwise} \end{cases} \tag{6.16}$$

where ζ is the vigilance parameter.

Winning neurons are now determined according to a vigilance level. The effect of the vigilance net is to constrain the ratio between the lengths of the longest and the average links in the TSP tour to be within some desirable interval.

6.6.2. The Procedure of Angéniol, Vaubois, and Le Texier

The procedure introduced by Angéniol et al. [90] is different from the guilty net in two aspects. The first difference is that the weight update equation does not have a fractional

learning rate parameter. The neighborhood function is the Gaussian kernel defined as:

$$\Lambda(j, j^*) = \frac{1}{\sqrt{2}} \exp\left(-\frac{d_{jj^*}^2}{\sigma^2}\right) \tag{6.17}$$

where σ (the standard deviation) is called the "gain parameter" and d_{jj^*} is defined as before. The value of σ is decreased at the end of each complete pass by a factor K_σ (a user-defined constant) i.e.,

$$\sigma \leftarrow K_\sigma \sigma, \qquad 0 < K_\sigma < 1. \tag{6.18}$$

When $\sigma \rightarrow \infty$, neurons on the ring move toward the city being presented with the same strength $1/\sqrt{2}$. On the other hand, as σ decreases only neurons closer to winning neuron j^* tend to move toward the city. Furthermore, as σ decreases the intensity with which the neurons move towards the city decreases. Thus the decrement of σ has the dual effect of decreasing the window size and of decrementing α, the adaptation parameter of the SOM. Typically, the algorithm begins with a high initial value of σ so that the neurons make large moves during the initial cycles. Afterwards, as σ becomes sufficiently low, the neurons on the ring settle down. It is to be noted that the only parameters that are to be adjusted are the initial value of σ, and K_σ.

The second and primary difference between this approach and the guilty net is the fact that this scheme permits creation and deletion of neurons as the algorithm proceeds. If a neuron wins the competition for two different cities in the same epoch, then a new neuron is created with the same coordinates as the winner one. This newly created neuron is inserted into the ring as the neighbor of the winner. Both of them are inhibited for the rest of the current epoch, which guarantees that they will disperse by the movements of the neighboring neurons. In the same way, a neuron is deleted, if it does not win a competition during three epochs.

The algorithm starts with only one neuron (located at the origin) in the Euclidean plane. Experimentally, the number of neurons created turns out to be less than twice the number of cities, but the algorithm terminates when the number of neurons is exactly equal to the number of cities, and when each neuron is associated with a single city.

With this model, Angéniol et al. solved the 30-city problem taken from Hopfield and Tank [95] and the five sets of 50-city problems taken from Durbin and Willshaw [101]. According to their simulation results, the algorithm found on the average solutions within 3 per cent of the optimum for the first data set. The results for the five sets of 50 cities show that, on the average, this approach performs as good as the elastic net, but is significantly faster.

There are also other approaches based on self-organizing maps in the literature which are similar to the approaches explained above. For a good review of them the reader is referred to Section 4 of [2], to Section 7.2 of [104], and to [106]. In addition, it is worth mentioning recent works by Budinich [114], and by Budinich and Rosario [115]; their neighborhood function considers not only the distances between the neurons on the ring (i.e., unlike the d_{jj*} of formula (6.17)) but also their physical distances (i.e., a function of the coordinates, such as the Euclidean distance).

7. THE KOHONEN NETWORK INCORPORATING EXPLICIT STATISTICS AND ITS APPLICATION TO THE EUCLIDEAN TRAVELING SALESMAN PROBLEM

7.1. Motivation

The primary contribution of KNIES is to incorporate statistical information *explicitly* into the well-known self-organizing map of Kohonen. The new neural network that is created has all the advantages of the traditional Kohonen network, but it is more accurate because it does not *only* depend on the *individual* data samples for its convergence. Additionally, it simultaneously incorporates the global properties of the sample points to enhance the convergence of the network.

7.2. KNIES_TSP

As it is mentioned earlier, KNIES by itself would not be too powerful in solving problems for which the SOM suffices. However, in the TSP, KNIES is a useful scheme because the quality of the final solutions depends on the transient stochastic distribution of the neurons on the ring. Use of this is made as follows.

First of all it is assumed that the number of neurons at any time is $M(t)$, and that this number can increase or decrease by the insertion or deletion of neurons. At any instant 't' we say that a neuron j located at $\mathbf{Y}_j(t)$ is associated with a point (city) $\Gamma_j(t)$ if:

$$\Gamma_j(t) = \min_i \|\mathbf{X}_i - \mathbf{Y}_j(t)\| \qquad j = 1, \ldots, M(t). \tag{7.1}$$

Thus, at this instant, $\Gamma_j(t)$ is the city closest to the neuron j, and the intention is that it is represented by $\mathbf{Y}_j(t)$.

At every time instant a city $\mathbf{X}_i$ is presented to the network. The neuron j^* closest to $\mathbf{X}_i$ is now determined as the winner and it is moved together with all its neighbors in the activation bubble toward $\mathbf{X}_i$ with a strength determined by an appropriate kernel function. In all the experiments the Gaussian kernel is used with a variance σ^2. However, rather than using the kernel associated with the coordinates of the neurons, the kernel associated with the indices is employed as described by Λ in Equation (6.17). Subsequently, the rest of the neurons outside this bubble are moved in such a way that the mean of the neurons currently on the ring coincides with the mean of the cities represented by these neurons. Notice that he mean is not forced to converge towards the mean of all the cities—it is only made to move onto the mean of the cities, the set $\mathcal{P}$, represented by the subset $\{\mathbf{X}_{\Gamma_j(t)} : j = 1, \ldots, M(t)\}$.

Observe now that the number of neurons $M(t)$ changes with time. Insertions and deletions are permitted in a manner analogous to the scheme by Angéniol et al. [90]. If a neuron wins the competition for two different cities in the same epoch, a new neuron is created with the same coordinates as the winner neuron. This newly created neuron is inserted into the ring as the neighbor of the winner. Both neuron are inhibited until they will disperse by the movements of the neighboring neurons. In the same way, a neuron is deleted, if it does not win a competition during a fixed number of (say, three) epochs (complete presentation of all the cities).

The difference between our insertion/deletion processes and Angéniol et al.'s is quite subtle. Consider the situation when a new neuron, k, is inserted because a neuron l, wins the competition twice. Thus $\mathbf{Y}_k(t)$ is exactly the same as $\mathbf{Y}_l(t)$ since they are both exactly at the same location. However, $\Gamma_k(t) \neq \Gamma_l(t)$ since they both represent completely different cities. The dispersing phase of KNIES now augments this effect since they are both in the ring and so they both contribute identically to the mean of the set of neurons although the cities which they represent are different. Thus the effect of the dispersion module causes the neurons to be at completely different locations than if they had been merely effected by Angéniol et al.'s procedure. In the same way, a neuron when deleted will no more be on the ring, and so its coordinates disappear when the mean of the neurons is made coincide with the mean of the cities they represent. Based on the above, one can argue that, in one sense, the present

scheme is a generalization of the algorithm due to Angéniol et al. because it introduces two modules, the attraction and the dispersion modules, (as opposed to a single one used by the latter)—to incorporate the learning of the position of the neurons and the identity of the cities which they represent.

The formal algorithm (**Algorithm** KNIES_TSP) works in principle just like Algorithm KNIES given in the previous chapter. The primary difference is the deletion and insertion of the neurons. It is formally given below.

Algorithm KNIES_TSP

Input : The coordinates of the N cities $\{X_i : 1 \leq i \leq N\}$.

Output : A solution to the TSP.

Parameters : ω, the width determining the window, $B_{j*}(t)$, σ, the standard deviation of the Gaussian kernel decremented as in KNIES, M is the initial number of neurons on the ring, K_σ is the parameter used to decrease the value of σ at the end of each epoch.

Notation : **Won** (j) is an array which records the number of times neuron j wins the competiton in any epoch. It is set to zero at the beginning of each epoch.

Insert (j) inserts a neuron at the same location as neuron j which is indexed as $j+1$.

Inhibit (j) is a Boolean array which indicates that neuron j is inhibited or not.

Delete (j) deletes neuron j from the current set of neurons.

Method

Begin

Initialize $M(0)$ with M and the location of the $M(0)$ neurons on a ring whose center is the mean of the cities.

Repeat

For $j = 1, \ldots, M(t)$ **Do**

Won(j) =0

Inhibit(j) = False

End For

For every point $\mathbf{X}_i \in \mathcal{P}$ **Do**

$j^* \leftarrow \text{argmin}_j \{\|\mathbf{X}_i - \mathbf{Y}_j(t)\|\}$

If (**Inhibit**(j^*)) **and** (neuron j^* overlaps with its inhibited neighbor)

go to jump

End If

Won(j^*) $\leftarrow$ **Won**(j^*) $+1$

If (**Won**(j^*) $= 2$) **Then**

MustInsert $\leftarrow$ True

Won(j^*) $= 0$

End If

$|B_{j*}(t)| = \omega\, M(t)$

For all $j \in B_{j*}(t)$ **Do**

$\Delta\mathbf{Y}_j(t) \leftarrow \Lambda(j, j^*)(\mathbf{X}_i - \mathbf{Y}_j(t))$, with

$$\Lambda(j, j^*) = \frac{1}{\sqrt{2}} \exp\left(-\frac{d_{jj^*}^2}{\sigma^2}\right)$$

$\mathbf{Y}_j(t) \leftarrow \mathbf{Y}_j(t) + \Delta\mathbf{Y}_j(t)$

End For

$\Gamma_j(t) = \min_i \|\mathbf{X}_i - \mathbf{Y}_j(t)\| \quad j = 1, \ldots, M(t)$

$\mathbf{X}_{total}(t) = \sum_j \mathbf{X}_{\Gamma_j(t)}$

$\mathbf{Y}_{total}(t) = \sum_j \mathbf{Y}_j(t)$

$\Delta\mathbf{Y}(t) = \mathbf{X}_{total}(t) - \mathbf{Y}_{total}(t)$

For all $j \notin B_{j*}$ **Do**

$\mathbf{Y}_j(t) \leftarrow \mathbf{Y}_j(t) - \frac{1}{M(t) - |B_{j^*}(t)|}\Delta\mathbf{Y}(t)$

End For

If (MustInsert) **Then**

Insert(j^*)

$M(t) \leftarrow M(t) + 1$

Inhibit(j^*) $\leftarrow$ True

Inhibit($j^* + 1$) $\leftarrow$ True

End If

jump: **End For**

For $j = 1, \ldots, M(t)$ **Do**

If (j has not won for three consecutive epochs) **Then**

Delete(j)

$M(t+1) \leftarrow M(t) - 1$

End If

End For

$\sigma \leftarrow K_\sigma \, \sigma$

$t \leftarrow t + 1$

Until Satisfied

TSP_Tour $\leftarrow$ Tour joining the cities in the order of their ring location

End Algorithm KNIES_TSP

7.3. KNIES_TSP_Global: A Simplification of KNIES_TSP

KNIES_TSP can be computationally expensive. This is just because we try to maintain the mean of the neurons to be the mean of the cities they represent, and so, we are continuously posed with the problem of identifying the city associated with each neuron. Consequently, $\{\Gamma_j(t) : j = 1, \ldots, M(t)\}$ has to be identified after the presentation of each city. A modification of KNIES_TSP (called KNIES_TSP_Global) approximates this by assuming that the migrations are always done towards the global mean of all the cities. The only difference between KNIES_TSP and KNIES_TSP_Global is that in the latter, the mean of the neurons on the ring always moves towards (not onto) the global mean of all the cities, instead of moving exactly onto the local mean only of the cities represented by the neurons currently on the ring.

This is accomplished by bookkeeping the amount by which the neurons inside the activation bubble are updated along the x and y coordinate axes. The total amount of updates along both coordinate axes which are computed by summing up the recorded quantities algebraically make up the components of a vector $\Delta \mathbf{Y}\,(t)$ which represents the total change in the coordinates of the neurons in $B_{j*}\,(t)$. This total change is equally distributed among

the neurons which are not in $B_{j*}(t)$ as follows:

$$\mathbf{Y}_j(t) = \mathbf{Y}_j(t) - \frac{1}{M(t)\cdot(M(t)-|B_{j^*}(t)|)}\Delta\mathbf{Y}(t) \qquad j \notin B_{j^*}(t) \tag{7.2}$$

The above assignment replaces their respective steps in KNIES_TSP to yield KNIES_TSP_Global. This procedure has also been implemented and tested. It is computationally faster than KNIES_TSP since it does not have to identify $\Gamma_j(t)$ after the presentation of each city, which is done by identifying the nearest city to a neuron on the ring. It is also quite accurate even though (in its transient behavior) it is theoretically a deviation from the principle motivating KNIES.

The figures 7.3.1, 7.3.2, and 7.3.3 show the evolution of the elastic band. The snapshots are taken while solving the instance `eil51` at epochs 8, 14, and 20.

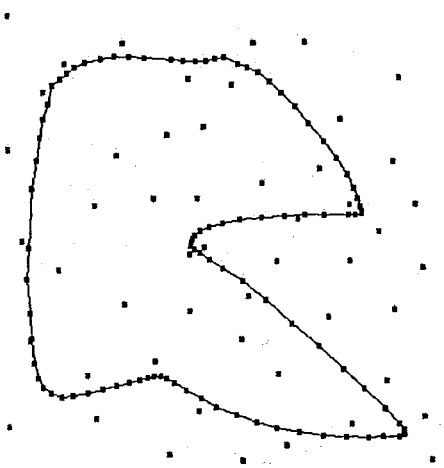


FIGURE 7.3.1. The band for instance `eil51` at epoch 8.

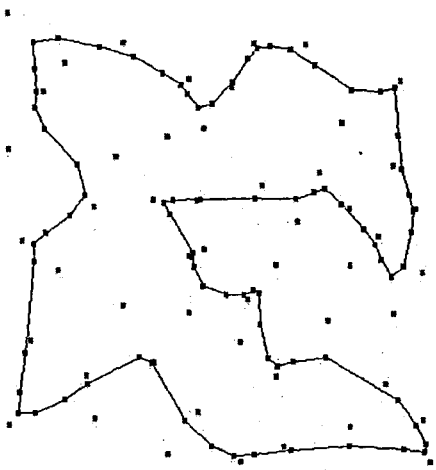


FIGURE 7.3.2. The band for instance `eil51` at epoch 14.

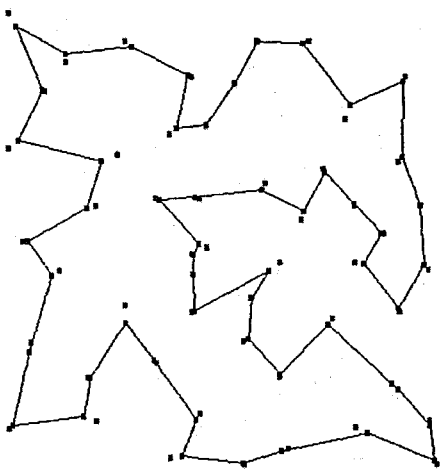


FIGURE 7.3.3. The band for instance `eil51` at epoch 20.

7.4. Computational Results

Both algorithms described in the preceding sections have been tested by using problem instances taken from TSPLIB [116]. The selected problems are geometric TSP instances which means that each city is represented by its coordinates on the two-dimensional Euclidean space and the distances among the cities are computed by using the Euclidean norm. Furthermore, the sizes of the problems range between 51 and 532. Table 7.4.1 includes the instances which make up the test bed. It is important to note that the optimum tour length is obtained by summing up the distances between the subsequent cities on the optimal tour where the distance between two cities $\mathbf{X}_i$ and $\mathbf{X}_j$ is computed by the following formula [116]:

$$d_{ij} = \lfloor \|\mathbf{X}_i - \mathbf{X}_j\| + 0.5 \rfloor \tag{7.3}$$

where $\lfloor a \rfloor$ denotes the largest integer less than or equal to a.

From this perspective, this study is also of documentary importance because, in the literature, almost all of the reported results on solution of the TSP by the neural network algorithms are based on randomly generated problems. All experiments were run on an HP-C110 workstation with 128 MByte RAM working within an HP-UX 10.2 environment.

In order to be able to compare the quality of the solutions obtained with our algorithms, all instances in the test bed were also solved by using the Pure Self-organizing Map (PSOM), the Guilty Net (GN) of Burke and Damany [110], and the approach of Angéniol, Vaubois and Le Texier (AVL) [90]. The neighborhood function which is used in all the methods is based on the lateral distance between two neurons on the ring rather than their Euclidean distance. More explicitly, the distance between the winner neuron and any other neuron on the ring is the absolute value of the difference between the indices of the two neurons. Furthermore, in all methods—except the GN which does not have it—the

TABLE 7.4.1. Problem instances used in the test bed.

Instance	Number of Cities	Optimum Tour Length
att532	532	27686
bier127	127	118282
eil51	51	426
eil76	76	538
eil101	101	629
kroA200	200	29368
lin105	105	14379
pcb442	442	50778
pr107	107	44303
pr124	124	59030
pr136	136	96772
pr152	152	73682
rat195	195	2323
rd100	100	7910
st70	70	675

parameter K_σ which is used to decrease σ at every epoch, is kept fixed at 0.8 throughout the iterations.

Table 7.4.2 summarizes the results obtained for the problem instances in the test bed. The first column specifies the instance as it is referred to in TSPLIB. The second column shows the optimal solutions for these instances. The numbers in the third to seventh column present the solutions obtained by using PSOM, GN, AVL, KNIES_TSP, (referred to as KL in the table, L standing for "local") and KNIES_TSP_Global (referred to as KG in the table, G standing for "global"), respectively. These values are the best values recorded after running the algorithms for a large number of different parameter settings. The figures in parentheses are the parameter values for which these best results are attained.

TABLE 7.4.2. Comparison of the results obtained for different algorithms instances.

Instance	**Opt. Value**	**PSOM** (M,σ,K_σ)	**GN** (λ,μ)	**AVL** (σ,K_σ)	**KL** $(M,\sigma,K_\sigma,\omega)$	**KG** $(M,\sigma,K_\sigma,\omega)$
att532	27686	No Con.	No. Con.	30217.6 (20,0.8)	29551.6 (200,30,0.8,0.15)	29569.8 (400,45,0.8,0.1)
bier127	118282	122211.7 (750,45,0.8)	155163.2 (20,0.5)	122673.9 (30,0.8)	121548.7 (100,15,0.8,0.1)	121923.7 (125,50,0.8,0.1)
eil51	426	443.9 (61,10,0.8)	470.7 (20,0.5)	443.5 (35,0.8)	438.2 (10,25,0.8,0.1)	438.2 (20,50,0.8,0.05)
eil76	538	571.2 (130,10,0.8)	614.3 (10,0.5)	571.3 (50,0.8)	564.8 (90,20,0.8,0.15)	567.5 (15,5,0.8,0.2)
eil101	629	688.7 (230,40,0.8)	771.9 (100,0.5)	671.4 (30,0.8)	658.3 (20,25,0.8,0.8)	664.4 (40,35,0.8,0.25)
kroA200	28568	30927.8 (450,25,0.8)	39370.2 (220,0.5)	30994.9 (50,0.8)	30200.8 (200,25,0.8,0.25)	30444.9 (160,10,0.8,0.05)
lin105	14379	15374.2 (310,30,0.8)	15469.5 (10,0.5)	15311.7 (35,0.8)	14664.4 (100,50,0.8,0.2)	14564.6 (100,35,0.8,0.05)
pcb442	50778	No Con.	No Con.	59649.8 (10,0.8)	56399.9 (500,30,0.8,0.1)	56082.9 (450,40,0.8,0.15)
pr107	44303	44504.3 (230,20,0.8)	80481.3 (400,0.5)	45096.4 (25,0.8)	44628.3 (100,45,0.8,0.1)	44491.1 (60,25,0.8,0.25)
pr124	59030	59612.8 (250,10,0.8)	86853.4 (200,0.5)	62050.3 (5,0.8)	59075.7 (25,25,0.8,0.10)	59320.6 (125,10,0.8,0.05)
pr136	96772	103878.0 (450,20,0.8)	135887.7 (250,0.5)	103442.3 (45,0.8)	101156.8 (75,40,0.8,0.15)	101752.4 (30,35,0.8,0.05)
pr152	73682	74804.2 (400,25,0.8)	105230.1 (620,0.5)	74641.0 (15,0.5)	74395.5 (180,30,0.8,0.05)	74629.0 (60,10,0.8,0.2)
rat195	2323	No Con.	3846.6 (260,0.5)	2681.2 (25,0.8)	2607.3 (200,25,0.8,0.25)	2599.8 (200,25,0.8,0.1)
rd100	7910	8137.9 (160,15,0.8)	8731.2 (45,0.5)	8265.8 (15,0.8)	8075.7 (80,20,0.8,0.25)	8117.4 (60,10,0.8,0.05)
st70	675	692.8 (160,15,0.8)	755.7 (30,0.5)	693.3 (10,0.8)	685.2 (40,10,0.8,0.05)	690.7 (30,45,0.8,0.05)

Figures 7.4.1, 7.4.2, and 7.4.3 illustrate the tours associated with (a) the optimal solutions and (b) the solution of KNIES_TSP for the problem instances `eil51`, `eil76`, and `eil101`, respectively.

