

A Thesis Submitted for the Degree of PhD at the University of Warwick

Permanent WRAP URL:

<http://wrap.warwick.ac.uk/167208>

Copyright and reuse:

This thesis is made available online and is protected by original copyright.

Please scroll down to view the document itself.

Please refer to the repository record for this item for information to help you to cite it.

Our policy information is available from the repository home page.

For more information, please contact the WRAP Team at: wrap@warwick.ac.uk

**The Design, Development and Application of a Combined Connectionist
Expert System and 'Pocket' Boltzmann machine approach to the
Dynamic Customer Assignment and Vehicle Routing Problem.**

Kevin Charles Gavigan

A thesis submitted to the University of Warwick for
admission to the degree of Doctor of Philosophy

Department of Engineering

June 1994

IMAGING SERVICES NORTH

Boston Spa, Wetherby
West Yorkshire, LS23 7BQ
www.bl.uk

CONTAINS DISKETTE

-UNABLE TO COPY-

**CONTACT UNIVERSITY IF
YOU WISH TO SEE THIS
MATERIAL**

Dedication and Acknowledgements

It is my pleasure and honour to dedicate this thesis to God the Father, Son and Holy Spirit; and to especially thank:-

- my wife, Mary Ellen Gavigan;
- my daughter, Rebecca Grace Gavigan;
- and my father and mother, Charles and Patricia Gavigan.

I would also like to thank: Mr Mike Hessey, for his valuable and enduring supervision and advice; Mr Robert Gumbley of Leyland Daf Vans, for his support and encouragement and Mr David Botterill and the staff at the Graduate Office for their friendly advice and patience.

Declaration

All of the original work described in this thesis is the result of the authors own independent research and has been completed since registering for this degree in April 1988. None of the work has been submitted for any other degree.

K. C. Gavigan

Abstract

This thesis describes the design, development and application of a combined connectionist expert system and 'Pocket' Boltzmann machine approach to a class of hard np -complete combinatorial optimization problems.

The problem of efficiently incorporating additional routing demands to existing consignments is encountered by a number of organizations, including those providing Dial-A-Ride bus services, parcel collection and delivery services and in some warehousing and Automated Guided Vehicle systems. The conventional approach to examples from this class of problems is either to attempt to solve them manually or to develop dedicated algorithms.

This thesis describes a novel connectionist expert system approach which can obtain very good or (near-) optimal assignments of multiple customers to multiple vehicles in examples of capacitated, stochastic, un-split demand, multiple-vehicle assignment and routing problems.

Given an initial assignment of customers, distributed across a fleet of capacitated vehicles, a set of modified 'Pocket' Boltzmann machines are generated, which attempt to determine optimal initial routes for each of the vehicles in the fleet. At random times, variable numbers of new, un-split demand customers, with random locations and requiring assignment, are introduced into the system.

A connectionist expert system is generated - whose connection weights are partially determined using incremental cost data computed by a set of 'Pocket' Boltzmann machines - which attempts to optimally assign as many as possible of the new, un-split demand customers to the available capacitated vehicles for the smallest increase in the value of a total cost function. It is also demonstrated that in computing the incremental costs, the Pocket Boltzmann machines can generate very good or near optimal tours that also satisfy a variable number of additional *hard* (mandatory) constraints.

TABLE OF CONTENTS:-

	<u>Page</u>
Title Page	i
Dedication and Acknowledgements	ii
Declaration	ii
Abstract	iii
Table of Contents	iv
List of Figures	ix
List of Tables	xii
List of Abbreviations	xiii
1.0 INTRODUCTION	1
1.1 The Problem Under Investigation and Current Position	1
1.2 The Approach	1
1.3 The Methodology	3
1.4 The Implementation	4
1.5 The Structure of this Thesis	4
2.0 PROBLEM DEFINITION	7
2.1 Introduction	7
2.2 The General Assignment Problem	7
2.3 Travelling Salesman Problem	8
2.4 The Vehicle Routing Problem	10
2.5 The Dynamic Customer Assignment Vehicle Routing Problem	11
2.6 Combined Assignment and Routing Problem and Related Research	15
2.7 Comments	17

TABLE OF CONTENTS (continued) :-

	<u>Page</u>
3.0 THE CONNECTIONIST PARADIGM	19
3.1 Introduction	19
3.2 Hopfield Networks	20
3.2.1 Introduction	20
3.2.2 The Discrete Hopfield Network Model	21
3.2.3 Energy Analysis in the Discrete Model	24
3.2.4 Encoding in the Discrete Model	26
3.2.5 Recall in the Discrete Model	27
3.2.6 The Continuous Hopfield Network Model	30
3.2.7 Energy Analysis in the Continuous Model	34
3.2.8 Encoding and Recall in the Continuous Model	35
3.2.9 Stability and Capacity	36
3.2.10 Strengths and Limitations	37
3.2.11 Solving the TSP using Hopfield and Tank's Method	38
3.2.12 Performance of Hopfield and Tank's Method	42
3.2.13 Conclusions	44
3.3 The Boltzmann machine	46
3.3.1 Introduction	46
3.3.2 Introduction to the Three layer Boltzmann machine	47
3.3.3 The Boltzmann machine Learning Algorithm	50
3.3.4 Simulated Annealing	55
3.3.5 Network Architecture and Cell Properties	58
3.3.6 Encoding in a three layer Boltzmann machine	60
3.3.7 Recall in a three-layer Boltzmann machine	69

TABLE OF CONTENTS (continued) :-**Page**

3.4	Solving the TSP using Aarts and Korst's Method	71
3.4.1	Introduction	71
3.4.2	Sequential Boltzmann machine and Consensus Maximization	73
3.4.3	The Asynchronous Parallel Boltzmann machine and Consensus Maximization	80
3.4.4	The Travelling Salesman Problem	82
3.4.5	Solving an example of the TSP on an Asynchronous Parallel Boltzmann machine	90
3.4.6	Performance of Aarts and Korst's Implementation	90
3.5	Application of the Boltzmann machine to the GAP, TSP and VRP	93
3.6	Parallel Boltzmann machine Research	93
3.7	Application of Other Connectionist Models to the TSP and VRP	94
3.8	Conclusions	96
4.0	CONNECTIONIST EXPERT SYSTEMS	99
4.1	Introduction	99
4.2	Gallant's Connectionist Expert System Model	100
4.2.1	Introduction	100
4.2.2	Connectionist Expert System Components	102
4.2.3	Knowledge Representation	104
4.2.4	Cell Properties and Network Structure	109
4.2.5	Network Generation	113
4.2.6	Network Training and the Pocket Learning Algorithm	116
4.2.7	The MACIE inference engine	125
4.2.8	Rule Extraction	130
4.2.9	Comments	132
4.3	Review of Connectionist Expert System Research	134
4.4	Review of Related Research	140
4.5	Comments	157

TABLE OF CONTENTS (continued) :-

	<u>Page</u>
5.0 THE OBJECT-ORIENTED PARADIGM	159
5.1 Introduction	159
5.2 Concepts and Terminology	160
5.2.1 Abstract Data Types	160
5.2.2 Objects and Classes	161
5.2.3 Encapsulation	164
5.2.4 Class Inheritance	166
5.2.5 Object Inheritance, Instance inheritance, Prototyping and delegation	169
5.2.6 Metaclasses	171
5.2.7 Object Instantiation (Creation) and Destruction	172
5.2.8 Message Passing	174
5.2.9 Polymorphism and Binding	175
5.2.10 Object Identity	179
5.2.11 Object Equivalence	182
5.2.12 Triggers	184
5.2.13 Related concepts	186
5.3 Advantages and Disadvantages for Conventional Application Development	186
5.4 Advantages and Disadvantages for KBS and Connectionist System Development	188
5.5 Conclusions	189
6.0 THE DESIGN DEVELOPMENT AND APPLICATION OF A CONNECTIONIST EXPERT SYSTEM AND 'POCKET BOLTZMANN MACHINE APPROACH TO THE DYNAMIC CUSTOMER ASSIGNMENT AND VEHICLE ROUTING PROBLEM	191
6.1 Introduction	191
6.2 Overview of the Design	192
6.3 Operation	196

TABLE OF CONTENTS (continued) :-

	<u>Page</u>
6.4 The Connectionist Expert System	202
6.4.1 Introduction	202
6.4.2 Description of Network Generation and Network Topology	202
6.4.3 Cell Properties, Connection Strengths and Threshold Values	207
6.4.4 Network Operation (Inferencing)	210
6.5 The 'Pocket' Boltzmann machine	215
6.5.1 Introduction	215
6.5.2 The 'Pocket' Boltzmann machine algorithm	215
6.5.3 Justification of the 'Pocket' Boltzmann machine algorithm	216
6.6 Implementation	220
6.6.1 Selection of a Suitable Development Environment	220
6.6.2 The Development Environment and Implementation and Operation of CONNEKT	222
6.7 Validation of the Pocket Boltzmann machine implementation	227
6.7.1 Introduction	227
6.7.2 Additional Implementation Details	228
6.7.3 Results	228
6.7.4 Analysis of Results	229
6.7.5 Conclusion	233
6.8 Performance of the Pocket Boltzmann machine implementation	235
6.8.1 Introduction	235
6.8.2 Results	235
6.8.3 Analysis of Results	240
6.8.4 Conclusion	240

TABLE OF CONTENTS (continued) :-**Page**

6.9	The Problem Examples and Results	242
6.9.1	Introduction	242
6.9.2	Problem 'A'	243
6.9.3	Problem 'B'	249
6.9.4	Problem 'C'	253
6.9.5	Problem 'D'	255
7.0	APPRAISAL AND EVALUATION	258
7.1	Introduction	258
7.2	Evaluation of Results and Appraisal and Critiscism of Performance	258
7.2.1	The Pocket Boltzmann Machine Model	258
7.2.2	The 'Pocket' Modification	259
7.2.3	The Connectionist Expert System	262
7.2.4	The Overall Performance of the Approach	263
7.3	Critiscism and Comparison with Other Approaches	264
7.4	Evaluation of Contribution	266
7.5	Suggestions for Further Research	267
8.0	CONCLUSIONS	269
	Appendices	275
	Bibliography	291
	References	294

List of Figures

<u>No.</u>	<u>Title</u>	<u>Page No.</u>
2.1	Tour for the k th vehicle at time t .	13
3.1	The topology of a discrete Hopfield Network	22
3.2	Ball-bearing on a bumpy surface (analogue of a simple network energy E landscape with two minima separated by an energy barrier).	28
3.3.	The topology of a continuous Hopfield Network.	31
3.4.	An Electronic implementation of the continuous Hopfield model.	33
3.5.	Topology of a Three-Layer Boltzmann machine.	59
4.1	Components of Gallant's connectionist expert system model.	104
4.2	Default dependency network with no dependency information.	105
4.3	Example dependency and training examples matrices.	106
4.4	Example Learning Matrix.	109
4.5	Example connectionist knowledgebase network.	111
4.6	The Perceptron Learning Algorithm.	118
4.7	The pocket learning algorithm.	121
4.8	Pocket Algorithm with ratchet.	122
4.9	The MACIE inference justification algorithm.	133
4.10	The topology of a RUBICON connectionist expert system network.	135
5.1	Simple class inheritance tree.	167
5.2	Simple class inheritance graph.	167
5.3	Example Semantic Network.	168

List of Figures (continued)

<u>No.</u>	<u>Title</u>	<u>Page No.</u>
5.4	Example object Graph with identical and shallow equal objects.	183
5.5	Example object graph with deep equal objects.	185
6.1	The CONNEKT customer entry/editing screen.	193
6.2	The CONNEKT vehicle entry/editing screen.	194
6.3	Topology of the CONNEKT Connectionist Expert System.	203
6.4	Variation in the Average Tour Length (pocket modification disabled).	230
6.5	Variation in the Frequency with which tours are obtained (pocket disabled).	230
6.6	The optimum tour for Hopfield and Tank's 10 city problem.	232
6.7	Variation in the Average Tour Length with the pocket modification enabled.	238
6.8	Variation in the Frequency with which tours are obtained with the pocket modification enabled.	239
6.9	Problem A - Locations of customers and initial tours for each of the vehicles.	244
6.10	Optimal assignment of customer C5 to vehicle V1.	247
6.11	Customer Co-ordinates and Initial tours for problem example B.	250
6.12	Optimal Assignment of customer C6 to vehicle V2 - with optimal new tour.	252
6.13	Optimal Assignment of customer C6 to vehicle V2-with non-optimal new tour.	254

List of Tables

No.	Title	Page No.
1.1	Components of the Methodology employed.	3
3.1	Statistics describing the results of applying the parallel Boltzmann machine model to instances of the 10 city and 30 city problems.	90
6.1	Co-ordinates of the cities in Hopfield and Tank's [1985] 10-city problem as determined by Wilson and Pawley[1988].	227
6.2a-c	Performance of the Pocket Boltzmann machine (pocket) disabled for different values of theta.	229
6.3	Aarts and Korst[1989a] Boltzmann machine implementing the LAP formulation.	231
6.4	Percentage difference in frequency and average tour length.	231
6.5	Corrected average tour length.	232
6.6a,b	Statistics describing generation of best (pocket) tours.	236
6.7a,b	Statistics describing generation of final state tours.	237
6.8	Assignment, availability and collection and destination locations of customers, and starting and end locations for vehicles in Problem A.	244
6.9	Statistics describing the generation of initial tours for vehicles $V1, V2$ in problem example A.	245
6.10	Statistics describing the assignment of a single customer C5 to one of two vehicles V1, V2.	246
6.11	Customer assignment, availability and collection and destination co-ordinates and vehicle co-ordinates for problem B.	250
6.12	Random co-ordinates and incremental costs for least and furthest strategies.	257

List of Abbreviations

Abbreviation	Description
ANS	Artificial Neural System
API	Application Programming Interface
CH	Continuous Hopfield Network
CONNEKT	Combined Object-oriented Neural Network and connectionist Expert sysTem
CPU	Central Processing Unit
DCAP	Dynamic Customer Assignment Problem
DCAVRP	Dynamic Customer Assignment and Vehicle Routing Problem
DLL	Dynamic Link Library
DMM	Distributed Memory Multiprocessor
GAP	General Assignment Problem
IBM	International Business Machines
LAP	Linear Assignment Problem
MACIE	MAtrix Controlled Inference Engine
MBNN	Modified Boolean Neural Network model
NT	New Technology
OLE	Object Linking and Embedding
PC	Personal Computer
QAP	Quadratic Assignment Problem
VLSI	Very Large Scale Integration
VRP	Vehicle Routing Problem
TCF	Total Cost Function

List of Abbreviations(continued)

Abbreviation	Description
TSP	Travelling Salesman Problem

1.0 INTRODUCTION

1.1 The Problem Under Investigation and Current Position

This thesis describes a novel connectionist approach for determining solutions to examples of a class of difficult, combinatorial optimization problems, specifically, Dynamic Customer Assignment and Vehicle Routing Problems (DCAVRP's).

This class of problems - as defined in this thesis - is created by combining two types of hard, *np*-complete problems (Garey and Johnson, [1979]), these are: a form of the General Assignment Problem (GAP) ; and the Vehicle Routing Problem (VRP).

The Dynamic Customer Assignment and Vehicle Routing Problem is concerned with the optimal assignment of new, un-split demand customers to a fleet of capacitated vehicles with pre-determined assignments and routes. This class of problems is of particular significance to organizations that wish to offer flexible distribution services, that are able to dynamically and efficiently accommodate new, previously unknown, demands. These could include for example, parcel collection and delivery services, Dial-A-Ride bus services and some Automated Guided Vehicle distribution systems.

The conventional approach to examples from this class of problems is either to develop and apply dedicated algorithms; or to solve them manually. Improving or even optimizing the performance of such systems may result in significant resource and cost savings.

1.2 The Approach

This thesis describes a general connectionist based approach which can develop very good or near-optimal solutions to examples from this class of problems. A connectionist expert system, based on Stephen Gallant's Matrix Controlled Inference Engine (MACIE) connectionist expert system model (Gallant [1985, 1986a, 1986b,

1987, 1988a, 1988b, 1990a, 1990b, 1993]), is used to determine which of the new customers should be selected and assigned to one or more of the available vehicles. The connectionist expert system is automatically generated in response to a particular example of the problem. The precise network topology and the state of units within the network reflect the status of individual entities in the problem domain. The values of a sub-set of the connection weights are determined using a set of dynamically generated 'Pocket' Boltzmann machines, which are used to compute the incremental cost of assigning a particular customer to a specific vehicle and to develop very good or near-optimal tours for each of the vehicles in the system. The assignment process terminates when either all of the unassigned customers have been assigned, or when all of the vehicles in the fleet are at full capacity.

The 'Pocket' Boltzmann machine is a novel modification of Aarts and Korst's [1989a, 1989b] implementation of the Boltzmann machine originally introduced by Ackley, Hinton and Sejnowski [1985]. The modification is inspired by Gallant's Pocket Algorithm (Gallant [1985, 1988a, 1993]) which Gallant has applied to the Perceptron and Back-propagation models. The 'Pocket' modification stores a copy (in a 'pocket') of the 'best' solution generated by the Boltzmann machine, that is, the one that best satisfies the *soft* constraints encoded in the Boltzmann machine network's pattern of connectivity and connection weight values; and a set of additional *hard* constraints. Each time a new solution is generated that satisfies *all* of the constraints, it is compared with the solution stored in the pocket, and if superior, replaces it.

The approach was applied, tested and evaluated by developing an object-oriented, 'graphical-user-interface' (GUI) application called the Combined Object-oriented Neural Network and connectionist Expert system (CONNEKT), which provides a environment which enables DCAVRP examples to be modelled, stored and solved.

1.3 The Methodology

The methodology employed in the implementation of the approach outlined above, and described in this thesis, is comprised of a number of component steps. These are listed in table 1.1.

<u>Step</u>	<u>Description</u>
1.	Development of a general knowledge of, the scope, and content of research in the areas of:- <ul style="list-style-type: none">• connectionist models;• connectionist expert systems;• the application of connectionism to combinatorial optimization;• algorithmic approaches to combinatorial optimization;• object-orientation and the simulation of connectionist systems.
2.	Analysis and definition of the problem.
3.	Literature search and review to support the development of a detailed understanding of the research into the solution of problems of this type.
4.	Development of a design proposal derived by extending and combining components of reviewed research.
5.	Evaluation of potential hardware platforms and software development tool(s).
6.	Selection, purchase and installation of suitable platform and development tool(s).
7.	Implementation of the design proposal.
8.	Proving the design and testing the implementation of the components of the design.
9.	Experimentation.
10.	Qualitative and quantitative evaluation of performance.
11.	Comparison with alternative approaches.
12.	Determination of areas for further research and development.
13.	Development of conclusions.

Table 1.1 : Components of the Methodology employed

1.4 Implementation

The Combined Object-oriented Neural NEtwork and connectionist expert sysTem, henceforth referred to as 'CONNEKT' was developed using Microsoft Visual C++ v2.0 to run under the Microsoft Windows NT (v4.0) operating system on a (single processor) personal computer (PC).

Although the approach enables very good or even near-optimal solutions to be obtained, it is worth stating, even at this early stage, that preliminary results indicate that the approach is computationally expensive when the 'Pocket' Boltzmann machines are simulated on a single processor (Pentium 90MHz) PC architecture. It is however argued that this processing 'bottleneck' could be significantly overcome, and hence the approach could become viable, if the 'Pocket' Boltzmann machines were implemented using parallel arrays of general purpose or dedicated Boltzmann machine processors.

1.5 The Structure of this Thesis

The work described in this thesis crosses a number of established boundaries and is formed from a conjunction of components of connectionism (including connectionist expert systems research); the object-oriented paradigm and operations research. It was therefore considered that reviews of relevant research from these areas should be included in context, that is, within the chapter dedicated to that research area.

The first chapter introduces the problem; describes the approach taken; the methodology employed; its implementation via CONNEKT and finally provides this description of the structure of the thesis.

Chapter two provides a formal description of the problem. The GAP, VRP and Travelling Salesman Problem (TSP) are first formally described; as these form the basis for the subsequent definition of the Dynamic Customer Assignment and Vehicle

Routing Problem. The penultimate section in chapter two, provides a review of research into combined assignment and routing problems, whilst the final section provides a brief commentary on some of the features of *np*-complete combinatorial optimization problems.

The third chapter introduces the Connectionist Paradigm and describes three connectionist models in detail. These are: Hopfield's discrete model (Hopfield [1982]), the Continuous Hopfield model (Hopfield [1984]), and the Boltzmann machine, originally introduced by Ackley, Hinton and Sejnowski [1985]. Hopfield's discrete model is described as it formed the basis for the development of Hopfield's continuous model, which was first used by Hopfield and Tank [1985], to demonstrate that connectionist networks could model and compute good solutions to combinatorial optimization problems, specifically to examples of the Travelling Salesman Problem. A description of the Boltzmann machine and its application to the TSP, is included as this forms the basis for the 'Pocket' Boltzmann machines implemented in CONNEKT.

The connectionist expert system component of CONNEKT is based on the Matrix Controlled Inference Engine (MACIE) connectionist expert system model, developed by Stephen Gallant [1985, 1986a, 1986b, 1987, 1988a, 1988b, 1990a, 1990b, 1993]. Gallant's model is described in detail in the fourth chapter, and follows a review of connectionist expert system research.

Connectionism shares a number of fundamental characteristics with the object-oriented paradigm. In both approaches, systems are developed which operate at the microscopic level, through the interaction (via forms of message passing) of typically many, small, self-contained components, each with its own attributes and behaviour. The macroscopic behaviour of both connectionist and object-oriented systems is similarly, a collective function of each of the individual behaviours and interactions of

some sub-set of the components (objects or nodes) that comprise the system.

Object-oriented development methodologies and languages therefore, provide a natural framework for the development of software simulations of connectionist networks. CONNEKT was developed, using (Microsoft Visual) C++ as an object-oriented application. The fifth chapter therefore, describes the key components and characteristics of the object-oriented approach, as this supports the subsequent description of the design, implementation, operation and performance of CONNEKT.

The sixth chapter provides a description of the research. Specifically, the design, implementation and operation of the connectionist expert system; 'Pocket' Boltzmann machines; and other components of CONNEKT, are described in detail. The chapter concludes with a description of the results of applying CONNEKT to examples of the Dynamic Customer Assignment and Vehicle Routing Problem introduced in section 1.1 and defined in section 2.5.

The penultimate, seventh chapter, provides an appraisal of the performance of CONNEKT; a comparison with other approaches; an evaluation of the contribution made; and a number of suggestions for further research. A number of conclusions are then introduced and expounded in the eighth and final chapter.

2.0 PROBLEM DEFINITION

2.1 Introduction

This chapter provides a formal description of the class of combinatorial optimization problems solved by CONNEKT, that is, Dynamic Customer Assignment and Vehicle Routing Problems (DCAVRP's). This class of problems is formed from the combination of two separate classes of hard, *np*-complete (Garey and Johnson,[1979]) problems. Specifically, these are a form of the General Assignment Problem (GAP) with a number of additional constraints, which is referred to as the Dynamic Customer Assignment Problem (DCAP), and the Vehicle Routing Problem (VRP).

The Vehicle Routing Problem, in the formulation used here, which follows Laporte, Nobert and Desrochers [1985], is in turn, comprised of a set of TSP's with a number of additional constraints.

The section that immediately follows defines the General Assignment Problem. Subsequent sections define the Travelling Salesman Problem, the Vehicle Routing Problem and the Dynamic Customer Assignment and Vehicle Routing Problem.

The penultimate section of this chapter, provides a review of research into combined assignment and vehicle routing problems, whilst the final section provides a brief commentary on a number of the features of *np*-complete combinatorial optimization problems.

2.2 The General Assignment Problem

The General Assignment Problem is a well known and widely studied, hard, *np*-complete, combinatorial optimization problem. The formulation of the GAP described here, essentially follows that of Fisher, Jaikumar and Van Wassenhove [1986]:

The problem is concerned with the assignment of l jobs, from the set of jobs,

$J = \{j_1, \dots, j_l\}$ to m available agents, which are members of the set of agents, $A = \{a_1, \dots, a_m\}$. If c_{ij} denotes the value of having the i th agent perform the j th job; b_i represents the resource availability of the i th agent; r_{ij} denotes the resource required by the i th agent to perform the j th job and:

$$x_{ij} = \begin{cases} 1 & \text{if the } i\text{th agent performs the } j\text{th job} \\ 0 & \text{otherwise} \end{cases} \quad [2.2.1]$$

The generalized assignment problem can then be formulated as:

$$T = \max \left(\sum_{i \in A} \sum_{j \in J} c_{ij} x_{ij} \right) \quad [2.2.2]$$

where:

$$\sum_{i \in A} x_{ij} = 1, \quad \forall j \in J \quad [2.2.3]$$

$$\sum_{j \in J} r_{ij} x_{ij} \leq b_i, \quad \forall i \in A \quad [2.2.4]$$

$$\sum_{i \in A} x_{ij} = 0 \text{ or } 1 \quad \forall i \in A, \forall j \in J \quad [2.2.5]$$

The objective in the General Assignment Problem, as defined in equation 2.2.2 is to find an optimal assignment of jobs to agents, that maximises the value of T , given the constraints that each job is wholly assigned to a single agent (equations 2.2.1, 2.2.3, 2.2.5), and the capacity of none of the agents can be exceeded (equation 2.2.4).

2.3 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is a classic example of a difficult optimization problem (Hopfield and Tank, [1985]). The problem consists of finding a short or minimal path length for a closed tour of n cities $\{A, B, C, \dots\}$ such that each city is visited exactly once, and the tour terminates at the city from which it started. The cities are separated by the (pairwise) distances $\{d_{AB}, d_{AC}, \dots, d_{BC}, \dots\}$. Each of the possible solutions to the problem can be described as some sequence of the form

C, F, A, \dots, D , with a corresponding total path length given by $d = d_{CF} + d_{FA} + \dots + d_{DC}$ or more formally, as in Aarts and Korst [1989b], by the cyclic permutation :

$$\pi = (\pi(1), \pi(2), \dots, \pi(k), \dots, \pi(n)) \quad [2.3.1]$$

where $\pi(k)$ denotes the successor city to city k in the tour, and:

$$\pi^l(k) \neq k \text{ for } l = 1, 2, \dots, n-1 \quad [2.3.2]$$

$$\text{and } \pi^n(k) = k \quad [2.3.3]$$

where: $\pi^l(k)$ denotes the successor city to city k at the l th step (position) on the tour.

The solution space is described by the set 'S' where:

$$S = \{\pi_1, \pi_2, \dots, \pi_p, \dots\} \quad [2.3.4]$$

and π_p denotes the p th cyclic permutation on n cities.

The distance between any pair of cities i, j can be described using the element(s) d_{ij} (and d_{ji}) of a symmetric $n \times n$ matrix \mathbf{D} . The distance between any city i and its successor $\pi(i)$ is then given by the element $d_{i, \pi(i)}$ of \mathbf{D} .

The total path length is given by the value of the cost function $f()$, where $f(\pi)$ is the length of the tour corresponding to cyclic permutation π , defined by the equation:

$$f(\pi) = \sum_{i=1}^n d_{i, \pi(i)} \quad [2.3.5]$$

The p th cyclic permutation, π_p , is globally optimal if it satisfies the condition:

$$f(\pi_p) \leq f(\pi_q), \text{ for all } \pi_q \in S \quad [2.3.6]$$

For the travelling salesman problem more than one cyclic permutation may satisfy the

condition specified by equation 2.3.6, and describe a tour that corresponds to a minimal value of the cost function. For convenience all cyclic permutations that correspond to globally optimal tours, denoted π_{opt} , are assigned to the set of optimal solutions S_{opt} where $S_{opt} \subseteq S$.

2.4 The Vehicle Routing Problem

The operations research literature contains a number of versions of the Vehicle Routing Problem (VRP). The version described in this section - and later combined with the Dynamic Customer Assignment Problem - essentially follows that described by Laporte, Nobert and Desrochers [1985], and is therefore comprised of a set of TSP's, as defined in section 2.3, with a number of additional constraints.

The Vehicle Routing Problem as formulated by Laporte, Nobert and Desrochers [1985]) has the following characteristics:

1. A symmetrical graph $G(N, E)$ where $N = \{1, 2, \dots, l\}$ represents a set of nodes, N , which correspond to the locations of some entity type(s), for example, cities or customers, and a set of undirected edges, $E = \{(i, j) : i, j \in N, i < j\}$. The matrix $\mathbf{D} = d_{ij}$, is the matrix of distances associated with the edges. The element, d_{ij} , stores the length of the edge between the node i and node j . The distance, d_{ij} is interpreted as the distance d_{ji} if $j > i$. \mathbf{D} is said to satisfy the triangle inequality if and only if: $d_{ij} \leq d_{ik} + d_{kj}, (i, j, k \in N)$. The set of nodes, N , contains a *depot* which is represented by the first node. Each node (customer), i , has a non-negative weight, r_i , representing a requirement or demand, associated with it. By convention, no requirements (demands) are made on any of the vehicles at the depot, that is, $r_1 = 0$.

2. Each member of a fleet of m identical vehicles, denoted by the set $V = \{v_1, v_2, \dots, v_m\}$, is based at the depot and has capacity n_{\max} , where $n_{\max} \geq \max(r_i)$.
3. Each node, except for the first node (the depot), must be visited exactly once and by a single vehicle.
4. Each vehicle starts and ends its journey at the depot. The maximum load (total sum of weights contained on a vehicles route) must not exceed the vehicle's capacity, n_{\max} , and may not exceed a pre-specified maximum upper limit, L .
5. The objective, is then, to minimize the value of a total cost function - which corresponds to the total distance travelled by all of the vehicles in the fleet - whilst satisfying all of the constraints.

If the set $\Pi_t = \{\pi_{t,1}, \pi_{t,2}, \dots, \pi_{t,m}\}$ denotes the set of cyclic permutations which correspond to routes (tours) for each of the vehicles in the fleet, at time t , then the total cost function for the vehicle routing problem is given by the expression:

$$g(\Pi_t) = f(\pi_{t,1}) + f(\pi_{t,2}) + \dots + f(\pi_{t,m}) \quad [2.4.1]$$

where $f(\pi)$ is defined as in the TSP, by equation 2.3.5.

2.5 The Dynamic Customer Assignment and Vehicle Routing Problem

The Dynamic Customer Assignment and Vehicle Routing Problem (DCAVRP) is formed from the combination of two hard, np -complete sub-problems, these are a particular form of the General Assignment Problem, referred to as the Dynamic Customer Assignment Problem (DCAP) and the Vehicle Routing Problem.

The Dynamic Customer Problem (as defined in this thesis), is a form of the General Assignment Problem with a number of additional constraints. In the DCAP, each of the members of the set of (un-split demand) customers $C = \{c_1, c_2, \dots, c_l\}$ (which correspond to jobs) are wholly assigned to a single member of the set of available vehicles $V = \{v_1, v_2, \dots, v_m\}$ (which correspond to agents). Each of the customers has associated with it, a *collection* point that is a member of the set, $P_c = \{p_{c1}, p_{c2}, \dots, p_{cl}\}$ and a *destination* point, that is a member of the set $P_d = \{p_{d1}, p_{d2}, \dots, p_{dl}\}$. The collection and destination points for the i th customer, c_i , are then respectively denoted by p_{ci} and p_{di} .

Initially and at any subsequent time t , each of the members of the set of customers, C , are either wholly assigned to a single vehicle or are unassigned. If $C_{a,k}$, where $C_{a,k} \subset C$, denotes the set of customers assigned to the k th vehicle at time t , and C_u denotes the set of unassigned customers, then, this can be expressed by:

$$\sum_{k \in V} C_{a,k} + C_u = C \quad \forall t = 0, 1, \dots, \infty \quad [2.5.1]$$

If the depot is denoted by the point, p_{depot} , and $P_{c,t,k} \subset P_c$ and $P_{d,t,k} \subset P_c$ respectively denote the set of collection points, and the set of destination points associated with the customers assigned to the k th vehicle at time t , then, the set of locations, at any time t , that the k th vehicle must visit, after leaving the depot and before returning to it, is given by the set $P_{t,k}$, where:

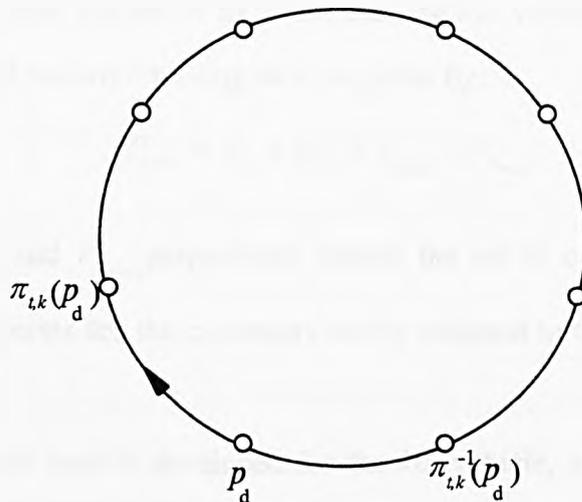
$$P_{t,k} = P_{c,t,k} + P_{d,t,k} + p_{depot} \quad \forall k = 1, 2, \dots, m, \forall t = 1, 2, \dots, \infty \quad [2.5.2]$$

A *valid tour* for each of the vehicles in the set, at any time t , is defined as one that starts at the depot, visits all of the collection and destination points of all of the customers assigned to the vehicle, at time t , and returns to the depot; in such a way that all other additional constraints are also satisfied.

The only additional constraints that must be satisfied are that for all customers, the tour must visit the customers collection point before it visits that customers destination point, and that at all times, the maximum capacity of the vehicle is not exceeded.

A valid tour for the k th vehicle at time t , is described by the cyclic permutation $\pi_{t,k}$ and illustrated in figure 2.1:

Figure 2.1 : Tour for the k th vehicle at time t .



where : $\pi_{t,k}(p_d)$ denotes the successor location to location p_d

$\pi_{t,k}^{-1}(p_d)$ denotes the successor location for which the successor on the tour is p_d

The length of the tour corresponding to the cyclic permutation, $\pi_{t,k}$, is $f(\pi_{t,k})$ and its value is given by equation 2.3.5. If $\Pi_t = \{\pi_{t,1}, \pi_{t,2}, \dots, \pi_{t,m}\}$ denotes the set of tours developed for each of the vehicles in the fleet, then a total cost function, $g(\Pi_t)$ is defined (in accordance with the VRP equation 2.4.1) to be:

$$g(\Pi_t) = \sum_{k \in V} f(\pi_{t,k}) \quad [2.5.3]$$

An *optimal tour* for the k th vehicle at time t , is any valid tour for that vehicle which has the shortest path possible length, that is, any tour that satisfies the condition specified in (the TSP) equation 2.3.6.

Now, at each time t , a random number of customers are randomly selected from the set of unassigned customers, C_{u_t} , and assigned to a sub-set of C_{u_t} , which is denoted by C_{req_t} , where $C_{req_t} \subset C_{u_t}$. This sub-set describes the set of customers that require assignment at time t . Once a customer is assigned to a particular vehicle, it is removed from the set of customers requiring assignment, and becomes a member of the set of customers assigned to that vehicle.

If, at time t , one or more of the customers requiring assignment are assigned to the k th vehicle, then the set of locations that the k th vehicle must now visit after leaving the depot and before returning to it, is given by:

$$P_{t+1,k} = P_{c_{t,k}} + P_{d_{t,k}} + P_{c_{req,t,k}} + P_{d_{req,t,k}} \quad [2.5.4]$$

where: $P_{c_{req,t,k}}$ and $P_{d_{req,t,k}}$ respectively denote the set of collection points, and the set of destination points for the customers newly assigned to the k th vehicle.

If a new valid tour is developed for the k th vehicle, visiting each of the locations in the set $P_{t+1,k}$, and this is denoted by the cyclic permutation $\pi_{t+1,k}$; then the length of the new tour is given by the value of the cost function, $f(\pi_{t+1,k})$ and the incremental increase in the value of the cost function for the k th vehicle, at time $t+1$, $I_{t+1,k}$, is given by the simple expression:

$$I_{t+1,k} = f(\pi_{t+1,k}) - f(\pi_{t,k}) \quad [2.5.5]$$

Now, the case for the k th vehicle must be generalized, since members of the set of customers requiring assignment, C_{req_t} , may have been assigned to any or all of the available vehicles in the fleet.

If $g(\Pi_t)$ and $g(\Pi_{t+1})$, respectively denote the total cost function (TCF) for the sets of valid tours, Π_t and Π_{t+1} , at time t and $t+1$, as defined by equation 2.5.3, then the total difference in the cost function resulting from assigning as many of the members

of C_{req} as possible, is given by the expression:

$$g'(\Pi_{t+1}, \Pi_t) = g(\Pi_{t+1}) - g(\Pi_t) = \sum_{k \in V} I_{t+1,k} \quad [2.5.6]$$

The objectives of the dynamic customer assignment and vehicle routing problem can now be defined as:

- Given some initial (possibly random) assignment of customers to vehicles, at every discrete point in time, t , to assign any new un-split demand customers to members of the fleet of capacitated vehicles in such a way as to minimize the increase in the total cost function (TCF) without exceeding the capacity of any of the vehicles; and at each point in time, to develop optimal valid tours for each of the capacitated vehicles in the fleet.

2.6 Combined Assignment and Routing Problem and Related Research

The research described in this section is confined to that research which is concerned with; or closely related to, combined assignment and vehicle routing problems. Research concerned with the application of connectionist models to examples of the TSP and VRP are described separately, in context, following a description of the relevant connectionist model.

Despite an extensive literature search, no examples of research into the precise formulation of the DCAVRP described in section 2.5 were found. However, a number of researchers have developed algorithmic solutions to problems that are related to the DCAVRP.

Some of the earliest work which relates to generalised assignment and vehicle routing problems is that of Fisher and Jaikumar, (published separately as Fisher [1981] and Jaikumar [1981]). They define the GAP and provide a review of the implementation

of the Lagrangian relaxation method; and then describe a method for vehicle routing that is based on a generalised assignment model, in which agents, corresponding to vehicles (trucks) and jobs; to items requiring delivery. Fisher, Jaikumar and Wassenhove [1986] describe a branch and bound algorithm (see for example, Winston [1992]) for the Generalised Assignment Problem. The bounds are obtained using Lagrangian relaxation with the multipliers set using an heuristic Multiplier Adjustment Method. They present results for the multiplier adjustment method which suggest that for easy problems, it requires comparable processing time when compared with the algorithm developed by Martello and Toth [1981]; whilst for all other problems, it is up to a factor of 10 times faster. Their results include statistics for their method applied to instances of the GAP, formulated as in their earlier work, as a vehicle routing problem. Their results indicate for example, that their approach can attempt to find the minimum distance travelled, when a fleet of 10 vehicles is scheduled to deliver a total of 140 customer orders, in less than 11 central processor unit (CPU) seconds on a DEC 10 computer.