With the exception of four cases, i.e., `lin105`, `pr107`, `pcb442`, and `rat195` the KNIES_TSP gives the best result for all instances. For the instance `eil51` both the KNIES_TSP and KNIES_TSP_Global yield the same solution whereas for the instances cited above the best solution is given by the KNIES_TSP_Global. For example, in the case of `eil76`, the optimal path length was 538. The path obtained by PSOM was 571.2, which was 6.17 per cent higher than the optimal. Similarly, the path obtained by the GN was 614.3, which was 14.18 per cent higher than the optimal. The results obtained by AVL was marginally worse than the PSOM solution—it yielded a path of 571.3. The best neural solution was obtained by KNIES_TSP which had a value of 564.8—a value 4.98 per cent higher than the optimal. This was slightly better than the solution given by KNIES_TSP_Global which yielded a value of 567.5—5.48 per cent higher than the optimal.

PSOM and GN exhibited convergence problems. Similar observations have already been reported for both methods [90, 112, 113, 117]. This problem becomes more severe as the problem size increases. Therefore, for the instances `rat195`, `pcb442`, and `att532` no solution could be obtained, and consequently the corresponding entries are missing in the table. Note that these two methods differ from the others by the lack of appropriate insertion/deletion mechanisms. PSOM begins with an initial number of neurons which should be equal to or greater than the number of cities. However, it is very unlikely that when the initial number of neurons is at least not as much as twice the number of cities, a final tour passing through the cities is reached. The reason for this nonconvergence is that the ring stabilizes at a position where two or more cities are mapped to a unique neuron while there exist some other neurons on the ring which do not represent any city at all. This problem may be overcome for small size problems by increasing the initial number of neurons. Unfortunately, this number becomes very large as the problem size increases.

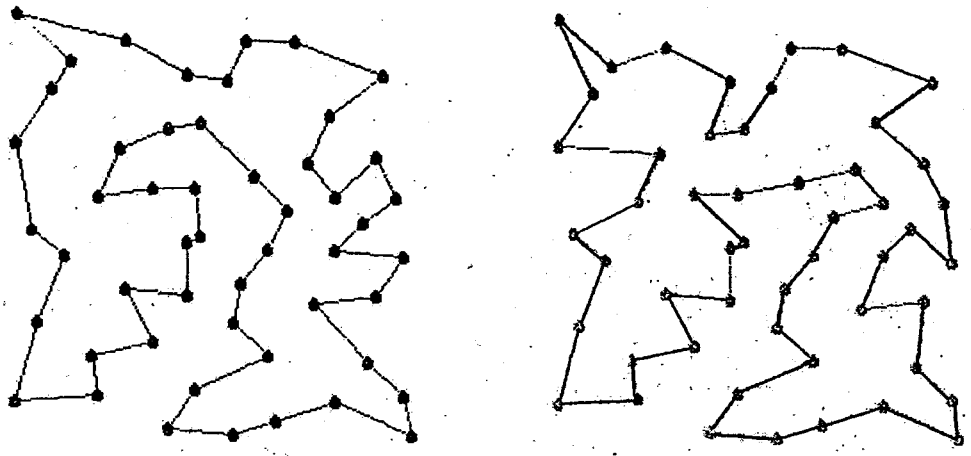
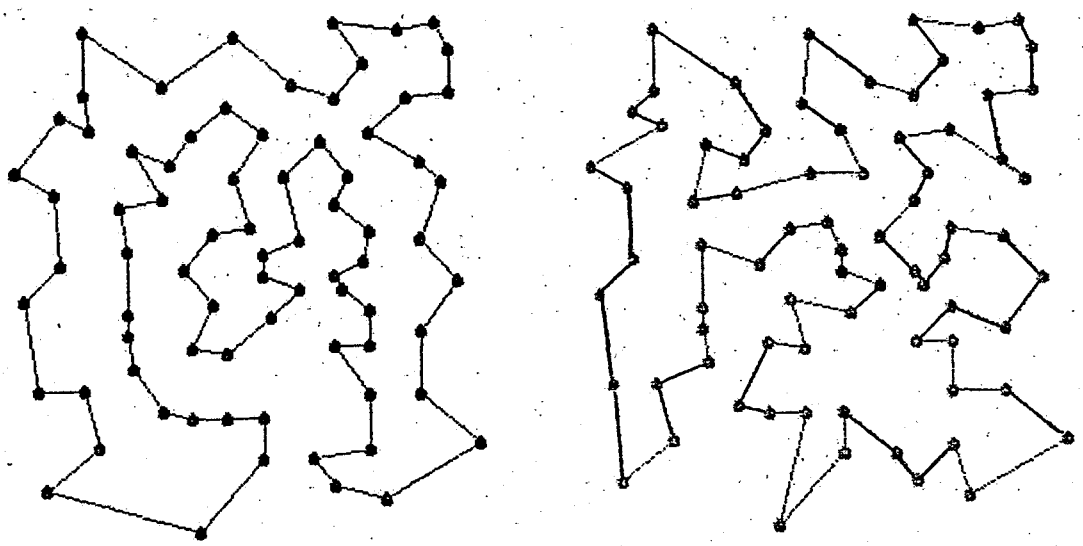


FIGURE 7.4.1. Tours for `eil51`.



(a) The optimal tour

(b) Tour obtained with KNIES_TSP

FIGURE 7.4.2. Tours for `eil76`.

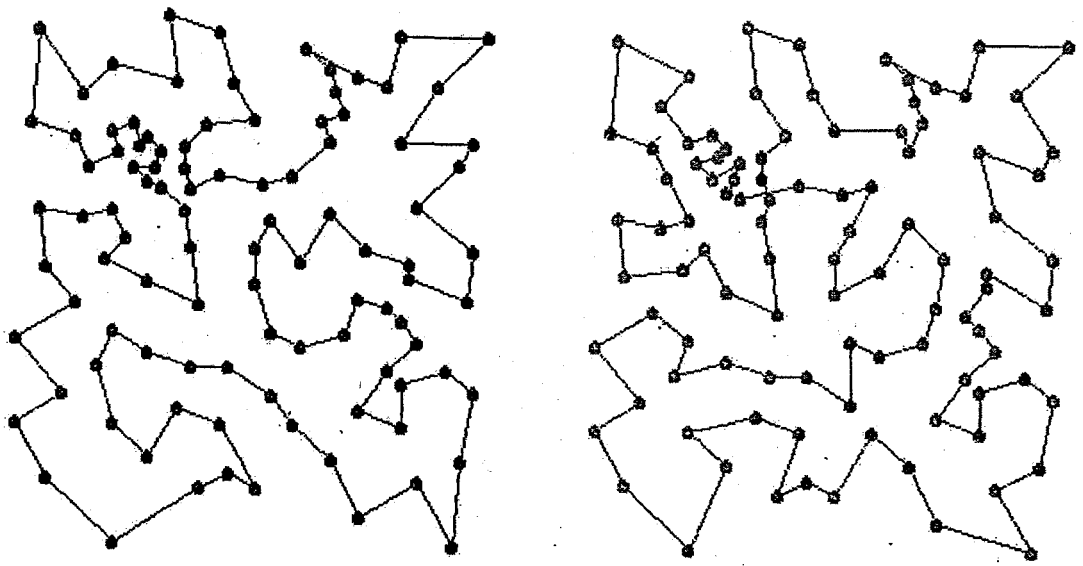


FIGURE 7.4.3. Tours for `eil101`.

In the GN approach small values of λ means the equivalence of GN to PSOM which explains the failure of convergence. Since the number of neurons is equal to the number of cities in this approach, there is no other way to facilitate the convergence than by increasing the value of the parameter λ which adjusts the weight given to the penalty term. Although this action forces the convergence of the algorithm especially for the small size problems, it deteriorates the quality of the solution by creating many crossings in the final tour. These phenomena can be best perceived in the figures 7.4.4, 7.4.5, and 7.4.6 which show the final shape of the band. Figures 7.4.4 and 7.4.6 exhibit the nonconvergence phenomenon for instances `eil51` and `eil76`, respectively. Figure 7.4.5 illustrates the case where a crossing occurs by solving the instance `eil51`. As it can be observed from the circular marks when the algorithm does not converge, there is at least one neuron (black dot) which represents more than one city (gray dot).

As can also be seen from Table 7.4.2, our implementation of the GN algorithm is run with $\mu = 0.5$ as proposed in the literature [110].

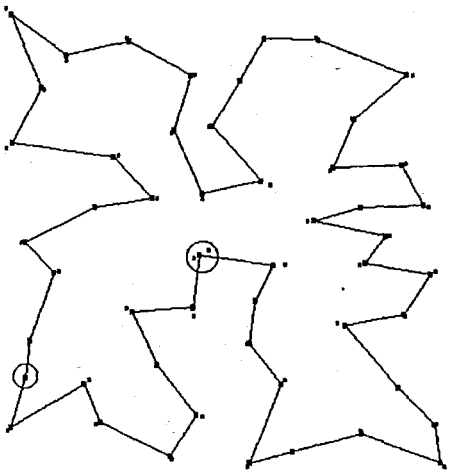


FIGURE 7.4.4. Nonconvergence for `eil51`.

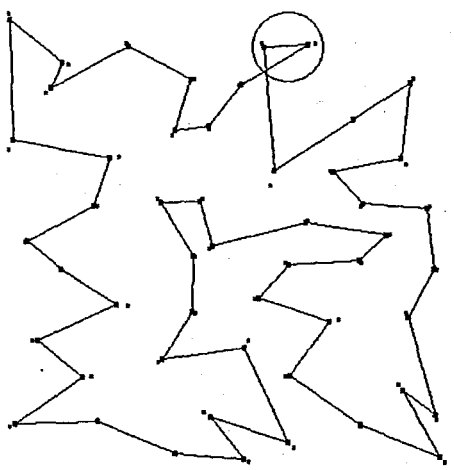


FIGURE 7.4.5. Edge crossing at `eil51`.

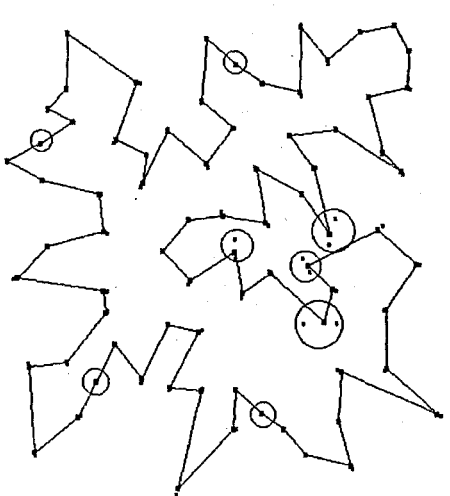


FIGURE 7.4.6. Nonconvergence for `eil76`.

Of all the algorithms reported prior to this paper AVL gives the best results. Using AVL, the tour always grows with smooth angles. At the beginning, when the number of neurons is small, the ring almost resembles a small organism running through the set of cities trying to determine the best tour (see Figure 7.4.7 and 7.4.8). Subsequently, as the process continues, the tour becomes more stabilized which is a direct consequence of the insertion/deletion mechanism (see Figure 7.4.9 and 7.4.10). The advantage of KNIES_TSP appears particularly effective at the beginning when the explicit use of the statistical information forces the tour to stabilize around the centroid of the cities, and the tour slowly extends towards the periphery.

The relative deviations from the optimal tour length are summarized in Table 7.4.3. Of course, it is impossible to make categorical statements about the various algorithms from the results presented here. However, to get an overall picture, we have computed the average deviations for PSOM, GN, AVL, KL, and KG, which are respectively 4.5 per cent, 32.61 per cent, 4.84 per cent, 2.73 per cent, and 3.17 per cent. These figures do not consider instances `att532`, `pcb442`, and `rat195` for which the pure self-organizing map (PSOM) and/or the guilty net (GN) did not converge for any value from our large parameter trial set. Based on these results, it is possible to draw the conclusion that KNIES_TSP (KL) performs better than the other methods. However, the performance of KNIES_TSP_Global (KG) is close to the KNIES_TSP's. This can also be observed from Table 7.4.2 where KG outperforms KL for `lin105`, `pcb442`, `pr107`, and `rat195`. This fact makes KG's use more attractive since it is computationally less expensive than KL.

Finally, it should be mentioned that the performance of all methods becomes worse as the number of cities increases. For example, in the case of the instance `att532`, PSOM and GN did not converge and AVL, KL, and KG yielded relative errors of 9.14 per cent, 6.74 per cent, and 6.80 per cent respectively.

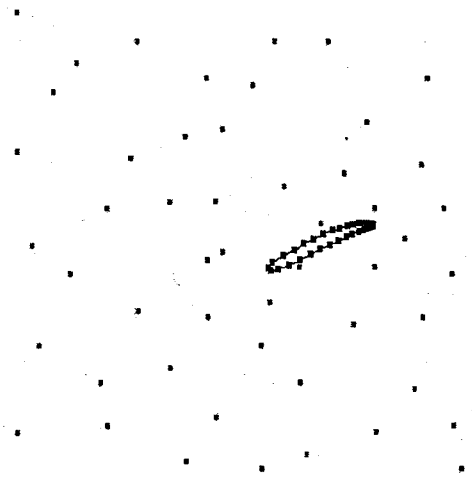


FIGURE 7.4.7. The band at epoch 1 (Angéniol et al.'s approach).

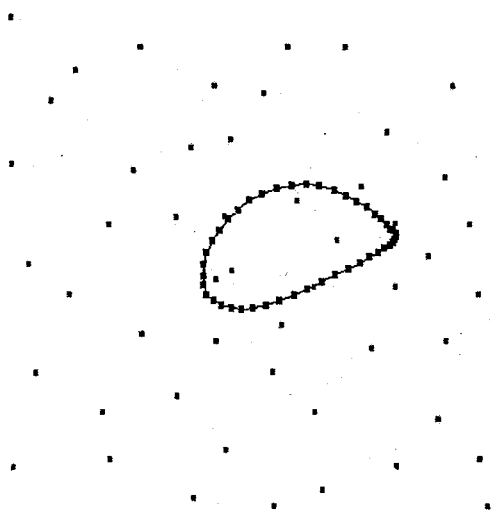


FIGURE 7.4.8. The band at epoch 3 (Angéniol et al.'s approach).

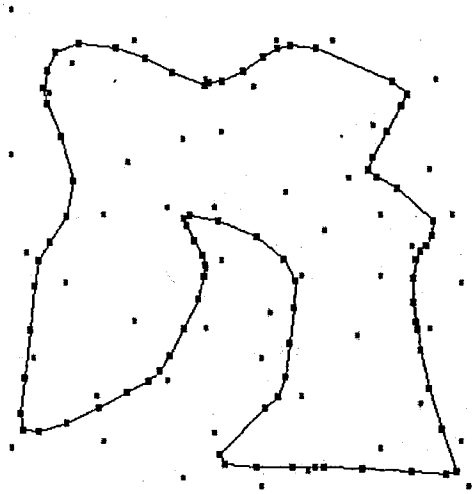


FIGURE 7.4.9. The band at epoch 10 (Angéniol et al.'s approach).

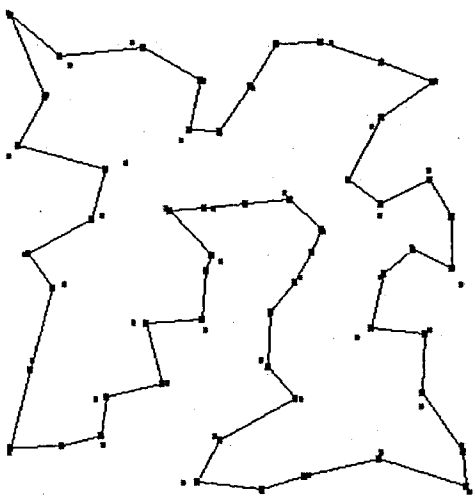


FIGURE 7.4.10. The band at epoch 16 (Angéniol et al.'s approach).

TABLE 7.4.3. Relative deviation from the optimal tour length (per cent) of the various algorithms.

Instance	PSOM	GN	AVL	KL	KG
att532	-	-	9.14	6.74	6.80
bier127	3.32	31.18	3.71	2.76	3.08
eil51	4.20	10.49	4.11	2.86	2.86
eil76	6.17	14.18	6.19	4.98	5.48
eil101	9.49	22.72	6.74	4.65	5.63
kroA200	8.26	37.81	8.49	5.72	6.57
lin105	6.92	7.58	6.49	1.98	1.29
pcb442	-	-	17.47	11.07	10.45
pr107	0.45	81.66	1.79	0.73	0.42
pr124	0.99	47.13	5.12	0.07	0.49
pr136	7.34	40.42	6.89	4.53	5.15
pr152	1.52	42.82	1.30	0.97	1.29
rat195	-	65.59	15.42	12.24	11.92
rd100	2.88	10.38	4.49	2.09	3.38
st70	2.64	11.96	2.71	1.51	2.33
Average Relative Deviations	4.50	32.61	4.84	2.73	3.17

8. KNIES AND ITS APPLICATION TO THE HAMILTONIAN PATH PROBLEM

8.1. The Hamiltonian Path Problem

The Traveling Salesman Problem has a close cousin, the Euclidean Hamiltonian Path Problem (HPP), which is also known to be *NP*-complete [1]. The HPP can be posed as follows: Given a set $\{\mathbf{X}_i : 1 \leq i \leq N\}$ of N cities with starting and terminal cities $\mathbf{X}_s$ and $\mathbf{X}_t$, respectively and distances for each pair of cities, what is the shortest path that starts at $\mathbf{X}_s$, terminates at $\mathbf{X}_t$, and visits each city exactly once?

The HPP has interesting applications such as on-line optimization of flexible manufacturing systems [118] and sequencing by hybridization of DNA [119]. The literature reports a few independent exact and heuristic algorithms, but for the most part, algorithms for the HPP utilize fast solutions to its cousin, the TSP. As in the case of all *NP*-complete problems, the exact algorithms are computationally expensive and often prohibitive even though they provide an optimal solution. The heuristic algorithms do not guarantee an optimal solution—they "quickly" find an approximate, near-optimal solution. A detailed, excellent survey of the techniques used to solve the HPP can be found in [118].

It is interesting to note that the independent solutions to the HPP are few; indeed, as explained in [93], most solutions utilize the underlying solution to the TSP. This is because the HPP can be solved using the solution to the TSP as follows:

1. The distance between $\mathbf{X}_s$ and $\mathbf{X}_t$ is set to an arbitrarily small value ($-\infty$), or,
2. A new city is added to the set of cities with distances between it and $\mathbf{X}_s$, as well as $\mathbf{X}_t$ are set to zero.

It is easy to see that since the path between X_s and X_t has to be included in the corresponding TSP, both of the above strategies will indeed yield a solution to the underlying HPP.

In this chapter the HPP is solved without resorting to an underlying TSP solution. The reasons for this are many. First motivation comes from the scientific nature of the problem itself. Second, in many real-life cases, it is well known that the cities are specified in terms of their coordinates (Cartesian, geographical or grid) and it is really quite artificial to assign a very large negative value to one single distance while all the other distances in the distance matrix are Euclidean (or geographical). A third reason involves the computation of the TSP for large problems. When the problem size is very large, it is often impossible to compute even an approximate solution to the problem in a reasonable time. In such cases, it is often advantageous to decompose the cities into clusters, and then to determine an entry and an exit city for each cluster are determined. The TSP can then be solved by computing the HPP for each cluster from the respective entry and exit points of clusters (see the next chapter). Clearly, it would be advantageous if the HPP could be solved in a fast and efficient manner for each of these clusters without having to resort to an artificial underlying TSP problem.

Apart from all of the above, the HPP is interesting in its own right because, the problem itself opens new avenues for attractive research problems. First, among these is the question of how the accuracy of an approximate solution is justified. Second, and more important, it is very interesting to note that various neural solutions to the TSP do not lead directly to corresponding solutions to the HPP. This is because the assumption that the ordering of cities has a cyclic property vanishes. Instead, the cities have to assume a linear ordering, and the two endpoints, which otherwise were close to each other on a TSP tour, can become very distant from each other on an HPP path.

Although there are many solution techniques based on neural networks (presented in Chapter 7) which yield approximate solutions to the TSP, none of them have been extended for the HPP except the one discussed below. The reason for this is primarily because the constraints that the TSP imposes permit the neurons to be placed on a ring, and force their migration to satisfy the cyclic nature of their locations with respect to each other.

Generalizing this for a "linear" structure is far from trivial since there is no straightforward way by which the energy functions can be correspondingly extended, and this is probably why no such solutions have been reported.

8.2. The Guilty Net Solution to the Hamiltonian Path Problem

Jeffries and Niznik [117] presented a scheme by which they compute the minimum cost Hamiltonian path on the plane. They did this by extending the basic ideas of the guilty net as given below.

A regression line going through the cities (i.e., data points) is first computed. The cities are then projected onto this line. The locations of these projections form the initial starting coordinates of the neurons. The algorithm then invokes the guilty net [110] by Burke and Damany (the details of the guilty net may be found in Chapter 7). In this algorithm the neurons are migrated until they stabilize at their final positions where they coincide with cities. However, the guilty net does not necessarily converge, and so when such a non-convergence condition is attained, neurons are inserted and deleted using the following technique:

1. If a neuron is not assigned to a city (data point) at the end of the guilty net algorithm, the neuron is deleted. Similarly, if multiple neurons are associated with the same city, the neuron furthermost to it is deleted.
2. If a neuron has more than one city associated with it, a new neuron is introduced, and each of these is associated with a specific city.

Finally, after the entire process converges, Jeffries and Niznik observed that the resulting solution often contained edge-crossings, and these were removed by invoking a 2-Opt-like heuristic strategy [93].

By using this simple strategy, Jeffries and Niznik [117] claim that they can yield near optimal Hamiltonian path and demonstrate their scheme for a simple data set consisting of 22 cities. It should be mentioned that their scheme does not solve the HPP since the endpoints of their path is not specified and fixed. In other words their algorithm tries to compute the shortest Hamiltonian path between all possible city pairs. Furthermore, the scheme is not an all-neural approach as it resorts to a final heuristic to remove the crossings. Finally, the scheme has the inherent drawbacks of the guilty net—it is not guaranteed to converge except after using the above mentioned Angéniol et al.'s like insertion/deletion steps.

This scheme is the only reported neural method that moves in the direction of solving the HPP without resorting to the underlying solution of the TSP. New self-organizing neural network approaches are now presented which indeed yield good approximate solutions to the HPP itself.

8.3. A Generalized Self-Organizing Map Solution to the Hamiltonian Path Problem

The self-organizing map as suggested by Angéniol et al. [90] is modified to yield a solution to the Euclidean Hamiltonian Path Problem (HPP) and is referred to as the Generalized SOM Solution (GSOM_HPP). This is done as follows.

The algorithm is initialized by using M neurons, $\{\mathbf{Y}_j(0) : 1 \leq j \leq M\}$, where M is a user specified input parameter. Two of these neurons, $\mathbf{Y}_1$ and $\mathbf{Y}_M$ are located at the coordinates of $\mathbf{X}_s$ and $\mathbf{X}_t$, the starting and terminal cities respectively, and are called the anchor neurons. The locations of the other $M-2$ neurons, $\{\mathbf{Y}_j(0) : 2 \leq j \leq M-1\}$, are on the straight line joining $\mathbf{X}_s$ and $\mathbf{X}_t$ where they are spaced equally. Thus,

$$\mathbf{Y}_j(0) = \mathbf{X}_s + (j-1)\frac{(\mathbf{X}_t - \mathbf{X}_s)}{M-1} \qquad j = 2, \ldots, M-1. \tag{8.1}$$

The cities are now presented to the algorithm one by one, and at each step a SOM procedure is invoked. If the winning neuron is an internal neuron (one which is not anchored at the endpoints) it is moved towards the presented city, and all the other neurons are moved towards the city as well using the Gaussian kernel with the specified variance whose value also decreases with the epochs. The crucial issue, of course, is that the anchor neurons at $\mathbf{X}_s$ and $\mathbf{X}_t$ are not migrated even if they become winners upon presentation of some city $\mathbf{X}_i$. It is also to be noted that at each epoch they win at least one competition which occurs upon presenting the starting and terminal cities. As usual, if a neuron does not win a competition during three complete passes (presentation of all cities once, also called cycle or epoch), it is deleted, and if a neuron wins a competition twice in the same epoch a new neuron is created at the same position. Unlike the deletion operation, the anchor neurons do participate in the creation process, but under slightly different conditions. Thus, if the winning neuron is one of the anchor neurons, ($\mathbf{Y}_1$ or $\mathbf{Y}_M$) a new neuron is created instantaneously at the same location. Unlike the anchor neuron, however, the newly created one is permitted to migrate.

When the algorithm terminates, the number of neurons can be larger than the number of cities because any city can be represented by more than one neuron located exactly on the city at the final configuration. Although this has no effect on the length of the final Hamiltonian path (the distance between two such neurons is zero), duplicate neurons may be eliminated by a pass over all the neurons.

8.4. KNIES_HPP: A KNIES Solution to the Hamiltonian Path Problem

As mentioned before, KNIES by itself would not be too powerful in solving problems for which the SOM suffices. This is because the distribution of the neurons will ultimately follow the distribution of the cities whether the global statistical properties are utilized or not. However, in the case of the TSP, KNIES is a useful scheme because the quality of the final solution depends on the transient stochastic distribution of the neurons. KNIES shall now be adapted for the HPP.

First of all, the initialization process is considered in order to determine the location of the neurons. As in the case of GSOM_HPP, the algorithm is initialized by using M neurons, where M is a user specified input parameter. Two of these neurons are located at the coordinates of $\mathbf{X}_s$ and $\mathbf{X}_t$, the starting and terminal cities respectively, and are the anchor neurons. The remaining neurons, namely $\{\mathbf{Y}_j(t) : 2 \leq j \leq M-1\}$, are first initialized according to equation (8.1) at $t = 0$. Then they are moved so that their mean falls exactly on the global mean of the cities. The intention is to maintain the mean of the neurons to be an invariant. This is done as follows. If $\overline{\mathbf{X}}$ is the mean of the coordinates of the cities, and $\overline{\mathbf{Y}}(0)$ is the mean of the coordinates of the neurons, $\mathbf{Y}_j(0)$ is set as:

$$\mathbf{Y}_j(0) = \mathbf{Y}_j(0) + \frac{M}{M-2}\left(\overline{\mathbf{X}} - \overline{\mathbf{Y}}(0)\right) \qquad 2 \leq j \leq M-1. \tag{8.2}$$

It is easy to see that as a result of the above assignment, the new $\{\mathbf{Y}_j(0) : 2 \leq j \leq M-1\}$ are all located on the same side of the line joining $\mathbf{X}_s$ and $\mathbf{X}_t$, and that they are located on the side where the global mean falls. This is distinct from the scheme of Jeffries and Niznik [117], where the initial neurons fall on the regression line. Furthermore, the mean of the cities and that of the neurons coincide and thus, we can attempt to maintain this as an invariant using the two modules of KNIES.

To show how this is done, it is assumed that the number of neurons at any time is $M(t)$, and that this number can be increased or decreased by the insertion or deletion of neurons. At any instant 't' a neuron j located at $\mathbf{Y}_j(t)$ is associated with a city $\Gamma_j(t)$ if:

$$\Gamma_j(t) = \min_i \|\mathbf{X}_i - \mathbf{Y}_j(t)\| \qquad j = 1, \ldots, M(t). \tag{8.3}$$

Thus, at this instant, $\Gamma_j(t)$ is the city closest to the neuron j, and the intention is that it is represented by $\mathbf{Y}_j(t)$.

At every time instant a city $\mathbf{X}_i$ is presented to the network. The neuron j^* closest to $\mathbf{X}_i$ is now determined as the winner and both this neuron as well as all its neighbors in the activation bubble are moved toward $\mathbf{X}_i$ with a strength determined by an appropriate kernel function. The Gaussian kernel that is used to define the neighborhood function in the previous chapter is employed again for running the experiments. However, the lateral distance between the neurons (the difference of the neuron indices) is used to define the kernel instead of the neuron coordinates as given by Λ as in (6.17). Decrementing σ takes care of both decreasing the width of activation bubble and also of the adaptation constant, $\alpha(t)$ which is used in the update equation.

The dispersing phase is as follows: Let $\mathbf{X}_{total}(t)$ be the vector sum of the coordinates of the cities represented by the neurons and $\mathbf{Y}_{total}(t)$ be the vector sum of the coordinates of the neurons at any epoch t. In order to bring the mean of the cities and the mean of the neurons to the same position, the difference between the two vector sums is evenly distributed among those neurons which are not in the activation bubble. It is interesting to note that unlike the TSP (where the neurons are either inside the bubble or outside it), in the case of the HPP these neurons fall either to the left of the bubble or to its right. Let $L(t)$ and $R(t)$ be the number of neurons respectively on the left and right hand sides of the bubble at any epoch t. These $L(t) + R(t)$ neurons are moved in such a way that the mean of the neurons currently on the band coincides with the mean of the cities they represent. Notice that we do not make the mean converge towards the mean of all the cities - it is only made to move onto the mean of the cities represented by the subset $\{\mathbf{X}_{\Gamma_j(t)} : j = 1, \ldots, M(t)\}$ of the set $\mathcal{P}$. It can be easily verified that this is achieved by updating:

$$\mathbf{Y}_j(t) = \mathbf{Y}_j(t) + \frac{\mathbf{X}_{total}(t) - \mathbf{Y}_{total}(t)}{L(t) + R(t)} \qquad j \notin B_{j^*}(t). \tag{8.4}$$

The above is surprising, because one would expect that the changes made on the left are proportional to $L(t)$, and that the changes made on the right are proportional to $R(t)$. It is indeed the case that the latter is true, but when the total change on the left is averaged by $L(t)$, and the total change on the right is averaged by $R(t)$, it turns out that all the neurons

"outside" the bubble are migrated by exactly the same amount. Thus, the dispersing phase is really a very straightforward operation.

The insertion/deletion strategy for internal neurons is the same as the one presented in the previous chapter for KNIES_TSP. The strategy for the anchor neurons is slightly different. If the winning neuron is one of the anchor neurons, ($\mathbf{Y}_1$ or $\mathbf{Y}_M$) a new neuron is created instantaneously at the same location. Unlike the anchor neurons, however, the newly created one is permitted to migrate. The KNIES_HPP algorithm is given below:

Algorithm KNIES_HPP

Input : The coordinates of the N cities $\{\mathbf{X}_i : 1 \leq i \leq N\}$, $\mathbf{X}_s$ and $\mathbf{X}_t$.

Output : A solution to the Euclidean HPP.

Parameters : W, the width determining the window, $B_{j*}(t)$, σ, the standard deviation
: of the Gaussian kernel decremented as in KNIES, M is the initial number of neurons on the ring.

Notation : **Won** (j) is an array which records the number of times neuron j wins the competiton in any epoch. It is set to zero at the beginning of each epoch.
Insert (j) inserts a neuron at the same location as neuron j which is indexed as $j+1$.
Inhibit (j) is a Boolean array which indicates that neuron j is inhibited or not.
Delete (j) deletes neuron j from the current set of neurons.

Method

Begin

Initialize $M(0)$ with M and the location of the $M(0)$ neurons on a piecewise linear line as:

$\mathbf{Y}_1(0) = \mathbf{X}_s$

$\mathbf{Y}_M(0) = \mathbf{X}_t$

For $j = 2, \ldots, M-1$ **Do**

$\mathbf{Y}_j(0) = \mathbf{X}_s + (j-1) \cdot \frac{(\mathbf{X}_t - \mathbf{X}_s)}{M-1}$

End For

For $j = 2, \ldots, M-1$ **Do**

$\mathbf{Y}_j(0) = \mathbf{Y}_j(0) + \frac{M}{M-2}\left(\overline{\mathbf{X}} - \overline{\mathbf{Y}}(0)\right)$

End For

$t = 0$

Repeat

For $j = 1, \ldots, M(t)$ **Do**

Won(j) =0

Inhibit(j) = False

End For

For every point $\mathbf{X}_i \in \mathcal{P}$ **Do**

$j^* \leftarrow \text{argmin}_j \|\mathbf{X}_i - \mathbf{Y}_j(t)\|$

If $((j^* = 1)$ or $(j^* = M(t)))$ **Then**

MustInsert $\leftarrow$ True

End If

If (Inhibit(j^*)) **and**

(neuron j^* overlaps with its inhibited neighbor) **Then**

go to jump

End If

Won(j^*)$\leftarrow$**Won**(j^*)+1

If (Won(j^*) $= 2$) **Then**

MustInsert $\leftarrow$ True

Won(j^*) $= 0$

End If

For all $j \in B_{j*}(t)$ **Do**

$\Delta\mathbf{Y}_j(t) \leftarrow \Lambda(j, j^*)(\mathbf{X}_i - \mathbf{Y}_j(t))$, with

$$\Lambda(j, j^*) = \frac{1}{\sqrt{2}} \exp\left(-\frac{d_{jj^*}^2}{\sigma^2}\right)$$

$\mathbf{Y}_j(t) \leftarrow \mathbf{Y}_j(t) + \Delta\mathbf{Y}_j(t)$

End For

$\Gamma_j(t) : j = 1, \ldots, M(t)$

$\mathbf{X}_{total}(t) = \sum_j \mathbf{X}_{\Gamma_j(t)}$

$\mathbf{Y}_{total}(t) = \sum_j \mathbf{Y}_j(t)$

$\Delta\mathbf{Y}(t) = \mathbf{X}_{total}(t) - \mathbf{Y}_{total}(t)$

For all $j \notin B_{j*}(t)$ **Do**

$\mathbf{Y}_j(t) \leftarrow \mathbf{Y}_j(t) + \frac{1}{M(t)-|B_{j^*}(t)|}\Delta\mathbf{Y}(t)$

End For

If (MustInsert) **Then**

Insert(j^*)

$M(t) \leftarrow M(t)+1$

If ($j \neq 1$) **and** ($j \neq M(t)$) **Then**

Inhibit(j^*) $\leftarrow$ True

Inhibit(j^*+1) $\leftarrow$ True

End If

End If

jump: **End For**

For $j = 2, \ldots, M(t)-1$ **Do**

If (j has not won for three consecutive epochs) **and** ($j \neq 0$) **Then**

Delete(j)

$M(t+1) \leftarrow M(t)-1$

End If

End For

Decrement σ

$t \leftarrow t+1$

Until Satisfied

END Algorithm KNIES_HPP

8.5. KNIES_HPP_Global : A Simplification of KNIES_HPP

KNIES_HPP has the same drawback as KNIES_TSP; it can be computationally expensive. This is just because we try to maintain the mean of the neurons to be the mean of the cities they represent, and so, we are continuously posed with the problem of identifying the city associated with each neuron. A modification of KNIES_HPP (called KNIES_HPP_Global) approximates this by assuming that we attempt to always migrate towards the global mean of all the cities. The only difference between KNIES_HPP and KNIES_HPP_Global is that in the latter, the mean of the neurons on the band always moves

towards (not onto) the global mean of all the cities, instead of moving exactly onto the local mean only of the cities represented by the neurons currently on the band. This is accomplished again by applying Equation (7.2) on page 125. The modified update scheme replaces its respective steps in KNIES_HPP to yield KNIES_HPP_Global. This procedure is computationally faster than KNIES_HPP since it does not have to identify the cities associated with the neurons currently on the band after the presentation of each city. Figures 8.5.1, 8.5.2, and 8.5.3 show the evolution of the elastic band. The snapshots are taken while solving the instance `eil51` at epochs 5, 10, and 17.

8.6. Measuring the Performance of the New Heuristics

The first major problem encountered when the neural HPP algorithms are tested is determining their accuracy. For HPP there does not exist a standard test library with known optimal solutions which is the case with the TSP. Although it is possible to transform a TSP instance with a known optimal solution (i.e., an optimal tour) into an HPP instance with an optimal path whose length is equal to the length of an optimal tour [120], any such transformation is useful for methods working with intercity distances rather than the city coordinates; and this is not the case with GSOM_HPP, KNIES_HPP and KNIES_HPP_Global as well as any other Kohonen type SOM method which solves the HPP and TSP. We are therefore forced to resort to statistical means to justify the accuracy of the solution. To do this the classical result due to Fisher and Tippett [121] is considered which has been adapted by various authors [122–125] to compare the efficiency of experimental results in combinatorial optimization.

8.6.1. Confidence Intervals for the Length of an Optimal Hamiltonian Path

Consider a set of S independent samples $\Sigma_1, \Sigma_2, \ldots, \Sigma_S$ each of size m drawn from a parent population which is both continuous and bounded from below by μ. If x_i is the

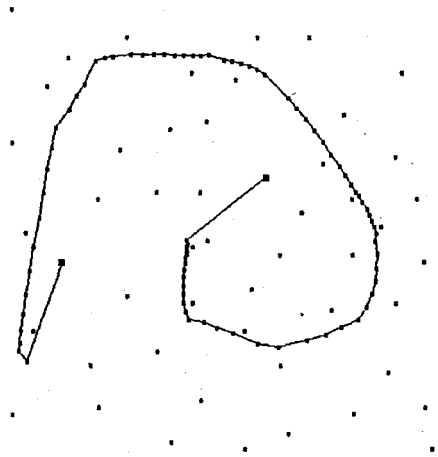


FIGURE 8.5.1. The band for instance `eil51` at epoch 5.

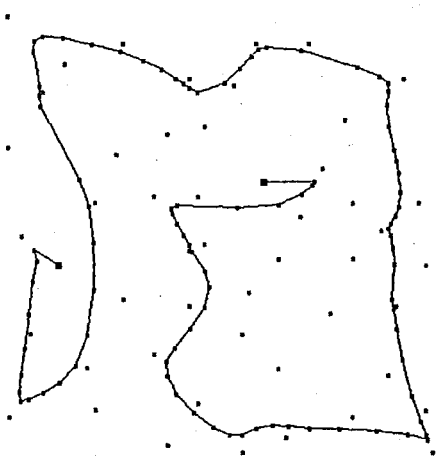


FIGURE 8.5.2. The band for instance `eil51` at epoch 10.

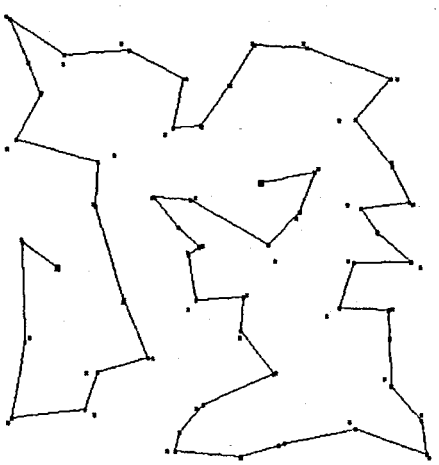


FIGURE 8.5.3. The band for instance `eil51` at epoch 17.

smallest value in sample i, and we let

$$v = min\{x_i : 1 \leq i \leq S\}, \tag{8.5}$$

then, as m gets larger the distribution for x_i converges to a Weibull distribution with location parameter γ [121]. The probability density function of the Weibull distribution is:

$$f(x) = \begin{cases} \left(\frac{\gamma}{\beta}\right)\left(\frac{x-\mu}{\beta}\right)^{\gamma-1} e^{-\overleftarrow{\frac{x-\mu}{\beta}}^{\omega_\gamma}} & \text{if } x \geq \mu \\ 0 & \text{otherwise} \end{cases} \tag{8.6}$$

where $\mu \geq 0$ is the location parameter, $\beta > 0$ is the scale parameter, and $\gamma > 0$ is the shape parameter. It is often easier to work with the cumulative Weibull distribution function given by:

$$F(x) = 1 - e^{-\overleftarrow{\frac{x-\mu}{\beta}}^{\omega_\gamma}} \tag{8.7}$$

The Weibull distribution with shape parameter $\gamma = 1$ is the same as the shifted (two-parameter) exponential distribution. Furthermore if the location parameter $\mu = 0$, then it becomes the exponential distribution with scale parameter β.