Federgruen and Zipkin [1984] address:

"the combined problem of allocating a scarce resource available at some central depot among several locations (or "customers"), each experiencing a random demand pattern, while deciding which deliveries are to be made by each of a set of vehicles and in what order."

They decompose the problem into an inventory allocation problem and a TSP for each vehicle and show that the r -opt methods described by Lin [1965] and Lin and Kernighan [1973] can be adapted to accommodate inventory allocation and then address the combined problem of allocating a scarce resource among several locations and planning the delivery of these resources using the fleet of vehicles. They present the results of 18 runs on an IBM4341 computer, for examples with up to 9 vehicles, and either 50 or 75 locations. They report that the time spent in allocation was a substantial fraction of the total, and that computation times varied between 3-7 and

7-16 CPU seconds for the two problem sizes.

Soumis et al. [1991] use a minimum cost flow model followed by heuristic tour construction and improvement procedures to solve the simultaneous origin assignment and vehicle routing problem. The problem consists of selecting units available at a series of origins and assigning them to satisfy demands at a series of destinations and then developing vehicle routes to support the transport of these units to their required destinations. They report that their approach allows larger problems to be solved, and present results involving the transport of 24,000 students working on an environmental cleanup project. They state that solutions to large test problems are shown to be 1% or 2% from the optimal solution.

As stated previously, neural network approaches to combinatorial optimization problems are described separately, in the chapter dedicated to connectionism (chapter three). A number of researchers have investigated and/or applied algorithmic approaches to various forms of the vehicle routing problem. these include:

Balakrishnan [1993]; Bertsimas and Van Ryzin [1993]; Potvin and Rousseau [1993]; Bertsimas[1992]; Desrochers et al. [1992]; Li et al. [1992]; Laporte [1992]; Dumas et al. [1991]; Savelsbergh [1990]; Golden and Wasil [1987]; Solomon[1987]; Golden and Assad [1986] and Laporte, Norbert and Desrochers [1985].

2.7 Comments

In the travelling salesman problem, if the number of locations to be visited is n , the number of distinct tours with potentially different path lengths is $n!/2n$. This is because a tour can start at any one of n different cities, there are then $n-1$ possible successors to the first city, $n-2$ potential successors to the second city, and so on until at the penultimate city only one choice remains. The total number of possible tours is

then: $n! = n(n-1)(n-2), \dots, 1$. However, $2n$ of these must have the same cost function value, as there is an n -fold degeneracy of the initial city in the tour, that is, n tours have an identical sequence except that the sequence starts at a different city. There is also a two-fold degeneracy of the tour sequence order, since, for each tour there is a tour with exactly the reverse sequence. Therefore, the number of distinct tours with potentially different path lengths is then $n!/2n$ [Hopfield and Tank, 1985].

The TSP is an example of a hard NP-complete optimization problem [Garey and Johnson, 1979] as the time required to find the best solution from the $n!/2n$ distinct tours, grows exponentially as the number of cities in the tour is increased. The important practical consequence of this, is that attempts to find an optimum or best solution through an exhaustive generation and examination of all possible solutions in the solution set S is impracticable for all but the simplest of problems.

The dynamic customer assignment and vehicle routing problems contain a (variable number of) travelling salesman sub-problems, it is clear then from the structure of the DCAVRP's described in section 2.3, that these are also examples of hard *np-complete* combinatorial optimization problems.

In this thesis a connectionist approach is taken to the solution of instances of the class of dynamic customer assignment and vehicle routing problems, described above. The application of connectionist techniques to combinatorial optimization problems is examined in chapter three. However, a number of algorithmic techniques have also been developed; the most influential of these are probably the simulated annealing algorithm [Kirkpatrick, Gelatt and Vecchi [1983] and the local search algorithms of Lin [1965], and Lin and Kernighan [1973]. The quality and performance of the connectionist approach taken in this thesis is evaluated and compared with a number of these other techniques, in chapter seven.

3.0 THE CONNECTIONIST PARADIGM

3.1 Introduction

Connectionist Networks are computational models comprised of typically many, simple, interconnected units and are in part inspired by the capabilities, capacity and operation of biological neural networks, like the human brain.

Connectionist Networks are also referred to using the terms *Artificial neural network systems (ANS)*, *Connectionist Systems*, *Parallel Distributed Processing*, *Simulated Neural Nets and Adaptive Systems*. Although a pedant could successfully argue that connectionist systems can be distinguished from ANS because they are not necessarily biologically inspired, for the purpose of this thesis the terms described above will be taken to be synonymous.

The remaining sections in this thesis are confined to the description of the design, development and application of three connectionist models. These are: Hopfield's discrete network model (Hopfield [1982]), the Continuous Hopfield network model (Hopfield [1984]) and the Boltzmann machine - originally developed by Ackley, Hinton and Sejnowski [1985]. Hopfield's discrete model is described because it provided the precursor for the development of the Continuous Hopfield network model, which was used by Hopfield and Tank [1985] to demonstrate that connectionist models could solve combinatorial optimization problems, and in particular, compute very good solutions to examples of the Travelling Salesman Problem. The Boltzmann machine model is described, because a novel, modification to this model, referred to as the 'Pocket' Boltzmann machine, is used in CONNEKT to compute the incremental cost of assigning customers to vehicles and solve the VRP component of the Dynamic Customer Assignment and Vehicle Routing Problem. It is able to do this as it can find very good or even globally optimal solutions to examples of the TSP, whilst simultaneously satisfying a number of additional constraints.

3.2 HOPFIELD NETWORKS

3.2.1 Introduction

In 1982 John Hopfield, a Professor of Biology and Chemistry at the California Institute of Technology and a consultant to AT&T's Bell Research Laboratories at Murray Hill, New Jersey, published a seminal paper which described a class of interconnected, auto-associative neural networks that had a number of emergent features, including [Hopfield,1982]:

"the capacity for generalisation, familiarity recognition, categorisation, error correction and time sequence retention."

Hopfield's original model is an example of a class of neural networks called *Discrete Auto-correlators*, *Hopfield Associative Memories* or *Hopfield Nets*. Hopfield later extended his discrete model [Hopfield,1984] and introduced, along with Cohen and Grossberg [1983], a class of networks which are now referred to as *Continuous Hopfield (CH) networks*.

Hopfield's paper [Hopfield, 1982], was influential because it introduced what was subsequently seen as a fundamental method of analysis based on the definition of a cost function, which he called the *energy* of the network ' E '. Hopfield was able to demonstrate: that provided the units in the network are symmetrically connected; and that only one unit at a time updates its firing (output) state; that each update reduces, or at worst does not increase the energy of the network. In other words, Hopfield was able to show that this class of networks operates by minimising an energy quantity, which causes them to settle into stable patterns of operation, that is, into stable states.

Hopfield also drew particular attention to two of the emergent properties of his auto-associative networks of interconnected, simple, non-linear devices. Firstly, that such systems have stable states which will always be entered, if the system is started in a

similar state, and secondly, that these states can be created by changing the strengths of the connections between the simple non-linear devices.

Hopfield's original discrete model, and his concept of energy are described in the section that immediately follows. Subsequent sections describe extensions to his earlier paper and examine how a continuous Hopfield network can quickly, find very good solutions to examples of difficult combinatorial optimization problems, like the travelling salesman problem. The principal strengths and weaknesses and the general applicability of this class of neural networks are examined; and finally a number of conclusions are drawn.

3.2.2 The Discrete Hopfield Network Model

Discrete Hopfield networks are single-layer, symmetric, non-linear, auto-associative, nearest-neighbour pattern encoders that use Hebbian learning to store binary spatial patterns of the form:

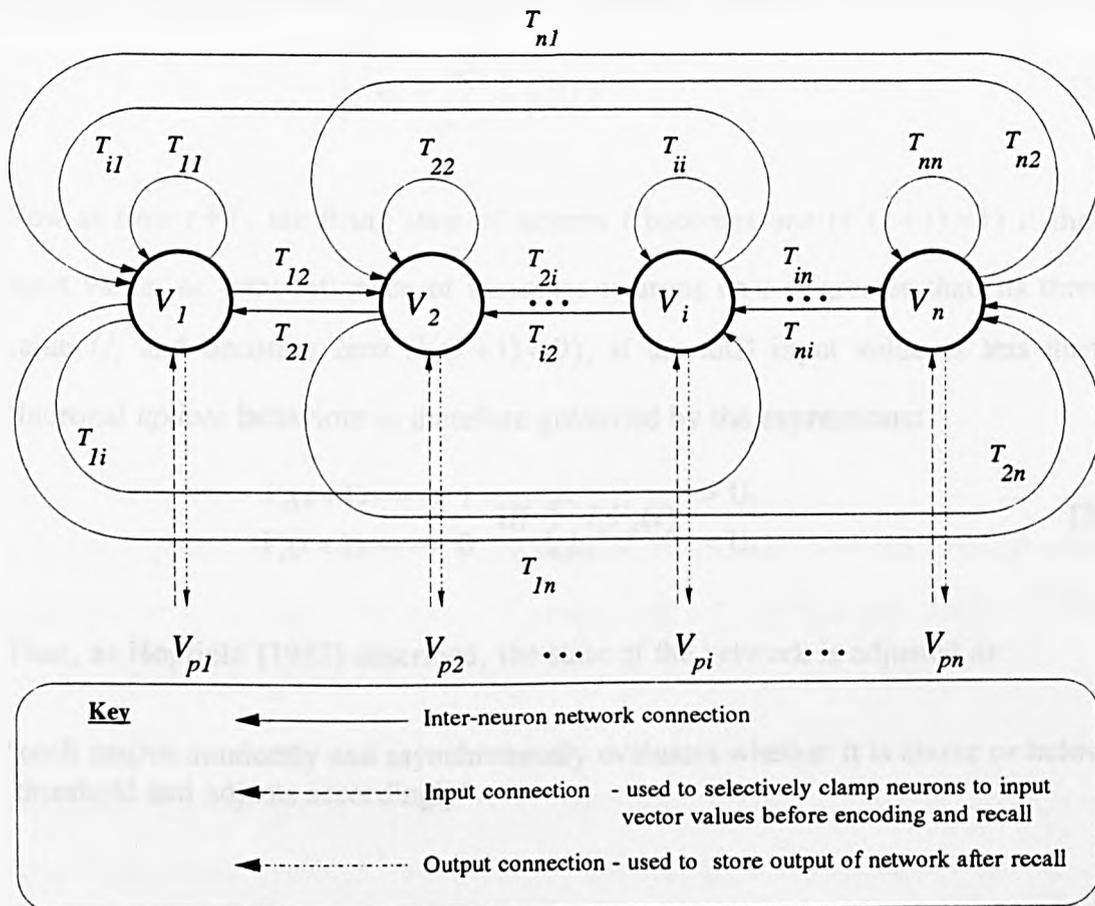
$$\vec{V}_p = (V_{p1}, V_{p2}, \dots, V_{pi}, \dots, V_{pn}) \quad \text{where } \vec{V}_p \text{ is the } p\text{th input vector pattern } p=1, 2, \dots, q, \\ \text{and } V_{pi} \in \{0, +1\} \text{ is the } i\text{th component of the } p\text{th} \\ \text{pattern}$$

As stated previously, Hopfield's discrete model is an example of a class of neural networks known as *discrete auto-correlators*. These were introduced as a theoretical idea by McCulloch and Pitts [1943] and were first rigorously studied by Amari [1972a, 1972b, 1977a, 1977b]. Other researchers, including Hopfield [1982], Little [1974] and Little and Shaw [1978] have studied the behaviour and dynamics of this class of neural networks .

The discrete Hopfield network model is comprised of non-linear processing units, called neurons, which like those of McCulloch and Pitts [1943] have two discrete

firing (or output) states. At any given time, the state (output) $V_i \in \{0, +1\}$ of the i th neuron is either, *firing at maximum rate* ($V_i = 1$) or *not firing* ($V_i = 0$). The network consists of a single layer of *fully-interconnected* neurons, that is, the output of each neuron is connected to the input of every other neuron via a weighted connection T_{ij} . All of the connections are *two-way*, allowing each neuron i to influence every other neuron j and *symmetric* ($T_{ij} = T_{ji}$). The topology of the discrete Hopfield model is described in figure 3.1. Hopfield's design precluded direct feedback connections from a neuron's output to its own input, that is, for all $i = 1, 2, \dots, n$, $T_{ii} = 0$, subsequent research, including that by McEliece et al, [1987] and Gindi, Gmitro and Parthasarathay [1988] have since shown that this restriction can be relaxed.

Figure 3.1 : The topology of a discrete Hopfield Network



Discrete Hopfield networks learn off-line and operate in discrete time. The neuronal

firing states are updated asynchronously, according to the following procedure:

At random times, but with a mean attempt rate W , each neuron i in a network of n neurons uses only local information to determine whether or not to adjust its state. It does this by comparing the sum of the inputs it receives from the other $n-1$ neurons in the network, with its threshold value, U_i . In Hopfield's original analysis [Hopfield,1982] the threshold value U_i was considered to be zero, although in a later extension [Hopfield,1984] non zero threshold values were allowed.

The input that the i th neuron receives from the j th neuron (the influence j has on i), at time t , is given by the product of the connection weight T_{ij} and the j th neuron's firing state, $V_j(t)$. The total influence on i , or total input value, u_i , is then the sum of the input values from each of the other $n-1$ neurons, that is:

$$u_i = \sum_{j=1, j \neq i}^n T_{ij} V_j(t) \quad [3.2.1]$$

Now at time $t+1$, the firing state of neuron i becomes one ($V_i(t+1)=1$) if the total input value, or total influence of the other neurons on i is greater than its threshold value U_i and becomes zero ($V_i(t+1)=0$), if the total input value is less than U_i .

Neuronal *update* behaviour is therefore governed by the expressions:

$$\begin{aligned} V_i(t+1) &\longrightarrow 1 && \text{iff } \sum_{j=1, j \neq i}^n T_{ij} V_j(t) > U_i \\ V_i(t+1) &\longrightarrow 0 && < U_i \end{aligned} \quad [3.2.2]$$

Thus, as Hopfield [1982] described, the state of the network is adjusted as:

"each neuron randomly and asynchronously evaluates whether it is above or below threshold and adjusts accordingly".

The instantaneous state of a network of n neurons can be represented by the vector:

$\vec{V} = (V_1, V_2, \dots, V_i, \dots, V_n)$. In Hopfield's original model $V_i \in \{0, +1\}$, although other

discrete auto-correlator networks have been developed with $V_i \in \{-1, +1\}$.

Hopfield associated each network state with a quantity which he called E (energy), and, as is described below, he was able to show that each time a neuron changed its state, E was reduced until it reached some locally minimum value. This ensured that the network would always settle into some stable state, that is, a state in which the firing state of each of the neurons in the network will remain unchanged.

3.2.3 Energy Analysis in the Discrete Model

In Hopfield's discrete model, at any one time, only one neuron is allowed to determine whether or not it should change its state. If the activation of the i th neuron, a_i , is given by the expression:

$$a_i = \sum_{j=1, j \neq i}^n T_{ij} V_j(t) - U_i \quad [3.2.3]$$

Then from examination of the neuronal *firing* or *update* rule, described by equation 3.2.2, the change in the value of the firing state of i , ΔV_i , is given by:

$$\begin{aligned} \Delta V_i &= 0 \text{ or } +1 & \text{if } a_i > 0 \\ \Delta V_i &= 0 \text{ or } -1 & \text{if } a_i < 0 \\ \Delta V_i &= 0 & \text{if } a_i = 0 \end{aligned} \quad [3.2.4]$$

The value of ΔV_i is always zero if the state of the i th unit does not change and non-zero, otherwise. In the latter case, the product of ΔV_i and the activation a_i is always positive, since ΔV_i and a_i are always either both positive or both negative.

Therefore, the change in energy, ΔE_i , that results from the change in firing state, ΔV_i , of any single neuron i , will always be reducing if ΔE_i is defined as:

$$\Delta E_i = -(\Delta V_i a_i) \quad [3.2.5]$$

that is:

$$\Delta E_i = -\Delta V_i \left(\sum_{j=1, j \neq i}^n T_{ij} V_j - U_i \right) \quad [3.2.6]$$

Hence the energy, E_i , of the i th neuron whose change of state led to the change in the energy of the network, by an amount, ΔE_i , is defined as:

$$E_i = -V_i \left(\sum_{j=1, j \neq i}^n T_{ij} V_j - U_i \right) = - \sum_{j=1, j \neq i}^n T_{ij} V_i V_j + V_i U_i \quad [3.2.7]$$

The total energy of the network can now be defined as the sum of the energy E_i for each of the neurons $i=1,2,\dots,n$, in the network. However, as the weights are symmetric ($T_{ij} = T_{ji}$), summing E_i over all n neurons leads to some of the summation terms appearing twice. Hence, Hopfield [1982] defined the global (or total) energy of the network E as:

$$E = -\frac{1}{2} \sum_{i=1, i \neq j}^n \sum_{j=1, j \neq i}^n T_{ij} V_i V_j + \sum_{i=1, i \neq j}^n V_i U_i \quad [3.2.8]$$

Now recalling that the connections weights are symmetrical. If only a single unit at a time, the i th unit, has the opportunity to update its state, the difference between the global energy of the network when that unit is not firing, $V_i = 0$, and when it is firing $V_i = 1$ can be determined locally by the i th unit, following equation 3.2.6, and is described by the expression:

$$\Delta E_i = \sum_{j=1, j \neq i}^n T_{ij} V_j - U_i \quad [3.2.9]$$

Hence, a unit will minimize its own contribution to the global energy, if it follows a firing rule which states that the unit should become firing ($V_i = 1$) if the total input from the other units exceeds its threshold, and become not firing ($V_i = 0$), if the threshold exceeds the total external input. This is precisely the firing or update rule that governs the behaviour of units in the network and is described by equation 3.2.2.

From Hopfield's analysis, it is therefore clear, that a network based on his discrete model (with symmetric connections and a single unit at a time updating its state using the firing rule described by equation 3.2.2), must reduce the value of E whenever a neuron changes its state. This process can continue only until the network reaches a stable state that corresponds to some local minimum of the energy function E .

3.2.4 Encoding in the Discrete Model

If the network is to resolve a user-defined problem, some mechanism must be found that enables the creation of stable states that correspond to solutions to the specified problem. In a Hopfield network, stable states, also referred to as *energy wells*, can be created either by calculating and assigning connection weight values, or by allowing the net to use some learning algorithm to adjust its own connection weights in response to the presentation of a set of example input vectors, a process called *training by example*.

The creation of energy wells by calculation, to solve an example of the travelling salesman problem, is described in section 3.2.11. Using training by example to encode binary spatial vector patterns p , of the form:

$$\vec{V}_p = (V_{p1}, V_{p2}, \dots, V_{pn}) \text{ where } V_{pi} \in \{0, +1\} \text{ or } \{-1, +1\}$$

is a relatively straightforward process and is described below:

Suppose that a network is to store q binary spatial vector patterns \vec{V}_p , each of length n bits. The q binary spatial vectors can be represented by the neuronal firing states of a Hopfield network of n neurons, where each pattern is represented by a particular stable state, provided that the number of patterns to be encoded, does not exceed the capacity of the network (described in section 3.2.9).

An $n \times n$ weight matrix \mathbf{W} can be defined and the values of each of the weights T_{ij} be

assigned to the elements, w_{ij} of \mathbf{W} . For each of the patterns, the weight matrix is updated, using the equation:

$$\mathbf{W} = \sum_{p=1}^q \vec{V}_p^T \vec{V}_p \quad [3.2.10]$$

or in point-wise notation:

$$w_{ij} = \sum_{p=1}^q V_{pi} V_{pj} \quad [3.2.11]$$

For each of the patterns, the procedure applies Hebbian learning [Hebb, 1949] by 'rewarding' the connection weights of each pair of neurons (representing a pair of vector elements) that are in the same state, by increasing the value of the weight by the outer product of the two elements. The incremental amount is always unity if $V_{pi} \in \{0,1\}$ or $\{-1,+1\}$

A straightforward example of the application of this encoding procedure is described, for example, by Simpson [1990]. Other encoding procedures have been implemented including those that apply the Widrow-Hoff (or Delta) rule [Widrow,1962], a simple example of which is given by Aleksander and Morton [1990].

3.2.5 Recall in the Discrete Model

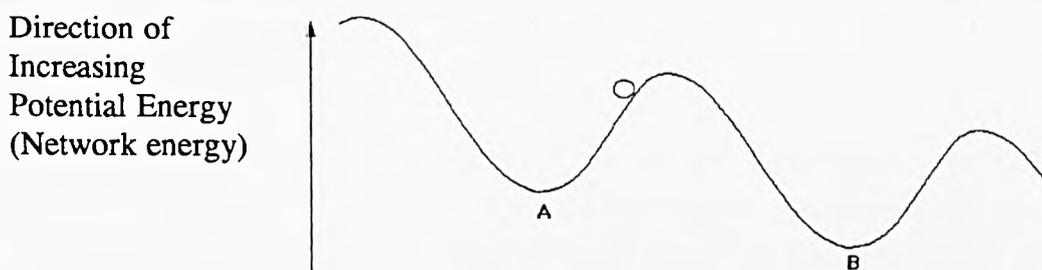
In the discrete model, the network recalls a particular memory or solves a particular problem by relaxing into a stable state that corresponds to the memory or to a solution of the problem. Depending on the application, the network may either be started with the firing states of its neurons set randomly, or with some or all of them clamped to the values of an input vector pattern. A complete binary spatial pattern (or memory) can be recalled by applying a partial or noisy copy of the pattern as an input to the network.

The input vector $\vec{V}_i = (V_{i1}, V_{i2}, \dots, V_{in})$ where $\vec{V}_i \in \{\vec{V}_p\}$ and $p = 1, 2, \dots, q$, is applied by clamping the firing states of each of the n neurons to the values of the corresponding elements of the vector \vec{V}_i . A neuron i is then selected at random, and is allowed to determine its new state according to the expression 3.2.2.

Neurons are repeatedly selected at random, until the firing states of all of the neurons in the network cease to change. At which point the network has relaxed into a stable state.

An analogy is often drawn between the operation of the network and the behaviour of a ball-bearing on a bumpy surface (see figure 3.2), where the bumpy surface is the analogue of the network's energy distribution. The analogy requires and assumes that ball-bearing has negligible inertia.

Figure 3.2 :Ball-bearing on a bumpy surface (analogue of a simple network energy E landscape with two minima separated by an energy barrier)



Troughs in the bumpy surface are analogues of stable states or energy wells. The initial input state, set by clamping certain of the neurons to the binary value of some input vector is analogous to positioning the ball-bearing at a particular point on the surface. Once released the ball-bearing, assuming negligible inertia, will descend to the bottom of the nearest minima and remain in that position. In the same way, the network, by applying the deterministic decision rule described by equation 3.2.2, performs gradient descent until it reaches a stable state at the bottom of the nearest energy well. Different starting positions can cause the ball-bearing (network) to come

to rest in either of the of the minima (energy wells), A or B, despite the fact that B is the deepest.

If the recall has been successful, the output vector $\vec{V}_o = (V_{o1}, V_{o2}, \dots, V_{on})$ constructed from the n neuronal firing states once the network has settled into a stable state, will correspond exactly to the training vector \vec{V}_p that was closest to the input vector \vec{V}_i .

In practice, the pattern of weights created by the encoding procedure or by calculation may inadvertently create *false energy wells*. These are stable states that do not correspond to any of the patterns in the training process or hence, to solutions to the problem under investigation.

3.2.6 The Continuous Hopfield Network Model

In 1984, Hopfield published a paper which described [Hopfield 1984]:

"a model for a large network of "neurons" with a graded response (or sigmoid-input output relation)"

The *Continuous Hopfield (CH)* network model was developed by extending the original discrete, stochastic model, and retains the content addressable memory and other emergent features of the earlier model [Hopfield,1984]. The discrete Hopfield model describes networks that operate in discrete time and are restricted to binary/bipolar inputs and responses. The continuous model is able to handle analogue data in continuous time. The extension has enabled the continuous model to be applied to a number of new application areas, including modelling and solving optimization problems.

The continuous network model is a single-layer, auto-associative, nearest-neighbour encoder, which operates in continuous time to store arbitrary analogue patterns [Simpson,1990], of the form:

$$\vec{V}_p = (V_{p1}, V_{p2}, \dots, V_{pi}, \dots, V_{pn}) \quad \text{where } \vec{V}_p \text{ is the } p\text{th input pattern } p=1,2,\dots,q,$$

$$V_i^0 \leq V_{pi} \leq V_i^1 \text{ is the } i\text{th component of the input,}$$

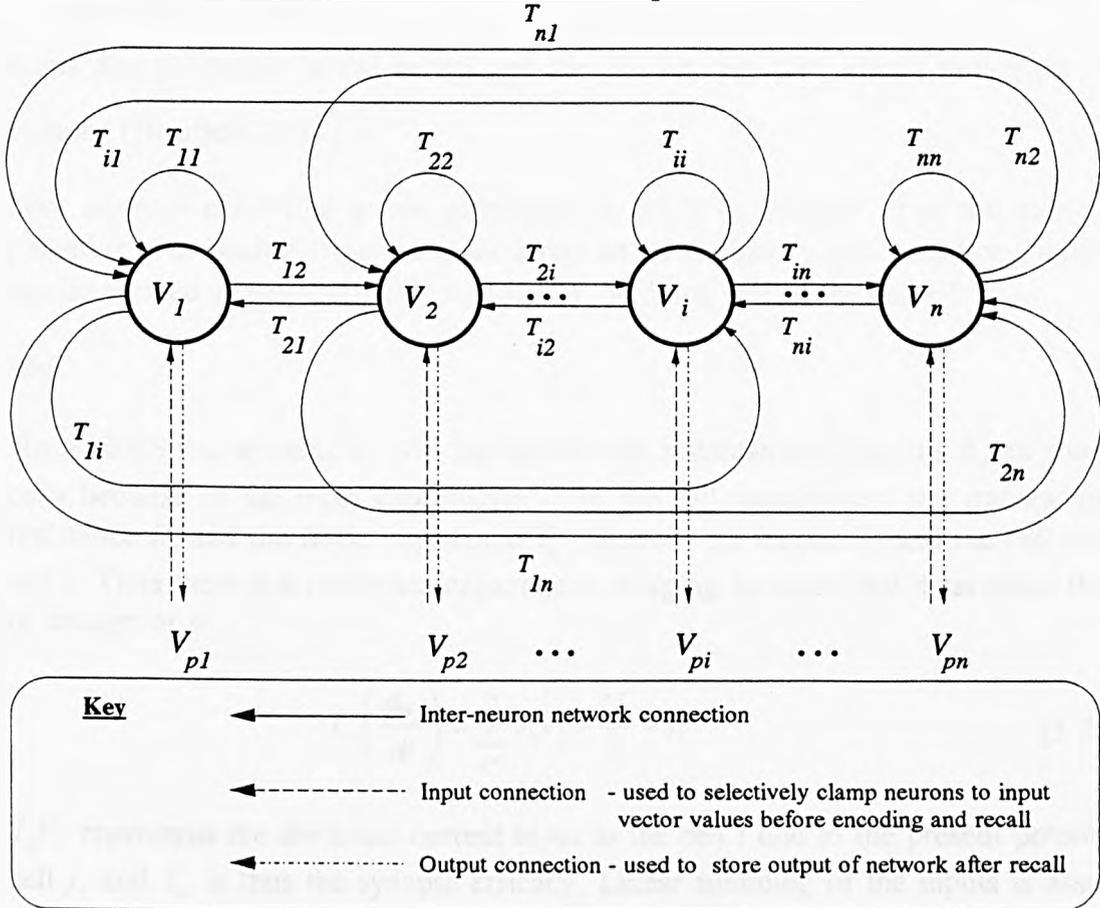
$$\text{where } V_i^0 < V_i^1 \text{ and } V_i^0, V_i^1 \text{ are constants.}$$

The topology of the continuous Hopfield model, described in figure 3.3, is the same single-layer feedback architecture as found in the discrete model.

In the continuous model [Hopfield,1984], the original discrete, two-state $V_i \in \{0,1\}$ McCulloch and Pitts neurons are replaced by neurons that can adopt any output value in the continuous range $V_i^0 \leq V_i \leq V_i^1$. Direct feedback connections are allowed ($T_{ii} \in \mathfrak{R}$) and the model incorporates extensions which allow neurons to have non-

zero threshold values, U_i ; and to receive a non-zero external input, I_i , in addition to the inputs from other neurons.

Figure 3.3 : The topology of a continuous Hopfield Network



Hopfield defined that the output value of the i th neuron, V_i , be determined by a (typically sigmoid), continuous and monotone-increasing function g of the instantaneous input u_i , that is:

$$V_i = g(u_i) \text{ which has asymptotes } V_i^0 \text{ and } V_i^1 \quad [3.2.12]$$

In the application of the continuous model to the travelling salesman problem (described in section 3.2.11), Hopfield and Tank [1985] used the sigmoid function:

$$g(u_i) = \frac{1}{2} \tanh(u_i/u_o) \quad [3.2.13]$$

where u_o is the gain, a positive constant controlling the steepness of the sigmoid

function. The inverse output-input function is also defined (for convenience) as:

$$u_i = g^{-1}(V_i) \quad [3.2.14]$$

Hopfield introduces and describes the continuous model by drawing analogies between terms and properties in the model and the characteristics of simple biological neural systems [Hopfield,1984]:

"For neurons exhibiting action potentials, u_i could be thought of as the mean soma potential of a neuron from the total effect of its excitatory and inhibitory inputs. V_i can be viewed as the short-term average of the firing rate of the cell i ."

and:

"In a biological system, u_i will lag behind the instantaneous outputs V_j of the other cells because of the input capacitance C of the cell membranes, the transmembrane resistance R , and the finite impedance T_{ij}^{-1} between the output V_j and the cell body of cell i . Thus there is a resistance-capacitance charging equation that determines the rate of change of u_i .

$$C_i \left(\frac{du_i}{dt} \right) = \sum_{j=1}^n T_{ij} V_j - \frac{U_i}{R_i} + I_i \quad [3.2.15]$$

$T_{ij} V_j$ represents the electrical current input to the cell i due to the present potential of cell j , and T_{ij} is thus the synapse efficacy. Linear summing of the inputs is assumed. T_{ij} of both signs should occur. I_i is any other (fixed) input current to neuron i ."

Hopfield's equation relates the change in potential across the body of the neuron with capacitance, C_i , to the total input current received by the neuron; using a formula describing the charge/discharge of a capacitor. If at a certain time t during the discharge of a capacitor of capacitance C through a resistance R , the potential difference (p.d.) across the capacitor is V and the charge on it is Q , then [Duncan,1987]:

$$Q = VC \quad [3.2.16]$$

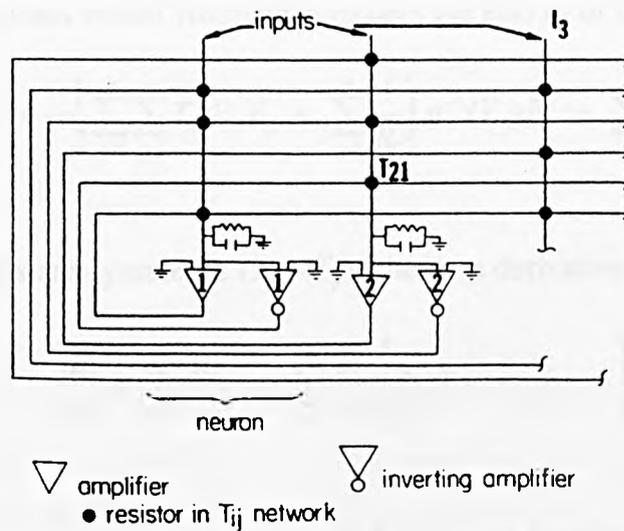
Now, the rate of change of potential is related to the rate of flow of charge, thus:

$$C \frac{dV}{dt} = \frac{dQ}{dt} \quad [3.2.17]$$

Hence, the right-hand side of Hopfield's equation (3.2.15) describes the current (or rate of flow of charge dQ/dt) received by the neuron as a result of the external input I_i and the output behaviour of the other neurons in the network. The left-hand side of equation 3.2.15 is simply the product of the rate of change of the input voltage u_i (potential difference) and the neuronal capacitance, C_i .

Hopfield [1984] implemented the continuous model, using an electrical circuit comprised of amplifiers and resistors. The diagram describing the implementation is reproduced in figure 3.4.

Figure 3.4 : An Electronic implementation of the continuous Hopfield model



In Hopfield's electronic implementation, excitatory signals are generated by the amplifiers whilst inverting amplifiers are used to provide the inhibitory signals. C_i and ρ_i , are respectively, the total input capacitance and the input resistance of

amplifier i . The magnitude of the connection weight $T_{ij} = 1/R_{ij}$, whilst the sign of T_{ij} is determined by the choice of position at the connection site of the positive or negative output of amplifier j (Hopfield [1984]).

The total input resistance for neuron i , R_i , is given by the expression:

$$\frac{1}{R_i} = \frac{1}{\rho_i} + \sum_{j=1}^n \frac{1}{R_{ij}} \quad [3.2.18]$$

Hopfield [1984] was able to confirm that the dynamic behaviour of his electronic implementation of the model, could indeed be described by equation 3.2.15. Hopfield also extended his energy analysis to demonstrate that networks based on his continuous model must also, eventually relax into a stable state. His analysis is described in the following section.

3.2.7 Energy Analysis in the Continuous Model

In the continuous model Hopfield considers the *energy* of the network E , where

$$E = -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n T_{ij} V_i V_j + \sum_{i=1}^n \frac{1}{R_i} \int_0^{V_i} g^{-1}(V) dV + \sum_{i=1}^n I_i V_i \quad [3.2.19]$$

If the weights are symmetric ($T_{ij} = T_{ji}$), the time derivative is:

$$\frac{dE}{dt} = \frac{\delta E}{\delta V_i} \frac{dV_i}{dt} = -\sum_{i=1}^n \frac{\delta V_i}{dt} \left[\sum_{j=1}^n T_{ij} V_j - \frac{u_i}{R_i} + I_i \right] \quad [3.2.20]$$

The term in parentheses in equation 3.7.20 is the same as the term on the right-hand side of equation 3.2.15, so:

$$\frac{dE}{dt} = -\sum_{i=1}^n C_i \frac{dV_i}{dt} \frac{du_i}{dt} \quad [3.2.21]$$

and substituting for u_i using 3.2.14:

$$\frac{dE}{dt} = -\sum_{i=1}^n C_i g^{-1}(V_i) \left(\frac{dV_i}{dt} \right)^2 \quad [3.2.22]$$

Now, C_i is a positive constant and g^{-1} is a monotone increasing function with a positive gradient, therefore, the right-hand side of equation 3.4.19 is always negative or zero, hence :

$$\frac{dE}{dt} \leq 0, \quad \frac{dE}{dt} = 0 \rightarrow \frac{dV_i}{dt} = 0 \text{ for all } i. \quad [3.2.23]$$

As Hopfield [1984] states (Eq. 10 corresponds to equation 3.2.23):

"Together with the boundedness of E, Eq. 10 shows that the time evolution of the system is a motion in state space that seeks out minima in E and comes to a stop at such points. E is a Lyapunov function for the system. This deterministic model has the same flow properties in its continuous space that the stochastic model does in its discrete space."

A number of other theorems including the Cohen-Grossberg [1983] stability theorem, can be used to prove the global stability of the Continuous Hopfield model, as demonstrated for example by Simpson [1990].

3.2.8 Encoding and Recall in the Continuous Model

In the continuous model the connection strength (conductance value) between the i th and j th neurons is the inverse of some resistance R_{ij} :

$$T_{ij} = \frac{\alpha}{R_{ij}} \quad [3.2.24]$$

where α is simply the sign of the connection value being emitted from the j th neuron.

The values of R_{ij} and hence T_{ij} can be determined by calculation or established using

a number of alternative encoding procedures including signal Hebbian learning or competitive learning (Simpson, [1990]).

The continuous Hopfield model performs recall by allowing each of the neurons in the network, in parallel, to simultaneously and continuously attempt to satisfy equation 3.2.25. In so doing, the system progressively reduces the energy of the network. This process continues until the state of the network corresponds to that at the bottom of some energy well. At which point the network has relaxed into a stable state.

$$C_i \left(\frac{da_i}{dt} \right) = \sum_{j=1}^n g(a_j) T_{ij} - \frac{a_i}{R_i} + I_i \quad [3.2.25]$$

where a_i is the activation of neuron i such that:

$$a_i = \sum_{j=1}^n T_{ij} V_j \quad [3.2.26]$$

3.2.9 Stability and Capacity

Through his energy analyses, Hopfield demonstrated that both the discrete and continuous network models will always relax into a stable state, that is, that the system is stable for all inputs. As stated previously, the Cohen-Grossberg [1983] stability theorem can also be used to prove that (auto-associators like) Hopfield's continuous model are globally stable.

The capacity of discrete and continuous auto-associator models is quite limited. Hopfield [1982] found that in a network of N neurons:

"About $0.15N$ states can be simultaneously remembered before error in recall is severe"

Amari and Maginu[1988] used Amari's [1974] method of statistical neurodynamics to show that the probability of perfectly recalling all stored patterns is unity provided

that the number of patterns stored does not exceed m , where:

$$m = \frac{N}{2 \log N + \log \log N} \quad [3.2.27]$$

3.2.10 Strengths and Limitations

Hopfield's discrete and continuous models are able to reconstruct entire patterns using only partial, incomplete or noisy input; are capable of error correction; and demonstrate fault tolerance. Both models have been applied in a number of application areas, including image, speech and signal processing, pattern classification and recognition. The continuous model has also been successfully applied to a number of types of combinatorial optimization problems.

As described in the previous section, both the discrete and continuous models suffer from the limitation of having only a quite limited storage capacity. They also suffer from the same hard learning problem as the perceptron [Rosenblatt, 1962]. This is because Hopfield's network models have no hidden units that can be used to develop internal representations of the problem.

The discrete model has the additional limitation that it can only store and recall binary pattern vectors, although unlike the continuous model which can accept analogue inputs, it can be trained on-line. The continuous model also lacks a straight-forward encoding scheme.