By arriving at S independent estimates (each of which is a local optimal value) of the length of an optimal Hamiltonian path, it may be argued that:

$$Pr[v - \beta \leq \mu \leq v] \approx 1 - e^{-S} \tag{8.8}$$

where S is the sample size, and μ and β are the location and scale parameters, respectively.

This is exactly the approach introduced first by Golden and Alt [123], and applied and tested for the TSP by Golden and Stewart [124] later. Los and Lardinois suggest two modifications on this approach [125]. First, instead of using S local optima, only a subset of size $S' \leq S$ *distinct* local optima and their values $x_{(1)} \leq x_{(2)} \leq ... \leq x_{(S')}$ are used to fit a Weibull distribution. The reason for this modification is the fact that the Fisher-Tippett theorem assumes the independence of S samples [121]; nevertheless having identical local optima in the sets $\Sigma_1, \Sigma_2, \ldots, \Sigma_S$ is equivalent to repeating the same sample several times. Second, allowing T to be any real number, they developed the following formula:

$$Pr\left[v - \beta/T \leq \mu \leq v\right] \approx 1 - e^{-S'/T^{\gamma}} \tag{8.9}$$

The main advantage of this formula over the one suggested by Golden and Alt (i.e., formula (8.8)) is its explicit dependence on the level of confidence. In other words, $100(1-\alpha)$ per cent confidence can be achieved by letting

$$T = \left(-\frac{S'}{\ln \alpha}\right)^{1/\gamma}. \tag{8.10}$$

However, there is a specific problem with the Los and Lardinois formula: it involves the shape parameter γ. According to the computational experiments reported in the literature, the computation of good estimators (e.g. maximum likelihood estimators (MLE)) for three-parameter Weibull distribution is not a trivial task [126]. The difficulty is mainly due to the location parameter μ. Once its value is determined the estimation of the other parameters (the scale parameter β and the shape parameter γ) becomes fairly easy.

In addition to MLEs some non-MLEs are also suggested for the shape parameter γ of the Weibull distribution [127–129]. These estimators are tested by Derigs while computing

confidence limits for the global optimum of the TSP and the quadratic assignment problem (QAP) [122]. Based on the results he advocates the use of Golden and Alt formula with the following estimators of the shape and scale parameters:

$$\widehat{\gamma} = \frac{\left(x_{(1)} \cdot x_{(S)} - x_{(2)}^2\right)}{\left(x_{(1)} + x_{(S)} - 2x_{(2)}\right)} \tag{8.11}$$

$$\widehat{\beta} = x_{(\lfloor 0.63S \rfloor + 1)} - \widehat{\gamma}. \tag{8.12}$$

These estimators are introduced by Zanakis [129]. Here $x_{(1)}$, $x_{(2)}$ and $x_{(S)}$ are respectively the smallest, the second smallest and the largest of S sample values. The main argument was that the lower confidence limits obtained by using these non-MLEs with Golden and Alt formula on the test problems were the greatest lower bounds on the length of optimal TSP tours (the gap between the lower bound and the length of an optimal tour) is the smallest. This was not the case for Los and Lardinois interval given with formula (8.10): it failed to give confidence intervals which cover an optimal value significantly more often than expected.

Given the intercity distances and endpoints of an Hamiltonian path, an arbitrarily small (i.e., $-\infty$) distance can be assigned between the first and last cities and one of the TSP heuristics is used with random initial conditions to generate S local optimal values. Note that, if the artificial distance is excluded, the sum of the distances associated with the arcs on the Hamiltonian path computed by one of the TSP heuristics can be used to estimate a confidence interval for the length of an optimal Hamiltonian path. In this work, being convinced by Derigs' results, Golden and Alt formula with non-MLEs, (Equations (8.11) and (8.12)) are used. Because of the reasons stated above, it becomes possible to get good interval estimates for the length of an optimal Hamiltonian path which can be used for measuring the performance of GSOM_HPP, KNIES_HPP and KNIES_HPP_Global.

Summing up the above arguments, in a given TSPLIB instance, first two cities are chosen randomly as the starting and terminal points of an Hamiltonian path. Then an arbitrarily small intercity distance is assigned for them and one of the known TSP heuristics is run many times with different initial conditions. Each run produces a local optimum, namely a near optimal Hamiltonian path whose length corresponds to one of the S sample values (i.e., $x_i : 1 \leq i \leq S$) used in the estimation of $100(1 - e^{-S})$ per cent confidence intervals for the length of an optimal Hamiltonian path. Note that once the endpoints are randomly selected they remain unchanged during the replications.

8.6.2. Verification of the Interval Estimates

A typical quality measure for a heuristic solution is the relative deviation of its value from the value of an optimal solution. Using such a measure the average of the relative errors obtained on a standard test library can be utilized as a good experimental performance measure for the heuristic: the smaller it is, the better the performance of the heuristic. This approach is adopted with one exception; we use the lower limit of the interval estimate instead of the length of an optimal Hamiltonian path to determine relative deviations. The essential valid reason for the use of lower confidence limits is the fact that optimal Hamiltonian paths are unavailable and very expensive to compute. Furthermore, it is clear that the estimation strategy is definitely more robust if the optimal value remains within the confidence interval.

To verify the above method the assumption that the confidence interval covers the length of the optimal path is tested first. For this purpose some of the TSPLIB instances are taken into account whose optimum tours are available. First, an edge on the given optimum tour is selected randomly, its length is made arbitrarily small and the remaining edges of the tour are treated as an optimal Hamiltonian path. The length of this path is simply the length of the optimum tour less the original length of the randomly selected edge, and its endpoints are the edge's endpoints. Then confidence intervals are computed by using one of the known TSP heuristics with the modified instance (the length of the randomly selected edge is very small and this guarantees their endpoints to be adjacent on a tour). To achieve this six TSP

heuristics are used. They are tour construction procedures: *nearest neighbor* (NNb), *nearest insertion* (NI), *farthest insertion* (FI), *cheapest insertion* (CI), *random insertion* (RI), and *Clarke-Wright savings method* (CW). How these heuristics construct a tour from scratch is summarized below. In all the heuristics c_{ij} denotes the distance between nodes i and j.

Nearest Neighbor (NNb)

Step 1. Start with any node as the beginning of a path.

Step 2. Find the node closest to the last node added to the path. Add this node to the path.

Step 3. Repeat Step 2 until all nodes are contained in the path. Then, join the first and last nodes.

Nearest Insertion (NI)

Step 1. Start with a subgraph consisting of node i only.

Step 2. Find node k such that c_{ik} is minimal and form subtour $i - k - i$.

Step 3. *Selection step.* Given a subtour, find node k not in the subtour closest to any node in the subtour.

Step 4. *Insertion step.* Find the edge (i, j) in the subtour which minimizes $c_{ik} + c_{kj} - c_{ij}$. Insert k between i and j.

Step 5. Go to Step 3 unless we have Hamiltonian cycle.

Farthest Insertion (FI)

Step 1. Start with a subgraph consisting of node i only.

Step 2. Find node k such that c_{ik} is maximal and form subtour $i - k - i$.

Step 3. *Selection step.* Given a subtour, find node k not in the subtour farthest from any node in the subtour.

Step 4. *Insertion step.* Find the edge (i, j) in the subtour which minimizes $c_{ik} + c_{kj} - c_{ij}$. Insert k between i and j.

Step 5. Go to Step 3 unless we have Hamiltonian cycle.

Cheapest Insertion (CI)

Step 1. Start with a subgraph consisting of node i only.

Step 2. Find node k such that c_{ik} is minimal and form subtour $i - k - i$.

Step 3. Find (i, j) in subtour and k not, such that $c_{ik} + c_{kj} - c_{ij}$ is minimal and, then insert k between i and j.

Step 4. Go to Step 3 unless we have Hamiltonian cycle.

Random Insertion (RI)

Step 1. Start with a subgraph consisting of node i only.

Step 2. Find node k such that c_{ik} is minimal and form subtour $i - k - i$.

Step 3. *Selection step.* Given a subtour, randomly select node k not in the subtour to enter the subtour.

Step 4. *Insertion step.* Find the edge (i, j) in the subtour which minimizes $c_{ik} + c_{kj} - c_{ij}$. Insert k between i and j.

Step 5. Go to Step 3 unless we have Hamiltonian cycle.

Clarke-Wright Savings Method (CW)

Step 1. Select any node as a base node and set up the $n - 1$ tours consisting of two. nodes each.

Step 2. For every pair of tours T_1 and T_2 compute the savings that is achieved if the tours are merged by deleting in each tour an edge to the base node and connecting the two open ends. More precisely, if (i, b) and (j, b) are edges in different tours then these tours can be merged by eliminating (i, b) and (j, b) and adding edge (i, j) resulting in a savings of $c_{ib} + c_{jb} - c_{ij}$.

Step 3. Merge the two tours giving the largest savings.

Step 4. Go to Step 2 as long as more than one tour is left.

Except the Clarke-Wright savings method and the cheapest insertion heuristics the time complexity of all the heuristics is $O(n^2)$. Clarke-Wright savings method and the cheapest insert heuristic both take $O(n^2 \log_2 n)$. Detailed information on the performances

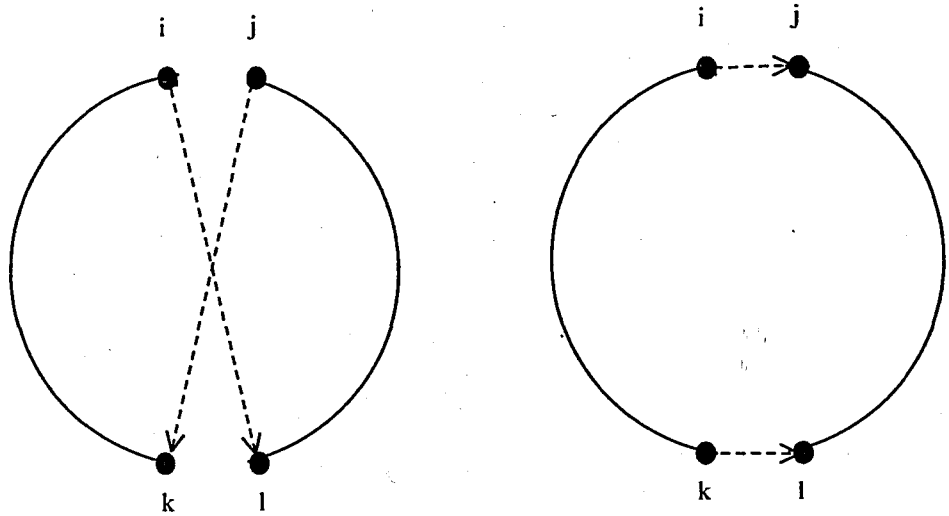


FIGURE 8.6.1. Exchange of edges (i,k) , (j,l) for edges (i,j) , (k,l).

of these heuristics can be found in an excellent work by Reinelt [93]. They are not only easy to implement but also the ones used in most of the previous studies [123,124]. Except for the Clarke-Wright savings method, the 2-Opt exchange heuristic [93] is applied on the solution generated by the heuristics for further improvement as advised in the literature [122,124].

The 2-opt exchange heuristic is one of the most famous simple local search algorithms. The other one is the 3-opt. Local search algorithms are specified in terms of a class of operations (exchanges or moves) that can be used to convert one tour into another. Given a feasible tour, the algorithm repeatedly performs operations from the given class, so long as each reduces the length of the current tour, until a tour is reached for which no operation yields an improvement. This tour is a locally optimal tour. This approach may also be viewed as a neighborhood search process, where each tour has an associated neighborhood of adjacent tours, i.e., those that can be reached in a single move,and one continually moves to a better neighbor until no better neighbors exist. The 2-opt exchange heuristic deletes two edges in the current tour, namely breaks the tour into two paths, then reconnects those paths in the other possible way. Figure 8.6.1 shows an example of a 2-opt exchange.

TABLE 8.6.1. Verification of the statistical method for procuring confidence intervals for the length of optimal Hamiltonain paths between X_s and X_t. Cases where the optimal value lies in the interval are marked with an "*".

Instance	**Endpoints** (X_s, X_t)	**Opt. Path Length**	**NNb**	**NI**	**FI**
att532	(112, 96)	27408.88	[27982.03, 28359.97] 28701.49	[29897.31, 30377.94] 30763.05	[28361.04, 28946.83] 29527.43
berlin52	(24, 48)	7528.55	[7318.04, 7643.15] * 7865.05	[7815.91, 8195.21] 8559.92	[6942.44, 7528.55] * 7977.02
eil51	(41, 13)	420.55	[416.41, 424.41] * 433.65	[417.4, 437.3] * 451.16	[399.50, 420.81] * 438.90
eil76	(11, 53)	537.39	[525.14, 543.75] * 560.89	[565.49, 583.07] 598.77	[523.23, 551.16] * 574.30
eil101	(54, 55)	633.83	[635.51, 651.09] 663.61	[665.60, 677.40] 694.02	[630.72, 653.25] * 673.53
kroA100	(34, 83)	21139.47	[20755.17, 21249.41] * 21647.81	[20973.48, 22432.01] * 23643.20	[20658.88, 21484.37] * 22244.06
lin105	(87, 66)	13999.78	[13752.08, 14155.66] * 14473.51	[13758.57, 14568.89] * 15082.25	[13462.44, 14211.07] * 14841.26
pcb442	(226, 411)	50591.51	[51406.66, 52378.21] 53183.72	[54252.59, 54830.73] 55306.98	[53296.61, 55108.42] 56579.12

All the TSP heuristics have been tested for the same problem instance (i.e., the endpoints of the randomly selected edge are the terminal cities) with fifty different initializations. In other words, 50 values, generated by each one of these heuristics, are used in the estimation of the shape and scale parameters of the Weibull distribution according to formulae (8.11) and (8.12). The estimates are then employed in establishing confidence limits for the length of an optimal Hamiltonian path according to Equation (8.8). Note that six heuristics yielded six samples of size 50. Hence six different confidence intervals are obtained. $S = 50$ is selected as suggested by Golden and Stewart [124]. This implies almost 100 per cent confidence, since $1 - e^{-S}$ is practically unity for $S = 50$.

The results as well as the test bed for the verification experiments are summarized in Tables 8.6.1 and 8.6.2. The first columns of both tables contain references for the TSPLIB

TABLE 8.6.2. Verification of the statistical method for procuring confidence intervals for the length of optimal Hamiltonain paths between X_s and X_t. Cases where the optimal value lies in the interval are marked with an "$*$".

Instance	**Endpoints** (X_s, X_t)	**Opt. Path Length**	**CI**	**RI**	**CW**
att532	(112, 96)	27408.88	[29112.36, 29880.26] 30545.77	[28344.20, 29056.11] 29668.59	[28854.35, 29585.69] 30220.21
berlin52	(24, 48)	7528.55	[7369.04, 7825.38] * 8247.02	[6787.68, 7528.55] * 8134.99	[7315.33, 7802.52] * 8213.32
eil51	(41, 13)	420.55	[400.02, 425.17] * 447.30	[414.51, 429.37] * 443.13	[403.77, 424.82] * 442.81
eil76	(11, 53)	537.39	[516.25, 551.94] * 581.56	[540.28, 560.23] 577.89	[529.48, 556.29] * 579.53
eil101	(54, 55)	633.83	[643.27, 664.32] 682.47	[616.88, 649.96] * 677.04	[634.43, 661.94] 684.88
kroA100	(34, 83)	21139.47	[20105.87, 22047.52] * 23617.26	[19968.92, 20164.70] 22187.48	[19718.48, 21480.60] * 22928.
lin105	(87, 66)	13999.78	[13194.70, 14343.56] * 15307.39	[13134.91, 14727.06] * 15157.35	[13729.92, 14433.60] * 15051.33
pcb442	(226, 411)	50591.51	[51927.08, 54248.71] 56091.78	[53599.85, 55162.32] 56480.68	[52382.87, 545117.87] 55593.77

TABLE 8.6.3. Confidence intervals for the length of optimal Hamiltonian paths between X_s and X_t.

Instance	Endpoints (X_s, X_t)	NNb	NI	FI
att532	(239, 451)	[28054.48, 28486.41] 28904.50	[29347.82, 29686.19] 30025.90	[27949, 28666.18] 29264.74
berlin52	(10, 31)	[7394.56, 7549.32] 7675.93	[7254.99, 7699.84] 7977.14	[7096.24, 7504.68] 7844.28
bier127	(14, 122)	[114970.39, 119102.93] 122972.70	[126551.31, 127988.16] 129220.59	[115506.66, 120347.06] 1243553.15
eil51	(5, 14)	[405.69, 422.95] 437.67	[400.13, 427.01] 451.79	[422.36, 430.90] 439.65
eil76	(60, 38)	[526.20, 548.07] 566.28	[560.02, 576.06] 590.07	[527.06, 555.70] 579.16
eil101	(7, 50)	[631.69, 649.28] 663.71	[634.38, 663.99] 687.67	[605.86, 639.24] 668.59
kroA100	(239, 451)	[20400.44, 21124.34] 21692.45	[21425.49, 22558.40] 23270.29	[19884.22, 21084.09] 22202.97
kroA200	(65, 134)	[28878.35, 29473.94] 30114.63	[32636.36, 33081.95] 33447.02	[29163.98, 30197.34] 31103.06
lin105	(37, 96)	[13885.61, 14338.09] 14814.50	[14199.53, 14442.39] 14728.55	[14088.46, 14529.35] 14858.00
pcb442	(315, 169)	[51315.36, 52195.61] 52947.12	[53860.96, 55416.70] 56689.56	[51761.16, 53771.12] 55585.26
pr107	(22, 91)	[39860.24, 40213.38] 40719.66	[40912.53, 41907.45] 42710.16	[39577.44, 40190.95] 40673.17
pr124	(8, 76)	[57309.63, 58823.04] 60338.93	[60665.69, 61191.00] 61605.94	[56243.79, 59101.27] 61514.62
pr136	(40, 105)	[100272.44, 102345.32] 104171.05	[97836.82, 100201.46] 102432.70	[97235.45, 101667.10] 103521.56
pr152	(103, 2)	[67171.17, 67701.35] 71765.97	[72013.72, 73121.88] 73941.44	[65791.64, 67701.35] 69297.67
rat195	(166, 128)	[2293.74, 2330.87] 2366.34	[2480.62, 2521.24] 2557.77	[2386.37, 2471.97] 2543.19
rd100	(26, 73)	[7685.11, 7996.26] 8289.27	[8343.94, 8497.79] 8677.11	[7823.07, 8157.85] 8455.82
st70	(7, 43)	[634.58, 671.35] 701.11	[684.37, 707.87] 720.42	[644.46, 677.09] 705.93

TABLE 8.6.4. Confidence intervals for the length of optimal Hamiltonian paths between X_s and X_t.

Instance	Endpoints (X_s, X_t)	CI	RI	CW
att532	(239, 451)	[28481.55, 29407.97] 30172.52	[28085.37, 28859.65] 29473.89	[28996.04, 29698.70] 30308.43
berlin52	(10, 31)	[7242.09, 7558.88] 7868.82	[6975.84, 7504.68] 7956.14	[7167.17, 7686.31] 8109.39
bier127	(14, 122)	[113237.42, 122426.45] 131539.83	[115126.03, 121040.12] 126051.13	[118105.45, 121833.82] 125360.92
eil51	(5, 14)	[393.62, 421.04] 445.14	[407.83, 425.27] 440.63	[399.27, 423.42] 443.81
eil76	(60, 38)	[531.70, 560.44] 584.44	[538.27, 559.73] 577.95	[527.52, 558.50] 585.01
eil101	(7, 50)	[635.90, 658.49] 676.83	[636.21, 655.82] 675.23	[640.07, 665.80] 687.16
kroA100	(239, 451)	[20135.53, 22024.95] 23520.29	[19817.05, 21047.05] 22039.85	[20320.10, 21690.25] 22851.23
kroA200	(65, 134)	[29217.27, 30908.65] 32374.05	28591.49, 29942.70] 31033.10	[29839.39, 30950.97] 31904.49
lin105	(37, 96)	[13856.84, 14475.32] 15020.69	[13954.35, 14596.09] 15149.65	[14101.05, 14830.78] 15494.39
pcb442	(315, 169)	[52549.31, 54145.65] 55559.21	[51390.02, 53847.92] 56480.68	[53838.73, 54838.37] 55728.19
pr107	(22, 91)	[39372.19, 40722.93] 41816.26	[39281.91, 40164.25] 40900.35	[40542.34, 42662.08] 44347.41
pr124	(8, 76)	[56409.17, 59750.24] 62513.57	[56651.56, 58959.58] 61326.05	[56773.39, 60619.76] 63613.63
pr136	(40, 105)	[93215.71, 97631.93] 101302.04	[94762.04, 98531.97] 101853.31	[92518.23, 97757.66] 101999.69
pr152	(103, 2)	[66267.42, 69220.83] 71765.97	[65580.40, 67854.34] 69956.44	[63990.73, 69611.87] 75018.80
rat195	(166, 128)	[2329.85, 2450.32] 2562.35	[2349.12, 2463.14] 2559.40	[2332.97, 2419.72] 2491.89
rd100	(26, 73)	[7485.46, 8151.63] 8776.48	[7541.79, 7949.27] 8304.01	[7758.64, 8222.84] 8621.75
st70	(7, 43)	[633.15, 671.79] 705.16	[643.68, 674.06] 700.06	[662.87, 693.66] 720.24

instances. The corresponding optimum tour lengths are given in Table 7.4.2. The pairs listed in the second column are the endpoints of the randomly selected edge whose length is made arbitrarily small. They represent the starting and terminal cities of an optimal Hamiltonian path. The length of an optimal Hamiltonian path between these endpoints, which is obtained by subtracting the length of the selected edge from the length of the optimum tour, is given in the third column. The starting and terminal cities are represented by their TSPLIB sequence number. The last three columns of both tables include the confidence intervals computed according to formula (8.9) as described in the previous section. As it can be observed, the optimal path length of the instances (column 3) is covered by 28 of the 48 interval estimates (columns 4–6). Successful intervals are marked with an "*".