Although Hopfield networks can relatively quickly compute solutions to examples of combinatorial optimization problems, particularly when compared to models like the Boltzmann machine, there are a number of problems associated with the approach. Firstly Hopfield and Tank's model has been shown to be inherently unstable, although a number of researchers have suggested improvements to the model. Some researchers have been surprised at the difficulty they encountered in developing software

simulations of Hopfield's continuous model, that actually found solutions to the problem it was intended to model. This is partly because the performance of the model can be significantly affected by small changes to one or more of the parameter values and also because in some simulations the smallest discrete change does not provide a fine enough simulation of a very small continuous change, and so causes the network to jump from the influence of one minima across the solution space to a random point on another minima.

However, the most serious limitation of Hopfield networks and similar auto-correlators, when applied to combinatorial optimization problems is that their initial state deterministically determines their final state. This is because they perform deterministic gradient descent in the solution space, and once the network state is in the influence of a minima, even if it is only a poor local minimum, the system is unable to escape from the influence of the local minima into deeper (and hence better) minima, even if these are immediately adjacent in the solution space.

3.2.11 Solving the TSP using Hopfield and Tank's method

Hopfield and Tank [1985] used Hopfield's continuous network model - whose topology and connectivity is described in section 3.2.6 and figure 3.3 - to model and solve examples of the travelling salesman problem.

They modelled and solved the problem by first developing a representation schema and then designing and applying a specific energy function whose terms would only be minimized if the network found good solutions to the problem. Energy wells corresponding to solutions of the problem were created by calculating the values of each of the connection weights.

Hopfield and Tank [1985] solved examples of the TSP with 10,15 and 30 cities, and described their representation using an example with the number of cities $n=5$:

"For n cities a total of n independent *sets* of n neurons are needed to represent a complete tour. This is a total of $N = n^2$ neurons. The output state of these n^2 neurons which we will use in the TSP computational network is most conveniently displayed as an $n \times n$ square array. Thus, for a 5-city problem using a total of 25 neurons, the neuronal state

	1	2	3	4	5	
A	0	1	0	0	0	(7)
B	0	0	0	1	0	
C	1	0	0	0	0	
D	0	0	0	0	1	
E	0	0	1	0	0	

shown above would represent a tour in which city C is the first city to be visited, A the second, E the third etc. [The total length of the 5-city path is $d_{CA} + d_{AE} + d_{EB} + d_{BD} + d_{DC}$.] Each such final state of the array of outputs describes a particular tour of the cities. Any city cannot be in more than one position in a valid tour (solution) and also there can be only one city at any position. In the $n \times n$ "square" representation this means that in an output state describing a valid tour there can be only one "1" output in each row and each column, all other entries being zero. Likewise, any such array of output values, called a permutation matrix, can be decoded to obtain a tour (solution)."

The firing state of a neuron in the X th row and the i th column is then V_{Xi} where each of the rows X represent a particular city $\{A,B,C,\dots\}$ and each of the columns i corresponds to a particular position in the tour $\{1,2,3,\dots,n\}$. For convenience, the row and column indices are of modulo n so for example $V_{X,n+1} = V_{X,1}$. This significantly simplifies the notation used to define the terms of Hopfield and Tank's energy function, (equation 3.2.29), since the neighbours of the first column (row) are the second and the n th, and similarly the neighbours of the n th column (row) are the $n-1$ th and the first column (row).

The firing states V_{Xi} of each of the neurons in the network are mapped onto the

corresponding elements in the permutation matrix d_{xi} according to the expressions:

$$d_{xi} = \begin{cases} 1 & \text{if } V_{xi} > \gamma \\ 0 & \text{if } V_{xi} < \sigma \end{cases} \quad \text{where } 0 \leq \sigma < \gamma \leq 1 \quad [3.2.28]$$

where: σ is close to but > 0 and γ is close to but < 1 .

Equation 3.2.28 states that if the firing state of the neuron is sufficiently close to one (zero) its value is assumed to be one (zero), and can be interpreted, once the network has reached a steady state, as meaning that the network has relaxed into a state in which the X th city represented is (not) visited at the i th step in the tour. This assumption is necessary because the value of the firing state of a neuron in the continuous model, has a continuous value in the range $[0,1]$ and is determined using a sigmoid function as in equation 3.2.13. This means that its value is only one (zero) if the total input to the neuron is positive (negative) and infinite.

The network connection weights must be calculated so as to ensure that the energy wells in the network correspond to good solutions to the particular example of the TSP under investigation. To enable the $N = n^2$ neurons in the network to compute a solution to the problem, the network must be described by an energy function in which the lowest energy state (the most stable state of the network) corresponds to the best path. This can be separated into two requirements. First the energy function must favour strongly stable states of the form of a permutation matrix, and secondly, of the $n!$ solutions corresponding to valid tours, the network must favour solutions that have short path lengths [Hopfield and Tank, 1985]

Hopfield and Tank [1985] designed an energy function which had four terms (see equation 3.2.29). The first three terms encourage the network to favour stable states that are valid tours, since they have minimal values, that is, they tend to zero, if only one neuron in each row and each column has a value which tends to one and the remainder have values which are close to zero. The first A term returns the sum of

the product of the firing states of each neuron and each of the other neurons in the same row, for each of the n rows. The term is zero if and only if (iff) there is a maximum of one 1 in each row and all other elements are zero. Similarly, the second (B) term is zero iff there is at most one 1 in each column and all other elements are zero. The third term is zero only if a total of exactly n elements have the value 1. The fourth term encourages the network to favour short paths, since its value is equal to the total path length - given that the network's response represents a valid tour.

$$E = \frac{A}{2} \sum_X \sum_i \sum_{j \neq i} V_{Xi} V_{Xj} + \frac{B}{2} \sum_i \sum_X \sum_{X \neq Y} V_{Xi} V_{Yi} + \frac{C}{2} \left(\sum_X \sum_i V_{Xi} - n \right)^2 \quad [3.2.29]$$

$$+ \frac{D}{2} \sum_X \sum_{Y \neq X} \sum_i d_{XY} V_{Xi} (V_{Y,i+1} + V_{Y,i-1})$$

The energy expression is mapped onto the network by setting the connection weights in accordance with the weight formula:

$$T_{Xi,Yj} = -A \delta_{XY} (1 - \delta_{ij}) - B \delta_{ij} (1 - \delta_{XY}) - C - D d_{XY} (\delta_{j,i+1} + \delta_{j,i-1}) \quad [3.2.30]$$

where $\delta_{ST} = 1$ if $S = T$ and is 0 otherwise and S, T are binary valued scalars.

The first (A) and second (B) terms respectively create inhibitory connections within each row and each column. The third (C) term provides global inhibition and the fourth (D) term, is a data term whose contribution applies an inhibitory influence on the symmetric weight between each pair of neurons in proportion to the distance d_{XY} between the cities X, Y in the example of the TSP under investigation.

A convenient way of storing the calculated connection weights is in a four dimensional array, $T[x][i][y][j]$. This can be visualised as an $n \times n$ matrix with a row corresponding to each of the n cities X , and a column representing the position in the

tour i . Each element of this matrix is itself an $n \times n$ matrix, with each row representing a possible successor city Y on the tour, and j the next tour position.

Once the connection weights have been calculated and set to appropriate values, a solution to the problem may be found by setting the network to an initial random state with a slight excitation bias introduced to prevent perfect symmetry. In Hopfield and Tank's [1985] implementation, the excitation bias applied to each of the N (operational amplifier) neurons was the external input current:

$$I_{xi} = +Cn \quad [3.2.31]$$

The excitation bias ensures that the initial state of the network is asymmetric and prevents the network from being unable to favour relaxing into any particular state - a situation analogous to a pencil standing perfectly upright and so not falling in any direction.

After the weights have been set and the excitation bias applied the network can be allowed to run until it settles into a stable state. Determining whether the network has found a solution to the TSP is straightforward and can be implemented semi-automatically. The expression 3.2.28 is used to map the neuronal firing states at equilibrium to the elements of the permutation matrix. If the matrix has the same form as that described in (7) (on page 39) then the network has found a valid solution to the problem. If the permutation matrix does not contain exactly a single one in each row and column, the user can examine the neuronal firing states and determine whether the network has found a valid solution if the values of γ and σ are relaxed.

3.2.12 Performance of Hopfield and Tank's Method

Hopfield and Tank [1985] found that their implementation of Hopfield's continuous model [Hopfield, 1984] successfully solved an example of the TSP with $n=10$ in 16

out of 20 trials. They investigated the quality of the solutions and found that despite the large number of possible distinct paths ($10!/20 = 181,440$) [Hopfield and Tank, 1985]:

"About 50% of the trials produced one of the two shortest paths. Hence the network did an excellent job of selecting a good path, preferring paths in the best 10^{-5} of all paths compared to random paths."

However, other researchers, including Wilson and Pawley [1988], have attempted digital simulations of Hopfield and Tank's continuous analogue application to the TSP and found the success rate to be much lower.

Behzad and Behrooz Kamgar-Parsi [1992a] present a systematic method for selecting suitable network parameter values. They consider Hopfield and Tank's formulation of the TSP and show that all valid solutions are unstable, and they state that as a result, the formulation is flawed. In a second paper [Kamgar-Parsi and Kamgar-Parsi, 1992b], they address the questions of how effective Hopfield nets are in solving combinatorial optimization problems; what types of problem they appear to be suited to, and how well the performance of the networks scale with the size of the problem. After simulating Hopfield and Tanks solution of the TSP, they report that [Kamgar-Parsi and Kamgar-Parsi, 1992b]:

"the number of times the network succeeds in finding valid solutions are considerably less than that found by Hopfield and Tank, but when the neural net finds a solution it is of remarkably good quality".

They state that marked improvements to the success rate can be obtained by changing certain of the network parameters. They offer an explanation for the significant difference in performance that appears to exist between analogue implementations of

Hopfield's continuous model, and discrete simulations of the model [Kamgar-Parsi and Kamgar-Parsi, 1992b]:

"We have overwhelming empirical evidence, based on Monte-Carlo estimates, that deeper minima of an analog network have generally larger basins of attraction, thus a randomly selected initial state has a higher chance of falling inside the basin of a deep minimum and finding a very good solution. Hopfield neural networks are complicated dynamical systems, and as yet there are no theoretical estimates for the sizes of these basins. Therefore, one cannot give the probability of finding the best solution. However, one can claim that analog networks greatly enhance the probability of finding very good solutions. This is true for both problems -TSP and clustering - we have discussed in this paper.

The success rate appears to be more dependent on the problem. For TSP the success rate is rather modest, in particular it scales poorly as the size of the problem increases"

A number of researchers have investigated the application of Hopfield's model to the TSP and related combinatorial optimization problems. References to a number of these are provided by Simpson [1990].

3.2.13 Conclusions

Analogue implementations of Hopfield's continuous network have a much higher success rate at finding solutions to the TSP, than digital simulations of Hopfield and Tank's method. In either case, the Hopfield continuous model deterministically performs gradient descent in the weight space and so descends to the bottom of the well that is closest to the initial state of the network. Once at the bottom of a well, the network unit update rule ensures that there is no possibility of escape. Hopfield's model therefore, can only find locally optimal solutions. The solution that it finds is the one that is closest to the initial network state.

Analogue implementations of the continuous model operate very quickly, and so the network can be allowed to run many times, starting with different initial states. As a consequence, the probability of finding a very good, or even globally optimal solution is increased. Digital simulations, particularly on single processor architecture

machines run more slowly and so rerunning the network many times is correspondingly less attractive and less practicable.

A number of researchers have also commented on the inherent instability of the formulation and the consequent difficulty in determining suitable network parameter values. As stated previously, some research has indicated that the approach also scales poorly. Despite, or perhaps because of these difficulties, a number of researchers, have suggested improvements to the formulation, and techniques for determining network parameter values.

Hopfield's continuous model and the formulation developed by Hopfield and Tank has a particular historical significance. However, other network models and formulations, including that of the Boltzmann machine, introduced by Ackley, Hinton and Sejnowski [1985] and described in the sections that follow, are able to obtain globally optimal solutions to the TSP.

3.3 THE BOLTZMANN MACHINE

3.3.1 Introduction

The Boltzmann machine, introduced by Ackley, Hinton and Sejnowski [1985] is closely related to Hopfield's discrete model. They both operate in discrete time, learn off-line and are nearest-neighbour pattern matchers capable of storing and retrieving binary spatial patterns. In addition, both models are able to model and find very good solutions to examples of combinatorial optimization problems, including the TSP.

The Boltzmann machine model has been implemented with a number of different network topologies and connectivity patterns. The sections that follow, firstly describe a three-layer Boltzmann machine model that can learn to represent and store an *environment* comprised of an arbitrary set of binary, spatial vector pattern pairs. The description is based on that provided by Simpson [1990]. The three-layer Boltzmann machine network is included because it supports the description of the essential properties of the Boltzmann machine, described by Ackley, Hinton and Sejnowski [1985] and in particular their implementation of the simulated annealing algorithm, developed by Kirkpatrick, Gelatt and Vecchi [1983].

These sections in turn, support the subsequent description of Aarts and Korst's [1989a,1989b] implementation of an auto-associative Boltzmann machine network; which uses the characteristic Boltzmann machine probabilistic decision rule and a simulated annealing cooling schedule, to obtain very good or even optimal solutions to combinatorial optimization problems, including the TSP.

A review of research into the application of the Boltzmann machine to the TSP; and research concerned with the implementation of Parallel Boltzmann machine models, are then described. Finally a number of conclusions are drawn and expounded.

3.3.2 Introduction to the three layer Boltzmann Machine

A Boltzmann machine can be trained, using the *Boltzmann machine learning algorithm* to represent and store an *environment*, comprising a set of r input, output binary vector pattern pairs, of the form, (\bar{I}_p, \bar{T}_p) , where $p=1,2,\dots,r$, the input vector is $\bar{I}_p = (i_{p1}, i_{p2}, \dots, i_{pi})$ and the corresponding output vector is $\bar{T}_p = (t_{p1}, t_{p2}, \dots, t_{pn})$. The *environment* that the network is required to learn is represented by the set of probabilities:

$$\{P^+(S_1), P^+(S_2), \dots, P^+(S_p), \dots, P^+(S_r)\}$$

where, $P^+(S_p)$ is the probability of occurrence of the p th pattern or state $S_p = (\bar{I}_p, \bar{T}_p)$ in the environment. The probabilities of the occurrence of the same patterns, in the network, when the net is running freely, that is, when none of its units are clamped, is given by the set:

$$\{P^-(S_1), P^-(S_2), \dots, P^-(S_p), \dots, P^-(S_r)\}$$

The superscript '+' indicates that the value is the probability of that state, found in the environment, and hence is the *desired* probability value that the network should attempt to reproduce when it is running freely. The superscript '-' indicates that the probability value is the *actual* probability of the occurrence of the pattern in the network, when it is running freely. The objective of the *Boltzmann machine learning algorithm* is to adjust the weights in the network until the difference between the contents of the two sets is reduced to a minimum value of zero, at which point, the contents of the two sets will be the same and the network, when freely running will have learnt to recall the states (pattern pairs) with the same probability as they appear in the environment.

There are two important differences between the three-layer Boltzmann machine

described in this thesis, and auto-associator models like Hopfield's [1982]:

- Boltzmann machines are comprised of both *visible* units, that act as the interface between the network and its environment, and *hidden* units which allow the network to solve hard learning problems by forming internal representations of the mappings between the binary spatial vector pattern pairs.
- The neurons in a Boltzmann machine obey a probabilistic firing rule which gives them the possibility of changing state even if this increases the overall energy of the network.

Ackley, Hinton and Sejnowski [1985] introduced the probabilistic firing rule by making a simple modification to Hopfield's discrete update rule. During recall, their modification can prevent the network from becoming stuck in *local* minima that are not globally optimal - an inevitable consequence of applying gradient descent and only allowing downhill moves. They achieved this by designing an update rule which allows the possibility of uphill (energy increasing) transitions, where the probability of such a transition is dependent on only the energy difference between the states and a network temperature value.

As with Hopfield's discrete update rule, if a change of state by the i th neuron reduces the global energy, the state transition is accepted. However, if the change of state increases the global energy by an amount ΔE_i , and hence is an uphill move, the probabilistic firing rule states that the transition is only accepted if:

$$P_i > P_{rand} \tag{3.3.1}$$

where:

$$P_i = \frac{1}{(1 + e^{-\Delta E_i/T(t)})} \tag{3.3.2}$$

and P_{rand} is a number *randomly* selected from the Boltzmann distribution given ΔE_i ,

described by equation 3.3.2.

Hinton and his colleagues called the quantity $T(t)$ the *temperature* of the network at time t , since its behaviour is analogous to temperature in a physical system. A procedure, called the *simulated annealing algorithm*, so named because its modus operandi is analogous to the physical annealing of metals to reduce stresses, is used to manage the progressive reduction of T over time, and consequentially, the progressive reduction of the probability of energy increasing state changes.

The algorithm initially sets the temperature of the network to a high value, and allows the units in the network to update their states in accordance with the probabilistic firing rule. The high temperature makes it likely that units in the system will be able to make transitions that increase the energy of the system (uphill steps), and hence enabling the network to escape from poor (shallow) local minima. A *simulated annealing* (or *cooling*) *schedule* slowly reduces the temperature over time, making energy increasing (uphill) steps progressively less likely. Provided that the annealing or cooling is applied sufficiently slowly, the most probable final state is the one with the lowest energy, that is the global minimum.

The sections that follow, provide a description of the Boltzmann machine learning (encoding) and recall algorithms. Subsequent sections describe the network and cell characteristics of the three-layer Boltzmann machine architecture.

3.3.3 The Boltzmann machine Learning Algorithm

As with Hopfield and Tank's [1985] implementation of the Continuous Hopfield model [Hopfield, 1984]; a Boltzmann machine can only resolve a user-defined problem, if some mechanism is found that enables the creation of deep energy wells (stable states), that correspond to solutions to the specified problem.

Stable states can be created either by calculating and setting the connection weights, or through the use of a *learning algorithm*. An implementation in which stable states are created by calculation is described in section 3.4.3. The Boltzmann machine learning algorithm, described in this section, is more complex both in theory and implementation than either the back-propagation learning algorithm [Rumelhart, Hinton and Williams, 1986] or Hopfield learning algorithms.

The three-layer Boltzmann machine architecture is comprised of two layers of *visible* (input and output) binary nodes and a single layer of *hidden* binary nodes. The visible nodes may be clamped to values set by the external environment, specifically to the values of the elements of input, output vector pairs, of the form $S_p = (\bar{I}_p, \bar{T}_p)$, where $p=1,2,\dots,r$. The only contact that the network has with the external environment is via the states of its visible units, which represent the set of constraints applied by environment, that the network should learn to satisfy. The states of the hidden nodes, and the connection weights from hidden to visible nodes, are used by the network to develop its own internal representation of the environment, by satisfying the externally imposed constraints.

If a network contains a total of $v = n + l$ visible binary units, where n is the number of visible units designated as inputs and l the number designated as outputs, the total number of different states that these v visible units can adopt is simply 2^v . Now, for the network to be fully trained, so that it accurately represents the environment, it must find a set of weights, that enable it to reproduce on its visible units, when

running freely, each of the vector pattern pairs $S_p = (\bar{I}_p, \bar{T}_p)$, $p = 1, 2, \dots, r$, with the same probability of occurrence $\{P_{(S_1)}^+, P_{(S_2)}^+, \dots, P_{(S_p)}^+, \dots, P_{(S_r)}^+\}$ as occurs in the environment. That is, the network must learn to minimise and ultimately reduce to zero the difference between these probabilities and the set of probabilities of the occurrence of the same global states being generated by the network when it is running freely, $\{P_{(S_1)}^-, P_{(S_2)}^-, \dots, P_{(S_p)}^-, \dots, P_{(S_r)}^-\}$.

As Hinton and Sejnowski [1986] state:

"A particular set of weights can be said to constitute a perfect model of the structure of the environment if it leads to exactly the same probability distribution of visible vectors with the network running freely *with no units clamped by the environment*."

An important property of Boltzmann machines is that the relative probability of two global states, S_p and S_q is determined solely by their energy difference and follows a Boltzmann distribution (Hinton and Sejnowski [1986]).

$$\frac{P(S_p)}{P(S_q)} = e^{-(E_{S_p} - E_{S_q})/T(t)} \quad [3.3.3]$$

where $P(S_p)$ is the probability of the global state, S_p , and E_{S_p} is the energy of that state, similarly $P(S_q)$ is the probability of the global state, S_q , with energy E_{S_q} .

The objective of the Boltzmann machine learning algorithm, is then to control the energy levels of the global states of the network, so that these states occur, when the net runs freely, with the same probability as they occur in training.

Now, an important consequence of the property described by equation 3.3.3, is that: the probability of the network being in a particular global state, after it has reached thermal equilibrium - given that it is allowed to run freely, and that it applies the probabilistic decision rule, (described by equations 3.3.1. and 3.3.2) - depends only on the energy of that state.

The probability of the network relaxing into a particular global state is therefore controlled by determining the energy of that state, which, in turn is altered by adjusting one or more of the network connection weights. Unfortunately, each weight contributes to the energy of more than a single global state. However, Hinton and Sejnowski [1986] state that, provided the network is allowed to reach thermal equilibrium, the effect on the probability of each global state, as the value of a single weight is changed, is given by the expression:

$$\frac{\partial \ln P_{S_p}^-}{\partial w_{ij}} = \frac{1}{T(t)} \left[o_{S_p i} o_{S_p j} - \sum_q P_{S_q}^- o_{S_q i} o_{S_q j} \right] \quad [3.3.4]$$

where:

$P_{S_p}^-$ is the *desired* or *predicted* probability, at thermal equilibrium, when the network is allowed to run freely (indicated by the subscript '-'), that the states of the visible units correspond to the values of the elements of the p th global state $S_p = (\bar{I}_p, \bar{T}_p)$

$P_{S_q}^-$ is the *desired* or *predicted* probability, at thermal equilibrium, when the network is allowed to run freely, that the states of the visible units correspond to the values of the elements of the q th global state $S_q = (\bar{I}_q, \bar{T}_q)$

w_{ij} is the strength of the connection (synaptic weight) from the j th to the i th unit

$o_{S_p i}$ is the binary state (output) of the i th neuron in the p th pattern (global state)
 $S_p = (\bar{I}_p, \bar{T}_p), p = 1, 2, \dots, r$

Equation 3.3.4 shows that only local information is needed to calculate the effect of a change in the value of a single network weight on the (natural log of the) probability of a particular global state. Unfortunately, as Hinton and Sejnowski [1986] state:

"it is not normally reasonable to expect the environment or a teacher to specify the required probabilities of entire global states of the network. The task that the network must perform is defined in terms of the states of the visible units, and so the environment or teacher only has direct access to the states of these units. The difficult learning problem is to decide how to use the hidden units to help achieve the required behaviour of the visible units."

Hinton and Sejnowski [1986] describe how the *Boltzmann machine learning algorithm* can enable a network to adjust its connection weights, so that when it is running freely, the probabilities of each of the global states is the same as that in the environment. It achieves this by minimizing a quantity G , called the *information theoretic measure* (or *asymmetric divergence* or *information gain*), which is a measure of the difference between the two sets of probabilities, $\{P^+(S_1), P^+(S_2), \dots, P^+(S_p), \dots, P^+(S_r)\}$ and $\{P^-(S_1), P^-(S_2), \dots, P^-(S_p), \dots, P^-(S_r)\}$.

The value of G , is given by the expression [Hinton and Sejnowski, 1986]:

$$G = \sum_p P^+(S_p) \ln \left[\frac{P^+(S_p)}{P^-(S_p)} \right] \quad [3.3.5]$$

where:

$P_{S_p}^+$ is the *actual* probability, that the states of the visible units correspond to the values of the elements of the p th global state $S_p = (\bar{I}_p, \bar{T}_p)$ when their states are determined by the environment

The measure of the difference between the two sets, G , is never negative. The natural log term ensures that the minimum value of zero is contributed to the sum, if the probabilities for each of the patterns are the same. The measure is asymmetric and weighted (by the first term, $P_{S_p}^+$) to give greater influence to more probable states.

Hinton and Sejnowski [1986] justify the weighting by stating that:

"When trying to approximate a probability distribution, it is more important to get the probabilities correct for events that happen frequently than for rare events. So the match between the actual and predicted probabilities should be weighted by the actual probability"

The network's model of the structure of the environment can be improved by making weight changes that perform gradient descent in ' G '. Hinton and Sejnowski [1986]

were able to demonstrate the important (and fortuitous) result, that in order to reduce the value of G by altering a particular connection weight, w_{ij} , between any pair of visible, hidden or visible and hidden neurons, i and j , only the temperature, $T(t)$ and local information describing the behaviour of the neurons at thermal equilibrium is required. Specifically they were able to show that [Hinton and Sejnowski, 1986]:

$$\frac{\partial G}{\partial w_{ij}} = -\frac{1}{T(t)} [p_{ij}^+ - p_{ij}^-] \quad [3.3.6]$$

where:

p_{ij}^+ is the probability, averaged over all environmental patterns and measured at equilibrium, that the i th and j th neurons are both firing (output=1) when the network is being driven by the environment

p_{ij}^- is the corresponding probability when the network is running freely.

These probability values can be determined entirely from the set of environment patterns $S_p = (\bar{I}_p, \bar{T}_p), p = 1, 2, \dots, r$, and hence an algorithm can perform gradient descent in G , by altering the value of a particular weight, w_{ij} , by first determining the value of $[p_{ij}^- - p_{ij}^+]$. If this value is positive (negative), the weight is increased (decreased) by an amount $\eta [p_{ij}^- - p_{ij}^+]$, where η is a *learning rate constant* whose value (between 0 and 1) is determined by experiment (Aleksander and Morton [1991]).

3.3.4 Simulated Annealing

Ackley, Hinton and Sejnowski [1985] recognised that, in a network like the one described by Hopfield [1982], there was no guarantee that the network will find the global minimum. This is because Hopfield's neuronal firing rule (equation 3.2.2) deterministically forces the network to perform gradient descent until it reaches the bottom of the energy well that is closest to its initial state. As Hinton and Sejnowski [1986] state:

"For hard problems, gradient descent gets stuck at *local* minima that are not globally optimal. This is an inevitable consequence of only allowing downhill moves. If jumps to higher energy states occasionally occur, it is possible to break out of local minima, but it is not obvious how the system will then behave and it is far from clear when uphill steps should be allowed."

They were able to overcome the problem by applying a method called *simulated annealing*, developed by Kirkpatrick, Gelatt and Vecchi [1983], which in turn was inspired by the work of Metropolis et. al. [1953]. Using the analogy of a ball-bearing on a bumpy surface, they reasoned that if a controlled amount of additional excitatory energy or 'noise' was introduced into the system there would be a probability that the network (like the ball-bearing) would be able to escape from a poor local minima, into the influence of a deeper and hence more stable minima.

Hinton and Sejnowski [1986] reasoned that if the ball-bearing (network), illustrated in figure 3.2, always moves downhill (performs gradient descent) and has no inertia, it will have an equal chance of ending up in either state A or B. This is because both minima have the same width and so the initial random starting position of the ball-bearing (network) is equally likely to lie in the influence of either of the minima. Furthermore, if the whole system were shaken, it would be more likely that the ball-bearing would be shaken from minima A to minima B, than the converse (B to A), because the energy barrier from the A side is lower. If the shaking is gentle, the probability of a transition from A to B will be many times as likely as a transition

from B to A, but both events will be very rare. So even though gentle shaking would ultimately lead to a very high probability that the ball-bearing (network) is in state B rather than A, it will take a very long time before this happens. Conversely, with violent shaking it is almost as easy for the ball-bearing (network) to escape from the deeper minima and cross the barrier (the wrong way) from B to A as move from A to B. However, with violent shaking, the much larger number of transitions, ensures that the ultimate probability ratio is approached much more rapidly, even though the ratio, and hence the probability that the ball-bearing (network) is in state B, will (in comparison with that for gentle shaking) not be very good.

Hinton and Sejnowski [1986] reasoned that a good compromise would be to start by shaking the system vigorously, and then progressively shake more and more gently.

Since [Hinton and Sejnowski, 1986]:

"This ensures that at some stage the noise level passes through the best possible compromise between the absolute probability of a transition and the ratio of the probabilities of good and bad transitions. It also means that at the end, the ball-bearing stays right at the bottom of the chosen minimum."

However they also point out that this view, of why annealing is helpful is misleading.

In the analogy, all of the states are laid out in one dimension, whereas [Hinton and Sejnowski, 1986]:

"Complex systems have high-dimensional state spaces, and so the barrier between two low-lying states is typically massively degenerate: The number of ways of getting from one low-lying state to another is an exponential function of the height of the barrier one is willing to cross. This means that a rise in the level of thermal noise opens up an enormous variety of paths for escaping from a local minimum and even though each path by itself is unlikely, it is highly probable that the system will cross the barrier."

Hinton and Sejnowski, [1986] also state that:

"At low temperatures there is a strong bias in favor of states with low energy, but the time required to reach equilibrium may be long. At higher temperatures the bias is not so favorable, but equilibrium is reached faster. The fastest way to reach equilibrium is generally to use simulated annealing: Start with a higher temperature and gradually reduce it."

An overview of the simulated annealing procedure is given by Kirkpatrick, Gelatt and Vechi [1983]:

"The simulated annealing process consists of first 'melting' the system being optimized at high effective temperature, then lowering the temperature by slow stages until the system 'freezes' and no further changes occur. At each temperature, the simulation must proceed long enough for the system to reach a steady state".

The initially high temperature value makes it likely that units in the system will be able to use the probabilistic firing rule to make uphill steps, that is, transitions that increase the energy of the system, and in so doing, enable the network to escape from poor (shallow) local minima. A *simulated annealing* (or *cooling*) *schedule* is used to slowly reduce the temperature over time. However, as Kirkpatrick et al. [1983] have stated, it is important that at each given temperature value, the system is allowed to achieve a close approximation to a *steady state* or state of *thermal equilibrium*.

The reason for this can be explained by first stating that [Aarts and Korst 1989a]:

"The simulated annealing algorithm can be formulated as a sequence of Markov chains, each Markov chain being a sequence of trials, where the outcome of a given trial only depends on the outcome of the previous trial and the value of the aforementioned control parameter."

The *control parameter* referred to by Aarts and Korst is the same quantity as $T(t)$, that is, the quantity analogous to temperature in the physical annealing process. They explain that given *infinitely long* Markov chains, as the temperature (control parameter) tends to zero, the system asymptotically converges to the set of optimal solutions with probability of 1. However, since [Aarts and Korst, 1989a]:

"Some conditions for asymptotic convergence (e.g. the infinite Markov chain length) cannot be met in practice. One commonly resorts to implementing the algorithm as a sequence of Markov chains of finite length, generated at decreasing values of the control parameter. Such a finite-time approximation requires the value specification of the following parameters:

- (1) the start value c_0 of the control parameter.
- (2) the decrement function of the control parameter.

- (3) the number of trials in each individual Markov chain
- (4) the stop criterion for terminating the algorithm

The set of parameters given above is usually referred to as the *cooling schedule*"

and:

"in a finite-time approximation the asymptoticity conditions are not attained, and thus convergence to an optimal solution is not guaranteed anymore, i.e. the algorithm obtains a local (possibly non global) optimum. For this reason the algorithm is viewed as an approximation algorithm."

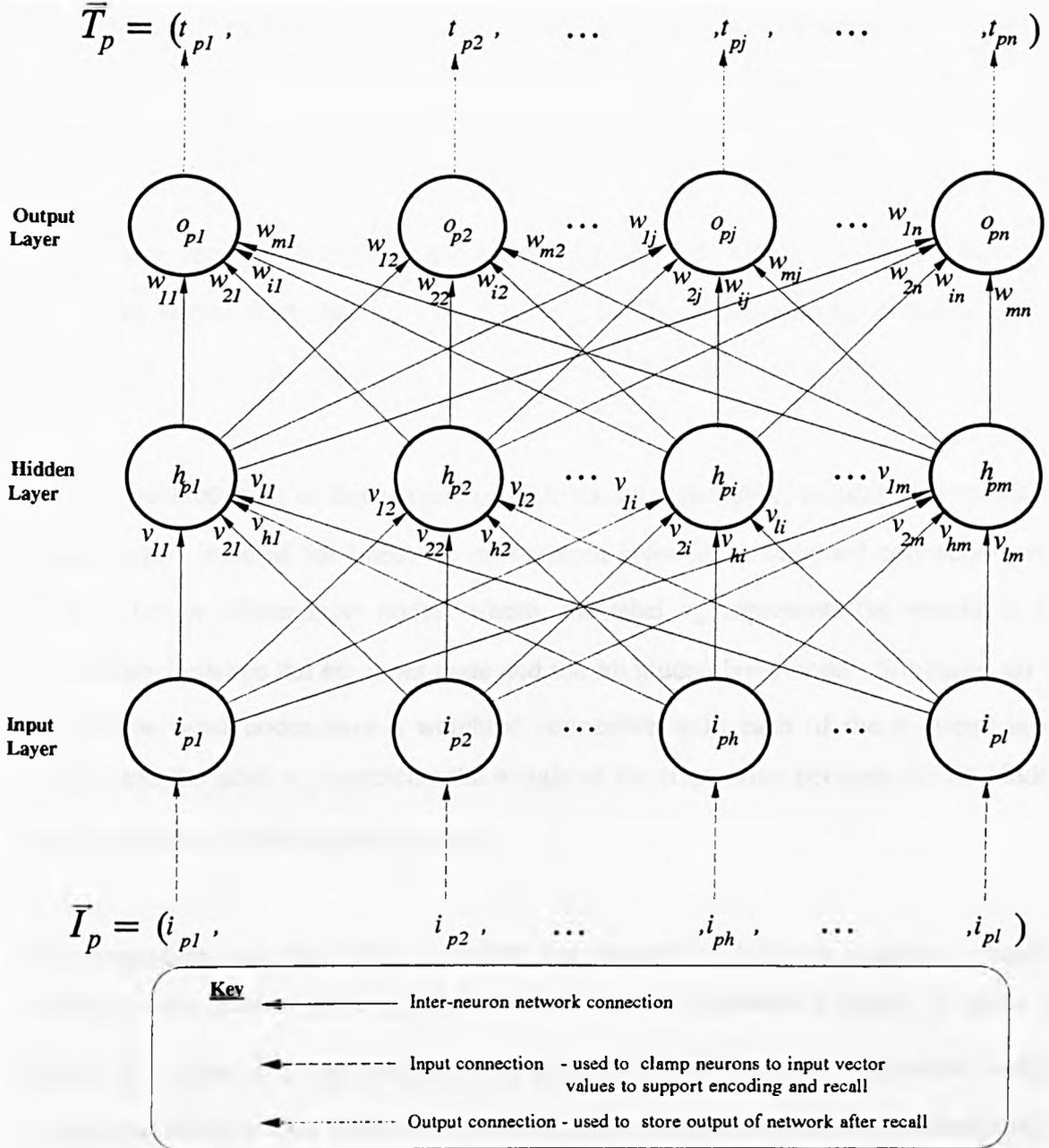
However, provided that sufficient time is given at each temperature, and the annealing or cooling is applied sufficiently slowly, the most probable final state is still the one with the lowest energy, that is the global minimum.

3.3.5 Network Architecture and Cell Properties

Like Hopfield's models, Boltzmann machines operate in discrete time, learn off-line and are nearest-neighbour pattern matchers capable of storing and retrieving binary spatial patterns. The Boltzmann machine is however, a heteroassociative pattern matcher which uses a combination of Hebbian learning and simulated annealing to encode arbitrary binary spatial patterns pairs of the form (\bar{I}_p, \bar{T}_p) , where $p=1,2,\dots,r$, the input vector is $\bar{I}_p = (i_{p1}, i_{p2}, \dots, i_{pn})$ represents the p th input vector pattern and the $\bar{T}_p = (t_{p1}, t_{p2}, \dots, t_{pn})$, the required network response (Simpson 1990).

The three-layer feedforward topology of a Boltzmann machine capable of associating binary spatial patterns is described in figure 3.5.

Figure 3.5. Topology of a Three-Layer Boltzmann machine



The firing states of hidden units, and visible units whose states are not clamped, may be updated by applying the probabilistic firing rule as described in section 3.3.1, (and by equations 3.3.1 and 3.3.2), or by applying the step function to the sum of the inputs to the unit, as described in steps 2.0 and 3.0 of section 3.3.6. The units in the three-layer Boltzmann machine described here, update their states sequentially following the learning and recall procedures described in sections 3.3.6 and 3.3.7.

3.3.6 Encoding in a three layer Boltzmann machine.

This section describes a sequential Boltzmann machine encoding (or learning) algorithm which can enable a three layer network, as illustrated in figure 6, to represent and store a set of r input/output binary vector pattern pairs.

The binary vector pattern pairs are denoted by (\bar{I}_p, \bar{T}_p) , for $p=1,2,\dots,r$; where I_p is the input vector such that $\bar{I}_p = (i_{p1}, i_{p2}, \dots, i_{pn})$. The corresponding output vector is $\bar{T}_p = (t_{p1}, t_{p2}, \dots, t_{pn})$.

The network consists of three layers of neurons: an *input* layer; a *hidden* layer; and an *output* layer. Each of the l neurons in the input layer has a weighted connection with each of the m hidden layer nodes, where, the label v_{hi} represents the weight of the connection between the h th input node and the i th hidden layer node. Similarly, all of the hidden layer nodes have a weighted connection with each of the n output layer nodes, and the label w_{ij} represents the weight of the connection between the i th hidden layer node to the j th output layer node.

The procedure described below, enables the network to learn to associate r binary spacial vector pattern pairs (\bar{I}_p, \bar{T}_p) , by adjusting the network weights, in order to reduce the value of a cost function. The procedure used to *recall* a particular output vector associated with a given input vector is quite straightforward and is described in section 3.3.7.

This recall procedure can be contrasted, with the computation performed by an auto-associative Boltzmann machine model, which applies a simulated annealing cooling schedule and a probabilistic decision rule, to attempt to perform global optimization and in so doing 'recall' an optimal solution to an example of the TSP.

The learning procedure is comprised of four distinct stages, these are:

- Initialization
- Calculation of the environmental probabilities p^+
- Calculation of the desired or target probabilities p^-
- Update the Network Connection Weights

A step by step description of each of the stages, is provided below:

Stage A: Initialization

Objective: Initialize the network (connection weights and neuron states)

Step Description

1.0 Create an $l \times m$ matrix, \mathbf{V} , to store the weights, v_{hi} , between the input and hidden layer neurons and a $m \times n$ matrix, \mathbf{W} , to store the weights, w_{ij} , between the hidden and output layer neurons.