Another interesting observation is the effect of the heuristics used in the generation of these intervals. All the heuristics performed equally well except the nearest insertion heuristic (column 5 in Table 8.6.1). Columns 4–6 of Tables 8.6.1 and 8.6.2 contain one additional number under the intervals: it designates the average length of 50 different Hamiltonian paths computed with the corresponding heuristic. If we measure the performance of a heuristic by the average values, it is reasonable to assign the one with the smallest average as the best. Thus the nearest neighbor followed by 2-Opt (NNb) is the winner for all test instances. Note that 5 out of 8 intervals generated by NNb includes the optimal value listed in column 3. Therefore, the use of lower confidence limits in measuring the performance of the HPP heuristics is indeed a good approach, as justified for the TSP and quadratic assignment problem (QAP).

8.7. Computational Results

The new neural methods have been tested by using problem instances taken from TSPLIB [116]. The selected problems are Euclidean instances which means that each city is represented by its coordinates on the two-dimensional Euclidean space and the distances between the cities are computed according to the Euclidean norm. Furthermore, the sizes of the problems range between 51 and 532 cities. The test bed includes the instances used

for the verification of interval estimates of the previous subsection as a subset. From this perspective, the results are of documentary importance because, in the literature, almost all of the reported results on the neural network algorithms for the TSP and its relatives are based on randomly generated test problems. All experiments were run on an HP-C110 workstation with 128 MByte RAM working within an HP-UX 10.2 environment.

In order to compare the quality of the solutions obtained with our algorithms, confidence intervals for all instances in the test bed are computed by using the same six TSP heuristics. They are summarized in Tables 8.6.3 and 8.6.4. The first column of the tables includes references to TSPLIB instances. The second column contains the endpoints of the Hamiltonian path for which the intervals are calculated; each endpoint is selected randomly (i.e., they are not the endpoints of a randomly selected edge on an optimal tour, which is the case in Section 9.6.2). The next three columns include average lengths of locally optimal Hamiltonian $(\mathbf{X}_s, \mathbf{X}_t)$-paths computed by the six TSP heuristics. The averages are taken over 50 solutions obtained by running the heuristics with 50 different random starting conditions. Recall that heuristic solutions are improved further by 2-Opt, except the one computed by Clarke-Wright heuristic. As it can be observed, in addition to average path lengths, each cell includes also an interval. These are the confidence intervals computed as explained before. Observe that the better the quality of the TSP heuristic (i.e., smaller average), the narrower is the confidence interval. This fact was also observed for the TSP [124].

Table 8.7.1 summarizes the results obtained on the test bed with the neural heuristics. The first column specifies the TSPLIB instance modified to obtain an HPP instance. The second column shows the starting and terminal cities for these instances. The third column consists of the confidence intervals used in the comparisons. They are selected from Tables 8.6.3 and 8.6.4, and represent the heuristics with the smallest average. Note that, except the ones obtained for `lin105`, `pr136`, and `pr152`, all are generated by using nearest neighbor followed by 2-Opt (NNb). The numbers in the fourth to sixth columns are obtained by using GSOM_HPP (referred to as GS in the table), KNIES_HPP, (referred to as KL in the table, L standing for "local") and KNIES_HPP_Global (referred to as KG in the table, G standing for "global"), respectively. These values are the best ones recorded after running the

TABLE 8.7.1. Comparison of the results obtained by neural HPP algorithms for various TSP instances. In each case the best results and the values of the parameters associated with these results are reported.

Instance	**Endpoints** (X_s, X_t)	**Test Interval**	**GS** $(M, \sigma, K_\sigma, iter)$	**KL** $(M, \sigma, K_\sigma, \omega, iter)$	**KG** $(M, \sigma, K_\sigma, \omega, iter)$
att532	(239, 451)	[28054.48, 28486.41]	29153.46 (160,20,0.8,22)	29281.37 (80,5,0.8,0.20,17)	29066.40 * (400,15,0.8,0.20,21)
berlin52	(10, 31)	[7394.56, 7549.32]	7660.33 (16,10,0.8,17)	7596.57 (56,5,0.8,0.20,16)	7522.04 * (48,5,0.8,0.15,15)
bier127	(14, 122)	[114970.39, 119102.93]	124153.90 (75,40,0.8,25)	121468.76 * (25,30,0.8,0.05,23)	121512.26 (125,45,0.8,0.05,26)
eil51	(5, 14)	[405.69, 422.95]	438.74 (50,30,0.8,22)	431.36 * (55,35,0.8,0.10)	432.13 (30,40,0.8,0.05,24)
eil76	(60, 38)	[526.20, 548.07]	575.02 (45,5,0.8,13)	566.03 * (60,25,0.8,0.25,21)	570.72 (60,30,0.8,0.15,21)
eil101	(7, 50)	[631.69, 649.28]	661.58 (40,50,0.8,24)	653.33 (80,45,0.8,0.10,24)	649.09 * (60,40,0.8,0.10,22)
kroA100	(92, 59)	[20400.44, 21124.34]	21219.57 (45,25,0.8,21)	21213.23 (90,40,0.8,0.20,24)	21129.63 * (45,10,0.8,0.10,18)
kroA200	(65, 134)	[28878.35, 29473.94]	29871.91 (160,35,0.8,25)	29732.98 * (80,25,0.8,0.20,24)	30174.88 (160,40,0.8,0.05,25)
lin105	(37, 96)	[14199.53, 14442.39]	14816.53 (20,5,0.8,15)	14619.93 * (80,10,0.8,0.15,19)	14666.96 (40,5,0.8,0.25,15)
pcb442	(315, 169)	[51315.36, 52195.61]	55642.20 (360,25,0.8,21)	55843.27 (450,30,0.8,0.20,23)	55379.15 * (450,50,0.8,0.05,25)
pr107	(22, 91)	[39577.44, 40190.95]	40445.14 (60,25,0.8,22)	40334.50 * (75,40,0.8,0.2,25)	40561.97 (30,25,0.8,0.25,24)
pr124	(8, 76)	[57309.63, 58823.04]	61392.61 (125,25,0.8,21)	59725.41 * (100,20,0.8,0.15,21)	60474.06 (50,5,0.8,0.20,18)
pr136	(40, 105)	[93215.71, 97631.93]	101946.20 * (120,20,0.8,20)	102126.91 (20,10,0.8,0.05,18)	103017.67 (80,45,0.8,0.25,25)
pr152	(103, 2)	[65791.64, 67701.35]	67750.98 * (120,35,0.8,25)	67771.59 (30,25,0.8,0.25,22)	68034.77 (90,25,0.8,0.20,20)
rat195	(166, 128)	[2293.74, 2330.87]	2611.38 (120,50,0.8,24)	2609.37 (40,15,0.8,0.10,20)	2575.93 * (40,20,0.8,0.25,21)
rd100	(26, 73)	[7685.11, 7996.26]	7958.17 * (40,15,0.8,20)	7958.94 (40,25,0.8,0.20,23)	7983.43 (20,20,0.8,0.20,21)
st70	(7, 43)	[634.58, 671.35]	693.19 (40,5,0.8,15)	685.03 * (40,10,0.8,0.15,19)	689.15 (50,20,0.8,0.10,21)

TABLE 8.7.2. Relative (per cent) deviation from the lower confidence limits. The associated parameters are specified in Table 8.7.1.

Instance	GS	KL	KG
att532	3.92	4.37	3.61
berlin52	3.59	2.73	1.72
bier127	7.99	5.65	5.69
eil51	8.15	6.33	6.52
eil76	9.28	7.57	8.46
eil101	4.73	3.43	2.75
kroA100	4.02	3.98	3.57
kroA200	3.44	2.96	4.49
lin105	4.34	2.96	3.29
pcb442	7.78	8.11	7.30
pr107	2.19	1.92	2.49
pr124	7.12	4.22	5.52
pr136	9.37	9.56	10.52
pr152	2.98	3.01	3.41
rat195	13.85	13.76	12.30
rd100	3.55	3.56	3.88
st70	9.24	7.95	8.60
Average Relative Deviation	6.21	5.24	5.54

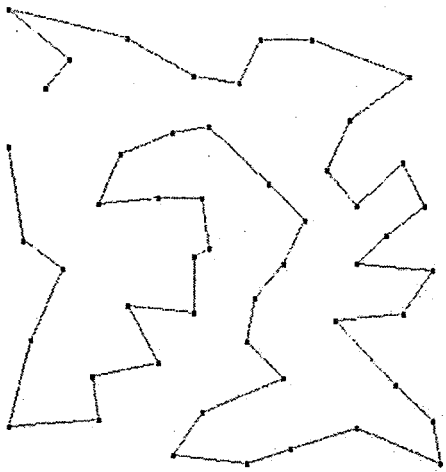
algorithms for a large number of different parameter settings. The numbers in parentheses are the parameter values for which these best results are obtained, and the corresponding number of epochs (denoted as "*iter*"). As a reminder M represents the number of neurons on the initial band connecting X_s and X_t, σ is the standard deviation and K_σ is the multiplier used to decrease its value from epoch to epoch, and ω determines the width of the activation bubble. In each row the cell with the smallest value is marked with an "*".

During the computations the neighborhood function is based on the lateral distance between two neurons on the piecewise linear line rather than their Euclidean distance. More explicitly, the distance between the winner neuron and any other neuron on the piecewise linear line is the absolute value of the difference between the indices of the two neurons. Furthermore, in all methods the parameter K_σ is kept fixed at 0.8 throughout the iterations; this is advised in the literature [90].

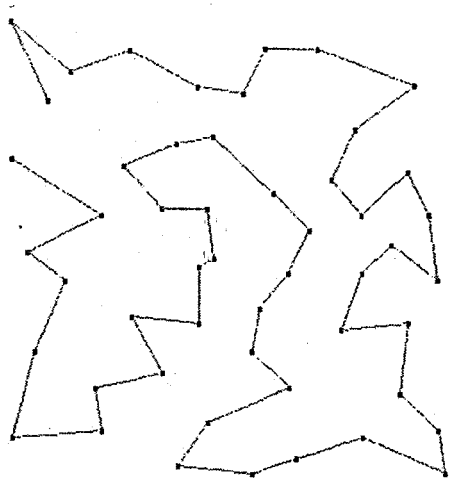
By a close examination it is possible to see that optimal path lengths obtained by GS, KL, and KG are outside the test interval. This is not surprising because none of the intervals is generated by using GS, KL, or KG. However, the values are not lower than the lower bound, which is used as a conservative estimate for the length of an optimal Hamiltonian path.

Figures 8.7.1, 8.7.2, and 8.7.3 illustrate the Hamiltonian paths associated with (a) optimal solutions and (b) the solution of KNIES_HPP for the problem instances `eil51`, `eil76`, and `eil101` by deleting the edges {41, 13}, {11, 53} and {54, 55} respectively. They are chosen randomly on the optimum tour listed in TSPLIB. Hamiltonian paths of figures 8.7.4, 8.7.5, and 8.7.6 are obtained by KNIES_HPP as well. In these cases, however, both of their endpoints are randomly selected, and thus they are not the endpoints of a randomly selected edge of an optimal TSP tour.

As it can be observed, KNIES_HPP (KL) is the winner for `bier127`, `eil51`, `eil76`, `kroA200`, `lin105`, `pr107`, `pr124`, and `st70`. In the remaining instances

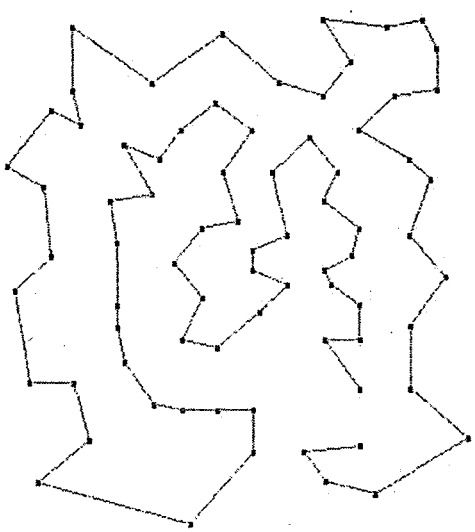


(a) Optimal

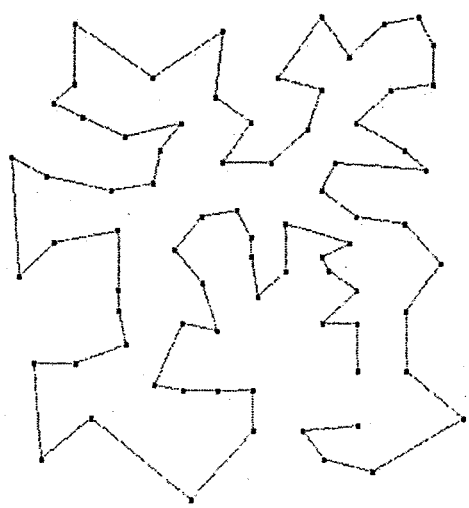


(b) Obtained with KNIES_HPP

FIGURE 8.7.1. Hamiltonian paths for `eil51` from 41 to 13.

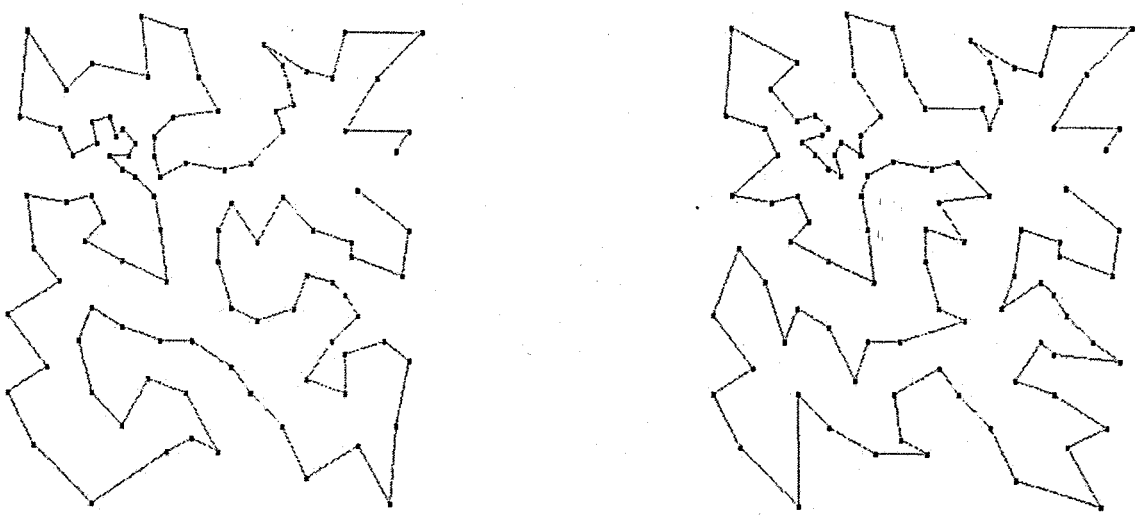


(a) Optimal



(b) Obtained with KNIES_HPP

FIGURE 8.7.2. Hamiltonian paths for `eil76` from 11 to 53.



(a) Optimal (b) Obtained with KNIES_HPP

FIGURE 8.7.3. Hamiltonian paths for `eil101` from 54 to 55.

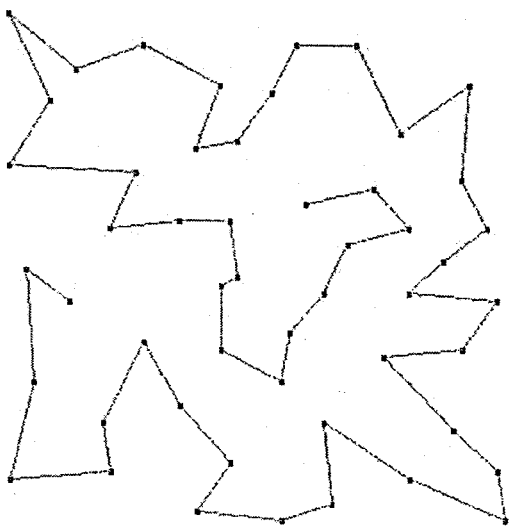


FIGURE 8.7.4. Hamiltonian path obtained with KNIES_HPP for `eil51` from 5 to 14.

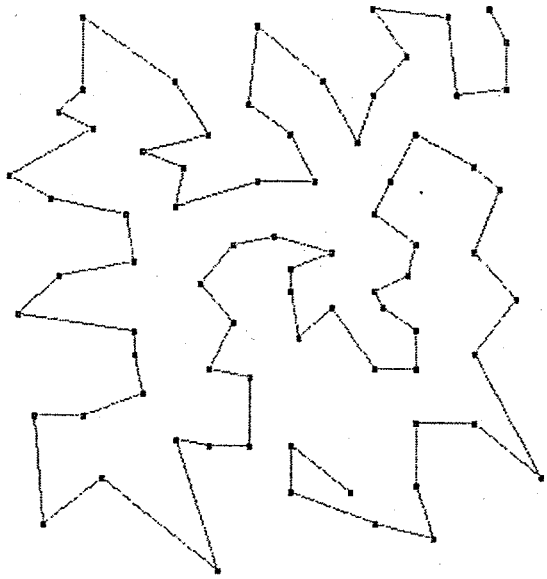


FIGURE 8.7.5. Hamiltonian path obtained with KNIES_HPP for `eil76` from 60 to 38.

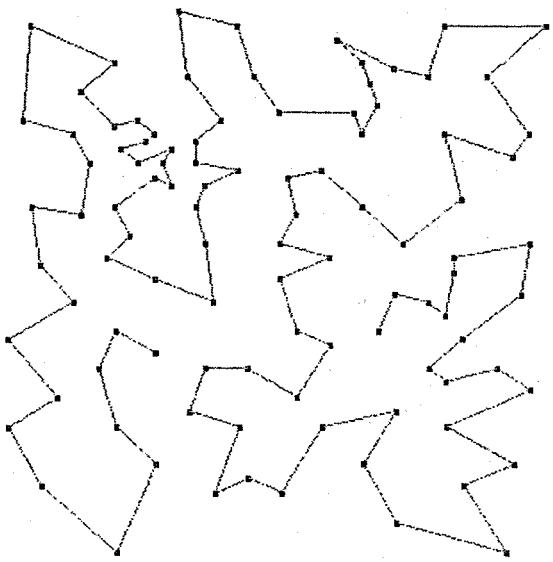


FIGURE 8.7.6. Hamiltonian path obtained with KNIES_HPP for `eil101` from 7 to 50.

KNIES_HPP_Global (KG) provides the best results except pr136, pr152, and rd100. For example, in the case of ei176, the lower confidence limit is 526.20. The length of the path obtained by KNIES_HPP is 566.03, which is 7.57 per cent higher than the lower confidence limit. Similarly, the length of the paths obtained by KNIES_HPP_Global and GSOM are respectively 570.72 and 575.02 corresponding to 8.46 per cent and 9.28 per cent of relative deviations from the lower confidence limits.

The results obtained are good. This is because all of them have been obtained without resorting to a solution to the TSP. Besides, since the neurons are ordered linearly (as opposed to cyclically), we have had to forfeit all the information which we would have otherwise obtained by processing the neurons that would have been neighbors on the ring. In spite of having to discard this information, the results obtained are always within a few percent of the lower confidence limits.

The relative deviations from the lower confidence limits are summarized in Table 8.7.2. These values are calculated according to the following formula:

$$\frac{Z_{NH} - LCL}{LCL} \times 100 \tag{8.13}$$

where Z_{NH} and LCL are respectively the length of the near optimal Hamiltonian path and lower confidence limit of the intervals listed in Table 8.7.1. Note that $NH \in \{GS, KL, KG\}$. Of course, it is impossible to make categorical statements about the various algorithms from the results presented here. However, to get an overall picture, the average deviation for GS, KL, and KG are computed, which are respectively 6.21 per cent, 5.24 per cent and 5.54 per cent. Based on these results, it is possible to draw the conclusion that KNIES_HPP (KL) performs better than the other two neural methods. However, the performance of KNIES_HPP_Global (KG) is close to the KNIES_HPP's. This can also be observed from Table 8.7.2 where KG outperforms KL for att532, berlin52, eil101,

kroA100, pcb442, pr152, and rat195. This fact makes KG's use more attractive since it is computationally less expensive than KL.

Finally, it should be mentioned that the performance of all methods is not "proportional" to the number of cities. For example, in the case of 532 city instance, att532, GS, KL, and KG yield relative deviations of 3.92 per cent, 4.37 per cent, 3.61 per cent respectively. However, these deviations are 8.15 per cent, 6.33 per cent. 6.52 per cent for the 51-city instance eil51.

9. DECOMPOSITION APPROACH TO THE EUCLIDEAN TSP

9.1. Clustering and the Traveling Salesman Problem

An idea to reduce the complexity of a large-scale traveling salesman problem instance is to decompose or partition it into smaller subproblems, which are easier to solve. The partitioning is performed by clustering the cities of the original problem in a way that structural properties of the problem instance are preserved. Generally speaking, the problem of size n is divided into k nonoverlapping clusters C_i of size n_i ($C_i \cap C_j = \emptyset \quad \forall\, i \neq j$), where $\max\{n_i : 1 = 1, \ldots, k\} \ll n$. An efficient clustering algorithm should furthermore fulfill most of the following additional restrictions:

(i) No intersection

$CH(C_i) \cap CH(C_j) = \emptyset \quad \forall\, i \neq j$, where $CH(\cdot)$ denotes the convex hull of a set.

(ii) Homogenous cluster sizes

$$\left\lfloor \frac{n}{k} \right\rfloor \leq n_j \leq \left\lceil \frac{n}{k} \right\rceil$$

(iii) Preserve preclustered structures

Every nearest neighbors should be in the same cluster:

$$d(x,y) = \min\{d(x,z) : z \in \cup C_j,\ z \neq x\} \Longrightarrow y \in C_i \qquad \forall i,\ \forall x \in C_i \tag{9.1}$$

Clustering is a huge topic and there are numerous mathematical clustering methods [130]. The most common non-neural clustering algorithms are summarized below:

(i) Geometric partitioning: Partition the given area into equal size. If

$$x_{\min} = \min\{x_i\},\ x_{\max} = \max\{x_i\},\ y_{\min} = \min\{y_i\},\ x_{\max} = \max\{y_i\} \tag{9.2}$$

then $l \times h$ rectangles $R_{i,j}$ of width $\frac{1}{l} \cdot (x_{\max} - x_{\min})$ and height $\frac{1}{h} \cdot (y_{\max} - y_{\min})$ with lower left corner at $\left(x_{\min} + \frac{i}{l} \cdot (x_{\max} - x_{\min}), y_{\min} + \frac{j}{h} \cdot (y_{\max} - y_{\min})\right)$ are defined.

(ii) Nearest neighbor clustering: Here the clusters are connected components of the graph which consists of the edges to the 2 $(3, 4, \ldots, k)$ nearest neighbors of each city. Using this procedure preclustered structures are preserved, but clusters of different size and very irregular shape are obtained.

(iii) Litke clustering [131]: Litke describes a heuristic that iteratively generates rectangular clusters of equal size by enlarging each cluster until it contains the desired number of cities. The disadvantages of this approach are that it generates intersecting clusters and that the rectangles generated during the final phase of the algorithm are extremely large.

(iv) Delaunay clustering [93]: The edges are sorted in ascending order with respect to their lengths. Starting with each city as a cluster the cities connected by an edge are put into the same cluster, until the maximum cluster size is reached. In this algorithm the cluster size and the number of clusters can be adjusted while the preclustered structures are preserved. The disadvantages of the approach are that clusters may intersect and may have an irregular shape.

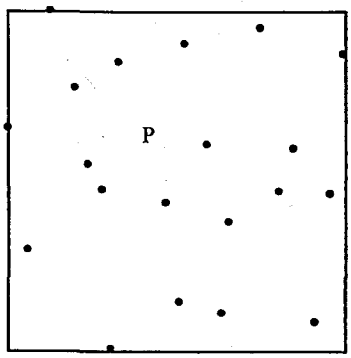
(v) Karp clustering [132]: In 1977 Karp introduced a TSP-heuristic in which the cities are partitioned recursively by horizontal and vertical cuts into non-intersecting rectangular clusters each of which contains no more than a certain number of cities. Then, for each cluster an optimal solution to the TSP defined on the cities of that cluster is computed, and in a final step all the subtours are glued together according to some scheme to form a tour through all cities. With this approach, the preclustered structures may be destroyed. Also, in the original algorithm cities on the edges of the rectangular clusters belong to both clusters, but the algorithm may be slightly modified to overcome the situation.

After the cities are partitioned into different clusters, it would be possible to proceed in two different ways. One way is (as is the case in the TSP-heuristic of Karp) to solve the traveling salesman subproblem for each cluster and then to join the subtours to a global tour. The traveling salesman subproblems can be solved using any efficient TSP heuristic (e.g., Lin-Kernighan algorithms [94]).

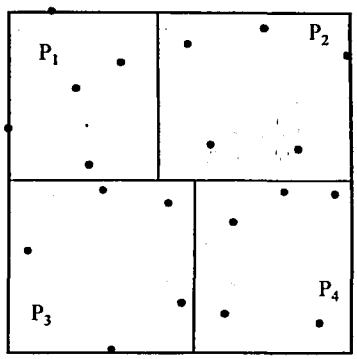
For the generation of the global tour the following procedure might be applied: We begin with an arbitrary subtour T_1. This subtour is then connected to another subtour T_2 to form a new tour In general, a subtour T_i is connected to the global tour $\hat{T}_i$ generated so far by removing one edge of T_i and one edge of $\hat{T}_i$ and then by replacing them by two new edges connecting T_i and $\hat{T}_i$ to form a new tour $\hat{T}_{i+1}$. This procedure, however, has an important disadvantage: the subtours are not optimized with respect to the edges connecting the clusters. So a better approach is to obtain the global tour *before* the subproblems are solved in order to integrate the obtained information into the local TSP-heuristic. In this way the local solution also considers the city n_{i_e} where the global tour enters the cluster, and the city n_{i_l}, where it leaves the cluster and so takes into account the outline of the global tour. As a result better global solutions are obtained. This latter approach is the second way of proceeding after the cities are partitioned and given below:

1. Compute the centroid or the mean of the cities in each cluster.
2. For every cluster determine the r nearest clusters in regard to the centroid.
3. Compute the convex hull of each cluster.
4. Compute the exact distances for the r nearest clusters (as the shortest distance between the nodes of the convex hulls).
5. Obtain a global tour through the clusters by a TSP-heuristic using the exact distances, if available, or otherwise the distances between the means which gives an entering and a exiting city for each cluster.
6. Apply a suitable heuristic to find the Hamiltonian path in every cluster connecting the entering to the exiting city visiting all the cities in that cluster.
7. Merge the edges between the exiting city of one cluster and entering city of the subsequent cluster in the global tour and the Hamiltonian paths to form a tour for the original problem.

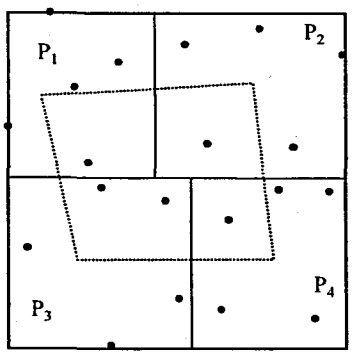
The flow of the above algorithm is illustrated in Figure 9.1.1.



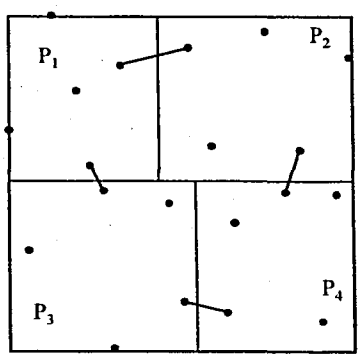
(a) The original problem



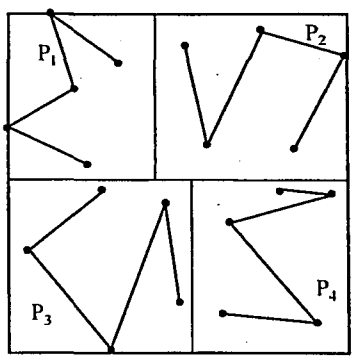
(b) The cities are partitioned into four clusters



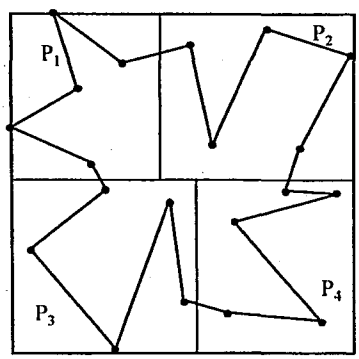
(c) The global tour



(d) The entering and exiting cities



(e) Hamiltonian paths through the cities in each cluster



(f) Merging the solutions in (d) and (e)

FIGURE 9.1.1. Decomposition approach.

9.2. An All-Neural Decomposition Approach

In this section an all-neural decomposition method which implements the steps of the basic decomposition approach by using neural networks is proposed. The first step of the decomposition approach is to partition the cities into clusters. The clustering is accomplished using vector quantization discussed in Section 4.1. Here the number of the clusters is a parameter and there are as many codebook vectors as the number of clusters. The codebook vectors are moved in the two-dimensional space until they find their final places and then the closest codebook vector is found for each city. Hence the input space is divided into clusters each of which is represented by a codebook vector.

At these moment the codebook vectors and the mean of the cities in a given cluster (the centroid of the cluster) coincide. Therefore the codebook vectors representing the clusters can be used to find the global tour through the clusters. However, in order to obtain a better discrimination of the clusters (i.e., near-optimal cluster boundaries) we make use of the intraregional and interregional polarizing explained previously in Section 4.1 in the context of road distance estimation.

The aim of the intraregional polarizing is to represent each cluster C_k by a number of codebook vectors M_k where M_k increases with the number of cities located in that cluster which is denoted as N_k. Specifically, $M_k = \lfloor 0.3\, N_k \rfloor$. If there are three or less cities in a cluster, then M_k is set equal to one. The set of codebook vectors for cluster k, $\{Q_{kj} : 1 \leq j \leq M_k\}$ are initially located on a circle the center of which is the mean of the cities belonging to that cluster. The codebook vectors are then updated according to the formula (4.13) which is given below once more.

$$Q_{k,j}(t+1) = \begin{cases} (1-\alpha(t))\, Q_{k,j}(t) + \alpha(t)\, P_{k,i} & \text{if } Q_{k,j} \text{ is the closest codebook} \\ & \text{vector to the data point } P_{k,i} \\ Q_{k,j}(t) & \text{otherwise} \end{cases} \tag{9.3}$$

$\alpha(t)$ is decremented linearly from unity for the initial learning phase and then switched to 0.2 and is decreased linearly for the fine-tuning phase. After the individual clusters have been represented by M_k codebook vectors they are tested to see whether they adequately classify the cities within their clusters. Therefore, the interregional polarizing phase has been employed where the codebook vectors do not find their places by learning only from the cities within their own clusters (as in the intraregional polarizing phase) but they are also migrated in such a way that they polarize away from the cities of the neighboring clusters.

All the updates of the interregional polarizing are performed according to the update formulae given in Equation (4.14). There are three parameters in these update equations; α, ϵ, and the diameter of the hypersphere W centered at the bisector of the two nearest codebook vectors. Except ϵ, which is kept constant at 0.25, experiments were performed with different values of parameters α and the diameter of W in order to see the effect of the interregional polarizing on the quality of the solution (i.e., tour length). Both α, and the diameter of W assumed values in the interval $(0, 1)$ with increments of 0.1.

The output of the interregional polarizing phase is the final partitioning of the cities into clusters. The next step is to determine the global tour through the clusters. To accomplish this, the centroid of each cluster is found by computing the mean of the cities located in that cluster and the algorithm KNIES_TSP_Global (discussed in Section 7.3) is invoked. KNIES_TSP_Global quickly yields a tour passing through the centroids. Hence, the sequence at which the global tour visits the clusters is determined. The next step is to find out the entering and exiting cities for each cluster. This is done as follows. For two subsequent clusters in the global tour the two nearest cities is found such that one city belongs to the first cluster and the other one belongs to the second cluster. These two cities constitute the bridge between the two clusters, in other words the final tour for the instance will exit one cluster and enter the subsequent cluster via these two cities.

After the special cities (entering and exiting) for each cluster are found, the remaining task becomes the determination of the Hamiltonian path between these cities which is done by running the algorithm KNIES_HPP_Global. When the Hamiltonian paths for each cluster

are glued together, the final tour is obtained. The decomposition approach is tested for the same instances from TSPLIB that were also used to assess the performances of the KNIES_TSP and KNIES_HPP approaches discussed in Chapters 7 and 8, respectively.

9.3. Computational Results

The crucial parameter for the decomposition approach is the number of clusters since the parameters of both the algorithms KNIES_TSP_Global and KNIES_HPP_Global are fixed. Particularly, the following values are used for the parameters of KNIES_TSP_Global that is used to determine the order in which the clusters are visited: $K_\sigma = 0.8$, $\omega = 0.2$, and $\sigma = 20$ initially. Furthermore, the initial number of neurons is set equal to the number of clusters. The same parameter setting is also adopted for KNIES_HPP_Global where $K_\sigma = 0.8$, $\omega = 0.2$, and $\sigma = 20$ initially. The initial number of neurons is set equal to 0.3 times the number of cities (including the entering and exiting cities) in the cluster for which the Hamiltonian path is to be found. If the number of neurons turns out to be less than three, then the algorithm is run with three neurons initially. These values for the parameters are selected since the algorithms KNIES_TSP_Global and KNIES_HPP_Global provided satisfactory results with these parameters for most of the instances which were solved in Chapters 7 and 8.

The steps of the all-neural decomposition approach is illustrated in the following figures while solving the instance `eil101` by decomposing it into 10 clusters.

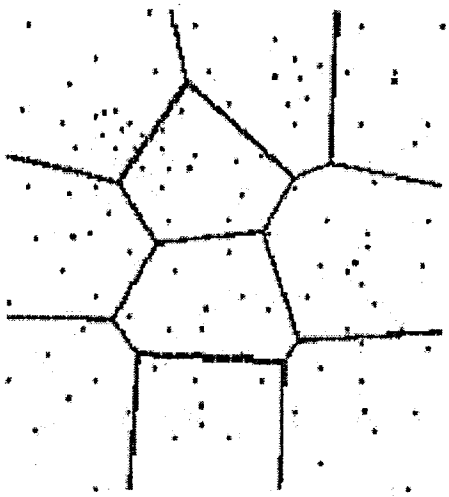
Figure 9.3.1 (a) shows the result of the partitioning. The cities are divided into 10 clusters. The darker dots represent the centroids of the clusters. Figure 9.3.1 (b) contains the global tour through the clusters computed by using KNIES_TSP_Global with the cluster centroids as the input. The global tour gives the order in which clusters are visited on the final tour passing through all the cities in the instance. Hence, it is now possible to determine the entering and exiting cities in each cluster. The exiting city of a cluster and the entering city of the subsequent cluster are connected and this connection constitutes the bridge between

the clusters (according to the order in which clusters are visited). This can be seen in Figure 9.3.1 (c). Figure 9.3.1 (d), Figure 9.3.2(a), and (b) are snapshots taken after the Hamiltonian paths are determined by KNIES_HPP_Global for one, two, and three clusters, respectively, and glued together with the bridges between these clusters. As the Hamiltonian path is found for each cluster and glued with that of the subsequent cluster, the TSP tour for the original problem grows gradually and gets its final shape given in Figure 9.3.2 (c). The final TSP tour for the instance `eil101` can be seen in Figure 9.3.2 (d) without the cluster borders.

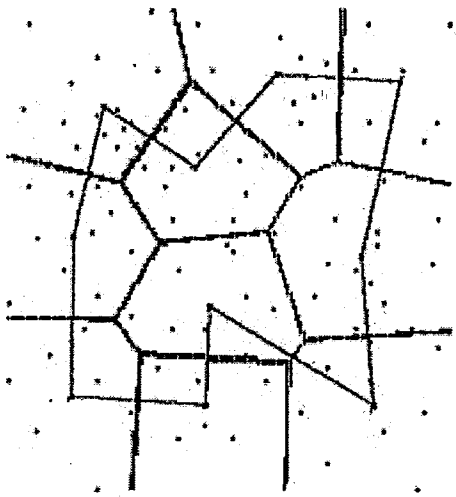
For each instance, experiments are performed with different number of clusters so that on the average no more than 50 and no less than 14 cities belong to each cluster which is achieved when the following is satisfied: $\lfloor 0.02\,N \rfloor \leq$ number of clusters $\leq \lfloor 0.07\,N \rfloor$. Here N is the problem size, i.e., the number of cities. This is because when the number of cities in any cluster exceeds 50 or so, then the computational time of determining the Hamiltonian path for that cluster increases. Hence, the number of clusters should not be less $\lfloor 0.02\,N \rfloor$. On the other hand, when the cities are divided into too many clusters, the direct consequence of which is the number of cities in each cluster gets smaller, then the preclustered structures are destroyed and we move away from the global optimal. Thus, the number of clusters has to be less than $\lfloor 0.07\,N \rfloor$. To give an example, the number of clusters varies between $\lfloor 0.02 \times 532 \rfloor = 10$ and $\lfloor 0.07 \times 532 \rfloor = 37$ for `att532`. As it is pointed out in the previous section the parameters of the interregional polarizing assumed the following values: $\epsilon = 0.25$, and both α and the diameter of W range between zero and one. Table 9.3.1 includes the best results obtained for `att532` as a function of number of clusters. The values of α and the diameter of W, which provide the best tour lengths are also given. The overall best result is achieved when the cities are divided into 13 clusters.

The overall best results obtained by the all-neural decomposition approach for different TSP instances and the best values provided by KNIES_TSP (referred to as KL) and KNIES_TSP_Global (referred to as KG) for the same instances are given in Table 9.3.2.