1.1 Assign random values in the range $[+1,-1]$ to each of the connection weights v_{hi} and w_{ij} and store these random values in the matrices \mathbf{V} and \mathbf{W} .

Stage B: Calculation of the environmental probabilities p^+

Objective: Calculate the actual environmental probability $p_{v_{hi}}^+$, averaged over all environmental patterns, that the h th input layer neuron and the i th hidden layer neuron are both firing (output=1) when the network is being driven by the environment. Also calculate the equivalent actual environmental probability, $p_{w_{ij}}^+$, that the i th hidden neuron and the j th output neuron are both firing, when the network is driven by the environment.

Step Description

2.0 Starting at time $t=1$, for each pattern pair (\bar{I}_p, \bar{T}_p) , $p=1,2,\dots,r$, repeat steps 2.1 to 2.8

2.1 Set the firing (output) states of the input layer neurons to the values of the elements of the vector $\bar{I}_p = (i_{p1}, i_{p2}, \dots, i_{pn})$, and the firing states of the output layer neurons to the values of the elements of the vector $\bar{T}_p = (t_{p1}, t_{p2}, \dots, t_{pn})$.

2.2 Change the energy of the network by selecting a hidden neuron h_i , at random and set its firing state o_{h_i} according to the expression:

$$o_{h_i} = \begin{cases} 1 & \text{if } o_{h_i} = 0 \\ 0 & \text{otherwise} \end{cases} \quad [3.3.7]$$

2.3 Calculate the change in the global energy of the network ΔE_i as a result of changing the state of the hidden neuron i :

$$\Delta E_i = \sum_{h=1}^l v_{hi} o_{h_i} + \sum_{j=1}^n w_{ij} o_{o_j} \quad [3.3.8]$$

Where o_{o_j} is the output or firing state of the j th output node o_j ,

2.4 If the change in energy $\Delta E_i < 0$, keep the change to the state, o_{h_i} , of the randomly selected hidden layer neuron.

If $\Delta E_i \geq 0$, accept the change only if the probability $P_i > P_{rand}$, where P_{rand} is a number chosen randomly from the Boltzmann distribution, P_i , given by:

$$P_i = e^{-\Delta E_i / T(t)} \quad [3.3.9]$$

where $T(t)$ is the positive valued *temperature* at time t , which controls the granularity of the search for the system's global minimum. Larger values of $T(t)$ mean that the search is coarser (analogous to the ball bearing being shaken more vigorously).

If $\Delta E_i \geq 0$ and $P_i \leq P_{rand}$ return the firing state o_{hi} to its original value, before step 2.2 was implemented.

2.5 Repeat the process described in steps 2.2 to 2.4 for another hidden layer neuron

2.6 Increment the time ($t=t+1$) and calculate a new value for the temperature by applying the (cooling schedule) equation:

$$T(t) = \frac{T_0}{1 + \log t} \quad [3.3.10]$$

Geman and Geman (1984) have shown that by applying equation 3.3.10, simulated annealing will find the global minimum of a given cost function, which for this algorithm, applied to the network described in figure 6, is the cost function defined by the equation:

$$G_h = \sum_{i=1}^l p_{vhi}^+ \ln \left[\frac{P_{vhi}^+}{P_{vhi}^-} \right] + \sum_{j=1}^m p_{wij}^+ \ln \left[\frac{P_{wij}^+}{P_{wij}^-} \right] \quad [3.3.11]$$

2.7 Repeat the process described in steps 2.2 to 2.6 until the system reaches thermal equilibrium, at which point it will have relaxed into the global minimum. Thermal equilibrium is detected by examining the change in energy ΔE_i for each of the neurons in the hidden layer. Thermal equilibrium has been reached when $\Delta E_i = 0$ for all neurons in the hidden layer.

2.8 Store the activation values o_h of each of the hidden layer neurons $i=1,2,\dots,m$ - after the system has achieved equilibrium with its input and output units clamped to the values of the elements of the p th pattern pair - as elements in the vector \bar{D}_p , where $\bar{D}_p = (d_{p1}, d_{p2}, \dots, d_{pi}, \dots, d_{pm})$ using the equation:

$$d_{pi} = o_h \quad [3.3.12]$$

As stated in step 2.0, the process described in steps 2.1 to 2.8 is repeated for each of the pattern pairs (\bar{I}_p, \bar{T}_p) , for $p=1,2,\dots,r$, enabling the collection of a set of state vectors $\{\bar{D}_1, \bar{D}_2, \dots, \bar{D}_p, \dots, \bar{D}_r\}$, for convenience these are stored as rows in the $r \times m$ matrix **D**.

3.0 The collected equilibrium states of the hidden layer neurons for each of the r patterns stored as the row (vector) \bar{D}_p in the matrix **D** are used to calculate the symmetric environmental probabilities p_{vhi}^+ and p_{wji}^+ , averaged over all environmental patterns and measured at equilibrium, that the i th and j th neurons are both firing (output=1) when the network is being driven by the environment.

3.1 First calculate the probability that the h th input neuron and the i th hidden layer neuron are both in the same state, using the equation:

$$p_{vhi}^+ = \frac{1}{r} \left[\sum_{p=1}^r \Phi(o_{i_{ph}}, d_{pi}) \right] \quad \begin{array}{l} \forall h = 1, 2, \dots, l \text{ input layer neurons and} \\ \forall i = 1, 2, \dots, m \text{ hidden layer neurons} \end{array} \quad [3.3.13]$$

where:

$o_{i_{ph}}$ is the output value of the h th input layer neuron when the input vector of the p th pattern is used to clamp its state, that is, when $o_{i_{ph}} = i_{ph}$

d_{pi} is the i th element (of the vector \bar{D}_p) in the p th row of the state matrix **D**

$\Phi()$ is the correlation function, where:

$$\Phi(x, y) = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} \quad [3.3.14]$$

and x and y are binary valued scalars.

3.2 Then calculate the probability of finding both the j th output neuron and the i th hidden layer neuron in the same state, using the equation:

$$p_{wij}^+ = \frac{1}{r} \left[\sum_{p=1}^r \Phi(o_{ipj}, d_{pj}) \right] \quad \begin{array}{l} \forall j = 1, 2, \dots, n \text{ output layer neurons and} \\ \forall i = 1, 2, \dots, m \text{ hidden layer neurons} \end{array} \quad [3.3.15]$$

where:

o_{ipj} is the output value of the j th output neuron when the p th pattern is used to clamp its state (that is, when $o_{ipj} = t_{pj}$)

The probabilities p_{vhi}^+ and p_{wij}^- , are determined using the values of \bar{D}_p (stored in \mathbf{D}) after the network has achieved equilibrium with the values of the elements of the r vector pattern pairs (\bar{I}_p, \bar{T}_p) used to clamp the states of the neurons in **both** the input and output layers.

Stage C: Calculation of the desired or target probabilities p^-

Objective: Calculate the *desired* or *target* probabilities p_{vhi}^- , averaged over all environmental patterns, that the h th input layer neuron and the i th hidden layer neuron are both firing (output=1) when the hidden and output layer units are allowed to run freely. Also calculate the equivalent desired probability, p_{wij}^- , that the i th hidden neuron and the j th output neuron are both firing, when units in the network are allowed to run freely.

This is necessary because the probabilities p_{vhi}^+ and p_{wij}^+ , may have been partly influenced by the internal structure of the network, that is, one part of the network may have been influenced by the internal state of other parts of the network (and vice versa) in addition to the influence exerted by the clamped states of the input and

output layer nodes. This influence can be determined and eliminated by repeating the process described in steps 2.0 to 3.2, with the exception that, for each of the r vector pattern pairs (\bar{I}_p, \bar{T}_p) **only** the input units are clamped to the values of the input pattern vector elements.

Step Description

4.0 Starting at time $t=1$, for each pattern pair (\bar{I}_p, \bar{T}_p) , $p=1,2,\dots,r$, repeat steps 4.1 to 4.8

4.1 Clamp **only** the output states of the input layer neurons to the values of the elements of the vector $\bar{I}_p = (i_{p1}, i_{p2}, \dots, i_{pl})$.

4.2 Change the energy of the network by selecting a hidden neuron h_i , at random and set its firing state o_{h_i} in accordance with expression 3.3.7.

4.3 Calculate the change in the global energy of the network ΔE_i as a result of changing the state of the hidden neuron i :

$$\Delta E_i = \sum_{h=1}^l v_{hi} o_{h_i} \quad [3.3.16]$$

4.4 If the change in energy $\Delta E_i < 0$, keep the change to the state o_{h_i} of neuron i in the hidden layer. If $\Delta E_i \geq 0$, accept the change only if the probability, $P_i > P_{rand}$ where P_{rand} is a number chosen randomly from the Boltzmann probability P_i , given ΔE_i , calculated using equation 3.3.9.

If $\Delta E_i \geq 0$ and $P_i \leq P_{rand}$ return the firing state o_{h_i} to its original value, before step 4.2 was implemented.

4.5 Repeat the process described in steps 4.2 to 4.4 for another hidden layer neuron.

- 4.6 Increment the time ($t=t+1$) and calculate a new value for the temperature by applying the cooling schedule equation 3.3.10.
- 4.7 Repeat the process described in steps 4.2 to 4.6 until the system reaches thermal equilibrium. Thermal equilibrium is detected by examining the change in energy ΔE_i for each of the neurons in the hidden layer and has been reached when $\Delta E_i = 0$ for all hidden layer neurons.
- 4.8 Store the activation values o_{ih} of each of the hidden layer neurons $i=1,2,\dots,m$ - after the system has achieved equilibrium with only its input units clamped to the values of the elements of the p th input vector pattern, - as elements in the vector \bar{D}_p , where $\bar{D}_p = (d_{p1}, d_{p2}, \dots, d_{pi}, \dots, d_{pm})$ using equation 3.3.13.

As stated in step 4.0, the process described in steps 4.1 to 4.8 is repeated for each of the input vector patterns pairs \bar{I}_p , for $p=1,2,\dots,r$, enabling the collection of a set of state vectors $\{\bar{D}_1, \bar{D}_2, \dots, \bar{D}_p, \dots, \bar{D}_r\}$, for convenience these are stored as rows in the $r \times m$ matrix \mathbf{D} (replacing its previous contents).

- 5.0 The collected equilibrium states of the hidden layer neurons for each of the r patterns stored as the row vector \bar{D}_p in the matrix \mathbf{D} , are used to calculate the probabilities p_{vhi}^- and p_{wij}^- , averaged over all environmental patterns and measured at equilibrium, that the i th and j th neurons are both firing (on, output=1) when only the inputs are clamped.
- 5.1 First calculate the probability that the h th input neuron and the i th hidden layer neuron are both in the same state, using the equation:

$$p_{vhi}^- = \frac{1}{r} \left[\sum_{p=1}^r \Phi(o_{iph}, d_{pi}) \right] \quad \begin{array}{l} \forall h = 1, 2, \dots, l \text{ input layer neurons and} \\ \forall i = 1, 2, \dots, m \text{ hidden layer neurons} \end{array} \quad [3.3.17]$$

5.2 Then calculate the probability of finding both the j th output neuron and the i th hidden layer neuron in the same state, using the equation:

$$p_{wij}^- = \frac{1}{r} \left[\sum_{p=1}^r \Phi(o_{ip}, d_{pi}) \right] \quad \begin{array}{l} \forall j = 1, 2, \dots, n \text{ output layer neurons and} \\ \forall i = 1, 2, \dots, m \text{ hidden layer neurons} \end{array} \quad [3.3.18]$$

p_{vhi}^- and p_{wij}^- are measures that describe the degree of correlation at equilibrium between the elements in the hidden layer of the network and between the input and output layer neuronal states when the hidden and output units are allowed to run freely.

Stage D: Update the Network Connection Weights

Objective: Update the network connection weights stored in the matrices **V** and **W**.

6.0 For each of the symmetric connection weights, v_{hi} , between each of the input layer neurons, $h=1,2,\dots,l$ and each of the hidden layer neurons, $i=1,2,\dots,m$, stored in the weight matrix **V**, update the weights value by adding the symmetric change in the connection weight, Δv_{hi} , calculated according to the expression:

$$\Delta v_{hi} = \eta (p_{vhi}^+ - p_{vhi}^-) \quad [3.3.19]$$

where η is positive constant that controls the network learning rate.

6.1 Similarly, for each of the symmetric connection weights, w_{ij} , between each of the hidden layer neurons, $i=1,2,\dots,m$, and output layer neurons $j=1,2,\dots,n$, stored in the weight matrix **W**, update the weight's value by adding the symmetric change in the connection weight, Δw_{ij} , calculated according to the expression:

$$\Delta w_{ij} = \eta (p_{wij}^+ - p_{wij}^-) \quad [3.3.20]$$

- 7.0 Repeat steps 2.0 to 6.0 until the changes in the connection strengths Δv_{hi} and Δw_{ij} become zero or sufficiently close to zero for all $h = 1, 2, \dots, l$; all $i = 1, 2, \dots, m$ and all $j = 1, 2, \dots, n$.

3.3.7 Recall in a three-layer Boltzmann machine

In contrast to the encoding procedure, the process of recalling an association between a pair of arbitrary binary spatial patterns (\bar{I}_p, \bar{T}_p) is relatively straightforward. A Boltzmann machine filters an input vector pattern \bar{I}_p , which clamps the value of the input layer neurons, through the hidden layer units to the output layer units, and in so doing recalls or computes the associated output vector pattern \bar{T}_p .

The steps of the recall procedure are described below:

Step Description

- 1.0 Clamp the states of the input layer nodes to the values of the input vector \bar{I}_p
- 2.0 Update the firing states (outputs) o_h of each of the hidden layer neurons according to the expression:

$$o_h = f\left(\sum_{h=1}^l v_{hi} o_h\right) \quad [3.3.21]$$

where $f(\)$ is the step function such that:

$$f(x) = \begin{cases} +1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \quad [3.3.22]$$

- 3.0 Finally set the firing states o_j of each of the neurons j in the output layer, according to the expression:

$$o_j = f\left(\sum_{i=1}^m w_{ij} o_h\right) \quad [3.3.23]$$

If the encoding and recall algorithms were implemented successfully, the firing states o_{o_j} of the neurons in the output, can be read and will have the same values as the elements of the vector \vec{T}_p , that was associated with the input vector pattern \vec{I}_p by the encoding procedure.

3.4 Solving the Travelling Salesman Problem using Aarts and Korst's Method

3.4.1 Introduction

Aarts and Korst [1989a, 1989b] propose a Boltzmann machine model as a massively parallel alternative to the sequential simulated annealing algorithm. They describe an approach that is tailored to the travelling salesman problem, but state that it can be applied to a more general class of combinatorial optimization problems [Aarts and Korst, 1989a].

"the structure of many combinatorial optimization problems can be mapped directly onto the structure of a Boltzmann machine by choosing the right connections between the computing elements. Next, by choosing the appropriate connection strengths, the corresponding cost function can be transformed into the consensus function associated with a Boltzmann machine.

Aarts and Korst [1989a, 1989b] present both a Sequential Boltzmann machine where only a single unit at a time is allowed to update its state and an Asynchronous Parallel Boltzmann machine where each unit, in parallel simultaneously and independently (without synchronization) updates its state. They describe how, in both models, units apply a probabilistic decision rule, in their attempt to collectively maximise the value of a consensus function. The principal difference between the two models is that in the sequential machine, all of the units share a single cooling schedule, whilst in the parallel Boltzmann machine the independence of each of the units is increased by allowing each of the units to update their states asynchronously using their own, unit dependent, cooling schedules.

Aarts and Korst [1989a] demonstrate that:

"near optimal solutions can be obtained by mapping the corresponding 0-1 variables onto the logic computing elements of a Boltzmann machine, and by transforming the cost function corresponding to the 0-1 programming formulations into the consensus function associated with the Boltzmann machine."

They describe two distinct 0-1 programming formulations for the TSP, as a linear assignment problem (LAP) and as a quadratic assignment problem (QAP). Both of the

formulations model the Travelling Salesman Problem using the same network topology, as that presented by Hopfield and Tank [1985]. Specifically, their model consists of a Boltzmann machine with $N = n^2$ units, which for convenience is considered to be arranged as a grid of n rows of n neurons, where each row corresponds to a city; and each column, to a position on the tour. As in Hopfield and Tank [1985], no training is involved as the network weights are predetermined by calculation and then fixed. In Aarts and Korst's representation, however, the units are not fully connected. Their model therefore requires fewer connections, specifically: $2n^3 - n^2$ for the QAP formulation and $2n(n-1)^2$ for the LAP formulation. This is compared with the n^4 connections specified in Hopfield and Tank's [1985] model.

Although Aarts and Korst, [1989a] state that both 0-1 programming formulations generate results that are approximately of equal quality in terms of the solutions they obtain and the computation time required. However, their published results show that the LAP formulation generated solutions more frequently. Specifically in 10 out of 11 (90.9%) runs compared with 5 out of 6 (83.3%) for the QAP formulation. As a consequence, the approach taken in CONNEKT and described here is therefore restricted to the formulation of the TSP as a linear assignment problem.

In the sections that follow, the sequential Boltzmann machine model, and the process of consensus maximisation, is initially described. The components of the asynchronous parallel Boltzmann machine model are then described, since the sequential Boltzmann machine model, with an asynchronous unit-dependent cooling schedule is implemented in CONNEKT.

3.4.2 Sequential Boltzmann machine and Consensus Maximization

Aarts and Korst [1989a] describe a Boltzmann machine network comprised of N logical units, which they represent as an undirected graph $G = (U, C)$, where $U = u_0, u_1, \dots, u_{N-1}$, denotes the vertex set of *neurons* or *units* and C , denotes the edge set $C \subseteq U \times U$, that is, the set of connections between the units in the network.

The units u_i and u_j are connected via the connection $\{u_i, u_j\} \in C$, the set of all network connections, which includes the set of bias connections, denoted, C_b , where $C_b \subset C$ and $C_b = \{\{u_i, u_i\} \mid u_i \in U\}$. A *configuration* k of the Boltzmann machine is uniquely determined by the *states* of the N units in the network, where each unit can adopt one of the binary values $[0, 1]$ and the state of the i th unit in configuration k is denoted by $r_k(u_i)$. The set of all possible configurations is denoted by the *configuration space* \mathfrak{R} and the total number of configurations $|\mathfrak{R}| = 2^N$.

Aarts and Korst [1989a, 1989b] define a connection to be *activated* in a given configuration k , if the product of the states of the units linked by the connection, u_i and u_j , is unity, that is, if $r_k(u_i) \cdot r_k(u_j) = 1$. Each connection, $\{u_i, u_j\}$, has a real valued connection strength $s(u_i, u_j) \in \mathbf{R}$ associated with it, which is considered to be a measure of the *desirability* that the connection $\{u_i, u_j\}$ is *activated*. If $s(u_i, u_j) > 0$ then activation of the connection is considered desirable and the connection is called *excitatory*. Conversely, if $s(u_i, u_j) < 0$ activation is considered to be undesirable and the connection is called *inhibitory*. The bias of the i th unit is given by the strength $s(u_i, u_i)$ of the bias connection (u_i, u_i) .

For a given configuration k , a *consensus function*, $C(k)$, where $C : \mathfrak{R} \rightarrow \mathbf{R}$, describes the overall desirability of all activated connections in the configuration, and is defined

as the sum of the strengths of the activated connections, that is:

$$C(k) = \sum_{\{u_i, u_j\} \in C} s(u_i, u_j) r_k(u_i) r_k(u_j) \quad [3.4.1]$$

As Aarts and Korst [1989b] describe:

"The consensus is large if many excitatory connections are activated, and it is small if many inhibitory connections are activated. In fact the consensus is a global measure indicating to what extent the units in a Boltzmann machine have reached a consensus about their individual states, subject to the desirabilities expressed by the individual connection strengths.

The objective of a Boltzmann machine is to reach a globally maximum configuration, i.e. a configuration with maximal consensus. To reach a maximal consensus, a *state transition mechanism* is introduced which allows the units to adjust their states to those of their neighbours. The adjustment is determined by a stochastic function of the states of the neighbours and the corresponding connection strengths. Hence, as in the simulated annealing algorithm, the state transition mechanism uses a stochastic acceptance criterion to escape from locally optimal configurations."

The first step in the state transition mechanism involves changing the state of some unit, u_i (from $0 \rightarrow 1$ or from $1 \rightarrow 0$), of a given configuration k , which creates a neighbouring configuration $k^{(i)}$, whose states are given by the expression:

$$r_{k^{(i)}}(u_j) = \begin{cases} r_k(u_j) & \text{if } j \neq i \\ 1 - r_k(u_j) & \text{if } j = i \end{cases} \quad [3.4.2]$$

The neighbouring configuration $k^{(i)} \in \mathfrak{N}_k$, where $\mathfrak{N}_k \in \mathfrak{N}$ and is called a *neighbourhood* of k , that is, the set of configurations that can be obtained from k , by changing the state of a single unit.

Now if the set of connections incident with unit u_i , excluding the bias connection $\{u_i, u_i\}$, is denoted by C_{u_i} and if the set, C' , is defined to be $C' = C - C_{u_i} - \{u_i, u_i\}$, then $\Delta C_{kk^{(i)}}$, the difference in the consensus between configurations k and $k^{(i)}$, is

given by the expression:

$$\Delta C_{kk^{(i)}} = C(k^{(i)}) - C(k) \quad [3.4.3]$$

$$= \sum_{\{u_i, u_j\} \in \mathcal{C}'} s(u_i, u_j) r_{k^{(i)}}(u_i) r_{k^{(i)}}(u_j) - \sum_{\{u_i, u_j\} \in \mathcal{C}} s(u_i, u_j) r_k(u_i) r_k(u_j) \\ = \sum_{\{u_i, u_j\} \in \mathcal{C}'} s(u_i, u_j) r_{k^{(i)}}(u_i) r_{k^{(i)}}(u_j) + r_{k^{(i)}}(u_i) \sum_{\{u_i, u_j\} \in \mathcal{C}_{u_i}} s(u_i, u_j) r_{k^{(i)}}(u_j) + r_{k^{(i)}}^2(u_i) s(u_i, u_i) \quad [3.4.3a]$$

$$- \sum_{\{u_i, u_j\} \in \mathcal{C}'} s(u_i, u_j) r_k(u_i) r_k(u_j) + r_k(u_i) \sum_{\{u_i, u_j\} \in \mathcal{C}_{u_i}} s(u_i, u_j) r_k(u_j) + r_k^2(u_i) s(u_i, u_i) \quad [3.4.3b]$$

The terms in expression 3.4.3a (3.4.3b), simply express the first (second) term in equation 3.4.3, and hence the consensus for configuration k ($k^{(i)}$) in terms of components for the sets of connections \mathcal{C}' , \mathcal{C}_{u_i} and $\{u_i, u_i\}$, respectively. Now, since only the i th unit changes state, and the set of connections \mathcal{C}' excludes all connections incident with u_i , the contribution to the consensus from the first terms in expressions 3.4.3a and 3.4.3b are identical and hence cancel out. Furthermore, either the initial state of the i th unit in configuration k is zero ($r_k(u_i) = 0$) or its new state, in configuration $k^{(i)}$ is zero ($r_{k^{(i)}}(u_i) = 0$). Therefore, both the second and third terms in either expression 3.4.3a or 3.4.3b are zero, that is:

$$\text{if } r_k(u_i) = 0 \Rightarrow r_{k^{(i)}}(u_i) = 1, \text{ then } \Delta C_{kk^{(i)}} = \sum_{\{u_i, u_j\} \in \mathcal{C}_{u_i}} s(u_i, u_j) r_{k^{(i)}}(u_j) + s(u_i, u_i) \quad [3.4.4]$$

$$\text{if } r_k(u_i) = 1 \Rightarrow r_{k^{(i)}}(u_i) = 0, \text{ then } \Delta C_{kk^{(i)}} = - \sum_{\{u_i, u_j\} \in \mathcal{C}_{u_i}} s(u_i, u_j) r_k(u_j) - s(u_i, u_i) \quad [3.4.5]$$

Now since only the i th unit changes state: $r_k(u_j) = r_{k^{(i)}}(u_j)$, expressions 3.4.4 and 3.4.5 can be expressed using the single statement:

$$\Delta C_{kk^{(i)}} = (1 - 2r_k(u_i)) \times \left[\sum_{\{u_i, u_j\} \in \mathcal{C}_{u_i}} s(u_i, u_j) r_k(u_j) + s(u_i, u_i) \right] \quad [3.4.6]$$

It is clear from examination of equation 3.4.6 that each unit can locally evaluate the effect on the overall consensus of a change in its state, since the change is expressed

in terms of the state of the unit, the state of the units it is connected to, and the strengths of the connections between them.

Aarts and Korst [1989a] define a *locally maximal configuration* or *local maximum* as a configuration $k \in \mathfrak{R}$, whose consensus cannot be increased by a single state transition, that is, where:

$$\Delta C_{kk^{(i)}} \leq 0 \quad \forall k^{(i)} \in \mathfrak{R}_k \quad [3.4.7]$$

The state transitions of units in a Boltzmann machine, when maximising the value of the consensus function (equation 3.4.1), can be described, as in simulated annealing, using Markov chains, where each Markov chain describes a sequence of trials, and where the outcome of each trial depends only on the outcome of the previous trial and on the value of a *control parameter* c , $c \in \mathbf{R}^+$, which is analogous to temperature in the physical annealing process [Aarts and Korst, 1989a, 1989b].

Aarts and Korst [1989a] describe how the simulated annealing algorithm starts with a given initial solution and continuously tries to transform the current solution into a neighbouring solution through the application of a generation mechanism and an acceptance condition. In their sequential Boltzmann machine model, at a given value of the control parameter, a new configuration is generated, with a probability called the *generation probability*, by selecting a single unit, u_i , which changes its state from $0 \rightarrow 1$ or from $1 \rightarrow 0$. As a result, a particular neighbouring configuration $k^{(i)}$ is temporarily created, whose states are defined by equation 3.4.2. The state is accepted and becomes permanent, that is, the configuration, $k^{(i)}$, becomes the new current configuration k , with some probability, called the *acceptance probability*.

The *acceptance probability* is a function of both the difference in consensus and the control parameter. As in the sequential simulated annealing algorithm, the larger the

value of the control parameter, the more likely that state changes, which include consensus decreasing transitions, occur.

The probability of transforming a configuration k into a neighbouring configuration l , at a given value of the control parameter c , is called the *transition probability*, $T_{kl}(c)$, which is defined as:

$$T_{kl}(c) = \begin{cases} G_{kk^{(l)}}(c) A_{kk^{(l)}}(c) & \text{if } l \neq k^{(l)} \\ 1 - \sum_{k^{(l)} \in \mathfrak{R}_k} G_{kk^{(l)}}(c) A_{kk^{(l)}}(c) & \text{if } l = k \\ 0 & \text{otherwise} \end{cases} \quad [3.4.8]$$

Where $G_{kk^{(l)}}$ and $A_{kk^{(l)}}$ respectively denote the *generation* and *acceptance probabilities*

In Aarts and Korst's model, all units are considered to be identical, and so the probability of a particular unit being selected, that is the *generation probability*, is for simplicity, chosen to be uniform over all of the available units and independent of both the configuration, k , and the value of the control parameter, c ; and is given by the expression:

$$G_{kk^{(l)}} = \frac{1}{N} \quad [3.4.9]$$

The acceptance probability is chosen as:

$$A_{kk^{(l)}}(c) = \frac{1}{1 + \exp(-\Delta C_{kk^{(l)}}/c)} \quad [3.4.10]$$

The expression 3.4.10 differs from the standard form of the equation used in the simulated annealing algorithm, where:

$$A_{kk^{(l)}}(c) = \exp(-\Delta C_{kk^{(l)}}/c) \quad [3.4.11]$$

The form described by expression 3.4.11 is used, for example, by Simpson [1990] in his description of a three layer, pattern classifying, Boltzmann machine encoding algorithm (described in section 3.3.6). However, the difference between the two forms is only minor. Both acceptance probability equations lead to the same stationary distribution, and hence have the same convergence properties. Aarts and Korst state that the expression described by equation 3.4.10 is only chosen because it reflects the typical sigmoid response of neurons in a biological neural network; and because the symmetric distribution of equation 3.4.10 can be more easily realized using dedicated hardware (Aarts and Korst [1989b]).

Aarts and Korst [1989b] prove that, given the transition probability equations 3.4.8, 3.4.9 and 3.4.10, the Boltzmann machine converges asymptotically - as the control parameter, c , tends to zero, ($c \downarrow 0$), and the number of transitions (the length of the Markov chain) tends to infinity, ($k \rightarrow \infty$) - to the set of optimal configurations $S_{opt} \in S$. As in the simulated annealing algorithm, it is not possible in practice, to satisfy all of the conditions for asymptotic convergence, particularly the condition requiring that the number of transitions is infinite, and hence, convergence to a configuration is not guaranteed.

In practice, Aarts and Korst implement a finite-time approximation, which as they describe [Aarts and Korst, 1989a]:

"is obtained by specifying a cooling schedule, i.e. the Boltzmann machine starts off at a (sufficiently large) initial value of c , and a (randomly) chosen initial configuration. Subsequently, a sequence of Markov chains of finite length is generated at descending values of c . Eventually, as c becomes close to 0, state transitions become more and more infrequent, and finally the Boltzmann machine stabilizes into a locally maximal configuration, which is taken as the final configuration. It should be noted that the final configuration will be a (near-)optimal one, since (as in the simulated annealing algorithm) the finite-time approximation no longer guarantees convergence to an optimal configuration."

In practice, since, as stated previously, [Aarts and Korst, 1989a]:

"Some conditions for asymptotic convergence (e.g. the infinite Markov chain length) cannot be met in practice. One commonly resorts to implementing the algorithm as a sequence of Markov chains of finite length, generated at decreasing values of the control parameter. Such a finite-time approximation requires the value specification of the following parameters:

- (1) the start value c_0 of the control parameter.
- (2) the decrement function of the control parameter.
- (3) the number of trials in each individual Markov chain
- (4) the stop criterion for terminating the algorithm

The set of parameters given above is usually referred to as the *cooling schedule*"

and:

"in a finite-time approximation the asymptotic conditions are not attained, and thus convergence to an optimal solution is not guaranteed anymore, i.e. the algorithm obtains a local (possibly non global) optimum. For this reason the algorithm is viewed as an approximation algorithm."

The initial value of the control parameter, c_0 , for the i th unit, is chosen such that approximately all of the proposed transitions are accepted, an expression that satisfies this condition, is given by [Korst and Aarts, 1989]:

$$c_0 = \sum_{\{u_i, u_j\} \in C_{u_i}} |s(u_i, u_j)| \quad [3.4.12]$$

The decrement rule for calculating the $t+1$ th value ($t=0,1,\dots$) of the control parameter c_{t+1} , is obtained by multiplying the t th value, by the constant α , and is given by:

$$c_{t+1} = \alpha c_t \quad \text{where the value of } \alpha \text{ is close to, but less than } 1 \quad [3.4.13]$$

Choosing an initial value for the control parameter, c_0 , that is larger than is necessary (for the network to be able to escape from a local maxima) does not affect the quality of the solution obtained, but simply extends the computation time.

Aarts and Korst [1989a] describe and implement the *conceptually simple cooling schedule*, introduced by Kirkpatrick, Gelatt and Vecchi [1983], in which the number of attempted transitions by a particular unit (and hence the number of trials in an individual Markov chain) are chosen to be constant. The decrement rule is applied after the units in the network have each attempted L transitions. A unit does not attempt any further transitions, if after a number, M , of consecutive trials, the unit has not accepted a transition. The process is terminated when all units have ceased attempting transitions.

3.4.3 The Asynchronous Parallel Boltzmann machine and Consensus Maximization

Aarts and Korst's [1989a,1989b] asynchronous parallel Boltzmann machine model is identical to the sequential Boltzmann machine except that each unit has its own unit dependent cooling schedule, which increases the independence of each unit, and that at each time t , units are allowed to change their states simultaneously, using information about the states of their neighbours at time $t-1$.

The initial value of the cooling parameter is then given by the expression:

$$c_0^{(i)} = \sum_{\{u_i, u_j\} \in C_{u_i}} |s(u_i, u_j)| \quad [3.4.14]$$

and the cooling schedule is described by:

$$c_{t+1}^{(i)} = \alpha c_t^{(i)} \quad \text{where the value of } \alpha \text{ is close to, but less than } 1 \quad [3.4.15]$$

Parallel Boltzmann machines are potentially much faster than serial machines, since the individual units can be modelled and their states updated in parallel using multiple processors. However, asynchronous simultaneous state changes can lead to erroneously calculated state transitions which result in the activation or deactivation of the connections that are not accounted for by the calculation of the difference in the network consensus, described by equation 3.4.3.

This is easily explained. In the sequential model, if only a single unit, i , changes its state and in so doing causes the configuration k to be transformed into the neighbouring configuration $k^{(i)}$ then the difference in consensus is that described by equation 3.4.3.

Now, in the asynchronous parallel model, suppose that a pair of neighbouring units, i and j , are both 'off' (output=0) and they both simultaneously use their last known view of local information, (that is based on the now false assumption that they are both 'off'), to locally determine that they should accept a state transition to the state of 'on'. In so doing the connection $\{i, j\}$ is activated. This activation was not accounted for by the local decisions of either of the units, and may even inadvertently reduce the value of the consensus. Clearly, equation 3.4.3 does not necessarily hold for either of the transitions, as the change in state is to a configuration that is not a neighbour of the previous configuration. Hence the proof of asymptotic convergence, described in the previous section, no longer holds.

Fortunately, the probability that erroneously calculated state transitions will occur decreases as the value of the control parameter, c , decreases, and follows directly from the choice of the equation describing the acceptance probability (equation 3.4.10). Aarts and Korst, [1989a] state that:

"The probability of two (or more) neighbouring units, accepting simultaneously a state transition, becomes very small, and consequently, states obtained by erroneously accepted state transitions will be corrected."

Aarts and Korst [1989a] comment that:

"We conjecture that allowing erroneously calculated state transitions does not affect the quality of the final result obtained by the Boltzmann machine. It merely may affect the speed of the convergence. This conjecture is supported by the results of computer simulations discussed in section 6, and by other results obtained for a number of different problems... However, a rigorous proof of the asymptotic convergence of asynchronous Boltzmann machines is considered an interesting open research topic."

Aarts and Korst[1989a] applied the following parameter values, when solving the examples of the TSP taken from Hopfield and Tank [1985] (for $n=10$ and 30 cities) .

$$\alpha=0.95, L=10, M=100$$

3.4.4 The Travelling Salesman Problem

Aarts and Korst [1989a,1989b] describe how a combinatorial optimization problem can be described by the pair (S,F) where S denotes the solution space, that is, the finite set of all possible solutions to the problem and F is a cost function $F : S \rightarrow \mathbf{R}$, which represents each solution, s , by some real number $F(s)$.

In the case of the TSP (which is formally defined in section 2.3), the cost function $F(s)$ determines the length of the tour for solution s . A Boltzmann machine can be used to find solutions to examples of the TSP if a correspondence is created between tours in the problem instance under investigation and configurations of the Boltzmann machine; where a configuration which has maximal consensus corresponds to a tour with minimal length. The objective, is then, to find a globally optimal (consensus maximising) solution s_{opt} , that corresponds to a minimal value of the cost function, such that:

$$F(s_{opt}) \leq F(s) \quad \forall s \in S \tag{3.4.16}$$

Each solution s has a neighbourhood of other solutions S_s , that is defined as the set of solutions that can be reached from s in precisely one step of the generation mechanism.

A Boltzmann machine is able to compute solutions to examples of the Travelling Salesman problem given that an instance of the TSP can be directly mapped onto the network, and that appropriate connections and connection strength values are chosen which ensure that the consensus of configurations of the Boltzmann machine, correspond to values of the cost function (tour length) in the TSP.

Aarts and Korst [1989a] rewrite the TSP as an instance of a 0-1 programming problem. If n denotes the number of cities, the topology of the Boltzmann machine that is chosen to represent the problem, is a grid of $n \times n$ units (or neurons). The units in the network are members of the set $U = \{u_{ij} \mid i, j = 0, \dots, n-1 \text{ where } i \neq j\}$ since in the LAP formulation the units $u_{ii} \mid i = 0, \dots, n-1$ are not defined..

Each of the 0-1 programming variables x_{ij} are assigned to, and represented by, the states of the corresponding unit u_{ij} in the set U . The $n \times n$ matrix $U = [u_{ij}]$ stores the state $r_k(u_{ij})$ of each of the units in the current configuration k , that corresponds to the value of the 0-1 variable, x_{ij} , which denotes whether the tour goes directly from city i to city j . The strength associated with the connection, (u_{ij}, u_{lm}) , between the units u_{ij} and u_{lm} , is $s(u_{ij}, u_{lm})$. The consensus function (3.4.1) can therefore be rewritten as:

$$C(k) = \sum_{\{u_{ij}, u_{lm}\} \in C} s(u_{ij}, u_{lm}) r_k(u_{ij}) r_k(u_{lm}) \quad [3.4.17]$$

In their formulation of the TSP as a linear assignment problem (LAP), the objective is to minimize the value of the cost function:

$$F = \sum_{i,j,p,q=0}^{n-1} d_{ij} x_{ij} \quad [3.4.18]$$

where: d_{ij} is an element of the distance matrix \mathbf{D} , whose values denote the length of the shortest path from city i to city j .

subject to:

$$\sum_{i=0}^{n-1} x_{ij} = 1, \quad j = 0, \dots, n-1 \quad [3.4.19]$$

$$\sum_{j=0}^{n-1} x_{ij} = 1, \quad i = 0, \dots, n-1 \quad [3.4.20]$$

and $X = [x_{ij}]$ is irreducible (i.e. there are no sub-tours). [3.4.21]

It is clear from equation 3.4.18 that the distance element d_{ij} only contributes to the value of the cost function where the value of $x_{ij} = 1$, that is when the tour includes a step that goes *directly* from city i to city j .

Each of the configurations of the Boltzmann machine uniquely represents a particular value assignment of the 0-1 variables in the linear assignment problem. A number of these configurations do not represent tours and hence solutions of the TSP.

For the network to generate a solution (corresponding to a valid tour), - as in Hopfield and Tank [1985] - the grid of neurons must contain exactly one unit whose firing state is '1', in each row and in each column of the grid, whilst the firing states of the remainder of the units is '0' (and there should be no sub-tours). If the network is to find good or optimal solutions to the problem, minimal length tours should be favoured. The network connection strengths should therefore be chosen so as to

ensure that the network has the following operational properties [Aarts and Korst, 1989a]:

- "(P1) A configuration corresponds to a local maximum of the consensus function if and only if it corresponds to a tour.
- (P2) The shorter the length of a given tour, the higher the consensus of the corresponding configuration.

From these two properties it follows that the consensus is maximal for configurations corresponding to an optimal tour, and that near-optimal (with respect to the consensus function) configurations correspond to near-optimal tours. Note that not every configuration of the Boltzmann machine defines a tour. However, properties P1 and P2 guarantee that, if the optimization algorithm eventually stops in a (near-optimal) local maximum of the consensus function, the configuration of the Boltzmann machine corresponds to a (near-optimal) tour."

The properties are implemented by ensuring a suitable connectivity pattern and connection weight values. Aarts and Korst [1989a] define the set of connections C , as the union of two disjoint sets, C_i and C_b .

C_i is the set of *inhibitory* connections, and is defined, for all $i, j, k, l = 0, \dots, n-1$ as:

$$C_i = \{ \{u_{ij}, u_{kl}\} | (i = k \wedge j \neq l) \vee (i \neq k \wedge j = l) \} \quad [3.4.22]$$

The inhibitory connections ensure that as the network converges on a (locally) maximal consensus configuration, only one unit in each of the rows and columns, has the value '1', and hence (provided that there are no sub-tours) configurations that have locally maximal consensus values correspond to valid tours.

This is achieved by completely connecting all of the units in the same row or column and by setting the associated connection strengths to relatively large negative values compared to the value of other connection weights. If one or more inhibitory connections are activated, their negative values reduce the value of the consensus. If the network configuration has only one unit firing in each row and column, and hence

corresponds to a tour, then none of the inhibitory connections are activated, and the consensus is not reduced.

The connection strengths for inhibitory connections, are given by:

$$\forall \{u_{ij}, u_{kl}\} \in C_i : s(u_{ij}, u_{kl}) < -\min\{s(u_{ij}, u_{ij}), s(u_{kl}, u_{kl})\} \quad [3.4.23]$$

which can be re-expressed as :-

$$\forall \{u_{ij}, u_{kl}\} \in C_i : s(u_{ij}, u_{kl}) = -\min\{s(u_{ij}, u_{ij}), s(u_{kl}, u_{kl})\} - \theta \quad [3.4.24]$$

C_b is the set of *bias* connections, and is defined, for all $i, j, k, l = 0, \dots, n-1$ as:

$$C_b = \{u_{ij}, u_{kl} \mid (i = k \wedge j = l)\} \quad [3.4.25]$$

Bias connections are feedback or loop connections that attempt to ensure that at least one unit in each row and column is firing. This is achieved by making the bias connections on each of the units, positive (excitatory), since this encourages the units to fire. The bias connection strength is then chosen as a function of the distance ' d_{ij} ', such that the smaller the distance, the larger the bias value:-

$$\forall \{u_{ij}, u_{ij}\} \in C_b : s(u_{ij}, u_{ij}) = -\max\{d_{ik} \mid k = 0, \dots, n-1\} - d_{ij} \quad [3.4.26]$$

However, Aarts and Korst [1989a] state that the choice of equation 3.4.26 results in a relatively small difference in consensus between short and long tours, compared to the consensus difference that occurs between tours and non-tours. This means that the control parameter value must be decremented slowly, if a near-optimal solution is to be obtained. As a consequence the network's convergence is rather slow.

In their computer simulations, Aarts and Korst [1989a] achieve faster convergence by using an alternative expression to determine the value of the bias connections. This is

based on subtracting the distance from city i to city j from the average rather than from the maximum distance value, that is:-

$$\forall \{u_{ij}, u_{ji}\} \in C_b \quad s(u_{ij}, u_{ji}) = \sum_{m=0, m \neq i}^{n-1} \frac{d_{im}}{(n-1)} - d_{ij} \quad [3.4.27]$$

This reduces the strengths of the bias connections and as a consequence also results in smaller inhibitory connection weight values, enabling the Boltzmann machine to more easily discriminate between short and long tours.

Now although maximisation of the consensus function is equivalent to minimization of equation 3.4.18, subject to equations 3.4.19 and 3.4.20, sub-tour elimination (equation 3.4.21) is not ensured and hence the final configuration may contain sub-tours.

Aarts and Korst [1989a] introduce the penalty function $P_{kk^{(j)}}$ which is associated with the transition from configuration k to $k^{(j)}$. This function alters the acceptance probability (equation 3.4.10) which becomes:-

$$A_{kk^{(j)}}(c) = \frac{1}{1 + e^{-(\Delta C_{kk^{(j)}} + P_{kk^{(j)}})/c}} \quad [3.4.28]$$

With the penalty function included, the definition of a local maximum is now a configuration k for which:-

$$\forall k^{(j)} \in \mathfrak{R}_k : \Delta C_{kk^{(j)}}(c) + P_{kk^{(j)}} < 0 \quad [3.4.29]$$

Aarts and Korst [1989a] state that:-

"The penalty function is considered as a function which changes the transition probability among configurations without affecting the value of the consensus function. Its value is chosen such that it penalizes transitions to configurations containing sub-tours."

To support this, they introduce the variables $y_k(u_{ij})$ and $z_k(u_{ij})$ which assign to each unit u_{ij} a binary number. They are defined recursively as:-

$$y_k(u_{ij}) = \begin{cases} 1 & \text{if } i = 0, \\ \bigvee_{m=0}^{n-1} y_k(u_{mi}) r_k(u_{mi}) & \text{otherwise} \end{cases} \quad [3.4.30]$$

$$z_k(u_{ij}) = \begin{cases} 1 & \text{if } j = 0, \\ \bigvee_{m=0}^{n-1} z_k(u_{jm}) r_k(u_{jm}) & \text{otherwise} \end{cases} \quad [3.4.31]$$

For a given configuration k , the variable $y_k(u_{ij})$ denotes whether or not there exists a directed path from city 0 (chosen arbitrarily) to city i . The variable has the value '1' if there is a directed path and '0' if no directed path exists. Similarly, the variable $z_k(u_{ij})$ denotes whether there is a directed path from city j to city 0.

A configuration k that satisfies equations 3.4.19 and 3.4.20 will correspond to a tour and have no sub-tours if:-

$$\sum_{i=0}^{n-1} y_k(u_{ij}) r_k(u_{ij}) = \sum_{i=0}^{n-1} z_k(u_{ij}) r_k(u_{ij}) = 1 \quad j = 0, \dots, n-1 \quad [3.4.32]$$

$$\sum_{j=0}^{n-1} y_k(u_{ij}) r_k(u_{ij}) = \sum_{j=0}^{n-1} z_k(u_{ij}) r_k(u_{ij}) = 1 \quad i = 0, \dots, n-1 \quad [3.4.33]$$

The value of the variables $y_k(u_{ij})$ and $z_k(u_{ij})$ denoting respectively whether a tour exists from city 0 to city i and from city j to city 0, are in turn dependent on the values of the variables $y_k(u_{mi})$ and $z_k(u_{jm})$ ($m = 0, \dots, n-1$) which need to be communicated to enable unit u_{ij} to locally determine whether to accept a proposed state transition. To enable this, the set of connections is extended by the addition of the set of *communication connections* C_c , which are defined as:-

$$C_c = \{ \{u_{ij}, u_{kl}\} \mid i = l \vee j = k \} \quad \forall i, j, k, l = 0, \dots, n-1 \quad [3.4.34]$$

There is no connection strength value associated with these connections and as such they do not contribute to the consensus. Aarts and Korst [1989a] state that:-

"Communicating and computing the variables $y_k(u_{ij})$ and $z_k(u_{ij})$ is very simple as compared to other calculations carried out by the units. Consequently, the time needed to propagate changes of the variables through the network is considered to be negligible. Due to the fact that transitions among configurations can still be computed locally the principle of massive parallelism is not affected. "

Aarts and Korst [1989a] define the following choice for the value of the penalty function , which they state enables the sub-tour elimination described above:-

$$P_{kk^{(w)}} = (1 - 2r_k(u_{ij})) \max\{d_{ij} | k = 0, \dots, n-1\} \left[\sum_{\{u_{ij}, u_{im}\}} Q_2 r_k(u_{im}) + Q_1 \right] \quad [3.4.35]$$

where :

$$Q_1 = \begin{cases} 1, & y_k(u_{ij}) \oplus z_k(u_{ij}) = 1 \\ -1, & y_k(u_{ij}) + z_k(u_{ij}) = 0 \\ 0, & \text{otherwise} \end{cases} \quad [3.4.36]$$

$$Q_2 = \begin{cases} -1, & ((y_k(u_{ij}) \oplus z_k(u_{ij})) \times (y_k(u_{im}) \oplus z_k(u_{im}))) = 1 \\ 0, & \text{otherwise} \end{cases} \quad [3.4.37]$$

and the symbol \oplus denotes the *exclusive-or* binary operator.

Aarts and Korst [1989a] provide the following explanation for the structure of these equations:-

"The variables Q_1 and Q_2 are chosen such that for each configuration k , not corresponding to a tour, there exists a neighbourhood configuration $k^{(ij)}$ with $\Delta C_{kk^{(w)}}(c) + P_{kk^{(w)}} > 0$.

In this way, the penalty function $P_{kk^{(w)}}$ is chosen such that transitions to configurations corresponding to subtours not visiting city 0 are penalized whereas transitions to configurations corresponding to subtours visiting city 0 are enhanced. In other words, subtours not visiting city 0 are broken up, and subtours visiting city 0 are enlarged until eventually all cities are visited (which corresponds to a feasible tour)."

3.4.5 Solving an example of the TSP

The first step in modelling and solving an example of the TSP on a sequential Boltzmann machine implementing an asynchronous unit-dependent cooling schedule, is to create a suitable network topology and set of weighted connections. If the example of the TSP has n cities, a network is created with $N = n^2$ binary $[0,1]$ units. Connections are created between the units in accordance with expressions 3.4.20, 3.4.22 and 3.4.24. The values of the connection strengths are set using equations 3.4.21, 3.4.23 and 3.4.26.

A solution is then obtained by allowing the network to run, that is, by allowing units to sequentially determine whether to change their states, by applying the probabilistic decision rule (equation 3.4.1 and 3.4.2). Each unit has its own cooling schedule. The value of the cooling parameter is decremented after the unit has completed L trials. Each unit ceases to attempt to change its state, if after M trials it has not accepted a transition. The process terminates when *all* units have ceased to generate transitions.

In the infrequent event that the network fails to settle into a configuration that corresponds to a tour, the Boltzmann machine is simply reset and restarted.

3.4.6 Performance of Aarts and Korst's Implementation

Aarts and Korst [1989b] describe the results of computer simulations of their asynchronous parallel Boltzmann machine model. They report that simulations carried out on a VAX 11/785 computer for examples of the TSP, with 10 and 30 cities respectively, taken from Hopfield and Tank [1985]. The computation times range from a few minutes for the 10 city problem, up to a few hours for the 30 city problem instance.

They investigated the quality of the solutions obtained by collecting a number of statistics. The statistics describing the performance of the system using the linear assignment formulation (LAP) are reproduced in table 3.1.

Statistic	10 Cities	30 Cities
Average Tour Length ($\bar{\ell}$)	2.783	5.643
Spreading of the Tour Length (σ)	0.097	0.434
Smallest observed tour length (l_v)	2.675	4.769
Largest observed tour length (l^{\wedge})	3.060	6.297
Sample Size (m)	100	40
Average number of iterations (I)	1.1	1.2
Smallest known value of the tour length (l_{\min})	2.675	4.299

Table 3.1 : Statistics describing the results of applying the asynchronous parallel Boltzmann machine model to instances of the 10 city and 30 city problems

Aarts and Korst [1989a, 1989b] demonstrate that an asynchronous parallel Boltzmann model, based on the sequential implementation described in section 3.4, is a feasible structure for approximately computing-near optimal solutions to examples of the TSP. They applied their parallel Boltzmann machine, to instances of the 10-city and 30-city problems investigated by Hopfield and Tank [1985], using both quadratic assignment problem (QAP) and a linear assignment problem (LAP) formulations.

They found that the quality of the solutions obtained compares favourably with those obtained by Hopfield and Tank [1985] but were slightly inferior to those obtained by either sequential simulated annealing [Kirkpatrick, Gelatt and Vecchi, 1983] or by Lin and Kernighan's [1973] local search heuristics - which typically require only a few seconds (Aarts and Korst, 1989b).

Korst and Aarts [1989] describe the application of their parallel Boltzmann machine implementation to the Max-Cut Problem, the Independent Set Problem and the Graph

Colouring Problem. They report that [Korst and Aarts, 1989]:

"Final solutions can be obtained by the Boltzmann machine which are comparable, in quality, to the solutions obtained by simulated annealing. For all three problems the Boltzmann machine uses considerably less computation time, provided the model is emulated on a parallel computer."

The relative performance of Aarts and Korst's asynchronous parallel Boltzmann machine implementation compared to the simulated annealing algorithm is worse than for other combinatorial optimization problems. Korst and Aarts [1989] state that:

"we conclude that it is much harder to obtain near-optimal results for the travelling salesman problem than for the graph problems discussed in this paper. Two reasons can be given to explain this feature.

First, choosing the appropriate connection strengths for the travelling salesman problem is difficult (cf. Aarts and Korst [1989a]). If the strengths are chosen to meet property P1, the convergence of the Boltzmann machine is slow due to the fact that the difference in the consensus function between good and bad tours is relatively small compared to the difference in consensus between tours and nontours. Choosing the connection strengths such that these differences are large results in a situation where final results are often infeasible.

Second, transferring a given tour into another one, using the given Boltzmann machine formulation, requires in many cases a number of steps in which configurations are visited corresponding to infeasible solutions, which often have low values of the consensus. This increases the probability of becoming trapped in configurations corresponding to locally optimal tours whose length deviates substantially from that of an optimal tour.

Evidently the deficiency mentioned above depends strongly on the construction that is used for the travelling salesman problem in the Boltzmann machine. It is however hard to think of other, more efficient constructions for this problem."

Korst and Aarts [1989] overall conclusion is that the Boltzmann machine is able to obtain results (to the three classes of problems described above) that are comparable with the simulated annealing algorithm, in terms of the quality attained, and require less time. In the case of the TSP, the sequential simulation of their parallel implementation takes significantly longer, a few minutes (for the 10-city), compared to a few seconds, and the average quality of solutions slightly inferior to those produced by conventional algorithms.

However, they suggest that the work of researchers, like Alspector and Allen [1987], who have presented a design for a VLSI chip with 5×10^5 gates, which could implement a Boltzmann machine with approximately 2000 units. They report that an estimate suggests that their chip will run approximately one million times faster than simulations on a VAX computer. They speculate that, other developments, for example those in optical technology, proposed by Ticknor and Barrett [1987], might even further increase this factor, by some orders of magnitude.

3.5 Application of the Boltzmann machine to the GAP, TSP and VRP

In addition to the work of Aarts and Korst, references to the work of other researchers that have been concerned with solving the TSP or the VRP include: Boseniuk and Ebeling [1988]; Pensini et al. [1989]; DeGloria et al. [1993] and Tesar et al. [1989].

3.6 Parallel Boltzmann machine Research

A number of researchers have conducted research into the implementation of parallel Boltzmann machine networks. These include:

Nang et al. [1991], who present an efficient mapping of a Boltzmann machine onto a Distributed Memory Multiprocessor (DMM) architecture which includes a Parallel Boltzmann Machine Convergence Algorithm and a Parallel Boltzmann Machine Learning Algorithm. They define a speed-up ratio S_p , which is the ratio of the time taken by their sequential learning algorithm and their parallel learning algorithm, in a network with p processors. They investigate the increase in the speed-up ratio as a function of the number of processors, p , for a number of Boltzmann machine topologies. They find that there is a cost-effective number of processors for a given topology, which is dependent on the ratio of C , the communication time for a processor to send and/or receive a message and M_a , the time taken to add two floating point numbers. They state that beyond this cost-effective number, the increase in the speed-up ratio is reduced.

Apolloni and de Falco [1991] describe their study of a parallel Boltzmann machine implementation in which each unit is updated independently of, but simultaneously with, the other units in the network. They state that their parallel implementation explores an "energy" landscape that is quite different to that for Ackley, Hinton and Sejnowski's [1985] sequential model and demonstrate that a learning rule can still be derived.

3.7 Application of Other Connectionist Models to the TSP and VRP

Since Hopfield and Tank's successful application of Hopfield's continuous network model (Hopfield and Tank [1985]), a number of other researchers have demonstrated that a number of different neural network models can be used to solve examples of combinatorial optimization problems, like the TSP. These include:

Durbin and Willshaw [1987] introduce the elastic net method in their description of an a parallel analogue approach to the TSP. Their approach uses an iterative procedure to non-uniformly, gradually elongate a circular closed path until it passes sufficiently close to all of the cities to define a tour. They report that their approach produces shorter tours than Hopfield and Tank's [1985] method and present results that show that the results of their implementation are comparable with simulated annealing, in both quality and computation time.

Boeres and de Carvalho [1992] propose a filtering mechanism, called the γ -filter in their implementation for the TSP of a faster elastic net method, based on that introduced by Durbin and Willshaw [1987]. They compare the performance of their implementation with the elastic net and the Lin-Kernighan heuristic algorithm [1973] for examples of the TSP with $n=100,200,300,400,500,750$ and 1000 cities. They state that the time consumed by the Lin-Kernighan heuristic is only comparable with that for elastic net algorithms for small values of n , and for larger values of n tends to increase disproportionately and that the quality of solutions obtained are slightly

inferior (by roughly 5 to 10%). They finally state that the γ -filter reduces the time taken to generate solutions compared to the original elastic-net method, their results suggesting that this improvement is also of the order of 10%.

Approaches based on Kohonen's self-organizing map are presented by Angeniol et al. [1988] and Ghaziri [1991]. The latter reports that their implementation was able to solve examples of the VRP with up to 250 cities (using 400 neurons and 700,000 iterations) in approximately 6.5 minutes CPU time, using a Sun SPARC Workstation. He states that the network parameters were easy to tune - in contrast to those in the Hopfield model and the Boltzmann machine.

Hueter [1991] presents an adaptive ring neural network, whilst Yu and Lee [1992] propose a parallel mean field annealing neural network and a new energy function for the TSP, which are both capable of generating near-optimal solutions to examples of the TSP.

Bhide et al. [1993] present a modified Boolean neural network model (MBNN) which they apply to examples of the TSP with 50 and 100 cities, and present results (from several 50 run sets) that demonstrate that the quality of the solutions obtained compare favourably with those generated by the simulated annealing algorithm.

Potvin and Shen [1991] describe a neural network approach to the vehicle dispatching problem; Shinozawa et al. [1991] discuss a connectionist approach for solving Dynamic TSP's and Szu [1991] describes a fast TSP algorithm based on a model which has Binary Neuron output and an analog neuron input.

Finally, Akiyama et. al [1989] present the Gaussian Machine which is comprised of neurons which have a graded response, as in the Hopfield network, but behaves stochastically like a Boltzmann machine, hence enabling the network to escape from local minima. They applied the Gaussian machine, and other models including the

Hopfield model and the Boltzmann machine model to the 10 city problem defined by Hopfield and Tank [1985]. They terminated each of 1000 trials after 100 steps each and found that their algorithm, on average generated solutions that are slightly shorter than those produced by Hopfield's model and significantly better than those generated by the Boltzmann machine. They concede, however that the selection of parameters for the Boltzmann machine may not be optimal. It should be noted that their results conflict with the seemingly more rigorous comparison, made by Aarts and Korst [1989b].

3.8 Conclusions

A number of Boltzmann machine networks with different network topologies and patterns of connectivity have been developed. These include three-layer feedforward models [Simpson, 1990], that learn to represent and store an environment comprised of a set of arbitrary, binary spatial vector pattern pairs and auto-associative Boltzmann machine networks [Aarts and Korst, 1989a, 1989b] that model combinatorial optimization problems, and attempt to perform global optimization by applying a probabilistic decision rule and a simulated annealing cooling schedule.

The introduction of a 'temperature' dependent amount of probabilistic noise to the state change equation means that, unlike Hopfield's models, the Boltzmann machine is not forced to deterministically perform gradient descent in the energy space, but rather allows state transitions which increase the network's total energy level. This feature enables Boltzmann machine networks to escape from the influence of (poor) local minima. Ackley, Hinton and Sejnowski [1985], describe how the relative probability of the Boltzmann machine being in one of a pair of states is related to the difference in energy between the two states. This leads to the important property that the most probable final state is the state with the lowest energy level.

If the network connections and weights are designed so that the state or states of the network with the lowest energy level correspond to optimal solutions to a particular problem, the most probable final state of the network will then correspond to the optimal solution to the problem. Hence, given an infinite number of transitions at each temperature and a suitable annealing schedule, the Boltzmann machine is able to perform global optimization. In practice a finite-time approximation is implemented which finds a locally (possibly non-globally) optimal solution.

The principal disadvantage of Boltzmann machines is the slowness of the encoding procedure when learning to relate arbitrary binary vector patterns and the slow simulated annealing cooling schedule when computing solutions to combinatorial optimization problems.

A second important disadvantage, is that although weight adjustment through gradient descent can be used to create a particular set of trained states which map a set of arbitrary binary input vector patterns onto a set of arbitrary binary output vector patterns, the procedure does not prevent the creation of false energy wells, which correspond to false, (i.e. incorrect) mappings; it simply reduces the probability that these have lower energies and are therefore less likely to occur than the minima that correspond to trained states.

Units in a Boltzmann machine can either update their states, one at a time, or alternatively, all of the units may simultaneously update their states in parallel. In the former case, the system may be referred to as a *Sequential* or *Serial* Boltzmann machine, whilst systems that implement the latter strategy are referred to as *Parallel* Boltzmann machines. A number of researchers, including Aarts and Korst [1989b] have described the application of sequential and/or parallel implementations of the Boltzmann machine model to the TSP.

Aarts and Korst's [1989b] state that their implementation of an asynchronous Boltzmann machine model, using either a quadratic assignment or linear assignment 0-1 programming formulation can develop very good or even optimal solutions to examples of the TSP. The quality of solutions obtained is at least as good as that reported by Hopfield and Tank [1985].

Korst and Aarts [1989] parallel Boltzmann machine implementation demonstrates that even though the selection of appropriate connection strengths to model examples of the TSP is difficult, and hence it appears to be harder to obtain near optimal solutions to examples of the TSP, than for other optimization problems, like graph colouring, parallel Boltzmann machine models can obtain near optimal solutions to examples of the TSP that compare favourably, in terms of quality, with those obtained by Hopfield and Tank [1985]. However, the quality of solutions are on average slightly inferior to those obtained by both sequential simulated annealing algorithms [Kirkpatrick, Gelatt and Vecchi, 1983] and heuristic algorithms like that of Lin and Kernighan [1973].

The time taken to generate solutions to the TSP also compares poorly with sequential simulated annealing and Lin-Kernighan heuristics, but research into Very Large Scale Integration (VLSI) designs that can support the operation of Boltzmann machine models offers the possibility that parallel Boltzmann machines could very quickly generate near-optimal solutions to examples of the TSP.

Finally, a number of issues remain unresolved, not least of which is the question of whether existing models will be able to be scaled up and still be able to compute very good or near-optimal solutions to reasonably large problems. If this proves to be the case, it is quite likely that new applications will be found and that consequent commercial benefits will accrue. However it is worth noting that such a development, would not resolve the apparent basic intractability of very large np -complete problems.

4.0 Connectionist Expert Systems

4.1 Introduction

Connectionist expert system research is both expanding and attracting the interest of researchers from a wide variety of disciplines. However, as a field of research, or even a research discipline, it is still relatively immature. One consequence of this is that much of the terminology used in the field, including the term 'connectionist expert system', has yet to be standardised. At its widest, connectionist expert system research encompasses a variety of ways in which conventional expert system and connectionist approaches can be combined. These include:

- Expert Systems that have connectionist knowledgebases and either traditional or neural inference engines.
- Expert systems that incorporate neural networks in the antecedents and/or consequents of rules.
- Expert systems created from the interconnection of multiple individual neural networks.
- Neural Networks that perform types of tasks previously associated with expert systems.
- Neural networks that use expert systems to perform pre-processing of input data and the post-processing of outputs.
- Neural network architectures that perform inductive and deductive reasoning
- The automatic generation of neural networks from expert systems.
- The automatic extraction of expert system rules from neural networks
- Connectionist expert system hardware

In this thesis, as in for example Gallant [1993], the term connectionist expert system is used more narrowly, to mean an expert system that has a connectionist network as its knowledgebase.

However, to ensure that all relevant research is included, this review describes both connectionist expert system research proper and important contributions describing the combination of connectionist and expert system approaches.

One of the earliest, and subsequently most sustained and significant contributions to connectionist expert system research is that by Stephen Gallant [1986a,1986b,1987,1988a,1988b,1990a,1990b,1993] and his work in collaboration with Hayashi [Gallant and Hayashi, 1991]. Gallant's model provides the basis for a major part of the connectionist expert system capability of CONNEKT and is described in some detail in the section that immediately follows. Subsequent sections review a number of examples of connectionist expert system and related research.

4.2 Gallant's Connectionist Expert System Model

4.2.1 Introduction

Gallant [1985,1988a,1993] presents a connectionist expert system model which has a discrete perceptron-based neural architecture as its knowledgebase and performs inferencing using a connectionist *Matrix Controlled Inference Engine* (MACIE).

He describes [Gallant, 1985] how the specification of rules in a *linear discriminant network* form, that is suited to linear discriminant or perceptron calculation [Rosenblatt, 1962; Minsky and Papert, 1969] can capture the antecedent and consequent clause relations of traditional expert system IF-THEN rules, along with additional constraints and support automatic (connectionist) learning techniques.

The nodes in Gallant's neural knowledgebase can have an associated semantic meaning, and may be assigned labels or names to represent a semantic feature or concept. The activity of the cell then has a corresponding semantic interpretation.

As Gallant [1993] states:

"By giving names to the nodes of the network we can specify a semantic interpretation for the activity of any cell."

Given a set of training examples or rules or a combination of the two, Gallant's methodology enables the automatic generation of strictly feedforward (acyclic) networks from either training examples, rules or from a combination of the two. The network topology may be simplified through the incorporation of dependency information obtained from a human domain expert.

The network (knowledgebase) is comprised of three types of node (or cell), these are: *input* (or *terminal*) nodes, *hidden* (or *intermediate*) nodes and *output* (or *goal*) nodes. Gallant's *pocket learning algorithm* uses supervised learning to adjust the weights on connections between nodes in the network knowledgebase. The algorithm can be applied to both separable and non-separable data. In the latter case his system can either attempt to find a set of weights that satisfies all of the rules and as many of the training examples as possible or can trigger the creation of additional hidden nodes in an attempt to make the mapping problem separable.

Gallant [1993] describes how 'IF-THEN' rules can be extracted from the knowledgebase, for example, to support the generation of explanations supporting the inferences made by his system, and in collaboration with Yoichi Hayashi [Gallant and Hayashi, 1991] present a system which provides confidence estimates by generating and examining the inferences of multiple connectionist expert systems.

4.2.2 Connectionist Expert System Components

Gallant's connectionist expert system model is comprised of three principal components: the *User Interface*, the *Matrix Controlled Inference Engine (MACIE)* and a neural *Knowledgebase*.

The user interface performs the same role as in a conventional expert system, as described for example by Giarratano and Riley [1989], Rich and Knight [1991], Winston [1992] and Parsaye and Chignell [1988]. As the name suggests, the user interface manages the input and output of information between the system and the user. This includes the entry of values, by the user, both as initial inputs and in response to questions raised by the inference engine, and outputs from the system which include: questions, the results of inferences and explanations.

The knowledgebase is a perceptron based feedforward connectionist network structure comprised of input, hidden and output nodes. All of the nodes in the network, - except additional hidden nodes added by the network generation and learning algorithm to enable non-separable relations to be learnt - have an associated semantic meaning and a name (label) to represent this meaning.

The particular semantic associations are of course application dependent. Gallant [1988,1993] describes an example system which diagnoses and recommends treatment for acute sarcophagal disease. In his example the input nodes represent symptoms, the hidden nodes are associated with possible diagnoses and the output nodes with potential treatments.

Each of the input nodes has a question associated with it. These may be used by the inference engine to elicit the value of unknown variables from the user and are stored in the knowledgebase in an 'input file' along with the semantic labels, training examples and any inter-node dependency information.

The other principal component in Gallant's model is the *Matrix Controlled Inference Engine* or MACIE. This performs the same role as the inference engine in a traditional expert system, and is capable of performing both forward chaining and backward chaining, generating confidence estimates and of explaining its reasoning.

MACIE operates by repeatedly implementing forward and backward chaining phases until sufficient (one or more) inferences have been made or until it determines that no inferences are possible.

The forward chaining phase operates through the propagation of activations through the feedforward neural knowledgebase. If more information is required before an inference can be made, the system executes a backward chaining phase. It is during this phase that the system selects questions to be put to the user to obtain the values of unknown variables, in order of those that are likely to make the largest contribution toward the completion of an inference.

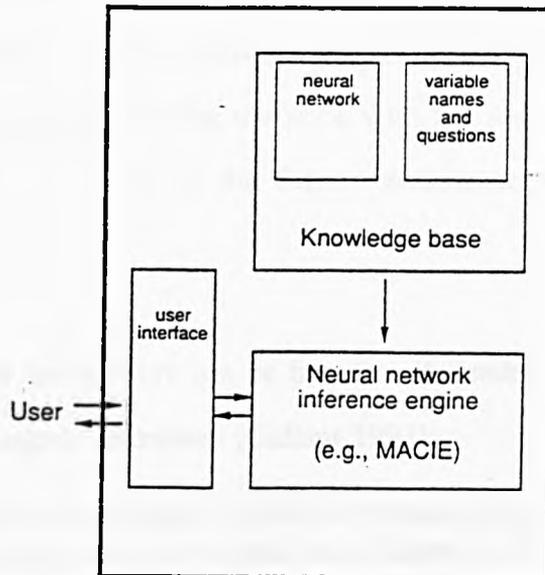
Gallant [1985,1988,1993] describes a number of heuristics that can be used to estimate the likelihood or confidence that an unknown variable will eventually be deduced to be *TRUE* or *FALSE* and presents algorithms that can be used (by MACIE) to extract IF-THEN rules from the neural knowledgebase in order to support its reasoning.

It is interesting to note that in his original (1985) paper, Gallant does not emphasise the connectionist nature of MACIE. In a later work [Gallant,1993] he explains:

"It is amusing to note that MACIE got its name because it was thought that "matrix controlled" would sound less objectionable than a name involving "perceptron" or "neural network"; the early 1980's were not hospitable times for such terms."

The component structure of Gallant's connectionist expert system model is illustrated in Figure 4.1 (reproduced from [Gallant, 1993]).

Figure 4.1 : Components of Gallant's connectionist expert system model



4.2.3 Knowledge Representation

In Gallant's model knowledge is stored in a neural network which acts as a knowledgebase for the connectionist Matrix Controlled Inference Engine. Gallant uses a *localist* representation, since in his model a particular node exclusively represents a single semantic concept or feature. This can be contrasted with the distributed representation of for example [Touretzky and Hinton, 1988].

The relationship or dependency between these nodes (concepts and features) is described by the weights associated with directed arcs (or connections) between nodes. The presence of a weighted directed arc from node u_j to node u_i indicates that the value of u_i is dependent on the value of node u_j and u_j is said to be a member of u_i 's *dependency list*. The degree of dependency is modelled by the magnitude of the connection weight. All connection weights are integer valued. A large positive (negative) weight indicates that the node with the dependency is strongly influenced by the presence (absence) of the feature associated with the node on which it is dependent.

The structure of the network can be based on dependency information derived from a human domain expert. However [Gallant 1993]:

"The dependency information is optional because every output cell may be specified as dependent upon every input cell as in figure 14.7. This figure shows a default dependency that is useful for some applications. However if more precise dependency information is available then dependency lists improve network generation algorithms since accidental correlations between unrelated variables are prevented from influencing the final network weights."

(Figure 14.7 from Gallant [1993] is reproduced immediately below as figure 4.2)

Figure 4.2 : Default dependency network with no dependency information

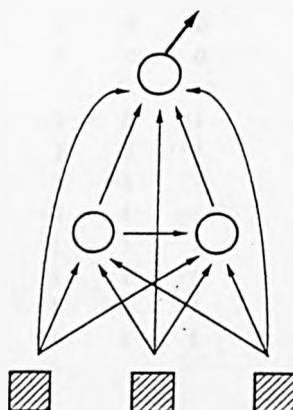


Figure 14.7
Default network with no dependency information.

The dependency network can also be represented in the form of an adjacency matrix where each row represents a hidden or output node and each column the value of each of the nodes in the dependency lists. In this representation the value '1' in a particular row and column indicates that the hidden or output node described by that row is dependent on the *presence* of the feature associated with that column, whilst the value 0 indicates that no such dependency exists. Training examples may also be stored in the same form, with values having the same meaning as in the dependency network. The additional value '-1' in a particular row/column simply represents a dependency on the *absence* of the particular feature associated with the column.

Example dependency and training matrices are described in figure 4.3. These are reproduced from [Gallant,1993] and describe dependency information and a number of training examples for an example network which provides diagnosis and suggests treatment for acute sarcophagal disease (illustrated below in figure 4.5).

Figure 4.3: Example dependency and training examples matrices

DEPENDENCY:	U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	U11	
	1	1	1	0	0	0	0	0	0	0	0	-> U7
	0	0	1	1	1	0	0	0	0	0	0	-> U8
	0	0	0	0	0	1	1	1	0	0	0	-> U9
	0	0	1	0	0	0	1	1	0	0	0	-> U10
	0	0	0	0	0	0	0	0	1	1	0	-> U11
EXAMPLES:												
	1	1	1	-1	0	-1	1	-1	1	-1	1	TE#1
	-1	-1	-1	1	1	-1	-1	1	1	1	-1	TE#2
	-1	-1	1	1	-1	1	1	1	-1	-1	-1	TE#3
	1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	TE#4
	1	-1	0	1	1	1	1	1	-1	1	1	TE#5
	1	-1	-1	1	1	-1	1	1	1	1	-1	TE#6
	1	1	1	-1	-1	1	1	-1	-1	-1	-1	TE#7
	-1	1	1	-1	1	1	-1	1	-1	-1	-1	TE#8

In Gallant's example [Gallant,1993] the columns in the training example and dependency matrix correspond to cells which represent the following features:

Node Semantic association

Input nodes - Symptoms/Facts

- u_1 : Swollen feet
- u_2 : Red ears
- u_3 : Hair loss
- u_4 : Dizziness
- u_5 : Sensitive arethra
- u_6 : Placibin allergy

Hidden nodes - Diseases

- u_7 : Supercilliosis
- u_8 : Namastosis

Output nodes - Treatments

- u_9 : Placibin
- u_{10} : Biramibio
- u_{11} : Posiboost

The column labels u_1 to u_{11} represent each of the nodes (features) in the dependency network. Each of the rows in the dependency matrix describes the dependency list for one of the hidden nodes u_7 and u_8 and the output nodes u_9, u_{10} and u_{11} . The first row in the dependency network, for example can be interpreted as indicating that the value of node u_7 is dependent only on the values of nodes u_1, u_2 and u_3 .

This can be represented in rule (IF-THEN) form as:

IF	U1 and
	U2 and
	U3
THEN	U7

which represents the medical diagnosis:

IF	Swollen feet AND
	Red ears AND
	Hair loss
THEN	Supercilliosis

The first row in the training example matrix, for example can be interpreted as representing:

The *presence* of the symptoms swollen feet ($u_1=1$), red ears ($u_2=1$) and hair loss ($u_3=1$) and the *absence* of dizziness ($u_4=-1$) and a placibin allergy ($u_6=-1$) indicates that the patient does not have the disease Namastosis ($u_8=-1$), but is suffering from Supercilliosis ($u_7=1$), and should be treated with Placibin ($u_9=1$) and Posiboost ($u_{11}=1$) and not with Biramibio ($u_{10}=-1$).

Gallant [1985] uses a *learning matrix* to store the connection weights generated automatically from training examples: He defines [Gallant, 1985]:

"a Learning Matrix, L, for a given set of Variables and Dependencies as a matrix of integers having one row for each non-Terminal variable and one column for each Variable, plus an additional (0th) column for constant terms. (see figure 3). Each row of L is simply the Linear Discriminant for that corresponding non-Terminal Variable. Thus row L_i determines the value of for its Intermediate or Goal Variable X according to the rule

$$X = \begin{cases} +1 \\ -1 \\ 0 \end{cases} \text{ if } L_i \bullet V = \sum_{j=0}^n L_{ij} * V_j \begin{cases} > 0 \\ < 0 \\ = 0 \end{cases}$$

where: X is the value of the hidden (intermediate) or output (goal) variable associated with the i th row

V is a vector of activation values corresponding to the elements in the i th row of matrix L

V_0 is defined to be +1 and

$L_{ij} \neq 0$ only if V_i depends upon V_j

(Figure 3 from Gallant [1985] is reproduced overleaf as figure 4.4)

Figure 4.4 : Example Learning Matrix

	C	T1	T2	T3	T4	T5	T6	I1	I2	G1
I1	-1	3	-3	3						
I2	1			3	3	3				
G1	-2						-4	2	2	
G2	-2							2	2	-4

Learning Matrix L for Plethoral Disease example. All other entries are 0.

As later sections describe, the learning matrix structure also supports the efficient re-evaluation (and hence propagation) of network cell activations. Given that the activation values of each of the inputs are set; the value of the first hidden or output value - associated with the first row in the matrix can be computed by taking the dot product of the first row from the learning weight matrix L with the activation vector V . Provided that the activation of the hidden or output variable is updated in the activation vector V , this process can be repeated for each of the rows of the learning vector in turn. This is a summary of the implementation of the forward chaining phase of the Matrix Controlled Inference Engine.

4.2.4 Cell Properties and Network Structure

Gallant's perceptron-based feedforward neural knowledgebase architecture is similar to that used in the PROSPECTOR expert system developed by Duda and Reboh [1983]. Specifically the knowledgebase of both systems is an *inference network* - a tree in which each of the nodes corresponds to an assertion.