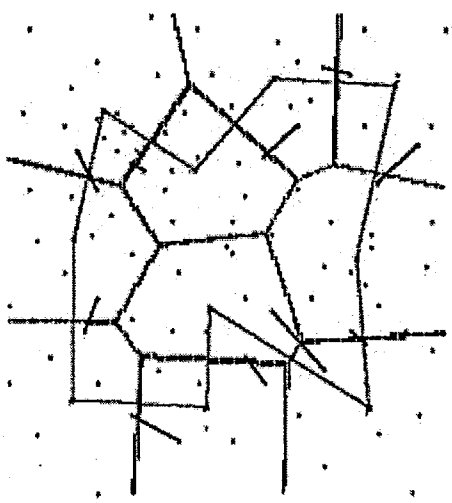
Table 9.3.3 contains the relative deviations of the three approaches from the optimal values. As it can be observed in the table the success of the decomposition approach



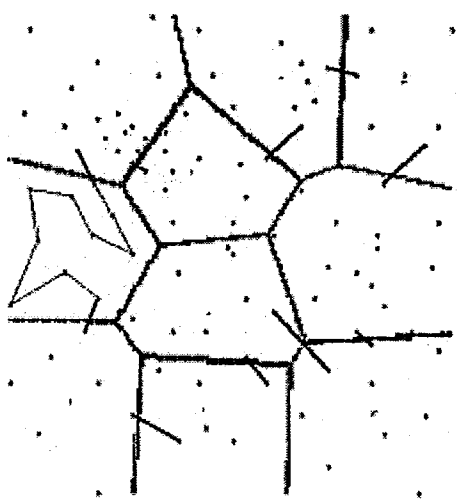
(a) The cities are partitioned into 10 clusters



(b) The global tour through the clusters

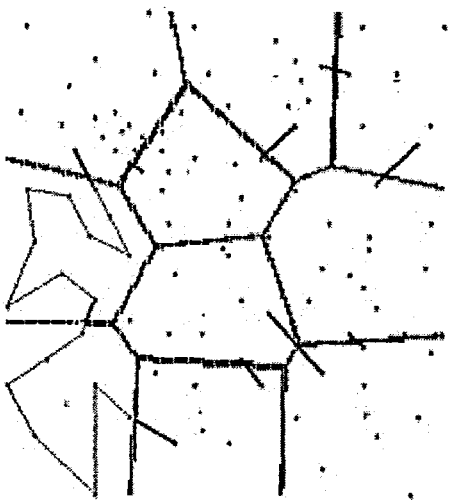


(c) The entering and exiting cities for each cluster and connections ("bridges") between clusters

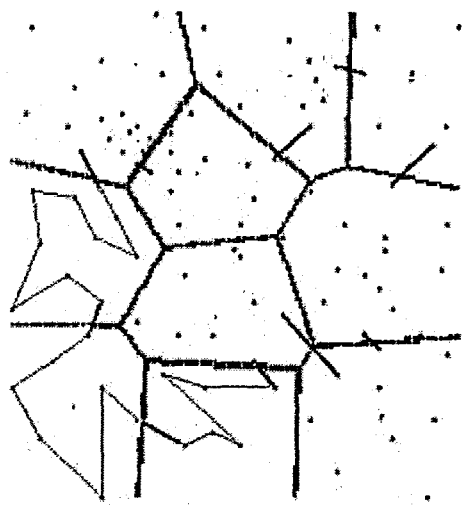


(d) Hamiltonian path between the entering and exiting cities for a cluster

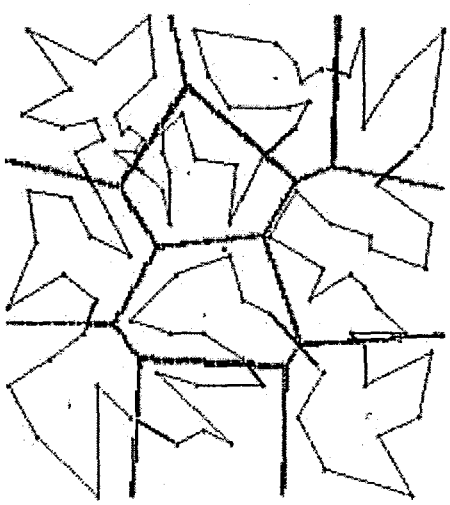
FIGURE 9.3.1. All-neural decomposition approach.



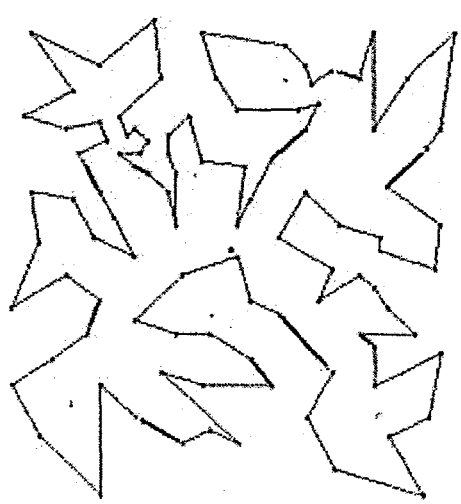
(a) Hamiltonian paths for two clusters are glued together



(b) Hamiltonian paths for three clusters are glued together



(c) Hamiltonian paths for every cluster are glued together



(d) The final tour for instance `eil101`

FIGURE 9.3.2. All-neural decomposition approach (continued).

TABLE 9.3.1. Best results for `att532` as a function of number of clusters.

Number of Clusters	Tour Length	α	Diameter of W
10	30018.16	0.7	0.6
11	29891.98	0.6	0.8
12	29927.55	0.7	0.6
13	29388.88	0.6	0.8
14	29883.40	0.2	0.6
15	29782.35	0.5	0.6
16	29628.22	0.4	0.6
17	30078.65	0.8	0.9
18	29821.84	0.3	0.5
19	30231.76	0.5	0.3
20	29921.52	0.8	0.9
21	30239.64	0.7	0.9
22	30581.60	0.5	0.7
23	30286.34	0.7	0.5
24	30269.26	0.1	0.6
25	30215.12	0.6	0.6
26	29949.10	0.9	0.8
27	30278.20	0.4	0.5
28	30017.96	0.9	0.6
29	30404.09	0.5	0.8
30	30507.37	0.2	0.8
31	30407.92	0.4	0.3
32	30874.54	0.8	0.2
33	29862.38	0.5	0.3
34	30299.79	0.5	0.3
35	30871.63	0.6	0.9
36	30148.04	0.1	0.9
37	30682.40	0.5	0.5

TABLE 9.3.2. Comparison of the results obtained for different algorithms instances.

Instance	**Opt. Value**	**Decomposition** (number of clusters)	**KL** $(M,\sigma,K_\sigma,\omega)$	**KG** $(M,\sigma,K_\sigma,\omega)$
att532	27686	29388.9 (13)	29551.6 (200,30,0.8,0.15)	29569.8 (400,45,0.8,0.1)
bier127	118282	126080.8 (9)	121548.7 (100,15,0.8,0.1)	121923.7 (125,50,0.8,0.1)
eil51	426	440.9 (5)	438.2 (10,25,0.8,0.1)	438.2 (20,50,0.8,0.05)
eil76	538	572.9 (6)	564.8 (90,20,0.8,0.15)	567.5 (15,5,0.8,0.2)
eil101	629	672.0 (4)	658.3 (20,25,0.8,0.8)	664.4 (40,35,0.8,0.25)
kroA200	28568	30184.9 (12)	30200.8 (200,25,0.8,0.25)	30444.9 (160,10,0.8,0.05)
lin105	14379	14693.1 (8)	14664.4 (100,50,0.8,0.2)	14564.6 (100,35,0.8,0.05)
pcb442	50778	54838.6 (3)	56399.9 (500,30,0.8,0.1)	56082.9 (450,40,0.8,0.15)
pr107	44303	49103.9 4	44628.3 (100,45,0.8,0.1)	44491.1 (60,25,0.8,0.25)
pr124	59030	60931.5 (3)	59075.7 (25,25,0.8,0.10)	59320.6 (125,10,0.8,0.05)
pr136	96772	98641.2 (8)	101156.8 (75,40,0.8,0.15)	101752.4 (30,35,0.8,0.05)
pr152	73682	76072.1 (15)	74395.5 (180,30,0.8,0.05)	74629.0 (60,10,0.8,0.2)
rat195	2323	2517.0 (2)	2607.3 (200,25,0.8,0.25)	2599.8 (200,25,0.8,0.1)
rd100	7910	8296.9 (9)	8075.7 (80,20,0.8,0.25)	8117.4 (60,10,0.8,0.05)
st70	675	699.8 (3)	685.2 (40,10,0.8,0.05)	690.7 (30,45,0.8,0.05)

TABLE 9.3.3. Relative deviation from the optimal tour length (per cent) of the various algorithms

Instance	Decompositon	KL	KG
att532	6.15	6.74	6.80
bier127	6.59	2.76	3.08
eil51	3.49	2.86	2.86
eil76	6.49	4.98	5.48
eil101	6.84	4.66	5.63
kroA200	5.66	5.72	6.57
lin105	2.18	1.98	1.29
pcb442	7.99	11.07	10.44
pr107	10.84	0.73	0.42
pr124	3.22	0.08	0.49
pr136	1.93	4.53	5.15
pr152	3.24	0.97	1.29
rat195	8.35	12.24	11.92
rd100	4.89	2.09	2.62
st70	3.67	1.51	2.33
Average Relative Deviations	5.41	4.19	4.42

increases as the problem size increases. For att532, pcb442, kroA200, rat195, the decomposition approach provides shorter tour lengths than both KNIES_TSP and KNIES_TSP_Global. If only these four instances are considered, the average relative deviations from the optimal tour lengths become 7.03, 8.94, and 8.93 for the decomposition approach, KNIES_TSP, and KNIES_TSP_Global, respectively. The best advantage of the decomposition approach which cannot be perceived in the table is its speed. Compared with the other two approaches it is possible to obtain solutions very quickly even for large problems. This is due to the fact that once the sequence of the clusters on the global tour and subsequently the entering and the exiting cities for each cluster are determined, the Hamiltonian paths between these cities within the clusters may be found simultaneously. It is even possible to solve the Hamiltonian path problems in a distributed manner on different computers.

10. CONCLUSIONS AND RESEARCH DIRECTIONS

In Chapters 4 and 5, neural networks are used first to estimate the actual distance between two points on the earth surface and second to compress or to quantize the input data prior to the estimation attempt. The former application is actually a nonlinear regression or nonlinear approximation to a multivariate function where the function is the unknown distance function between two points within a given region whereas the second application is related to the classification or clustering.

For both of these problem types neural networks have shown comparable or better performance than traditional statistical techniques although statisticians do not feel comfortable with the use of neural networks as statistical techniques because there is no strong theoretical basis for the statistical results provided by the neural networks even though there have been significant theoretical contributions in neural networks in the areas of learning algorithms, the structure of neurons, the transfer or activation function etc.

A possible research topic related to the learning vector quantization might be the investigation of different distance measures in finding the winning neuron upon the presentation of an input vector. The most widely used distance (dissimilarity) measure which is also used in this study is the Euclidean distance. However there are many other distance measures available such as the ℓ_p-distance (see Chapter 2), the Mahalonobis distance [52], and the Hamming distance that is particularly useful when the input vectors are symbols. The choice among these distance measures is application dependent and may require a clear understanding of the problem at hand.

Almost every class of combinatorial (and noncombinatorial) optimization problem has been tackled by neural networks over the last decade while the majority of the research is focused on the solution of the traveling salesman problem since this problem has become a benchmark. Chapters 6–9 contain this type of work where new neural algorithms are

proposed to solve the Euclidean traveling salesman problem and its cousin the Euclidean Hamiltonian path problem

The algorithms KNIES_TSP and KNIES_HPP are based on Kohonen's self-organizing map. Unlike all the other neural approaches that are also based on the self-organizing map and try to solve the TSP, however, KNIES (Kohonen Network Incorporating Explicit Statistics) utilizes not only the local information that is used when presenting a city to the network (which is the case in other approaches), but also the global information latent in the entire data which is some statistical information extracted from the cities, i.e., their mean. As a result, KNIES includes a dispersing module in addition to the ordinary attracting module. Because of this module the neurons individually find their places both statistically and topologically, and also collectively maintain their mean to be the mean of the cities they represent. Experimental results for problems obtained from TSPLIB either directly (for the Euclidean traveling salesman problem) or by modifying the instances for randomly selected starting and terminal cities (for the Euclidean Hamiltonian path problem) indicate that these algorithms are very accurate.

For large size problems the neural approaches tend to be more time consuming. Decomposing the original problem into subproblems and solving each subproblem separately by paying attention to the quality of the overall solution turns out to be a very effective strategy. The method presented in Chapter 9 is the first all-neural decomposition approach for solving large size traveling salesman problems. Instances selected from TSPLIB for the experiments in order to assess the performance of KNIES are again used for the evaluation of the efficiency of the decomposition approach. All the instances can be solved very quickly and with no deterioration in the solution quality. On the contrary, as the problem size increases the decomposition approach yields better results compared to KNIES in much shorter time.

A future research in this area may be in the direction of finding a more intelligent way in determining the entering and exiting cities in each cluster. The solution quality is affected by both the quality of the Hamiltonian paths joining the entering and exiting cities

in the clusters and the bridges between the clusters, i.e., the connections from one cluster to the subsequent one. However, these two aspects are not independent since the entering and exiting cities also determine the input to the Hamiltonian path problem to be solved for each cluster. Therefore, their selection becomes very crucial from the solution quality point of view. The decomposition approach introduced here does not modify clusters as well as the entering and leaving cities for each cluster once they are determined. It is evident that mechanisms that modify the cluster borders and thus result in a different decomposition of the cities so as to obtain a better configuration prior to locating the entering and exiting cities will improve the solution quality considerably.

In a recent survey of meta-heuristics, Osman and Laporte [133] report that while neural networks are a very powerful technique for solving problems of nonlinear regression, classification and pattern recognition, they have not been as successful when applied to optimization problems and are not competitive with the best meta-heuristics from the operations research literature. For this reason, there is a never-ending attempt to improve the solution quality by novel techniques. In this regard, a promising direction lies in the hybridization of neural networks with meta-heuristics such as genetic algorithms and simulated annealing in such a way that the advantages of each of the techniques can be combined to overcome the limitations [134, 135].

An accurate evaluation of the capabilities of neural networks to obtain near-optimal solutions to optimization problems is necessary so that neural networks are employed in practical situations where their advantages over existing techniques can be exploited.

APPENDIX

The following publications appeared as a result of the research found in this thesis.

INTERNATIONAL REFEREED JOURNALS

Alpaydın E., İ.K. Altınel, and N. Aras, "Parametric Distance Functions vs. Nonparametric Neural Networks for Estimating Road Travel Distances," *European Journal of Operational Research*, Vol. 92, No. 6, pp. 230–243, 1996.

Altınel İ.K., J. Oommen, N. Aras, "Vector Quantization for Arbitrary Distance Function Estimation," *INFORMS Journal on Computing*, Vol. 9, No. 4, pp. 439–451, 1997.

Altınel İ.K., J. Oommen, N. Aras, "Discrete Vector Quantization for Arbitrary Distance Function Estimation," *IEEE Transactions on Systems, Man and Cybernetics—Part B: Cybernetics*, Vol. 28, No. 4, pp. 496–510, 1998.

Aras, N, İ.K Altınel, and J. Oommen, "Kohonen Network Incorporating Explicit Statistics and its Application to the Traveling Salesman Problem," *Neural Networks* (to appear).

Altınel, İ.K., N. Aras, and J. Oommen, "Fast, Efficient and Accurate Solutions to the Hamiltonian Path Problem Using Neural Approaches," *Computers and Operations Research* (to appear).

NATIONAL REFEREED JOURNALS

Altınel İ.K., and N. Aras, "Türkiye İçinde Karayolu Uzaklıklarının Yaklaşık Hesabı," *Transactions on Operational Research*, Vol 6, No. 2, pp. 121–136, 1994.

Altınel İ.K., N. Aras, and J. Oommen, "Vektör Niceleme ve Uzaklıkların Kestirimi," *Galatasaray Üniversitesi Mühendislik Bilimleri Dergisi*, Vol. 1, No. 1, pp. 77–96, 1998.

INTERNATIONAL CONFERENCES WITH REFEREED PROCEEDINGS

Altınel İ.K., N. Aras, and E. Alpaydın, "Comparing Distance Functions and Neural Networks for Estimating Road Travel Distances," *The 9th International Symposium on Computer and Information Sciences ISCIS '94*, pp. 508–515, 1994.

Aras N., İ.K. Altınel, and J. Oommen, "A New Self-Organizing Approach to the Traveling Salesman Problem," *Proceedings of the 12th International Symposium on Computer and Information Sciences ISCIS '97*, pp. 385–392, 1997.

Aras N., İ.K. Altınel, and J. Oommen, "Arbitrary Distance Function Estimation Using Discrete Vector Quantization," *Proceedings of 1997 International Conference on Neural Networks ICNN '97*, pp. 1272–1277, 1997.

Oommen J., N. Aras, and İ.K. Altınel, "Solving the Traveling Salesman Problem using the Kohonen Network Incorporating Explicit Statistics," *Proceedings of WIRN/VIETRI-98, the Tenth Italian Workshop on Neural Nets*, Italy, pp. 273–282, 1998.

Oommen J., İ.K. Altınel, and N. Aras, "Kohonen-like Neural Solutions to the Hamiltonian Path Problem," *Proceedings of the Irish Artificial Intelligence and Cognitive Science Conference AICS '99,* Dublin, September 1999, pp. 51–57, 1999.

NATIONAL CONFERENCES WITH REFEREED PROCEEDINGS

Aras N., İ.K. Altınel, and E. Alpaydın, "Estimation of Road Travel Distances," *Proceedings of Research Symposium '95*, pp. 185–189, 1995.

Altınel İ.K., and N. Aras, "Türkiye İçinde Karayolu Uzaklıklarının Yaklaşık Hesabı," *3rd National Transportation Symposium*, pp. 175–190, 1995.

Aras N., İ.K. Altınel, and J. Oommen, "Adaptive Clustering by Vector Quantization for Arbitrary Distance Function Estimation," *Proceedings of the 5th Turkish Symposium on Artificial Intelligence and Neural Networks TAINN'96*, pp. 53–62, 1996.

Altınel İ.K., N. Aras, and J. Oommen, "Karayolu Uzaklıklarının Kestiriminde Vektör Niceleme Yönteminin Kullanılması," *Proceedings of the 18th National Operations Research and Industrial Engineering Symposium YA/EM'96*, pp. 270-273, 1996.

REFERENCES

1. Garey, M.R., and D.S. Johnson, Computers and Intractability, W.H. Freeman, New York, 1979.

2. Potvin, J.-Y., "The Traveling Salesman Problem: A Neural Network Perspective," *ORSA Journal on Computing*, Vol. 5, pp. 328–348, 1993.

3. Ghaziri, H., "Supervision in the Self-organizing Feature Map: Application to the Vehicle Routing Problem," in I.H. Osman and J.P. Kelly (Eds.), Metaheuristics: Theory and Applications, pp. 651–660, Kluwer, Boston, 1996.

4. Dagli, C.H., S. Lammers, and M. Vellanki, "Intelligent Manufacturing Using Neural Networks," *Journal of Neural Network Computing*, Spring, pp. 4–10, 1991.

5. Lee, Y.-H. and S. Kim, "Neural Network applications for Scheduling Jobs on Parallel Machines," *Computers and Industrial Engineering* 25/1–4, pp. 227–230, 1993.

6. Cherkassky, V., Y. Lee, and H. Lari-Najafi, "Self-organizing Network for Regression: Efficient Implementation and Comparative Evalution," *Proceedings of International Joint Conference on Neural Networks*, Seattle, WA, Vol. 1, pp. 79–84, 1991.

7. Kaparthi, S., N.C. Suresh, and R.P. Cerveny, "An Improved Neural Network Leader Algorithm for Part Machine Grouping in Group Technology," *European Journal of Operational Research*, Vol. 69, pp. 342–356, 1993.

8. Erkut, H., and S. Polat, "A Simulation Model for a Urban Fire Fighting System," *Omega*, Vol. 20, pp. 535–542, 1992.

9. Francis, R.L., L.F. McGinnis, and J. A. White, *Facility Layout and Location : An Analytical Approach*, 2nd edition, Prentice Hall, Englewood Cliffs, 1992.

10. Love, R.F., J.G. Morris, and J. Wesolowsky, *Facilities Location : Models and Methods*,

North - Holland, New York, 1988.

11. Microanalytics, Truckstops2 Tutorial Documentation, 2300 Clarendon Blud., Arlington, Virginia, and 1986 Queen St. E., Toronto, Ontario, 1994.

12. Roadnet, Roadnet Technologies Inc., 10540,York Road, Huntvalley, Maryland, 21030.

13. Love, R.F., and J.G. Morris, "Modelling Inter-City Road Distances by Mathematical Functions," *Operational Research Quarterly*, Vol. 23, pp. 61–71, 1972.

14. Altınel, İ.K., N. Aras, A. Alie, G. Cangür, R. Özel, and A. Yücel, Estimating Road Travel Distances in Türkiye, Research Paper Series No : FBE-IE-03/94-03, Department of Industrial Engineering, Boğaziçi University, İstanbul, 1994.

15. Berens, W., "The Suitability of the Weighted L_p norm in Estimating Actual Road Distances," *European Journal of Operational Research*, Vol. 34, pp. 39–43, 1988.

16. Brimberg, J., R.F. Love, and J.H. Walker, "The Effect of Axis Rotation on Distance Estimation," *European Journal of Operational Research*, Vol. 80, pp. 357–364, 1995.

17. Love, R.F., and J.H. Walker, "An Empirical Comparison of Block and Round Norms for Modeling Actual Distances," *Location Science*, Vol. 2, pp. 21–43, 1994.

18. Ward, J.E., and R.E. Wendell, "A New Norm Measuring Distance Which Yields Linear Location Problems," *Operations Research,* Vol. 28, pp. 836–844, 1980.