For simplicity Gallant makes the *hierarchical assumption* that the network contains no directed cycles, and hence is a *feedforward network*. This is implemented in

practice by the network generation algorithm, which prevents cycles by eliminating selected arcs and by letting the training examples implicitly specify any mutual dependencies or through the use of choice variables (winner-take-all groups).

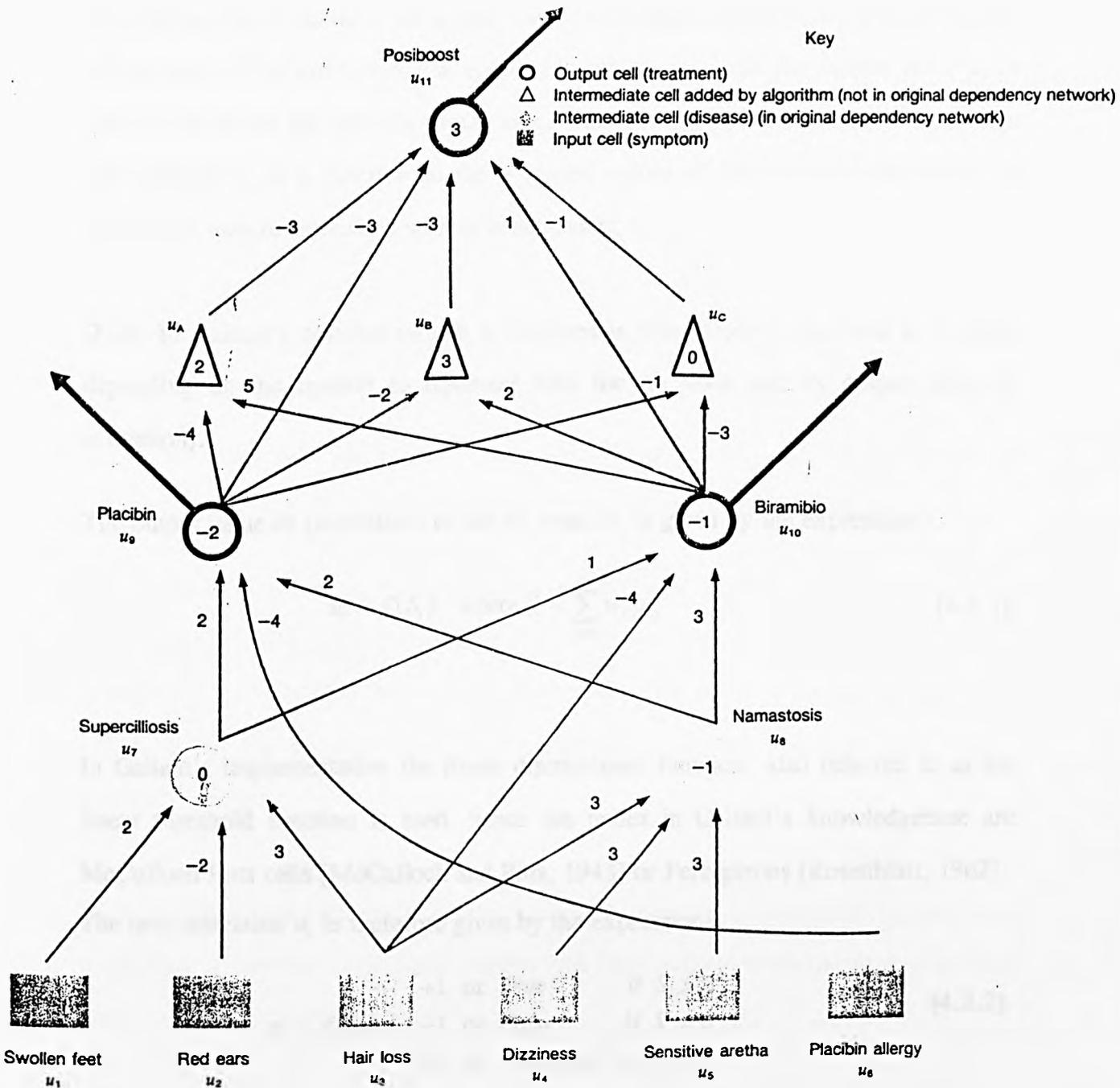
The structure of an example knowledgebase network, reproduced from Gallant [1988], is illustrated overleaf in figure 4.5.

Each of the input, hidden and output nodes can have the value +1,-1 or 0. The value '+1' ('-1') indicates that the concept or feature associated with the node is *TRUE* (*FALSE*). For example, if the node represents a medical symptom a value of '+1' ('-1') indicates that the assertion that the symptom is present (not present) is *TRUE* (*FALSE*). A value of '0' indicates either that the concept or feature associated with the node is *Unknown* or that it has been declared by the user to be *Unavailable* for the whole of the duration of a particular run of the expert system.

Each of the nodes in the knowledgebase network is assigned a unique, contiguous numeric index value, starting with the first terminal variable node u which is labelled u_1 . As in a number of connectionist networks, each knowledgebase network contains an additional *bias* node u_0 that provides an input to all of the hidden and output nodes in the network. The value of the bias node is set permanently to +1. This enables the bias or threshold value of each of the nodes u_j to be assigned to the weight $w_{j,0}$. The bias node has no associated semantic meaning and is used to simplify the implementation. It is not normally displayed on network diagrams.

In principle (and by default where no dependency information is specified) each of the hidden and output nodes can only receive a weighted input from each of the nodes in the network with a numeric index lower than its own. This strategy ensures that the network structure is feedforward and contains no cycles.

Figure 4.5 : Example connectionist knowledgebase network



The value of each of the hidden and output node threshold weights is determined - along with all of the other weights in the network - by the *pocket algorithm* (Gallant, 1985, 1988, 1993]in response to training examples and/or rules. Gallant's Pocket algorithm is described in section 4.2.6.

The activation of any node u_i is determined in the same way as in Multilayer Perceptron (MLP) models, using only local information. Specifically it is a function of the sum of the product of the connection weight $w_{j,i}$ and the output value u_j of each of the nodes that provide inputs to u_i . Another way of stating this, is to say that the value of u_i is a function of the weighted values of the variables on which the concept or feature associated with u_i is dependent.

(N.B. In Gallant's notation (which is followed in this chapter), the label u_i is used, depending on the context to represent both the i th node and its output value or activation).

The output value or (activation) of the i th node, u_i is given by the expression:

$$u_i = f(S_i) \quad \text{where } S_i = \sum_{j \geq 0} w_{i,j} u_j \quad [4.2.1]$$

In Gallant's implementation the linear discriminant function, also referred to as the linear threshold function is used, hence the nodes in Gallant's knowledgebase are McCulloch Pitts cells [McCulloch and Pitts, 1943] or Perceptrons [Rosenblatt, 1962].

The new activation u'_i is therefore given by the expression:

$$u'_i = f(S_i) = \begin{cases} +1 \text{ or } True & \text{if } S_i > 0 \\ -1 \text{ or } False & \text{if } S_i < 0 \\ 0 \text{ or } Unknown & \text{if } S_i = 0 \end{cases} \quad [4.2.2]$$

The dynamic properties of the knowledgebase network are succinctly described by Gallant [1988]:

"An iteration of the network consists of re-evaluating the each cell in index order and changing its activation before the next cell is re-evaluated. One iteration suffices to

bring the network to steady state because of the lack of directed cycles and because of our indexing scheme.

We call this network model a *linear discriminant network*. It is one of the simplest possible connectionist models, since there is no feedback and all computations are performed using integer arithmetic."

4.2.5 Network Generation

The first step in Gallant's approach [Gallant, 1993] to the development of a (decision support) connectionist expert system is for a domain expert to identify the significant features that influence the outcome of the decision. Gallant cites as an example the problem of determining whether to approve or reject a particular credit card purchase, given available data including the cost of the item, the type of purchase, the customer's past history of charges and payments, current credit status etc.

The identification of an appropriate set of features requires both creativity and knowledge of the specific problem domain. However determining which of a set of features of the available data to consider would seem to be more appropriate for a human expert than the production of IF-THEN rules and associated confidence factors [Gallant, 1993].

Once the significant features have been identified, training examples can be generated by extracting the values that these features had from historic customer data describing 'good' and 'bad' purchase decisions.

The next step requires the specification of names for the input, hidden and output cells corresponding to each of the features (or *variables of interest*). The name is a semantic label representing the feature associated with the cell. Each of the input cells also has a question associated with it, which may be used by the (MACIE) inference

engine during backward chaining to elicit a value from the user for the variable associated with the cell.

The first step in generating the network is to create a cell for each of the features and possible outcomes (decisions). Network generation and learning can be simplified through the use of dependency information acquired from a domain expert. As stated previously, for each of the hidden (intermediate) and output (goal) nodes, a dependency list of nodes can be specified, which contains only those nodes that are sufficient to compute the value of the particular node.

The dependency information for each of the nodes u_i specifies a *dependency network* in which for every node u_j in the dependency list of u_i there exists a directed arc from node u_j to u_i . The dependency information is however optional because by default every output cell may be specified as dependent on every input cell as in figure 4.2. However, if more precise dependency information is available then dependency lists improve network generation algorithms since accidental correlations between unrelated variables are prevented from influencing the final network weights [Gallant, 1993].

Given that the network generation and learning problem is an *easy learning problem*, that is, an example of a set of [Gallant,1993] :

"problems where we are given a network, and where the training examples specify input values and correct activations for all output and *intermediate* cells. What makes these problems "easy" is that we can decompose them into single-cell problems"

An initial network can be created by connecting the nodes using weighted directed arcs to reflect the dependencies specified by the domain expert, or in the absence of dependency information by connecting the nodes to create a default dependency network.

The next step involves attempting to separately and independently train each of the nodes in the network using the training examples (and rules).

If the single cell relationships are all separable then *perceptron learning* [Rosenblatt, 1961] or the *pocket algorithm* [Gallant, 1985,1988,1993] can be used to find a set of weights for each of the intermediate and output cells such that all of the relationships are learnt [Gallant, 1993].

However a particular cell in the dependency network may not be capable of perfectly modelling a given set of internally consistent training examples, because not all Boolean functions (for example XOR) are separable. If some or all of the single cell learning problems prove to be non-separable, that is, if no set of weights can be found by the pocket algorithm that satisfies all of the training examples, it may be sufficient to use the *pocket algorithm with ratchet* to find a set of weights that satisfies as many of the training examples as possible.

If a closer fit is required the single-cell learning problem can be treated as an *expandable network problem* and additional intermediate cells can be added using Gallant's *distributed method* [Gallant, 1993] or any one of a number of *constructive* algorithms, including for example the *Tower algorithm* [Gallant, 1986] to form a final network that is capable of modelling any self-consistent set of training examples [Gallant, 1988].

Gallant [1986,1993] demonstrates that:

"with arbitrarily high probability, the tower algorithm will fit non contradictory sets of training examples with input values restricted to $\{+1,-1\}$, provided that enough cells are added and enough iterations are taken for each added cell. Furthermore each added cell will correctly classify a greater number of training examples than any prior cell."

However, Gallant [1993] states that if a dependency network is given and the intermediate cell activations are unknown then there is no alternative to using a fixed network learning algorithm, such as backpropagation.

Once the network has been generated and the connection weights 'learnt' using the *pocket learning algorithm* or the *pocket learning algorithm with ratchet*, the network is ready to act as an expert system knowledgebase supporting inferencing by the Matrix Controlled Inference Engine (MACIE), the first (forward chaining) stage of which was described briefly above.

Gallant's *pocket learning algorithm* and the refinement which results in the *pocket learning algorithm with ratchet* are described in the section which immediately follows.

4.2.6 Network Training and the Pocket Learning Algorithm

Gallant's pocket algorithm [1985, 1988a, 1993] is a procedure that generates connection weights for discrete networks from training examples, rules or from a combination of examples and rules. It is a single cell supervised learning procedure that can find an optimal set of weights when classifying separable data and a good set of weights for non separable data.

It is suitable for 'easy learning' problems' where the training examples specify values for the hidden and output cells, since this enables the problem to be decomposed into a set of independent single cell problems. Each cell can then be considered separately and a set of weights for the connections to the cell generated using the sets of example training inputs and associated desired activations. However the (Boolean) function that

a particular cell may be required to learn will not necessarily be separable.

Gallant's pocket learning algorithm is based on Rosenblatt's [1961] perceptron learning algorithm, but incorporates an important modification that ensures that unlike the former, the pocket algorithm when attempting to classify non separable data is not 'poorly behaved'.

The term 'poorly behaved' is attributed to the perceptron learning algorithm (described in figure 4.6) because the algorithm may reject an optimal set of weights, which correctly classify as many training examples as possible, and replace them with the worst possible set of weights, which misclassifies all examples, in a single step. This means that even after a large number of iterations there is no guarantee as to the quality of the weights produced by perceptron learning algorithm.

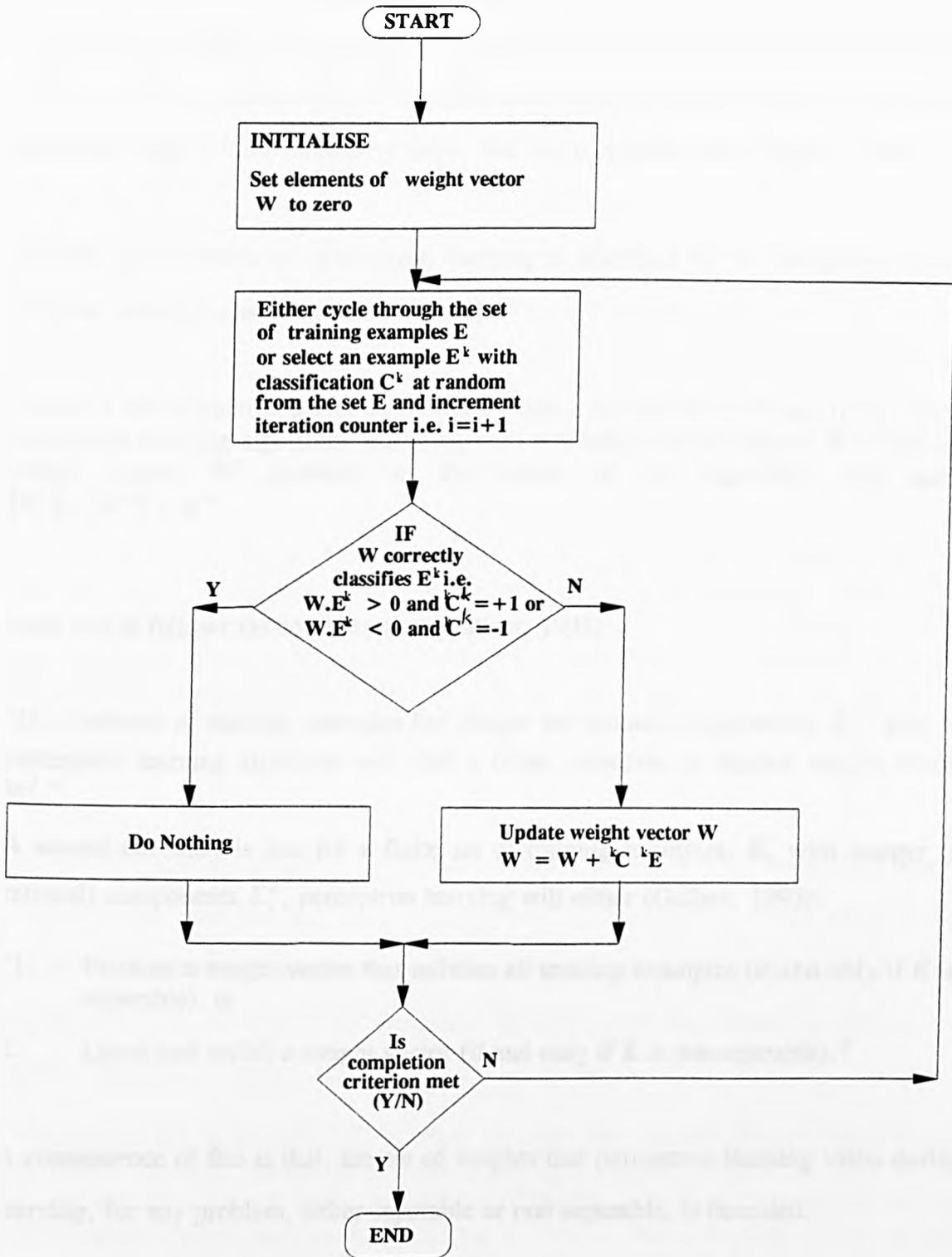
Although perceptron learning is poorly behaved when classifying non separable data, it nevertheless has a number of useful properties. The *perceptron convergence theorem* may be stated as [Gallant, 1993] :

"If E is a (possibly infinite) set of training examples, each of which has a length of at most K . If there exists a set of weights, W^* , and a number $\delta > 0$ such that $W^* \cdot E^k \geq \delta$ for all $E^k \in E$ then the perceptron learning algorithm will execute the change step $W = W + C^k E^k$ at most $\left(\frac{K \|W^*\|}{\delta} \right)^2$ times."

and as an immediate corollary [Gallant, 1993]:

"For a finite set of separable training examples, E , the perceptron learning algorithm will produce a set of weights, W that satisfies $W \cdot E^k > 0$ for all $E^k \in E$ after a finite number of change steps."

Figure 4.6 The Perceptron Learning Algorithm



N.B Various completion criterion are possible, including continuing until all examples are correctly classified (assumes separability) and/or until a number of iterations have been performed and/or each example has been used a minimum number of times etc.

The practical consequence of the property described by these theorems is that if any set of weights exists that correctly classify a finite set of training examples, that is, if the data is separable, the perceptron learning algorithm will always succeed in finding this (or possibly another set) of weights that correctly classifies all of the training examples, after a finite number of steps, and hence in finite time (Gallant, 1993).

Another characteristic of perceptron learning is described by the *perceptron cycling theorem*, which states that [Gallant, 1993]

"Given a set of training examples E , there exists a number M such that if we run the perceptron learning algorithm beginning with any initial set of weights W^0 , then any weight vector W^t produced in the course of the algorithm will satisfy $\|W^t\| \leq \|W^0\| + M$ "

from which follows the corollary that [Gallant, 1993]

"If a finite set of training examples has integer (or rational) components E_j^k , then the perceptron learning algorithm will visit a finite collection of distinct weight vectors W^t ."

A second corollary is that for a finite set of training examples, E , with integer (or rational) components E_j^k , perceptron learning will either (Gallant, 1993):

1. Produce a weight vector that satisfies all training examples (if and only if E is separable), or
2. Leave and revisit a weight vector (if and only if E is nonseparable)."

A consequence of this is that, the set of weights that perceptron learning visits during learning, for any problem, either separable or non separable, is bounded.

Gallant correctly identified that the characteristic 'poor behaviour' of the perceptron learning algorithm resulted from the fact that the algorithm only utilises negative reinforcement, and completely ignores examples that are correctly classified.

Gallant's modification to the perceptron learning algorithm eliminates the characteristic 'poor behaviour' by taking account of the number of correct classifications a particular set of weights has achieved. Gallant [1993] describes how this is achieved:

"The pocket learning algorithm takes correct classifications into account by keeping a separate set of weights, w^{Pocket} , "in your pocket" along with the number of consecutive iterations for which w^{Pocket} correctly classified the chosen training example. Now whenever the current perceptron weights, W have a longer run of correct classifications, we replace the pocket weights w^{Pocket} by W ."

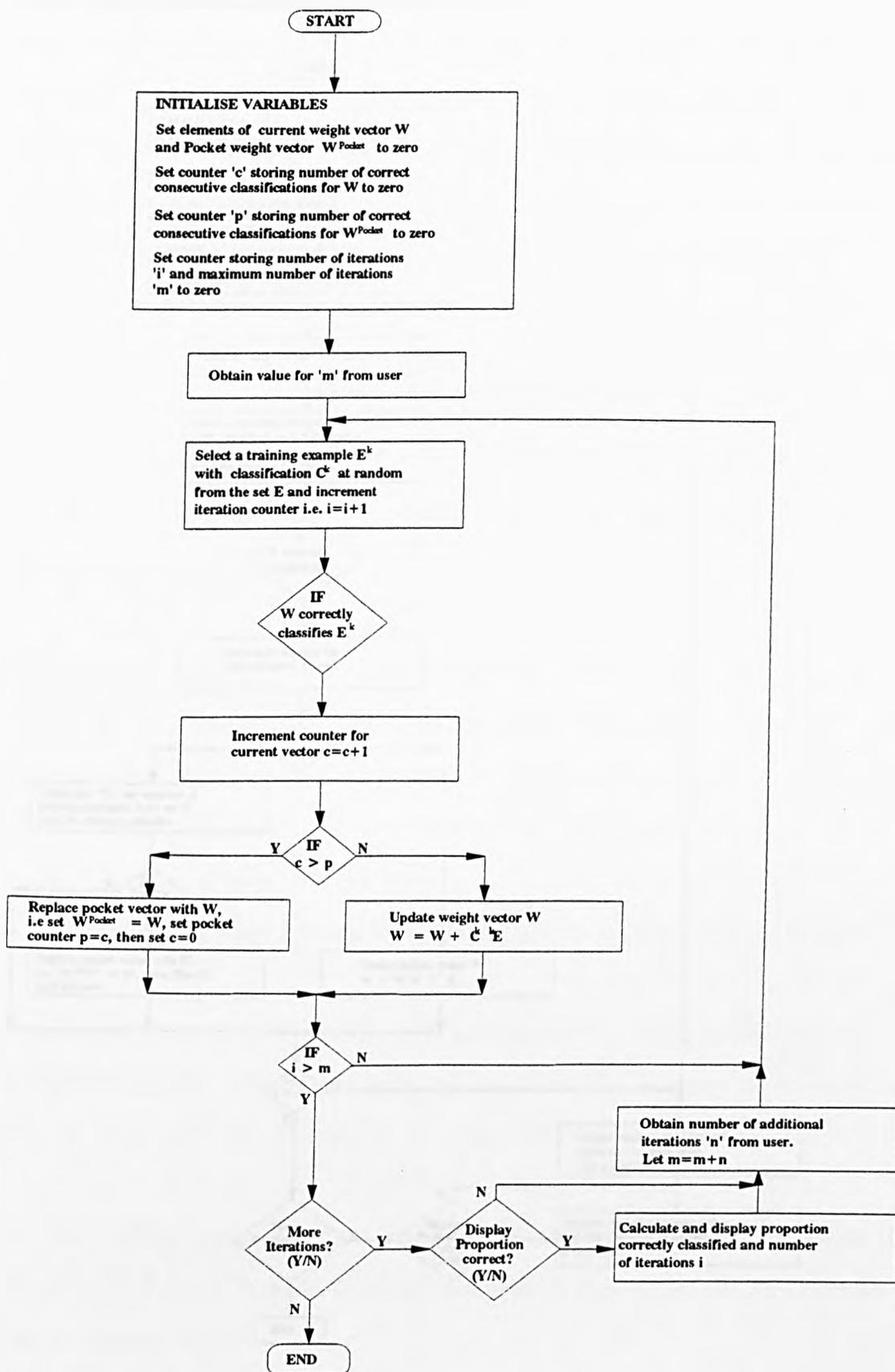
However, as Gallant states [Gallant, 1993]:

"When running the pocket algorithm there is nothing to prevent a bad set of weights from having a "lucky run" of correct responses and then replacing a good set of weights currently in the pocket. Even worse, if the lucky run is a long one then the bad weights can hang around in the pocket for a large number of iterations. We know that the probability of this happening becomes arbitrarily low as the number of iterations increases, but in practice it is common for such temporary retrograde steps to occur."

The pocket learning algorithm is described overleaf in figure 4.7. Gallant was able to overcome this problem, for finite training sets by introducing a refinement to his algorithm, which he called the *pocket algorithm with ratchet*. As with his improvement to the perceptron learning algorithm, the refinement is simple yet effective [Gallant, 1993]:

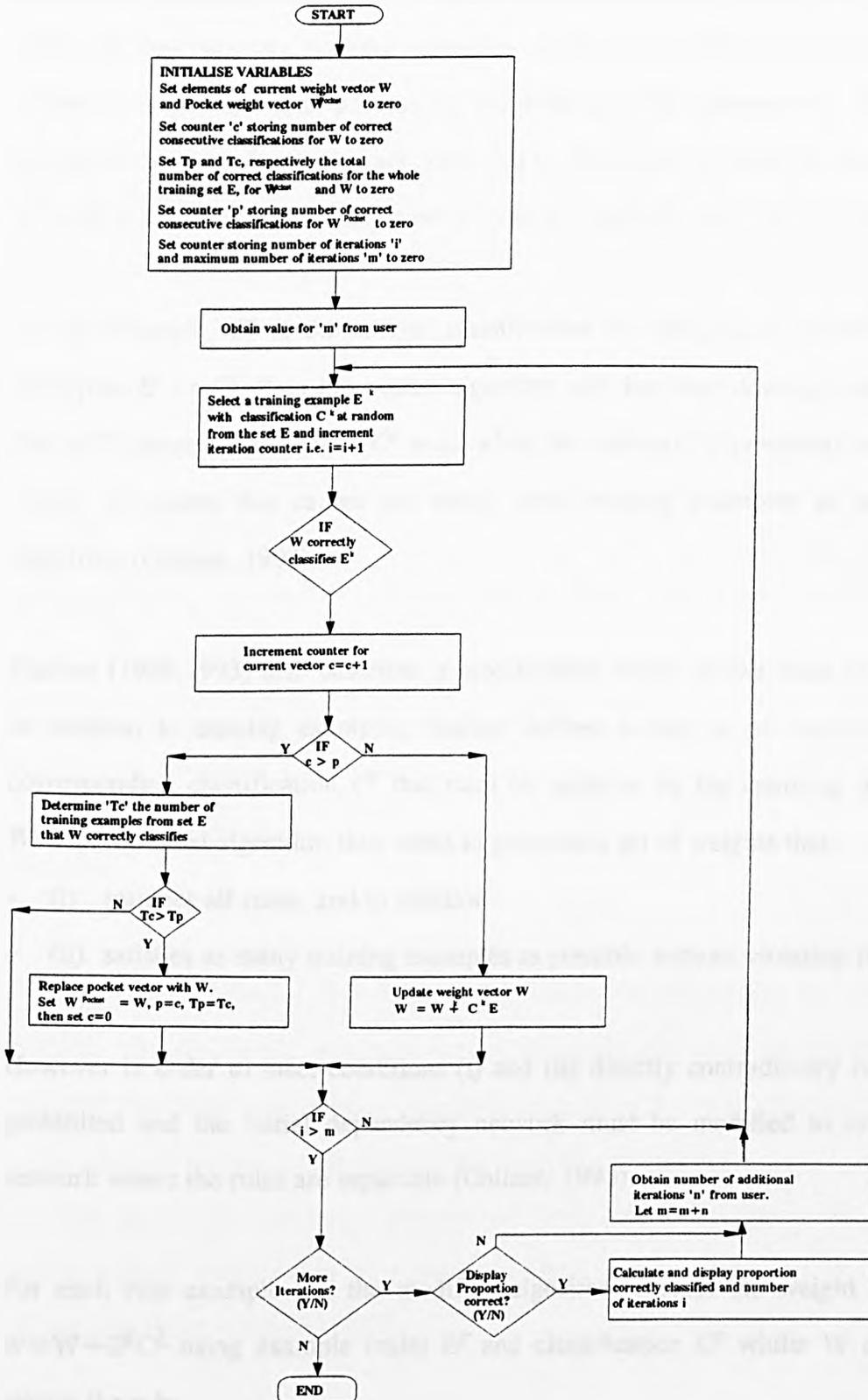
"Whenever a set of weights, W , is about to enter the pocket we can check to see whether W actually classifies more training examples than the current set, w^{Pocket} . If not, we do not allow W into the pocket. (Thus the quality of the pocket weights ratchets up and never decreases). Of course if there are many training examples then such a ratchet check is expensive, but as the run length of correct classifications for the pocket weights increases, the time between such checks goes up exponentially. This makes ratchets a big winner in practice, so for finite sets of training examples the ratchet version of the pocket algorithm should be used."

Figure 4.7: The pocket learning algorithm



The pocket algorithm with ratchet is described in figure 4.8

Figure 4.8 : Pocket Algorithm with ratchet



A feature of the pocket algorithm is that it converges to a set of weights that correctly classifies a randomly chosen training example with maximum likelihood. This means that the pocket algorithm is a reasonable candidate to apply to the classification of noisy or contradictory training examples. A set of training examples is said to be noisy if a probabilistic process is involved in their generation. Noisy training examples are sometimes contradictory, that is, there may be training examples E^r and E^s with corresponding classifications C^r and C^s where $E^r = E^s$ but $C^r \neq C^s$.

If, for example, C^r is the correct classification for the greater number of training examples $E^r (=E^s)$ then the pocket algorithm will however develop a set of weights that will generate the output C^r even when the network is presented with the input vector E^s unless this causes too many other training examples to be incorrectly classified (Gallant, 1993).

Gallant [1988,1993] also describes a modification which allows rules to be specified in addition to training examples. Gallant defines a *rule* as an example E^r with a corresponding classification C^r that *must* be satisfied by the resulting weight vector W . The modified algorithm thus seeks to generate a set of weights that:

- (i) satisfies all rules, and in addition
- (ii) satisfies as many training examples as possible without violating (i) above.

However in order to meet conditions (i) and (ii) directly contradictory rules must be prohibited and the initial dependency network must be modified to create a final network where the rules are separable (Gallant, 1993).

For each rule example E^r , the modified algorithm repeats the weight update step $W = W + E^k C^k$ using example (rule) E^r and classification C^r whilst W continues to violate the rule.

Gallant [1993] also modified the basic algorithm to enable it to learn to classify weights for cells in a winner-takes-all group.

The convergent property of the pocket algorithm has been formally described by Gallant[1993] in the *pocket convergence theorem*, which states that:

"Given a finite set of input vectors $\{E^k\}$ and corresponding desired responses $\{C^k\}$ and a probability $p < 1$, there exists an N such that after $n \geq N$ iterations of the Pocket Algorithm, the probability that the pocket coefficients are optimal exceeds p ."

The practical consequence of which, is that [Gallant,1993] :

"If the problem is separable, the pocket algorithm will produce an optimal set of coefficients by the perceptron convergence theorem. Otherwise, perceptron learning will go from coefficient set to coefficient set, but eventually one of the repeatedly visited optimal sets will have a longer run of correct responses than any other particular non optimal set with arbitrarily high probability. Since there are only a finite number of non optimal sets of coefficients, as the number of iterations grows an optimal set will, with probability 1, have the longest run of correct responses.

It is important to note that for even medium sized problems we are not likely to compute an *optimal* set of coefficients after any reasonable number of iterations. Thus the convergence to optimality is in theory only, and the production of optimal weights is not to be depended upon for actual problems. Nevertheless, experience has shown that the weights are very likely to be *good* weights that approach an optimal number of correct classifications. This makes the algorithm a good one for practical problems."

Gallant's model includes therefore, an algorithm which enables the cells in 'easy learning' networks to independently automatically learn to compute separable and non separable functions from training examples, or rules or a combination of both.

In the case of separable functions these weights should be optimal, and for non-separable functions, even in the presence of noisy and contradictory data it is able to generate good weights that can correctly classify more examples than a standard statistical package (Gallant, 1993).

4.2.7 The MACIE inference engine

The objective of the MACIE inference engine is to infer one or more conclusions given that a set of inputs clamps the values of some or all of the networks input nodes.

The first stage in the operation of MACIE is the *initialisation* phase. This is where the system uses the question strings associated with each node (see figure 4.1) to prompt the user to initialise the input variables from their default value of *unknown* (also referred to as *not-yet-known*) (0) to the value of *true* (+1) or *false* (-1).

As Gallant states [1988]:

"Initialization is important since it focuses the subsequent problem solving. By contrast, systems based on decision trees cannot take full advantage of initial information because such information does not change the order in which the nodes are examined."

During the backward chaining phase (described below) the system may ask the user to supply the value of a node whose value is currently *not-yet-known* (or *unknown*). The user can respond with *true* (+1), *false* (-1) or *unavailable* (0) (also referred to in [Gallant,1988] as *unobtainable*). This means that the value of this variable, and hence the activation of the associated node, will not be available at any time during the current session.

Nodes that are *unavailable* are taken by the inferencing system to be known but with an activation of zero. The set of nodes whose values are known '*{known}*' at any point in time is therefore comprised of all of the nodes in the network that have the value *true*, *false* or *unavailable*. The nodes whose values are *not-yet-known*, which initially includes all terminal nodes not set by the user and all non-terminal nodes, form the complimentary subset *{unknown}*.

Clearly at every point in time each node must be a member of only one of these mutually exclusive sets - although nodes can cease membership of one set and become members of the other as the result of inferences and as new information is entered by the user.

The second stage in the inference process is called the *forward chaining* phase. This is where the value of each of the non-terminal nodes is determined in (label order) sequence and in so doing the terminal (input) values are propagated up through the nodes in the hierarchical network.

As stated previously, each of the non-terminal nodes in the network can have one of three values, true (+1), false (-1) or unknown (0). The first step in determining whether a new value for a non-terminal node can be inferred from its inputs, involves the calculation of the current total weighted input of the node u_i , $KNOWN_i$ using the expression:

$$KNOWN_i = \sum_{j=0, u_j \in \{known\}}^{j < i} w_{i,j} u_j \quad [4.2.3]$$

where : u_j is the value (activation) of the j th node $\{+1, -1, 0\}$
 $\{known\}$ is the subset of all nodes whose value is currently known, that is those that have the value true, false or unavailable
 $w_{i,j}$ is the weight on the connection from node j to node i

The new value (activation) of the i th node can be determined (or inferred) using the sum of the weighted inputs to the node, $KNOWN_i$, according to the expression:

$$\begin{array}{lll} \text{True (+1)} & \text{if} & KNOWN_i > 0 \\ \text{False (-1)} & \text{if} & KNOWN_i < 0 \\ \text{Unknown (0)} & \text{if} & KNOWN_i = 0 \end{array} \quad [4.2.4]$$

It is interesting to note that equations 4.2.3 and 4.2.4 are the same as those that describe the behaviour of McCulloch-Pitts cells [McCulloch and Pitts, 1943] and Perceptrons [Rosenblatt, 1961]

However Gallant introduced an additional qualification to the perceptron update rule, and it only after this condition is satisfied that MACIE enables the node to change its state (that is, make an inference based on its inputs).

Before an inference can be reliably made, the maximum possible size of the weighted input to node u_i from the nodes whose values are currently *not-yet-known* must be determined. This value is called $MAX_UNKNOWN_i$, and is calculated according to the expression:

$$MAX_UNKNOWN_i = \sum_{j=0, u_j \in \{unknown\}}^{j < i} |w_{i,j}| \quad [4.2.5]$$

The value of $KNOWN_i$ gives the current weighted input to the i th node whilst the value of $UNKNOWN_i$ is the largest amount by which this value might change, in either direction, based on any possible combination of the currently unknown information.

Now if the magnitude of the known weighted input ($KNOWN_i$) is greater than the largest amount by which this value can change ($MAX_UNKNOWN_i$), then the sign of the known value cannot change.

Therefore, MACIE enables a node to change its state according to equation 4.2.4 provided that:

$$| KNOWN_i | > MAX_UNKNOWN_i \quad [4.2.6]$$

The semantic interpretation is that the node is allowed to make the *conservative* inference (one whose result cannot be changed by additional information) provided

that the partial evidence for the inference is sufficient to ensure that the result cannot be changed by the introduction of subsequent information.

Gallant [1988] illustrates this process using the sarcophagal diagnosis example described in figure 4.5 and section 4.2.3 (on page 111).

"if the patient has swollen feet ($u_1=+1$), but neither red ears ($u_2=-1$) nor hair loss ($u_3=-1$) then we can conclude that supercilliosis is present ($u_7=+1$) because

$$0 + (2)(1) + (-2)(-1) + (3)(-1) > 0"$$

Where the first term '0' is the threshold of u_7 , and the second, third and fourth terms are respectively the connection weight to node u_7 and the value of nodes u_1 , u_2 and u_3 .

MACIE allows each of the non-terminal in the network, in index order, to determine whether an inference can be made, and where it can, to set its value according to the result of the inference. In this way the values of the terminal (input) nodes can propagate up through the network and influence the values of the goal (output) nodes.

At any time during the run an estimate can be made of the likelihood that an unknown variable u_i will eventually be deduced to be *true* or *false*. Gallant [1988] warns that this value, called $\text{Conf}(u_i)$:

"is useful for comparing two unknown values (but cannot be interpreted as the probability that u_i will eventually be found to be true)."

There are a number of heuristics that can be used to calculate $\text{Conf}(u_i)$. One of the simplest is:

- For a node with a known value $\text{Conf}(u_i) = u_i$ [4.2.7]
- For an unknown input node $\text{Conf}(u_i) = 0$ [4.2.8]

For other unknown nodes, $\text{Conf}(u_i)$ is calculated in index order according to the equation:

$$\text{Conf}(u_i) = \frac{\sum_{j=0}^{i-1} w_{i,j} \text{Conf}(u_j)}{\sum_{j:u_j \in \{\text{unknown}\}} |w_{i,j}|} \quad [4.2.9]$$

From which it is clear that: $-1 \leq \text{Conf}(u_j) \leq +1$ [4.2.10]

The relative likelihoods of all of the nodes in the network can be calculated in bottom-up pass through the network.

Now, if at the end of the forward chaining process there is insufficient information to infer values for one or all (if required) of the goal nodes, MACIE executes a *backward chaining* phase.

In the backward chaining phase, MACIE uses the questions associated with each of the input nodes (see figure 4.1) to elicit additional information from the user. One of the advantages of MACIE over traditional expert systems is that MACIE attempts to find the value of the unknown variable that is most likely to contribute toward the an inference being completed.

Gallant [1988] suggests a number of heuristics for selecting an input cell with unknown activation, perhaps the simplest of which is:

"(1) Select the unknown output variable u_i such that $|\text{Conf}(u_i)|$ is maximum. (This strategy starts with an output cell close to having its value set.) We say u_i is being *pursued*.

(2) If pursuing cell u_i , find the unknown cell u_j with the greatest absolute influence on u_i . In other words find a j yielding

$$\max_j |w_{i,j}| : u_j \text{ unknown.}$$

If u_j is an input variable, ask the user for its value (employing the character string question for u_j in the knowledgebase). Otherwise pursue u_j and repeat (2).