19. Aras, N., "Estimating Actual Road Distances in İstanbul and Türkiye," Unpublished M.S. Thesis, Boğaziçi University, 1993.

20. Love, R.F., and J.G. Morris, "Mathematical Models of Road Travel Distances," *Management Sciences*, Vol. 25, pp. 130–139, 1979.

21. Mittal, A.K., and V. Palsule, "Facilities Location with Ring Radial Distances," *Institute of Industrial Engineers Transactions*, Vol. 16, pp. 59–64, 1984.

22. Perreur, J., and J. Thisse, "Central Metrics and Optimal Location," *Journal of Regional Science*, Vol. 14, pp. 411–421, 1974.

23. Altınel, İ.K., and N. Aras, "Road Distance Estimation Between Two Locations in Türkiye," *Transactions on Operational Research*, Vol. 6, No.2, pp. 121–136.

24. Krarup, J., and P.M. Pruzan, "The Impact of Distance on Location Problems," *European Journal of Operational Research*, Vol. 4, pp. 256–269, 1980.

25. Berens, W., and F. Körling, "Estimating Road Distances by Mathematical Functions," *European Journal of Operational Research*,Vol. 21, pp. 54–56, 1985.

26. Berens, W., and F. Körling, "On Estimating Road Distances by Mathematical Functions–A Rejoinder," *European Journal of Operational Research*, Vol. 36, pp. 254–255, 1988.

27. Brimberg, J., P.D. Dowling, and R.F. Love, "The Weighted One-Two Norm Distance Model: Empirical Validation and Confidence Interval Estimation," *Location Science*, Vol. 2, pp. 91–100, 1994.

28. Brimberg, J., and R. F. Love, "A New Distance Function for Modeling Travel Distances in a Transportation Network," *Transportation Science*, Vol. 26, pp. 129–137, 1992.

29. Ward, J.E., and R.E. Wendell, "Using Block Norms for Location Modeling," *Operations Research*, Vol. 33, pp. 1074–1091, 1985.

30. Murtagh, B.A., and M.A. Saunders, MINOS 5.1 User's Guide, Technical Report No : SOL 83 - 20R, Stanford University, Stanford, California, 1983 (revised 1987).

31. Love, R.F., and J.G. Morris, "On Estimating Road Distances by Mathematical Functions," *European Journal of Operational Research*, Vol. 36, pp. 251–253, 1988.

32. Kirkpatrick, S., C.D. Gelatt, and M.P. Vecci, "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671–680, 1983.

33. Zurda, J.M., *Introduction to Artificial Networks*, St. Paul, MN: West, 1993.

34. Hertz, J., A. Krogh, and R.G. Palmer, *Introduction to the Theory of Neural Computation*, Addison Wesley, 1991.

35. Funahashi, K., "On the Approximate Realization of Continuous Mapping by Neural Networks," *Neural Networks*, Vol. 2, pp. 183–192, 1989.

36. Hornik, K., M. Stinchcombe, H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, Vol. 2, pp. 359–366, 1989.

37. Rumelhart, D.E., G.E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in: D.E. Rumelhart, J.L. McClelland and the PDP Research Group (eds.) *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol 1., pp. 318–362, MIT Press, 1986.

38. Silverman, B.W., *Density Estimation for Statistics and Data Analysis*, Chapman and Hall, 1986.

39. Duda, R.O., and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley, 1973.

40. Geman, S., E. Bienenstock, and R.Doursat, "Neural Networks and the Bias/Variance Dilemma," *Neural Computation*, Vol. 4, pp. 1–58, 1992.

41. Stone, C.J., "Consistent Nonparametric Regression," *The Annals of Statistics*, Vol. 5, pp. 595–645, 1977.

42. Alpaydın, E., "GAL: Networks That Grow When They Learn and Shrink When They Forget," *International Journal of Pattern Recognition and Artificial Intelligence*, Vol. 8, pp. 391–414, 1994.

43. Specht, D.F., "A General Regression Neural Network," *IEEE Transactions on Neural Networks*, Vol. 2, pp. 568–576, 1991.

44. Alpaydın, E., "Multiple Networks for Function Learning," *IEEE International Neural Network Conference*, San Francisco, March 1993, Vol 1, pp. 9–14, 1993.

45. Mani, G, "Lowering Variance of Decisions by using Artificial Neural Network Portfolios," *Neural Computation*, Vol. 3, pp. 484–486, 1991.

46. Perrone, M.P., "Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization," Ph.D. Dissertation, Brown University, 1993.

47. Wolpert, D.H., "Stacked Generalization," *Neural Networks*, Vol. 5, pp. 241–259, 1992.

48. Breiman, L., Stacked Regression, TR-367, Department of Statistics, University of California, Berkeley, 1992.

49. Zhang, X., J.P. Mesirov, and D.L. Waltz, "Hybrid System for Protein Secondary Structure Prediction," *Journal of Molecular Biology*, Vol. 225, pp. 1049–1063, 1992.

50. Brimberg, J., and G.O. Wesolowsky, "Probabilistic L_p Distances in Location Models," *Annals of Operations Research*, Vol. 40, pp. 67–75, 1992.

51. Fukunaga, K, *Introduction to Statistical Pattern Recognition*, 2nd edition, Academic Press, San Diego, 1990.

52. Kohonen, T, *Self-Organizing Maps*, Springer-Verlag, Berlin, Germany, 1995.

53. Kohonen, T., "The Self-Organizing Map," *Proc. IEEE*, Vol 78, pp. 1464–1480, 1990.

54. Fildes, R.A. and J.B. Westwood, "The Development of Linear Distance Functions for Distribution Analysis," *Journal of the Operational Research Society*, Vol. 29, pp. 585–592, 1978.

55. Gray, R.M., "Vector Quantization," *IEEE ASSP Mag.*, Vol 1, pp. 4–29, 1984.

56. Linde, Y., A. Buzo, and R.M. Gray, "An Algorithm for Vector Quantization," *IEEE Trans.*

Communication COM-28, pp. 84–95, 1980.

57. Makhoul, J., S. Rouchos, and H. Gish, "Vector Quantization in Speech Coding, " *Proc. IEEE*, Vol. 73, pp. 1551–1588, 1985.

58. Alpaydın, E., İ.K. Altınel and N. Aras, "Parametric Distance Functions versus Nonparametric Neural Networks for Estimating Road Travel Distances," *European Journal of Operations Research*, Vol. 92, pp. 230–243, 1996.

59. Oommen, J., "Absorbing and Ergodic Discretized Two Action Learning Automata," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 16, pp. 282–293, 1986.

60. Oommen, J., and J.K. Lanctôt, "Discretized Pursuit Learning Automata," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 20, pp. 431–438, 1990.

61. Oommen, J., N. Andrade, and S.S. Iyengar, "Trajectory Planning of Robot Manipulators in Noisy Workspaces Using Stochastic Automata," *International Journal of Robotics Research April 1991* pp. 135–148, 1991.

62. Lanctôt, J. K., and J. Oommen, "Discretized Estimator Learning Automata," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 22, pp. 1473–1483, 1992.

63. Tsetlin, M. L., *Automaton Theory and the Modelling of Biological Systems*, Academic Press, NewYork, 1973.

64. Lakshmivarahan, S., *Learning Algorithms: Theory and Applications*, Springer-Verlag, NewYork, 1981.

65. Narendra, K.S. , and M.A.L. Thathachar, *Learning Automaton: An Introduction*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

66. Oommen, J., and D.C.Y. Ma, "Stochastic Automata Solutions to the Object Partitioning Problem," *The Computer Journal*, Vol. 35, pp. A105–A120, 1992.

67. Altınel, İ.K., J. Oommen, and N. Aras, "Vector Quantization for Arbitrary Distance Function Estimation," *INFORMS Journal on Computing*, Vol. 9, No.4, pp. 439–451, 1997.

68. Oommen, J., İ.K. Altınel, and N. Aras, "Arbitrary Distance Function Estimation Using Vector Quantization," *Proceedings IEEE International Conference on Neural Networks*,Vol. 6, pp. 3062–3067, 1995.

69. von der Malsburg, Ch., "Self-organization of Orientation of Sensitive Cells in the Striate Cortex," *Kybernetic*, Vol. 14, pp. 85–100, 1973.

70. Amari, S. A., "Topographic Organization of Nerve Fields," *Bulletin of Mathematical Biology*, Vol. 42, pp. 339–364, 1980.

71. Kohonen, T., "Automatic Formation of Topological Maps of Patterns in A Self-organizing System," *Proceedings of 2nd Scandinavian Conference on Image Analysis*, Espoo, Finland, pp. 214–220, 1981.

72. Kohonen, T., "Self-organized Formation of Topologically Correct Feature Maps," *Biological Cybernetics*, Vol. 43, pp. 59–69, 1982.

73. Freeman, J.A., and D.M. Skapura, *Neural Network Algorithms, Applications, and Programming Techniques*, Addison-Wesley, Reading MA, 1991.

74. Kohonen, T., K. Makisara and T. Saramaki, "Phonotic Maps—Insightful Representation of Phonological Features for Speech Recognition," *Proceedings of the Seventh International Conference on Pattern Recognition*, pp. 182–185, 1984.

75. Kohonen, T., K. Torkkola, M. Shozokai, J. Kangas and O. Venta, "Microprocessor Implementation of a Large Vocabulary Speech Recognizer and Phonetic Typewriter for Finish and Japanese," *Proceedings of European Conference on Speech Technology*, pp. 377–380, 1987.

76. Kohonen, T., "The Neural Phonotic Typewriter," *Computer*, Vol. 21, pp.11–22, 1988.

77. Samarabandu, J. K. and O. E. Jakubowicz, "Principles of Sequential Feature Maps in Multi-Level Problems," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN-90-WASH-DC II-683-II686, 1990.

78. Orlando, G. A., R. Mann, and S. Haykin, "Radar Classification of Sea-ice Using Traditional and Neural Classifiers," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN-90-WASH-DC, II-263-II-266, 1990.

79. Neumann, E.K., D.A. Wheeler, A.S. Burnside, A.S. Bernstein, and J.C. Hall, "A Technique for the Classification and Analysis of Insect Courtship Song," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN-90-WASH-DC, II-257-II-262, 1990.

80. Marks, K.M. and K.F. Goser, "Analysis of VLSI Process Data Based on Self-Organizing Feature Maps," *Proceedings of Nuero-Nimes '88*, pp.337–347, 1988.

81. Tryba, V., K.M. Marks, U. Rütcker, and K. Goser, "Selbst-organizierende Karten Als Lernende Klassifizierende Speicher," *IFG Fachbericht*, Vol. 102, pp. 407–419, 1988.

82. Hemani, A., and A. Postula, "Scheduling by Self Organization," *Proceedings of the International Joint Conference on Neural Networks*, IJCNN-90-WASH-DC, II-543-II-548, 1990.

83. Graf, D.H., and W.R. Lalonde, "A Neural Controller for Collision-free Movement of General Robot Manipulators," *Proceedings of the IEEE International Conference on Neural Networks*, pp. 177-I-84, 1988.

84. Graf, D.H., and W.R. Lalonde, "Neuroplanners for Hand / Eye Coordination," *Proceedings of the International Joint Conference on Neural Networks*, II-543-II-548, 1989.

85. Martinetz, J., H.J. Ritter, and K.J. Schulten, "Three-dimensional Neural Net for Learning Visuomotor Coordination of a Robot Arm," *IEEE Transactions on.Neural Networks*, Vol. 1, pp. 131–136, 1990.

86. Ritter, H.J., J. Martinetz and K.J. Schulten, "Topology Conserving Maps Learning Visuo-

motor Coordination," *Neural Networks*, Vol. 2, pp. 159–168, 1989.

87. Ritter, H.J. and K.J. Schulten, "Topology Conserving Mapping for Learning Motor Tasks," *Proceedings of Neural Networks for Computing, AIP Conference*, pp. 376-380, 1986.

88. Wong, Y., "A Comparative Study of the Kohonen Self-Organizing Map and the Elastic Net," *Computational Learning Theory and Natural Learning Systems*, Vol. 2, pp. 401–413, 1996.

89. Ritter, H.J., "Asymptotic Level Density for a Class of Vector Quantization Processes," *IEEE Transaction on Neural Networks*, Vol. 2, pp. 173–175, 1991.

90. Angéniol, B., C. Vaubois, and J.Y. Le Texier, "Self-Organizing Feature Maps and the Traveling Salesman Problem," *Neural Networks*, Vol.1, pp. 289–293, 1988.

91. Papadimitriou, C.H., "The Euclidean Traveling Salesman Problem is *NP*-Complete," *Theoretical Computer Science*, Vol. 4, pp. 237–244, 1978.

92. Lawler, E. L., J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 1985.

93. Reinelt, G., *The Travelling Salesman. Computational Solutions for TSP Applications*, Springer-Verlag, Berlin, 1994.

94. Lin, S. and B. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, Vol. 21, pp. 498–516, 1973.

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

96. Hopfield, J.J, "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-state Neurons," *Proceedings National of Academy of Sciences, USA, Biophysics*, Vol. 81, pp. 3088–3092, 1984.

97. Tank, D.W., and J.J. Hopfield, "Simple 'Neural' Optimization Networks: An A/D Con-

verter, Signal Decision Circuit, and a Linear Programming Circuit," *IEEE Transactions on Circuits and Systems*, Vol. 33, No. 5, pp. 533–541, 1986.

98. Aiyer, S.V.B., M. Niranjan, and F. Fallside, "A Theoretical Investigation into the Performance of the Hopfield Model," *IEEE Transactions on Neural Networks* Vol. 1:2, pp. 204–215, 1990.

99. Brandt, R.D., Y. Wang, A.J. Laub, and S.K. Mitra, "Alternative Networks for Solving the Traveling Salesman Problem and the List-Matching Problem," *Proceedings of the International Conference on Neural Networks* Vol. 2, pp. 330–340, 1988.

100. Cuykendall, R. and R. Reese, "Scaling the Neural TSP Algorithm," *Biological Cybernetics*, Vol. 60, pp. 365–371, 1989.

101. Durbin, R. and D. Willshaw, "An Analogue Approach to the Travelling Salesman Problem using an Elastic Net Method," *Nature*, Vol. 326, pp. 689–691, 1987.

102. Simic, P.D., "Statistical Mechanics as the Underlying Theory of Elastic and Neural Optimization," *Network*, Vol. 1, pp. 363–374, 1990.

103. Vakhutinsky, A.I., and B.L. Golden, "A Hierarchical Strategy for Solving Traveling Salesman Problems Using Elastic Nets," *Journal of Heuristics*, Vol. 2, pp 67–76, 1995.

104. Johnson, D.S. and L.A. McGeoch, "The Traveling Salesman Problem: a Case Study," in E. Aarts and J.K. Lenstra (Eds.), *Local Search in Combinatorial Optimization*, pp. 215–310, Wiley, Chichester, 1997.

105. Cichocki, A., and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, Wiley, Chichester, 1994.

106. Smith, K., "Neural Networks for Combinatorail Optimization: A Review of More Than a Decade of Research," *INFORMS Journal on Computing*, Vol. 11, No. 1, pp. 15–34, 1999.

107. Fort, J.C., "Solving a Combinatorial Problem via Self-Organizing Process: An Application

of the Kohonen Algorithm to the Travelling Salesman Problem," *Biological Cybernetics*, Vol. 59, pp. 33–40, 1988.

108. Hueter, G.J., "Solution of the Traveling Salesman Problem with an Adaptive Ring," *Proceedings of the IEEE International Conference on Neural Networks*, San Diego, CA, pp. I-85–92, 1988.

109. Ritter, H.J., and K.J. Schulten, "Kohonen's Self-organizing Maps: Exploring Their Computational Capabilities," *Proceedings of the International Joint Conference on Neural Networks*, pp. 2455–2460, 1988.

110. Burke, L.I. and P. Damany, "The Guilty Net for the Traveling Salesman Problem," *Computers & Operations Research*, Vol. 19, No. 3/4, pp. 255–265, 1992.

111. DeSieno, D., "Adding a Conscience Mechanism to Competitive Learning," *Proceedings of the IEEE International Conference on Neural Networks* 1, 117–124, 1988.

112. Burke, L.I., "Neural Methods for the Traveling Salesman Problem: Insights From Operations Research," *Neural Networks*, Vol. 7, pp. 681–690, 1994.

113. Burke, L.I., "Conscientious Neural Nets for Tour Construction in the Traveling Salesman Problem: The Vigilant Net," *Computers & Operations Research*, Vol. 23, No. pp. 121–129, 1996.

114. Budinich, M., "A Self-Organizing Neural Network for the Traveling Salesman Problem That is Competitive with Simulated Annealing," *Neural Computation*, Vol. 8, pp. 416–424, 1996.

115. Budinich, M., and B. Rosario, "A Neural Network for the Traveling Salesman Problem with a Well Behaved Energy Function," in S. W. Ellacott, J . C. Mason, I. J. Anderson (Eds.), *Mathematics of Neural Networks, Models, Algorithms and Applicaitons*, Kluwer Academic Publishers, Boston–London–Dordrecht, 1997.

116. Reinelt, G., "TSPLIB—A Traveling Salesman Problem Library," *ORSA Journal on Com-*

puting, Vol. 3, No.4, pp. 376–384, 1991.

117. Jeffreys, C., and T. Niznik, "Easing the Conscious of the Guilty Net," *Computers & Operations Research*, Vol. 21, pp. 961–968, 1994.

118. Ascheuer, N., Hamiltonian Path Problems in the on-line Optimization of Flexible Manufacturing Systems, Technical Report No. TR 96–3, ZIB, Berlin, 1996.

119. Waterman, M.S., *Introduction to Computational Biology*, Chapman and Hall, Cambridge, 1995.

120. Ascheuer N., private communication, 1998.

121. Fisher, R. and L. Tippet, "Limiting Forms of the Frequency Distribution of the Largest or Smallest Number of a Sample," *Proceedings of Cambridge Phil. Society*, Vol. 24, pp. 180–191, 1928.

122. Derigs, U., "Using Confidence Limits for the Global Optimum in Combinatorial Optimization," *Operations Research*, Vol. 33, pp. 1024–1049, 1985.

123. Golden, B. and F.B. Alt, "Interval Estimation of a Global Optimum for Large Combinatorial Problems," *Naval Research Logistics Quarterly*, Vol. 26, pp. 69–77, 1979.

124. Golden, B. and W.R. Stewart, "Empirical Analysis of Heuristics," in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D. B. Shmoys (Eds.), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, 1985.

125. Los, M. and C. Lardinois, "Combinatorial Programming, Statistical Optimization and the Optimal Transportation Network Problem," *Transportation Research*, Vol. 16B, pp. 89–124, 1982.

126. Zanakis, S.H.,"Extended Pattern Search with Transformation for the Three-Parameter Weibull MLE Problem," *Management Science*, Vol. 25, pp. 1149–1161, 1979.

127. Cooke, P., "Statistical Inference for Bounds of Random Variables," *Biometrika*, Vol. *66*, pp. 367–374, 1979.

128. Dannenbring, D., "Estimating Optimal Solutions for Large Combinatorial Problems," *Management Science*, Vol. 23, pp. 1273–1283, 1977.

129. Zanakis, S.H.,"A Simulation Study of Some Simple Estimators of the Three-Parameter Weibull Distribution," *Journal.Statistical Computational Simulation*, Vol. 9, pp.101–106, 1979.

130. Mirkin, B., *Mathematical Classification and Clustering*, Kluwer Academic Publishers, Boston–London–Dordrecht, 1996.

131. Litke, J.D., "An Improved Solution to the Traveling Salesman Problem with Thousands of Nodes," *Communications of the ACM*, Vol. 27, pp. 1227–1236, 1984.

132. Karp, R., "Probabilistic Analysis of Partitioning Algorithms for the Traveling Salesman Problem in the Plane," *Mathematics of Operations Research*, Vol. 2, pp. 209–224, 1977.

133. Osman I.H. and G. Laporte, "Metaheuristics: A Bibliography," *Annals of Operations Research*, Vol. 63, pp. 513–623, 1998.

134. Schaffer, J.D., D. Whitley, and L.J. Eshelman, "Combinations of Genetic Algorihtms and Neural Networks. A Survey of the State of the Art," *Proceedings of the International Workshops on Combinations of Genetic Algorithms and Neural Networks*, pp. 1–37, 1992.

135. Yao, X., "Evolutionary Artificial Neural Networks," *International Journal of Neural Systems*, Vol. 4, pp. 203–222, 1993.