Since we have been careful to prevent directed loops in the connectionist network, no variable can be pursued more than once without a question being asked. Therefore, this method of backward chaining quickly chooses an unknown variable to ask the user with no need for backtracking."

However Gallant states that [1988,1993] :

"For confidence estimates and for backward chaining, the choice of heuristics does not appear to affect the practical performance of the inference engine very much."

Once the user has entered the value of a previously unknown variable or declared an unknown variable to be unavailable then the forward chaining phase again attempts to infer a value (a conclusion) for one or more of the output nodes.

The process of alternately propagating the values of nodes through the network and asking questions to elicit the value of unknown variables, continues either until sufficient inferences have been made or until MACIE determines that no inferences are possible.

4.2.8 Rule Extraction

In Gallant's network knowledgebase, IF-THEN rules are not represented explicitly, rather they are generated by the inference engine as required. Gallant [1993] suggests a number of reasons for extracting rules from neural networks. These include supporting the generation of explanations for inferences and in so doing responding to the criticism that [Gallant, 1993]:

"neural networks are unable to explain their conclusions"

Rule extraction can also enable a human expert to examine a representation of the distributed information stored in the neural knowledgebase, in order to refine it or even learn from it. If the expert disagrees with a particular rule, he or she may be able to correct or refine the rule. This can then be used to prepare a new training example which is used to regenerate the knowledgebase and potentially improve the performance of the expert system.

An important goal of a rule-extraction algorithm could be to automatically generate a rule that supports a particular inference. Such a rule should be *valid*, that is, it must always hold regardless of the values of the variables included in the rule and of the value of any other variables; and should be *maximally general*, in the sense that unnecessary conditions (antecedents) are removed from the rule - until the removal of further conditions would render the rule invalid.

The basic idea underlying the methodology described by Gallant, and used in the MACIE inference justification algorithm, is that once the system has inferred that a particular node is true or false, a minimal subset of the currently known information (node values and associated weights) can be selected that is sufficient to support the inference.

To support this process, Gallant [1993,p318] defines a *contributing variable* as :

"one that does not move the weighted sum in the wrong direction. If we are trying to explain an inference C_i for cell u_i , then cell u_j , is contributing if

$$C_i w_{i,j} u_j \geq 0"$$

Whilst the *size of the contribution* is defined as $| w_{i,j} |$.

Given an inference, a rule which would provide the same inference is generated by: firstly, selecting all of the contributing variables supporting the inference. These can then be ranked in order of the *size of the contribution* that they make toward the inference, starting with the most influential first.

A rule is then formed by making the outcome of the inference the consequent of the rule, and by adding antecedents, created by forming clauses from the contributing variables, starting with the most influential, until there are sufficient clauses to support the inference. This approach (described overleaf in figure 4.10) ensures that the rules are both valid, since they are only generated to support a conservative MACIE inference, and maximally general, since they contain the minimum number of contributing variables required to support the inference.

4.2.9 Comments

Gallant [1986a,1986b,1986c,1987,1988a,1988b,1988c,1990a,1990b,1993] has made possibly one of the most sustained and significant contributions to connectionist expert system research, to-date. He presents a connectionist expert system model which has a discrete perceptron-based neural architecture as its knowledgebase and performs inferencing using a connectionist *Matrix Controlled Inference Engine* (MACIE). He also describes an approach based on his novel 'Pocket' algorithm which enables connectionist expert system networks to be generated from a set of training examples or rules and a method for the extraction of IF-THEN type rules can be extracted from the connectionist knowledgebase.

The connectionist expert system implemented in CONNEKT which assigns customers to vehicles is based on the perceptron-based connectionist expert system architecture described by Gallant. It is conjectured that an attempt to automatically generate the network based on the entities and constraints in a DCAVRP problem would make an interesting research topic.

Finally as has been previously stated the 'Pocket' algorithm developed by Gallant and applied by him to perceptron and back-propagation networks provided the basis for the development of a 'Pocket' algorithm for the Boltzmann machine, as described in section 6.5 of this thesis.

Figure 4.9 The MACIE inference justification algorithm

Given: Cell u_i inferred to have value $C^i = \pm 1$.

1. Set $CURRENT = w_{i,0}$ (the bias for u_i).
 Set $VARS_UNUSED = \{j | w_{i,j} \neq 0 \text{ and } j \neq 0\}$
 = set of nonbias cells connected to u_i that are not used in the rule.
 Set $UNKNOWN = \sum_{j \in VARS_UNUSED} |w_{i,j}|$
 = max value that $CURRENT$ can change by.
2. If $|CURRENT| > UNKNOWN$ then stop; clauses generated so far give a valid justification that is (usually) maximally general.
3. Find an input $j \in VARS_UNUSED$ such that
 $C^i w_{i,j} u_j \geq 0$
 and $|w_{i,j}|$ is maximized. (Ties are broken arbitrarily.)
4. Output a rule condition using cell u_j and its value (possibly "unavailable").
5. Set
 $CURRENT = CURRENT + w_{i,j} u_j$
 $UNKNOWN = UNKNOWN - |w_{i,j}|$
 $VARS_UNUSED = VARS_UNUSED - \{j\}$.
6. Go to step 2.

4.3 Review of Connectionist Expert System Research

The combination of connectionist and expert system techniques provides a significant and increasingly active area of research. In 1993 alone, in excess of hundred papers were published describing research in this and closely related fields. A review of all connectionist expert system research is beyond the scope of this thesis. The work selected for review in this thesis was selected either because it is significant and influential in its own right, or because it is representative of this large and growing body of work.

The first work that is examined is that of Samad [1988], who describes a novel connectionist architecture for implementing a rule-based system, called RUBICON, which uses both distributed and local representations. RUBICON has a conceptually simple connectionist architecture which uses layers of nodes to represent attributes, values, attribute-value pairs, antecedents, rules and consequents. Samad's architecture allows any rule to contain an arbitrary number of antecedent and consequent expressions, including negated expressions, provides support for the addition and removal of working memory elements, and can enable rules to be executed based on partial matches. Samad describes how these features distinguish RUBICON from previous connectionist attempts at developing rule-based systems [Samad, 1988]

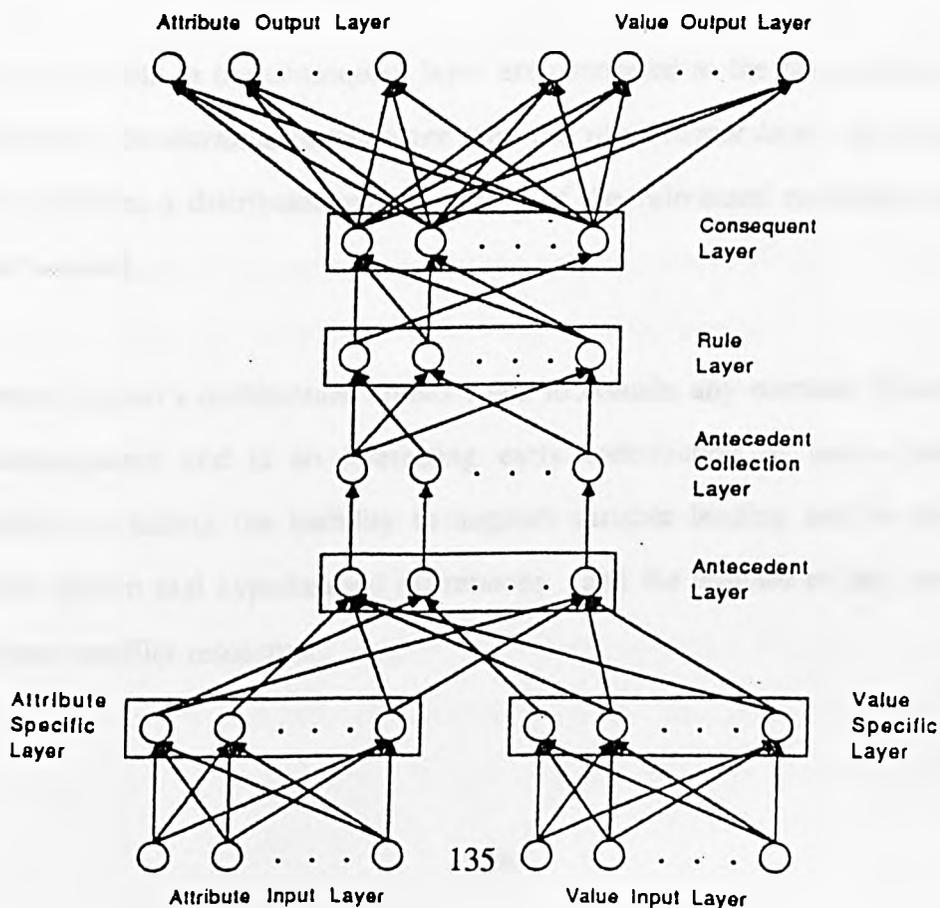
"the Distributed Connectionist Production System developed by Touretzky and Hinton (1986) is a complex system with many thousands of units arranged in five "spaces", and requires that all rules have exactly two expressions in the antecedent; PLATO/ARISTOTLE (Voevodsky,1987) cannot perform chained inferences, only allows one consequent expression per rule, and does not allow negated expressions; the "connectionist expert system" described in Gallant (1988) uses a connectionist knowledgebase but a separate inferencing procedure and has no concept of working memory"

However Samad [1988] acknowledges that:

"Each of these schemes also has some features that are lacking in RUBICON, such as variable binding and certainty measures"

RUBICON's architecture is multilayered (see Figure 4.11) and comprises two input layers, a single output layer and four internal intermediary layers. The units in both of the input layers and the output layer are split into two groups. In each case one of the groups represents *attributes* and the other *values*. Attributes and values appear at the first input layer as a distributed network of activation. The second input layer, which contains two winner-takes-all groups known as the *attribute specific layer* and the *value-specific layer* are comprised of nodes that respectively represent a unique attribute name and a unique attribute value. Connections from the attribute and value specific layers to the first internal layer, the *antecedent layer*, which is also a winner-takes-all group, allow any number of antecedent expressions to be formed, with each unit representing a unique attribute-value pair (appearing in the antecedent of any rule). The next layer is the (non winner-take-all) *antecedent collection layer* which can be thought of as the working memory of the model since each of its units becomes and remains active if the corresponding unit in the *antecedent layer* becomes active.

Figure 4.10: The topology of a RUBICON connectionist expert system network



An antecedent layer unit (and consequently a antecedent collection layer unit) will also become activated if the attribute value pair presented to the network is a sufficiently close partial match to the pair associated with the unit, to overcome some threshold value. The activations of units in the antecedent collection layer therefore represent the set of attribute value pairs or partially matched variants, that have been presented to the network as inputs

The next layer is the *rule* layer. This contains a group of refractory (once fired they cannot re-fire) winner-take-all units. Each rule unit has a positively weighted connection enabling it to receive excitatory signals from each of the antecedent units - that correspond to antecedent expressions in the rule's clause. Each rule unit in turn has a positive excitatory connection to units in the last of the internal layers, the *consequent* layer. This is also a group of refractory winner-takes-all units, which represent the set of consequents for the particular rule (unit). A small random component and the winner-take-all behaviour of the consequent enables each of the consequents of a rule to fire, one at time, in some undetermined order.

Finally the units in the consequent layer are connected to the two groups of output layer units , the *attribute output layer* and the *value output layer*, the activation of which provides a distributed representation of the rule-based reasoning performed by the network.

Although Samad's architecture allows rules to contain any number of antecedents and consequents and is an interesting early contribution, it has a number of limitations including the inability to support variable binding and to distinguish between known and hypothesised expressions, and the absence of any mechanism to support conflict resolution.

Oka [1991] proposes a two level hybrid cognitive model, called the *C/U model* which uses localised symbolic processing on the *conscious* level and distributed sub symbolic processing on the *unconscious* level. Knowledge in the symbolic or conscious level is separated into 'chunks' and a connection exists between each of the chunks in the conscious layer and a *corresponding node* in the distributed unconscious level. The unconscious level also contains other nodes that are not linked to chunks in the conscious level.

Both levels of the model execute in parallel. Efficient processing on the conscious level is facilitated by only allowing chunks which are activated by the unconscious level to be accessed. On the unconscious level parallel distributed computation is performed by the spreading of activations via weighted connections under the influence of signals, both from the external world and from the conscious level.

Oka briefly states that he has developed some example implementations of the *C/U model* and describes an implementation method using a parallel logic programming language.

Ray [1991] describes the development of a connectionist expert system for aircraft engine fault diagnosis developed in Foxbase on a PC. The network attempts to infer one of a finite set of causes (for an equipment fault) from a finite set of symptoms. The network has a two layer feedforward architecture, with each of the nodes in the input layer corresponding to one of a set of possible symptoms. Each of the nodes in the output layer corresponds to a member of the set of possible causes. There are no connections between units in the input (symptom) and output (cause) layers.

The network is trained using the pocket learning algorithm [Gallant, 1986] using training data comprised of engine case histories and consultation data. The consultation data is prepared by domain experts who rate the strength of the

relationship between possible causes and symptoms using a continuous scale in the range -1 (symptom absent) to +1 (symptom present). Once trained, the weight between a possible cause node and a symptom should reflect the 'causal strength' relating the components of that cause/symptom pair. Ray describes the degree that a symptom is subordinate to a particular cause as the *degree of membership*. This concept is the same as that employed in CONNEKT to describe the degree of membership of an instance to a collection/concept.

Ray's inference engine performs inference by first forward chaining using the symptoms entered by the user, and then backward chaining in the attempt to confirm or deny hypotheses (causes) by asking the user questions. This process continues until Gallant's [1986,1988] satisfiability criterion are met, that is until a conclusion is reached or the system becomes satisfied that no conclusion can be reached.

Ray describes a number of problems with his system. These include:

- the system may generate too many different causes for some symptoms.
- no distinction of confidence or 'believability' is drawn between inferences that identify a cause that has a grade of membership that is significantly greater than those associated with any other cause and an inference in which there are a number of leading contenders with only slightly different grades of membership.
- the system learns and considers only *shallow* knowledge obtained from developing a representation of the relationships in the (cause and symptom) data as opposed to *deep* model-based knowledge.
- the system cannot interact with graphics software, hence a graphical approach cannot be used to represent the state of the network during consultation

- The network is a feedforward network with two layers and no hidden units

Ray reports that his system has been tested with various field applications and that domain experts are able to confirm approximately 75% of the system's diagnoses.

A two layer feedforward neural architecture and a diagnosis reasoning procedure based on the shunting competitive equation of Grossberg [1988] is proposed by Wang and Ayeb [1992] for the computation of cause to effect diagnoses. In their proposed architecture, each of the nodes in one of the layers represents a cause and each of the nodes in the other, an effect. The activity of the node represents the degree to which the cause or effect associated with the node is present. Interconnections are allowed between the cause nodes if they have at least one common effect.

The proposed diagnosis reasoning procedure has a number of operational stages:

- 1 Initialise the activities of each of the effect nodes,
- 2 Initialise the activities of each of the cause nodes.
- 3 Update the activities of each of the cause nodes to take account of inhibitory influences from the other cause nodes.
- 4 Firstly update the weights between the nodes in the cause and effect layers to reflect the current variations of the network's belief in the plausibilities of the causes, and secondly modify each of the weights contributing to the inhibitory influence between competing causes, in proportion to the variations in activity of the two cause nodes.
- 5 Prune (eliminate) the influence of effects on causes where the connection weight is below a threshold value (by setting the weight to zero).
- 6 Terminate the reasoning process if each of the remaining causes is exclusive, otherwise repeat steps 3 to 6

The key steps in Wang and Ayeb's procedure are Stages 4 and 5 since these apply a form of winner-takes-all competition between nodes and eliminate non-essential causes which have achieved only limited support.

The authors suggest that areas suitable for additional investigation include: analysis of the neural dynamics of the proposed network and the determination of a suitable level of information abstraction.

4.4 Related Research

Cohen and Hudson [1990] describe a neural network learning algorithm which they apply to a three layer feedforward network in which connections are allowed between *interaction nodes* in the hidden layer. The network and algorithm are implemented in an automated medical decision making aid which is applied to the problem of determining the prognosis of patients with melanoma. Their model was able to determine that of 109 parameters that were considered relevant, only 7 contributed to the prognosis. When only these seven variables were used the system was able to correctly determine the prognosis of patients in 79% of cases, which compares favourably with previous statistical methods, which yielded approximately 65% accuracy.

Cohen and Hudson suggest that expert supplied rules could be combined with the neural network algorithm to take advantage of all possible sources of information and propose an interface that could directly elicit values for the seven variables by producing an interaction (that is, presenting questions) in the form of an expert system.

Schreinemakers and Touretzky [1988] present ELSIE a Common Lisp based system that is comprised of an OPS-5 based expert system module, a neural network simulation module and a knowledge acquisition module.

Their system is applied to the problem of the detection of udder infection in cow's, using data describing a series of measurements of milk production and counts of the number of white blood cells per volume of milk, for each cow.

The diagnosis proper is performed by a neural network which has 36 input nodes, 6 hidden nodes and 3 output nodes. The network is trained using 18 milk production values and 18 cell count values from 116 cows, which were classified as either 'healthy', 'subclinically sick' or 'clinically sick' by a domain expert based only the input data.

The diagnosis process is managed by the expert system module which invokes the neural network to perform the diagnosis for a particular cow. A construct called DEFNETWORK is used to:

- allocate a name to the neural network, (which is used by the expert system to support invocation),
- to describe the names of files that contain network description details and the values of connection weights
- define the name of an associated data structure called a WME (described below)
- Define a set of fields including the number and type (input or output) of fields used by the neural network.

Data is passed between modules in ELSIE using named working memory elements or WME's (pronounced 'woomies') which correspond to a record type in Pascal or C. The format of each WME is tailored to satisfy its particular interface role.

The expert system manages the process of populating the fields of the WME with the milk production and cell count readings for a particular cow, and the invocation of the neural network to perform the diagnosis.

The rule-based knowledge manager uses a set of OPS-5 rules that examine the neural networks diagnosis - contained in the output fields of the WME - to detect improperly classified cows. This can occur if the network is unable to form a diagnosis for (classify) a particular cow, in which case the data is examined by an expert veterinarian who provides a diagnosis. A new training example is then generated by combining the domain expert's diagnosis with the data from the cow descriptor WME. New training examples are used by the knowledge manager to retrain the neural network, with a resulting incremental expansion of the knowledge available for capture by the network.

Their system initially generated the same classification as the domain expert in 87% of the cases. However this figure was subsequently revised up to 98% after the expert veterinarian revised his diagnosis for a number of disputed cases.

Schreinemakers and Touretzky [1988] state that they intend to confirm that the network generalises correctly by expanding the number of cows for which data is recorded to allow separate training and test sets to be formed, and to extend the network to selectively take account of other factors considered significant in diagnosis by veterinary experts.

Hillman [1991] describes an architecture in which knowledge-based systems are used to pre and post process data for a series of cascading neural networks. In Hillman's architecture, as in traditional systems analysis and knowledge engineering, the application domain is decomposed into separate chunks or areas of knowledge that represent the problem and solution set. These chunks of knowledge are represented by neural networks which are then integrated to represent the larger domain of knowledge.

Hillman describes the principal advantages of the cascading network over the single network approach as :

- conforming to the traditional problem solving approach by decomposing larger problems into smaller, more manageable problems.
- enabling the developer to more easily comprehensively test the behaviour of the system - since each of the (relatively small) component networks can be isolated and tested separately.
- Reducing the difficulty in developing neural training sets by focusing on specific problems
- Provides for ease of maintenance by supporting the modular replacement of knowledge.

Liang and Jin [1993] describe a 'handcrafted' cellular neural architecture which successfully solves an example of the Missionaries and Cannibals Problem (MCP). The problem is decomposed into a number of stages or sub-goals. A set of neural cells, comprised of *state* neurons, which describe the status of the sub-goal and *constraint* neurons which are used to encode the constraints that each of the sub-goals must satisfy if the problem is to be solved, are designed to satisfy each of the sub-goals - given the set of applicable constraints. The neural cells are then combined using only connections between immediately neighbouring stages (cell groups)

A solution is found by clamping the states of the neurons in the initial and final stages to values corresponding to the initial and goal states of the problem. A

parallel stochastic update method is used to enable the activations to propagate through the network (from both ends) and to avoid local minima. They prove that any stable state of the network must correspond to a feasible solution of the MCP and in simulations starting from different initial states demonstrate that the network can find all possible solutions. However they have not yet proven that the network will always converge into a stable state.

Smieja [1991] presents an architecture for a type of neural expert module called an *Authority*. An Authority is comprised of a number of *Minos* modules. Each Minos module contains two neural networks a *worker network* and a *monitor net*. The worker net is responsible for performing some computation given a set of inputs and providing a response or *answer*, whilst the monitor uses the same set of inputs to determine the degree of confidence that the Minos module has in its answer. The Authority employs its collection of Minoses as a 'panel of experts' and accepts the answer provided by the most confident Minos, which it then transmits to higher levels in the system hierarchy

Smieja states that the preliminary results from some initial simulations showed that an Authority with three Minos modules performed well in the expert division of random mappings, shift experiments and hand-written digit recognition, although the results from the last application are qualified - as they require further analysis.

Smieja's approach is in part inspired by the suggestion that [Smieja, 1991]:

"all neural networks are in some respect unreliable, and 100% performance will like as not never be achieved from a flexible undedicated neural network module; the key is to *recognise* where the unreliability lies, gauge it and base consequent higher level decision making on the degree of fuzziness or *error* of the answer."

A number of other researchers have also attempted to reduce the complexity of neural network training by decomposing or partitioning the problem into functions that can be solved by smaller more easily trained networks. Green and Noakes

[1989] describe ASLANN (*A System for the Linked Assembly of Neural Networks*) which they demonstrate using the application of back-propagation networks to the routing of an integrated circuit.

Suddarth and Holden [1991] describe how network learning in a connectionist expert system model can be enhanced by the decomposition of larger non-monotonic functions into smaller monotonic ones, implemented using an architecture of networks instead of a single network. They also describe a technique which they call '*the use of rule-injection hints*' which imposes constraints on the learning process, can reduce training time and improve the probability of rule extraction by controlling the training set presented to back-propagation networks and analog associative memories. They suggest that a hybrid system, that is [Suddarth and Holden 1991]:

"a specific structure in which a system is divided into sub functions, each of which can be modelled by either a set of rules or by a neural network rather than by using a single, large network"

Which when combined with the use of hints can [Suddarth and Holden 1991] :

"create the possibility of a "quantity first then quality" design methodology, where highly complex models are created by starting with approximations which are then improved over time with the addition of knowledge. Knowledge is input in the form of hints, as rules, or by dividing complex tasks into simpler (strictly monotonic) ones"

Meng [1993] describes a hybrid neural network/rule-based '*neural production system*' called NEURO-NAV and its application to vision-guided robot navigation. In Meng's architecture both neural networks and symbolic expressions are used to form the antecedents and consequents of rules. The basic syntax of the rule base is the same as in traditional rule-based systems but the internal representations of the antecedent and consequent terms can include references to neural networks whose definitions are stored in a separate 'Database of Neural Networks'.

Their (back-propagation) networks were trained off-line to perform tasks that would be difficult to encode in a traditional rule-based system, for example the detection of an approaching junction (or other landmarks) in a camera image.

Once the neural networks have been trained NEURO-NAV is able to use the hybrid rule-base to perform simple navigation by operating in either a *direction driven mode* where it follows a set of user defined directions or in *autonomous mode* where it uses a combination of basic reactive navigational skills and high level reasoning on a symbolic model of its controlled environment, to find its own way to a new user-defined position.

A number of researchers describe architectures in which neural networks are used to classify complex inputs, and feed the neural classification into an expert system.

Kanal and Raghavan [1992] describe a *Multi-Level Hybrid Neural Network Radar Cross Section Classifier* which uses majority voting to determine an aircraft classification from a radar signature. A second system uses an expert system to identify forests, water bodies and fields using inputs from a geographical database and from a neural network classifier which receives pre-processed radar and optical images.

Kanal and Raghavan [1992] state that in their experience:

"hybrid systems provide a key to achieving significantly better performance than individual component methodologies in solving complex pattern recognition problems"

Several researchers are investigating the automated extraction of rules from neural network architectures. One of the principal motivations behind this approach is the belief that neural networks could help overcome the traditional KBS knowledge

acquisition bottleneck. Given suitable learning, representation and rule-extraction strategies, neural network systems could be used to automatically acquire knowledge which could then be extracted into a more comprehensible (e.g. rule-based) form. Domain expert(s) could then more easily verify the knowledge and even learn new insights from suggested relationships.

Boucheau and Bourguine [1988] demonstrate that a multilayer connectionist structure in which the nodes in the network are assigned semantic meaning can form the basis of a strategy for extracting logical rules from the network. In addition Kane and Milgram [1993] present methods to extract rules from trained multilayer networks and describe the training of 40 networks with different architectures to learn different logical rules. They found that the networks were able to satisfactorily learn logical truth tables but that false rules could also appear.

Another example of the successful extraction of rules from a neural network is described by Nottola, Condamin and Naim [1991]. They used a binary threshold neural network using an adapted back-propagation algorithm, combined with the ID3 decision tree generation algorithm, to extract rules from examples and assist in the standardisation of a methodology for the financial assessment of companies across all of the practical experts or 'agents' of an organisation. They suggest that their approach can help overcome:-

- the '*knowledge gap*' or discrepancy between the practical knowledge of an agent or "practical expert" performing a function, and that of the central department agent or "theoretical expert" attempting to standardise the application of the function.
- the *model gap* resulting from the system being modelled evolving more quickly than the model resulting in a mismatch.
- and the *objectives gap* which can occur if local agents and local management have objectives that are not taken into account by the "theoretical agent".

- Problems of bias caused when the formulation of knowledge by practical experts is influenced by theoretical models, when these models are not applied in practice.
- Problems arising from the existence of a large number of experts with apparently conflicting knowledge formulations.

They found that their combined method produced a more generalised and structured rule-base than that derived from the application of the ID3 algorithm alone. Additionally, the intermediate features generated by the network were of considerable interest to the human experts and could potentially prompt the development of new concepts further assisting the process of company evaluation.

Fu [1992] describes *Knowledgetron* a novel intelligent system which derives rule-based expert systems from neural networks which have initially been trained using error backpropagation. A second training pass uses a strategy which clusters hidden units in the network, - based on similarity between their input weight vectors. The objective of which is to more compactly encode the information in the network, and hence make it easier to extract rules , whilst preserving the network's performance.

Fu's approach is intended to translate an entire adapted network into a rule-based system with equivalent performance. In this sense, Fu's approach is more extensive than that of either Gallant [1988] or Saito and Nakano[1988].

Omlin, Giles and Miller [1992] describe how neural networks can learn to classify long strings of regular language and how a suitable heuristic can be used to extract rules describing the learned grammar.

Sun, [1991] proposes a connectionist architecture called CONSYDERR which is able to account for some of the common patterns found in common-sense reasoning and partially remedies the problem of brittleness, typical of symbolic systems.

The architecture of CONSYDERR is comprised of two levels CL and CD:

CL is a connectionist network in which rules are represented as links between pairs of nodes which respectively represent the rule's *condition* and its *conclusion*. The representation in CL is localist and corresponds roughly to reasoning at the conceptual level.

In contrast CD is a connectionist network with a distributed representation which roughly corresponds to sub-conceptual level reasoning. A set of units is used to represent a particular rule or concept. The sets of units representing different concepts may overlap, the extent of any overlap is then a measure of the degree of similarity between the concepts.

The two layers are allowed to interact during fixed cycles in which weighted links are formed between a neuron representing a particular concept in the localised network CL with all of the nodes in CD representing the same concept. This is repeated for each of the nodes in CL. The activations of the neurons in each of the two networks are then simultaneously allowed to settle down and finally the activation of neurons in CD are allowed to flow back into CL, and are combined with the activations of the other CL nodes. This procedure enables the rule links encoded in the CL layer to be duplicated diffusely in CD.

Sun demonstrates that an architecture of this type can reduce the degree of brittleness, a characteristic, typical of symbolic rule-based systems. Sun also suggests that connectionist models of reasoning are not necessarily simply an

alternative implementation to symbolic systems but can provide better computational models for common-sense reasoning.

Myllymaki et al [1990] describe a knowledge representation scheme designed to perform limited inference in the presence of partial or even inconsistent information. The scheme is implemented using a *neural language* called NEULA. They suggest that neural network programming need not simply mean the type of *low level* programming used to set interconnection weights and activation levels but argue that neural networks can be programmed to perform concept property inference. In their representation, a concept C is represented using labelled collections of (property, value) pairs $C = \{(P_1, V_1), (P_2, V_2), \dots, (P_n, V_n)\}$. Concepts can be arbitrarily complex since the value of a particular property can also be a concept.

In the implementation of NEULA each of the concepts and statements describing the concept are assigned to a particular neuron in the network. Weighted connections are then established between each of the concept neurons C and each of the neurons PV that represent a statement describing that concept. The weights are positive if the statement describes a property/value pair that the concept has. Negative weights are used to connect the concept with property/value pairs that it does NOT have. Echo connections are created in the opposite direction to support inferencing from property/value pairs to concepts.

Inferencing in the network is via *activation propagation* using the formula:

$$a_i^{k+1} = a_i^k + (1 - |a_i^k|) \left(\sum_j w_{ij} a_j^k \right) \quad [4.3.1]$$

where a_j^k denotes the activity of the j th neuron after the k th update and w_{ij} is the weight of the connection between the i th and j th neurons.

In response to a user defined query, selected units are clamped to the values '1' or '-1'. The activation propagates through the network as each of the neurons are allowed to update their activations by applying the formula described in equation 4.3.1. The precise formulation of the equation ensures that neurons whose activations are either 1 or -1 remain set at these values. The computation is terminated when the activity level of any of the neurons that represent possible solutions to the query is within some value, ϵ of either 1 or -1.

In experiments conducted on a system with approximately 300 neurons and with the value of $\epsilon=0.1$, they found that the system always terminated quickly, with an upper limit to the number of synchronous steps, $k \approx 10$.

A number of researchers including Giles and Omlin [1993] have demonstrated - through training a recurrent neural network to recognise a known non-trivial, randomly generated regular grammar - that recurrent networks can be beneficially applied to rule refinement. Specifically that they can preserve genuine knowledge and correct through training, inserted prior knowledge that was wrong - that is inserted rules that were not in the randomly generated grammar.

In their paper "Artificial Intelligence for Explosive Ordnance Disposal System (AI-EOD)", Madrid, Williams and Holland [1992] describe an application which uses both neural and expert system components to help Explosive Ordnance Disposal Technicians to correctly identify damaged or partially obscured munitions in the field.

Basic information about munitions is collected and stored in a file and a 'Knowledge acquisition' expert system is used to efficiently collect data describing the identifying attributes of individual munitions. These sets of possible attribute value aggregates were then stored in a second file. The information in the two files was

used to configure a neural network whose neurons represented attribute-value pairs. These could include for example neurons representing the statements 'yes there are fins', and 'plastic body casing'. Neurons representing the same attribute are clustered together with inhibitory connections between them and from each neuron an excitatory connection is made to each of the other neurons that would contribute to the identification of a particular munition.

The magnitude of individual weights represent the relative importance of a particular input signal. For example, the diameter of a device is considered to be of greater significance than the colour of the paint that may or may not still be on the munition. Analog data is handled differently to discrete attribute-value pairs. The closer a particular analog value is to a design specification (e.g. a diameter) the greater the associated signal that is passed to the receiving neuron(s). The further the signal is from the specified value the weaker the transmitted signal is until it finally becomes negative with the effect of dampening resonance patterns for munitions that have analog valued attributes that are strongly at variance with the user's inputs.

The neural network is distributed (via CD-ROM) to EOD Technicians in the field. and a user interface program asks the technician specific questions about an unidentified item of ordnance and uses the known and estimated responses to activate corresponding neurons in the network. Inhibitory and excitatory signals propagate through the network and excite neurons that are consistent with the known information and suppress the remainder. Eventually the propagation of the signals flows back to the originating neurons and further excites them (and suppresses the other neurons in the same cluster). The conclusion of the network is then given by the dominant resonant harmonic or pattern of activity inferred from the inputs from the user.

Madrid, Williams and Holland [1992] found that their approach can enable a 100% correct ordnance identification rate compared with a 98% identification rate using the previous microfiche and paper based system. They also state that identification could also be performed almost twice as quickly (5 minutes compared to 9.6 minutes) using their system.

Various researchers suggest that the performance of rule-based systems could be significantly improved by mapping the knowledge onto inherently parallel neural network architectures. A number of techniques and applications are described for example by Fu [1989]. A two stage process, where the expert system is first translated into a parallel symbolic representation which is then used to create a real-valued or binary neural network is presented by Stottler and Henke [1992]

Pham and Degoulet [1988] present MOSAIC a '*Macro-connectionist Organization System for Artificial Intelligence Computation*'. MOSAIC is an expert system generator. An expert system created by MOSAIC can be defined as a set of fragments of knowledge called *Neuronics*. Each neuronics corresponds to a concept. Neuronics can have user defined behaviour which can be triggered if the neuronics is suitably excited by the behaviour of other neuronics. Neuronics can co-operate to perform inferencing and support a distributed implementation of the principal components of an expert system (knowledge-base, working memory, and inferencing engine). Neuronics share some of the characteristics of C++ object instances including: encapsulation; communication via a form of message passing; and the ability to represent an instance of an abstract data type. However in their approach there is no equivalent to the object oriented concepts of class, inheritance and polymorphism.

Liow and Vidal [1991] describe a novel *Dual Network Expert System* architecture that combines the learning and inferencing capabilities of neural networks with a user friendly interactive interface. They present a three layer network consisting of input, output and hidden units. The units are fully interconnected between layers. However the units in the hidden layer are separated into two sets, one of which supports forward chaining (or data driven inference) and the other backward chaining (or goal-directed reasoning). The inputs of the net correspond to the presence or absence of particular attributes (e.g. medical symptoms) and the outputs to particular hypotheses (diagnoses). The dual network is trained and weights modified using classical error backpropagation, with training being performed in both directions alternately modifying the weights to and from each of the sets of hidden units.

Once the training has been completed the user can map a set of symptoms onto the network by the clamping the activations of input units that correspond to the pattern of symptoms. The activations are then allowed to propagate forward through the network, via the forward chaining set of hidden units and establish a pattern of activity on the outputs. This pattern of activity (on the outputs) is then propagated back through the network via the other set of hidden units to *all* of the inputs. At which point, units that were initially unknown (not clamped by the user) will have specific values.

They interpret the activation of the output units as the *probabilities of the diagnosis based on the symptoms* and the activations of the inputs as the *likelihood of symptoms not yet detected or that may develop*. The user is given the opportunity to examine the state of the inputs (which correspond to symptoms), and set previously unknown values, perhaps following additional laboratory tests, before the network performs the next bi-directional update. This process is repeated until the hypothesis under scrutiny reaches a stable activation level.

Khosla and Dillon [1993] describe an integrated model for real time alarm processing in a real-world terminal power station, in which a generic neural expert system model, an object model and a Unix operating system process model are combined to develop an Alarm Processing System (APS) for an electricity supply network.

Their integrated model employs an object-oriented design structure and models the performance of the alarm system using a Unix Operating System object, Expert System objects, combined neuro-expert objects and instances of various other object types which are used to model the operation of particular features of the alarm/supply network.

The APS was developed using Neuron Data's Nexpert Object development tool which provides the class and object structure, which are supplemented by external C routines.

Healy [1989] describes a number of the limitations of typical contemporary expert systems, illustrated using the example of an expert system designed to assist a fighter pilot in aerial combat. These include the restriction that the rule-base does not automatically incorporate experience, that procedures for handling unexpected events must be prespecified and the problem of performance, - generating an appropriate response in real-time given complex inputs like radar images and instrumentation readings, combined with potentially lengthy chains of inference and only a single sequential processor.

Healy proposes that a massively parallel neural architecture could provide high speed, self-programming computation which is robust under loss or degradation of individual components. He acknowledges that to be generally useful the network

must prove itself to be capable of learning and/or retrieving patterns in a noisy, changing environment, it must avoid chaotic activity, converge into a stable pattern and demonstrate reliably repeatable performance.

Healy describes a '*Parallel Predicate Model*' based on the adaptive resonance theory (ART) model of Carpenter and Grossberg [1987a,1987b]. He describes how the properties of instances of real-world entities can be stored in the network as elements of a binary spacial vector. Each of the elements corresponds to a particular property or predicate and the value of each of the elements describes whether the proposition that the entity has that property is true (1) or false (0). In the Parallel Predicate Model representation, the vector $\bar{X} = [1000101\cdots 01\cdots]$ signifies that the entity satisfies the first predicate (has the first property), does not satisfy the second, third or fourth, satisfies the fifth and so on. Healy describes how the ART model with outstar connections which form the (template) pattern $V^{(j)}$ when presented with the a new input vector pattern \bar{X} will change to incorporate the new pattern. Specifically $V^{(j)}$ will be replaced by the conjunction $\bar{V}^{(j)} \wedge \bar{X}$. De Morgan's law, which states that :

$$\overline{X \wedge Y} = \bar{X} \vee \bar{Y} \text{ where } \bar{X} \text{ is the complement of } X$$

can be applied to allow disjunctive patterns to be stored in the network, although some pre-processing is required to generate the complement of the *parallel predicate* (binary spacial input vector pattern) describing the entity being encoded.

Tsai, Parlos and Fernandez [1990] present a novel associative memory called ASDM - *Adaptive Architecture based on the Sparse Distributed Memory* and describe how it can be used as a neural expert system that can 'learn' rules in addition to those acquired from human experts.

Other examples of neural network expert system implementations include the random optimisation approach used by Looney [1993], and backpropagation

systems developed by Yoon et al [1990] for the instruction of medical students in the diagnosis of a form of skin disease.

4.5 Comments

The preceding sections reviewed the work of one of the most significant contributors to the field of connectionist expert system research, and a sample of research from a number of other authors.

Connectionist expert system research, is attracting growing interest and contributions from researchers in a wide variety of disciplines, including the pure sciences, engineering, medicine and the social sciences.

As has already been stated, at its widest connectionist expert system and related research encompasses:

- Expert Systems that have connectionist knowledgebases and either traditional or neural inference engines.
- Expert systems that incorporate neural networks in the antecedents and/or consequents of rules.
- Expert systems created from the interconnection of multiple individual neural networks.
- Neural Networks that perform types of tasks previously associated with expert systems.
- Neural networks that use expert systems to perform pre-processing of input data and the post-processing of outputs.
- Neural network architectures that perform inductive and deductive reasoning
- The automatic generation of neural networks from expert systems.
- The automatic extraction of expert system rules from neural networks
- Connectionist expert system hardware

As is clear from the sample reviewed in this thesis a number of different network architectures and applications have been developed. However, many of these, like the model developed by Gallant, and the connectionist expert system implemented in CONNEKT are still based on the relatively simple perceptron model.

It is conjectured that as the field matures, connectionist models that are specifically designed to support the implementation of connectionist expert systems will increasingly be developed.

5.0 THE OBJECT-ORIENTED PARADIGM

5.1 Introduction

There is no widely accepted definition of what precisely constitutes an object-oriented approach. However, Booch [1991] provides a definition of object-oriented programming (OOP) that may be generally acceptable:

"Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships."

By extension, object-orientation could then be defined as: 'the software modelling and development (engineering) disciplines that enable the construction of complex systems through the application of object-oriented programming' (Khoshafian and Abnous [1990]).

Object-orientation can be described as a 'data-centred' approach, in which the application is designed around the data upon which it operates, as Blum [1992] states:

"It's unlikely that the fundamental data that you are working on will change significantly. However the functionality (what you do with that data) will be constantly enhanced and modified as time goes on. Typical structured programming techniques, such as 'functional decomposition', concentrate on what a program *does*, rather than on what it does it *to*. Such design methods require the large-scale overhaul of the software when changes to the functionality are required (as they inevitably are)."

Although a number of object-oriented programming (OOP) languages have been developed the two most widely used are probably, Smalltalk [Goldberg and Robson, 1983] and C++ [Stroustrup, 1986].

5.2 Concepts and Terminology

The application of object-oriented terminology is still some way from being formally standardised. Unfortunately, a single concept can be associated with a number of different (implementation dependent) terms, and a small number of terms have more than one meaning.

To maintain consistency within this thesis, the terminology associated with the C++ development environment is generally used. However where a different, widely applicable equivalent exists this is introduced in parentheses and may subsequently be used interchangeably.

5.2.1 Abstract Data Types

During the 1970's it became clear that to support the development of large, complex software systems a more powerful abstraction mechanism than the 'procedure' was required and the concept of the *abstract data type* [Hoare, 1972; Liskov and Zilles, 1974] was developed, as Hughes states [Hughes, 1991]:

"As with procedures, the basic concept of an abstract data type is to separate the user's view of the abstraction from details of its implementation in such a way that implementations of the same abstraction can be substituted freely. This is achieved by encapsulating the representation of the data object being abstracted with a set of operations that manipulate it, and restricting users to those operations. Code which uses the abstract data types will not be affected by changes in its implementation."

Hughes [1991] provides the following description of an abstract data type:

"- An abstract data type (ADT) defines a class of objects which have the same abstract data structure, by the operations applicable to them and the formal properties of those operations"

Support for Abstract Data Types (ADT's) is one of the most important features of the object-oriented approach. Languages that support abstract data typing must satisfy the following [Parsaye et al., 1989].

1. *Object classes.* Every datum (object) must be the element of an abstract data type (the object's class)
2. *Information Hiding.* Furthermore an object is accessed and modified only through the external interface routines and operations defined for its abstract data type. The internal implementation details, data structures and storage elements used to implement the objects of an abstract data type and its operations are not visible to the clients accessing and manipulating the objects.

Similar to many conventional programming languages, the built-in types of the language (e.g. integers, characters, etc.) also satisfy the following:

3. *Completeness.* The operations associated with an abstract data type correctly and fully define the behaviour of the abstract data type as intended by the programmer."

The concept of abstract data typing is implemented in many object-oriented languages, including C++, Smalltalk and Eiffel, using the *class* construct.

5.2.2 Objects and Classes

In conventional procedural programming, active procedures perform operations on passive data items. In the object-oriented approach the procedures that model the behaviour of a particular entity and the data variables that describe its state are unified

in a single object, which Thro [1991] defines as a:

"Programming unit composed of data structures (instance variables) and methods (operating procedures and functions)."

Every object is an *instance* of a particular *class*, where a class can be defined as [Budd, 1991]:

"An abstract description of the data and behaviour of a collection of similar objects. The representatives of the collection are called *instances* of the class.

The class construct is used to implement encapsulated sets of objects, that model real-world entities, which exhibit the same behaviour. An object's class, is in fact an object type definition that can be used like a template, to create functioning instances of objects with a particular *state* and *behaviour*. At any point in time an object's state is determined by the values stored in its *member data variables* (also referred to as *properties* or *instance variables*), whilst its behaviour is defined by its *member functions* (also known as *operations* or *methods*).

The distinction between an object's class and an instance of the object is rendered indubitable by an example provided by [Tello, 1989]:

"We might say that whereas a class provides the blueprint or genetic code for creating cows, it is the actual cow instance, not the cow class, that gives milk."

Classes can be arranged in *type* graphs or hierarchies, where *subclasses inherit* all of the *properties* and *behaviour* of their *superclass(es)*. Subclasses can redefine their inherited characteristics and add additional properties and behaviour, enabling them to model more *specialized* versions of their superclass types.

In C++, the declaration and definition of a class are generally treated as two separate sets of operations. The class declaration, or *specification*, informs the compiler that a new class - which may or may not be derived from other previously declared classes - exists and that it is comprised of a specific set of member data variables and member function definitions. The member function definitions are described in the *body* of the class. Class declaration specifications (in AT&T Version 3.0 compliant versions of C++) have the general form:

```
class class_name
{
private:
    private components (member functions and data items)

protected:
    protected components (member functions and data items)

public:
    public components (member functions and data items)
}
```

The *private*, *protected* and *public* labels determine the levels of access or visibility granted to the members (both functions and data) declared in their respective sections.

The data members can be declared to be of *fundamental* (also referred to as *basic*) data types: like *char*, *int*, *float*, and *double*; arrays of basic data types; or *pointers* to the memory addresses of fundamental data types. Data members can also be defined as variables of, or pointers to, instances of (previously declared) classes, or arrays of class instances. A data member can even store pointers to instances of its own class.

Similarly, member functions can be declared which have return variables of fundamental or user-defined types, pointers to fundamental or user-defined types or with no return type (void). (For further details see for example [Lippman, 1991; Schildt, 1987]).

5.2.3 Encapsulation

Encapsulation or information hiding can be defined as [Booch, 1991]:

"the process of hiding all of the details of an object that do not contribute to its essential characteristics"

In C++ encapsulation is implemented by declaring data members to be *private* or *protected*. They can then only be accessed in a controlled way, via a set of *public* or *protected* member functions that collectively comprise the class interface.

The private members of an object instance can only be accessed by member functions and friend functions (described below) of that instance. Protected members can only be accessed by member and friend functions of their class, and by classes derived from it through the process of class inheritance (described in section 5.2.4). Public members can be accessed and where appropriate modified by any function (within their scope).

Encapsulation enables the internal implementation of a class and its instances to be separated, by the interface provided by the class member functions, from external references to the object. This can help prevent the accidental modification of data, and in conjunction with inheritance and polymorphism (both described below) can enable the creation of a single interface but potentially many (context dependent) implementations.

In C++ a class can have one or more functions or classes declared to be a *friend* of the class. Friend functions and classes provide additional flexibility, but at the expense of partially violating encapsulation, albeit in a tightly controlled way.

Friend functions are declared by preceding the function declaration in a class's specification by the keyword 'friend'. A class can be defined as a friend to another

class by including a statement of the following type in the class declaration:

friend class *class_name*;

All of the friend functions, and all of the member functions of any friend class(es), are given special access privileges to the private and protected members of the class containing the friend declarations.

Some of the properties of friend functions are described by Microsoft [1991]:

"Friend functions are not considered class members; they are normal external functions that are given special access privileges. Friends are not in the class's scope, and they are not called using the member selection operators (. and ->) unless they are members of another class."

However, the special access granted to friend functions and classes does not extend beyond the class in which the privileges are granted. As Microsoft [1991] state:

" "friendship" cannot be inherited, nor is there any 'friend of a friend' access"

Friend functions and classes increase the power of the C++ language, but only by providing direct access to the private and protected members of an object - and hence only through a controlled violation of encapsulation.

The decision to introduce friend functions was not incontrovertible, however, the violation of encapsulation that they enable is limited. As Lafore [1991] states:

"**friend** functions are controversial. During the development of C++ arguments raged over the desirability of including this feature. On the one hand, it adds flexibility to the language; on the other, it is not in keeping with the philosophy that only member functions can access a class's private data.

How serious is the breach of data integrity when **friend** functions are used?

A friend function must be declared as such within the class whose data it will access. Thus a programmer who does not have access to the source code for the class cannot make a function into a **friend**. In this respect the integrity of the class is still protected."

5.2.4 Class Inheritance

One of the most powerful features of object-oriented systems is that they can allow components of the model for one entity type or entity instance to be shared or *inherited* by other entity types or instances.

A number of object-oriented languages, including C++, Smalltalk, Eiffel and Ada, use a process called *class inheritance*, or often simply referred to as *inheritance*.

Class inheritance enables a class to inherit the member data (attributes or properties) and member function (method or behaviour) definitions of other classes.

If a class *D* is inherited from a class *B*, then *D* is called the *derived* class and *B*, its *base* class. Another way in which this may be described is that: *D* is the sub-class of *B*; which in turn, is *D*'s *super-class*.

The declaration and definition for a derived class can include member data variables and member functions, over and above those inherited from its base class(es). The derived class may also be able to modify the behaviour of some of all of the member functions that it inherits, a process known as *function overloading* (described in section 5.2.9). The derived class can also act as the base class for other classes in the hierarchy.

Class inheritance enables complex hierarchies to be developed, in which the behaviour of the most general classes, at the top of the hierarchy, is increasingly specialized, with classes in the lower level of the hierarchy modelling increasingly more specific and specialized entity types.

Some languages, for example, Smalltalk are *single-inheritance* systems, as they allow a class to have only a single base class. Other languages (including C++) are *multiple-inheritance* systems, since they allow a class to be derived from more than one base class. Single-inheritance systems can support only *tree* like inheritance hierarchies (see figure 5.1), whilst multiple inheritance systems can support hierarchies that are either tree's or directed acyclic graphs (DAG's) (see figure 5.2).

Figure 5.1 : Simple class inheritance tree

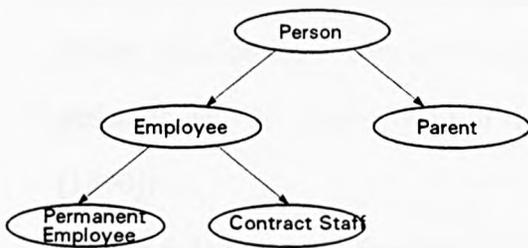
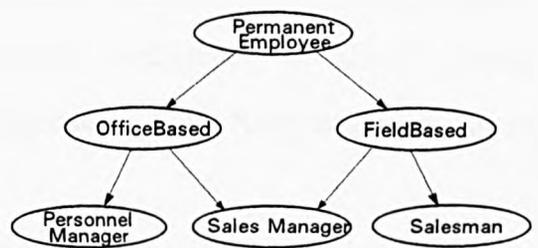


Figure 5.2 : Simple class inheritance graph



Classes can be either publicly or privately derived from their superclass(es). As Hughes [1991,p130] explains:

"The keyword **public** or **private** placed before each base class indicates the access control to be applied to the inherited members. **Public** means that the inherited members retain the status that they have in the base class, while **private** causes the inherited members to become private members of the derived class irrespective of their status in the base class."

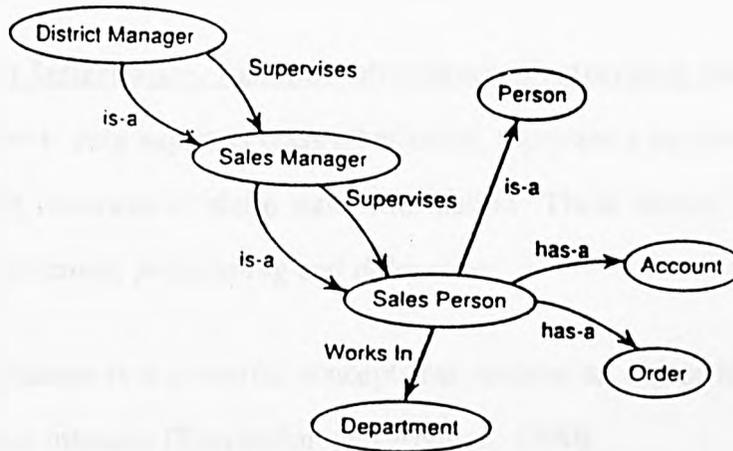
The types of the member data items specified in the base class declarations, are inherited by the class's sub-classes. In C++ the type cannot be changed, but in Eiffel, which is also a strongly-typed language, the inherited type can be redefined provided that the new type is a sub-type of the type in the base class (Khoshafian and Abnous [1990]).

The instances of a class can be thought of as a set of objects of the same (abstract data type), which share the same behaviour. Objects instantiated from a class with a

super-class have the relationship, *is_a*, with all of the *ancestors* of their class, that is with all of the classes from which it is directly or indirectly derived. Each of the classes in the hierarchy can also contain member data variables which describe the attributes of the entity type modelled by the class. An object instance instantiated from a class in a class hierarchy, therefore also has the relationship, *has_a*, with each of the entity types represented by its attributes.

A class inheritance hierarchy represents the same relationships as the semantic network representation, first introduced by Quillian [1968] which is widely used to model relationships between entities in artificial intelligence. A simple example semantic network is described in figure 5.3 (reproduced from Khoshafian and Abnous [1990]).

Figure 5.3: Example Semantic Network



A semantic network for SalesPerson.

In C++ the *types* of the member data items - specified in the base class declarations - are inherited by the class's sub-classes. In C++ the type cannot be changed, but in Eiffel, which is also a strongly-typed language, the inherited type can be redefined

provided that the new type is a sub-type of the type in the base class [Khoshafian and Abnous,1990,p101]

In CONNEKT, all objects are instantiated from classes derived using class inheritance from classes in the Microsoft Foundation Class (MFC v2.0) class library. This is a requirement for the classes and instances implementing the graphical user interface when developing in Microsoft Visual C++.

The classes and instances that model vehicles, customer and locations and those that actually provide the connectionist expert system and 'Pocket' Boltzmann functionality are all derived from the MFC class CObject. This provides the application classes with a set of inherited capabilities including functionality to support the storage (serialization) and retrieval (deserialization) of object instances to and from hard disk storage.

5.2.5 Object Inheritance. Instance inheritance. Prototyping and delegation

Although C++ only supports class inheritance, there are a number of techniques that enable object instances to share state information. These include: *object inheritance*, *instance inheritance*, *prototyping* and *delegation*.

Object inheritance is a powerful concept that enables an object to inherit the state of another object instance [Khoshafian and Abnous, 1990]:

"In general terms, an object O inherits the state of an object O' if the value of an instance variable i' defined in O' *determines* the value of the same instance variable in O."

Instance inheritance is a more tightly constrained application of object inheritance [Khoshafian and Abnous, 1990]:

"an instance O' can inherit from an instance O only if the class of O' is a subclass of the class of O."

A more flexible alternative to class, object and instance inheritance can be provided by prototypes [Parsaye et al, 1989]:

"In a prototype system one first thinks of a particular prototypical object and then draws similarities and/or distinctions for objects that are like the prototype in some ways and different in others. Any object can become a prototype. The approach is to start with individual cases and then subsequently specialize or generalize. Lieberman (1986) elegantly describes the distinction between this approach and the object-oriented (set-oriented) approach.

Prototype systems allow creating concepts first, then generalizing them by saying what aspects of the concept are allowed to vary. Set-oriented (object-oriented) systems require creating the abstraction description of the set (class) before individual instances can be installed as members."

In a prototypical system selected properties of an instance can be *delegated* to another object instance. This means that the values for the delegated properties in the object instance are determined by the values of the corresponding properties in the prototype object.

Khoshafian and Abnous [1990] report that there has been some discussion as to whether prototyping/delegation or inheritance are the most appropriate techniques for modelling a given domain and state that Lieberman [1986] has argued:

"that delegation is more powerful than inheritance since it can easily be shown that inheritance can be modelled through delegation. The sharing of methods that is supported through the class-based systems can be achieved by having objects delegate their operations or methods to a common ancestor. Delegation appears to be more powerful than inheritance because it is more general; in addition to sharing behaviour, objects can also share state. Delegation also appears to be more general than object inheritance because arbitrary objects can delegate to one another.

However as Khoshafian and Abnous [1990] point out, Stein [1987] has shown that classes can be used to model delegation.

The differences of approach were examined at the OOPSLA '87 conference in Orlando, Florida and led to the development of the *Orlando Treaty* [Stein, Lieberman, and Ungar, 1989]. This acknowledges that different approaches are applicable in different situations since there are essentially two types of code sharing: *unanticipated sharing* and *anticipatory sharing*. They suggest that delegation based systems are more suitable for unanticipated sharing, but that class-based systems are best for anticipatory sharing. As Stein, Lieberman, and Ungar [1989] explain:

"... no definite answer as to what set of choices is best can be reached. Rather, that different programming situations call for different combinations of these features; for exploratory, experimental programming environments, it may be desirable to allow the flexibility of dynamic, explicit, per object sharing; while for large relatively routine software production, restricting to the complementary set of choices - strictly static, implicit and group oriented - may be more appropriate."

Some of the comparative advantages and disadvantages of the two approaches are described by Khoshafian and Abnous [1990]:

"There is a tradeoff of space versus execution time between the two strategies, with prototype systems typically requiring less space but more time to bind methods or obtain attribute values. By contrast, systems using class inheritance have faster method lookup but may require more space. Also, if the class-based system is strongly typed, there is an additional tradeoff, that of safety versus flexibility. The bottom line is that dynamic systems can be used to build prototype systems, but production quality and high performance systems will have to use strongly typed class-based systems such as C++ or Eiffel."

5.2.6 Metaclasses

Several object-oriented languages, including Smalltalk, but not C++, support the concept of metaclasses. A metaclass is a class whose instances are also classes. Smalltalk is a 'pure' object-oriented language because [Hughes, 1991]:

"Everything in a Smalltalk program is an object, including every class. That is, every class is viewed as an instance of a higher level class called a metaclass. Thus a Smalltalk program can be described as a tree of objects where the root is a built-in

superclass called *Object*. The root of the sub-tree containing only classes is a special built-in metaclass called *Class*."

In addition to acting as a template describing the attributes and behaviour of a class (type), a metaclass can contain *class instance variables* which can be used to store global (or group) information that applies to all of the instances of the class. These can be accessed and maintained using *class methods*. Examples of class instance variables could include: a count of the number of instances instantiated and aggregates of member data variables, for example the total salary bill for the instances of an *Employee* class.

In C++, a class instance variable can effectively be created by declaring a member data item to be *static*. This ensures that only one copy of the variable is created, which is then shared between all of the instances of the class. However, the variable must be initialized once only, at global scope using the class name and the C++ scope resolution operator '::'.

5.2.7 Object Instantiation (Creation) and Destruction

Instantiation is the process of creating object instances from class definitions. A class is said to be instantiated [Tello, 1989] :

"when it has actually been used to stamp out one or more objects of its type"

A key part of the process of creating a new object instance involves special functions called *constructors* which cause free, volatile memory to be allocated to store the new object instance and set up pointer(s) to addresses in the allocated memory area. In C++, a constructor is automatically invoked whenever a new object instance is created and normally initialises some or all of the object instances' data members.

In C++, *constructor* functions always have the same name as their class. A class can have several constructor functions, but each must have a different argument signature. The constructors of classes that are derived from one or more other classes can invoke the constructors of their base class(es) when creating a new instance of the derived class.

Destructor functions are called to destroy objects by deallocating the volatile memory used to store them. The destructor function's name is always the same as the name of its class, but preceded with a tilde (~). The destructor for a hypothetical class CPerson would therefore be called ~CPerson(), and its definition would be of the form:

```
~CPerson() { body of destructor function };
```

If required, specific 'clean-up' tasks can be specified in the body of the destructor function. One of the most important of these is the specific deletion (de-allocation of memory) for objects created by the user (on the heap using the C++ **new** operator). This is necessary to prevent *memory leaks*, that is, memory remaining allocated after a function goes out of scope or even after the application has ended.

Destructors are called (in Microsoft C/C++ v7.0) when one of the following events occurs [Microsoft 1991b]:

- "- An object allocated using a **new** operator is explicitly deallocated using the **delete** operator"
- and:
- "- A local (automatic) object with block scope goes out of scope.
 - The lifetime of a temporary object ends.
 - A program ends and global or static objects exist.
 - The destructor function is explicitly called using the destructor function's fully qualified name."

When a destructor is called, the statements in the body of the function are executed first. If the destructor function's class is derived from other classes, the destructors for these classes are invoked in the reverse order of their appearance in the class declaration defining the classes superclasses. If any of these classes are derived from other classes the process is repeated until the destructors of each of the class's ancestors have been invoked.

5.2.8 Message Passing

The overall behaviour of an object-oriented application is determined by the behaviours of the individual objects within it, which communicate by passing messages. Along with encapsulation, the message passing protocol prevents the status of an object from being modified accidentally. As Tello [1991,p6] states:

"Objects are seen as passing messages that they can either accept or reject. Generally an object accepts messages it recognizes and rejects those it does not. In this way the types of interaction between objects is carefully controlled. Because all interactions follow this protocol, code external to an object has no opportunity to interfere in any way with the object's functioning in unpredictable and undesirable ways."

When a message is issued, the run-time system has to determine which method or member function should be invoked, a process known as *method lookup*. This is defined by Budd [1991] to be:

"The process of locating a method matching a particular message, generally performed as a part of the message-passing operation. Usually, the run-time system finds the method by examining the class hierarchy for the receiver of the message, searching from bottom to top until a method is found with the same name as the message"

In C++, message passing is implemented through the invocation of member functions. A function can be bound to an object instance either when the program is compiled, in which case it is known as *static*, *early* or *compile-time binding* or when the program is running, in which case it is called *dynamic*, *late* or *run-time binding*.

Run-time binding supports a powerful form of polymorphism, and is described below.

5.2.9 Polymorphism and Binding

When applied in a programming context, the term polymorphism indicates that [Khoshafian and Abnous, 1990]

"the language construct can assume different types or manipulate objects of different types"

Polymorphism is a feature of a number of language types, including procedural languages like Basic, and C. However in most procedural languages polymorphic behaviour is limited to only a few functions. For example, in most (if not all) versions of BASIC the functions that support basic input and output, and operators that perform arithmetic and assignment are polymorphic. That is, a programmer can use them to operate on literals and variables of different fundamental types, without having to have regard for the differences between the types .

In contrast, in object-oriented languages, like C++, polymorphic behaviour is often extended to a large range of operations on both basic and user defined types.

One of the simplest forms of polymorphism occurs when the same function name and argument signature is used in two or more unrelated classes. When the member function is invoked, the system ensures that the correct function is called by examining (at compile-time) the declaration type, that is, the class, of the object instance invoking the function, and associating the call with the member function for that class.

A powerful form of polymorphism is called *function overloading*. Function overloading can support either static or dynamic binding.

The simplest form of function overloading occurs when member functions with the same name but with different argument signatures are declared and defined within a single class, that is not part of a class hierarchy or graph. This is allowed (by the compiler) provided that the combination of their function name and argument signature is unique.

For example, a class definition will often include a number of constructor declarations and definitions. These enable objects to be instantiated with different initial states. The C++ compiler simply uses the argument signature to determine which of the functions should be statically bound.

A more sophisticated and powerful form of function overloading can occur when classes are related in a hierarchy or graph. Again in the simplest case, if a class 'D' and its base class 'B' both contain a function 'FunctionA()' and only the base class contains a second function 'FunctionB()', then if an instance of both classes is created, and each invokes both FunctionA() and FunctionB(), each of the instances will be bound at compile time to its *own* version of FunctionA(), but both will be statically bound to the *base* class function, FunctionB().

This occurs because, at its simplest, the compiler uses the (class) **type** of the object instance invoking the function to determine which function to use in the binding process. If the class does not contain the function, then the system searches the inheritance hierarchy or graph, starting with the class itself, and progressively examining each of its ancestors until it finds a match.

The most powerful form of function overloading uses dynamic or run-time binding. If a developer declares a function in a base class to be *virtual* by inserting the keyword immediately before the function declaration, the system will use the class of the

object assigned to the variable invoking the function, to dynamically determine which function to call. C++ is a strongly typed language, and the compiler performs type checking to ensure that the variable contains an instance of the base class, or of a class derived from it. This extremely powerful programming feature can be illustrated using a simple example:

Suppose that a graphics application contains a class *GraphicElement* which has a definition for a virtual member function called *Draw()*. Suppose also that the classes *Line*, *Arc*, *Ellipse*, and *Polygon* are derived from *GraphicElement* and that each overloads the *Draw()* virtual function to draw a particular type of graphic element (the type that corresponds to their name). Such an application could contain an array of pointers to class *GraphicElement* to store references to instances of the eponymously named classes, derived classes.

The application display could then be re-drawn, simply by iterating through the array and invoking the *Draw()* function for each pointer. The run-time system dynamically binds each function call, to the version of the *Draw()* function that is defined in the class, that is the class of the instance pointed to by the pointer stored in the array element. This means, for example, that a pointer to a *Line* element will cause the *Line* class's version of *Draw()* to be invoked

In the example above, it is unlikely that an instance of the class *GraphicElement* element would ever be created. The class is in fact an example of an *abstract* class, which exists purely to act as a collection for the attributes and behaviour of a set of related classes. In C++, the class can be prevented from being instantiated, that is, only instances of its derived classes would be allowed if one or more of its member functions are declared to be *pure virtual* functions, by appending the suffix '=0' to the function declaration.

C++ supports another form of polymorphism, called *operator overloading*. This extends function overloading to allow most of the basic operators to be overloaded. A simple example could be overloading the assignment (=) and addition (+) operators for a 'String' class to enable it to add together (concatenate) two or more strings. The following expressions, where all objects are instances of class String, would then be legal:

```
StringX = StringY ;           // Assign the value of StringY to StringX

StringA = StringB + StringC + StringD; // Concatenate StringB, StringC and
// StringD and assign value to StringA
```

The implementation of operator overloading can dramatically simplify the manipulation of object instances. Other examples could include the creation of overloaded assignment, addition, subtraction, multiplication and equality operators for classes modelling vectors, matrices and complex numbers.

The principal advantage of using polymorphic operators that can manipulate user defined types, and polymorphic functions that can operate on a range of basic and user defined types is that it simplifies the initial creation and the subsequent maintenance of application code.

A number of object-oriented languages, excluding C++, but including Ada [Ichbiah et al.,1979], and Eiffel [Meyer, 1988] support a more constrained form of polymorphism called *parametric polymorphism* or *genericity*.

Systems that support parametric polymorphism support the definition of parameterized types (or classes) and allow methods to be defined that are able to perform the same type of operation on parameters of different types. An example could be a single

method that can calculate the sum of a list of integers, or of a list of real numbers. The principle advantage of parametric polymorphism is that it combines the flexibility and advantages of code sharing for generic types with the power of strong typing (Khoshafian and Abnous, [1990]).

5.2.10 Object Identity

A fundamental property of an object instance is described by the concept of object identity. An object's identity uniquely defines and specifies an object instance and supports many of the operations performed by, and on object instances. For example, the existence of unique object identities enables the run-time system to support communication between objects via message passing (member function invocation), allows programmers to construct arbitrary graph-structured or complex objects, based on sub-objects. and can even support the serialization and subsequent retrieval of persistent objects.

Schemes that assign identities to objects should ideally ensure that all objects have *location independence, value independence and structural independence* [Hughes, 1991]:

- "- *Location independence* implies that an object's identity is preserved despite changes in its physical location, whether in volatile or persistent store.
- *Value independence* involves the preservation of object identity through changes in its value.
- *Structure independence* involves the preservation of object identity through changes in its structure."

Objects that have structural independence, that is, objects that can undergo structural modifications through changing their class are said to have *strong built in identity*.

There are three principal ways in which an object's identity can be specified. These are through the use of *memory references*, *identifier keys* and *surrogate identity*:

i) memory references/addresses.

This is the technique used in many object-oriented and conventional programming languages (including C++) where an object is identified and referenced using its address in volatile memory.

A memory address is imposed on an object by the external run-time system and is in fact different in concept to identity. If an object needs to be moved in memory, all references to the object must be updated, hence location independence is not supported, however this technique does provide both structural and value independence.

Implementations that use pointers to memory addresses can also suffer from the serious problem of 'dangling references'. This occurs when an area of memory is allocated, for example to an object, and the object is deleted or moved without updating all of the pointers that referenced its memory address. Dangling pointers can potentially cause key program or system areas to be corrupted, with the result that the program or system crashes.

Some programming languages, for example Smalltalk-80 [Goldberg and Robson, 1983], achieve a degree of location independence, by addressing the object [Hughes, 1991]:

"by means of a pointer which is not the virtual address of the object, but rather is an entry in an object table. This is equivalent to indirect addressing and has the advantage of allowing individual objects to move within a single address space without the necessity to change their identity. Full structure and value independence are also provided by this approach. However since the pointers are not unique, the scheme does not permit the sharing of objects among multiple programs."

ii) *identifier keys*.

Tuples in conventional database systems are frequently identified through the values in a set of *key* attributes. The same technique can be applied to objects, using the values stored in a subset of the objects member variables. However, it may be necessary for artificial key attributes to be introduced to ensure that all objects can be uniquely identified. This technique provides location independence, however, value and structural independence can be jeopardized if the values in key attributes are modified.

iii) *surrogate identity*. A number of authors expound the importance of surrogate identities, for example Hughes [19910,p171]:

"A powerful technique for supporting identity is through the use of surrogates... Surrogates are system-generated globally unique identifiers, independent of physical location. They therefore offer full location independence. Also, if a surrogate is associated with each *object*, regardless of its complexity (rather than with a tuple as in Codd's RM/T), then they provide full value and structure independence."

An object is assigned its globally unique surrogate identity by the run-time system when it is instantiated. The identity is conceptually independent of its name, address or location, status, and class or type.

The independence of surrogate object identifiers is supported by preventing users from exercising direct control over them. This can prevent the occurrence of problems, including the violation of referential integrity, which can occur, for example when system or user specified key values are changed.

The high degree of independence offered by surrogate identities, has led to the technique being implemented in a number of object-oriented database systems including, for example, ODE [Agrawal and Gehani, 1989].

In CONNEKT all application objects including customer, vehicle, location, connectionist expert system and Pocket Boltzmann machine instances are assigned a unique (surrogate) object identity on instantiation.

5.2.11 Object Equivalence

The ability to determine whether two or more objects are equal is of fundamental importance in object-oriented systems. In the object-oriented model, there are a number of different ways in which two (or more) objects can be considered to be equal, they can for example, be *identical*, *shallow-equal*, or *deep-equal*.

- "1. *Identity predicate (identical)*. The identity predicate corresponds to the equality of references or pointers in conventional languages: It checks whether the object identities are the same. With the semantics of object identity, if the object identities are the same then the objects are the same."
2. *Shallow equality predicate (shallow-equal)*. Two objects are shallow equal if their *states* or contents are identical. That is, two objects are shallow equal if they are instances of the same class and the values they take for every instance variable are identical - corresponding instance variables cannot merely have the same object contents but must be identical objects."
- "3. *Deep equality predicate (deep equal)*. The third, and in a sense the weakest genre of equality is purely value-based deep equality. There are two flavors of deep equality, depending whether the predicate is checking for isomorphism of the object graphs.

In its simplest form deep equality ignores object identities and checks whether:

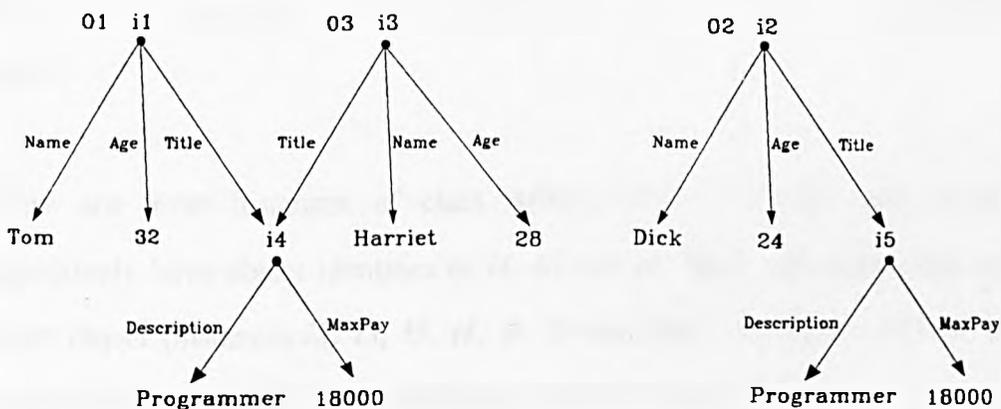
- a. two objects are instances of the same class (have the same structure or type) and
- b. the values of the corresponding base objects are the same."

The different equality types can be illustrated, using two examples. The first is adapted from an example given by Khoshafian and Abnous [1991].

Suppose that a class *EMPLOYEE* has three attributes, *name*, *age*, and *job-title* and that three *EMPLOYEE* instances have been created, called *Tom*, *Dick* and *Harriet*, with surrogate object identities of *i1*, *i2*, and *i3* respectively. Each *EMPLOYEE* instance stores their *name* and *age* as literal values, and contains a reference to instances of a class *JOB-TITLE*. The class *JOB-TITLE*, also has two attributes, *description* and *maxpay*, which respectively store a description of the *EMPLOYEE*'s job-title, and the maximum pay that can be awarded to an individual with that job-title.

All of the Employees are programmers and hence have the same job title. However, (perhaps over time) two 'Programmer' instances of class *JobTitle* have been created and have been assigned object identities of *i4* and *i5*. This situation is described graphically in figure 5.4, where the aggregate objects *Tom*, *Dick* and *Harriet* are respectively labelled **O1**, **O2** and **O3**, and only the object identifiers of the structured *EMPLOYEE* and *JOBTITLE* object instances are displayed.

Figure 5.4: Example Object graph with identical and shallow equal objects



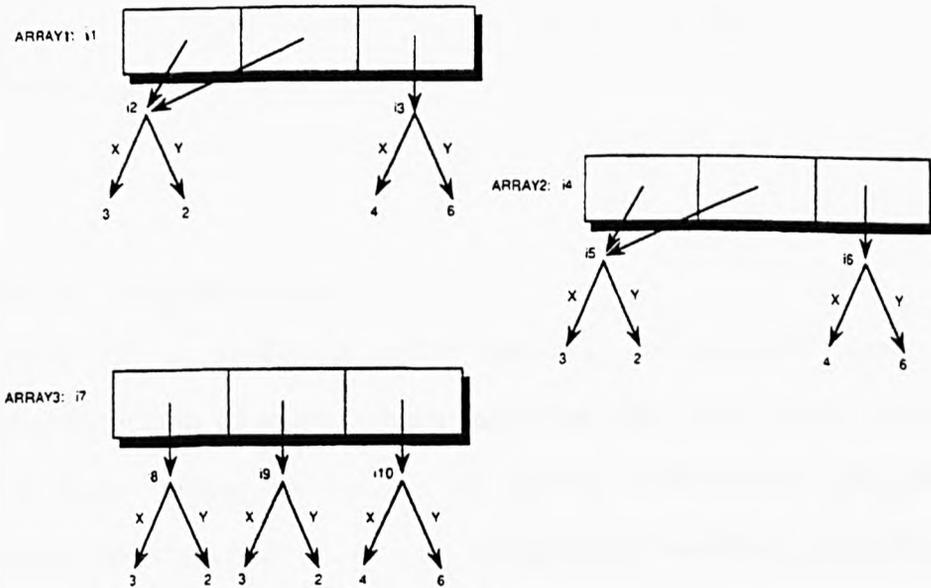
In the example, both *Tom* and *Harriet's* object identity references to their *JOB-TITLE* instance are *identical*, since they both refer to object **i4**. Dick's reference is to object **i5**, which is clearly a *different* object, even though it has the same attribute values as **i4**. The *JOB-TITLE* object referenced by Dick's *job-title* attribute does not therefore have *identical identity* with that referenced by *Tom* and *Harriet*, even though objects **i4** and **i5** are *shallow-equal*, since they are instances of the same class and the values they take for every instance variable are identical.

It should be emphasised that for two objects to be shallow-equal, the values of all member instances must be **identical**. If, for example, the *JOBTITLE* class had itself included a reference to another object instance. Objects **i4** and **i5** would only be shallow-equal, if they both referenced the same (identical) object. If the references were to *two* different objects, even if the referenced objects had identical contents, and hence were shallow-equal, **i4** and **i5** would could not be classed as shallow-equal, but would still be deep-equal.

Two types of deep equality can be illustrated using an example of a class called *ARRAY* (taken from Khoshafian and Abnous, [1990]) that contains a one dimensional array, of three elements. The array stores references to object instances of class *POINT*. Each *POINT* has two (integer) values, representing the 'X' and 'Y' co-ordinates of the point.

There are three instances of class *ARRAY*, *Array1*, *Array2* and *Array3*, which respectively have object identities of **i1**, **i4** and **i7**. Each *ARRAY* element references a *POINT* object (instances **i2**, **i3**, **i5**, **i6**, **i8**, **i9** and **i10**). The values of the 'X' and 'Y' co-ordinates of the points are described overleaf in figure 5.5

Figure 5.5 : Example Object graph with deep equal objects



The corresponding elements in each of the arrays, contain references to different objects that share the same 'X' and 'Y' co-ordinates. Each of the arrays, therefore, stores the same sets of co-ordinate values and as a consequence all three arrays are **deep equal**. However, it is obvious that *Array1* and *Array2*, also have the same structure, since the first two elements in each array reference a single object. There is therefore, in a structural sense, a greater degree of similarity between their object graphs, than when the graph for either, is compared with that for *Array3*. *Array1* and *Array2* can therefore, additionally be described as **Isomorphically deep equal**.

5.2.12 Triggers

Triggers, which are also referred to as *alerters* or *monitors* are procedures that monitor the state of a system, usually a database, waiting for a set of conditions to become true. Once the conditions are satisfied, the trigger *fires* and initiates a set of associated actions.

Triggers are a feature of ODE [Agrawal and Gehani, 1989] which is an object-oriented database extension to the C++ programming language. In ODE, triggers fire *once-only* by default, although a time-out action and a *perpetual* status can

optionally be specified. Once-only triggers are deactivated once they have fired and need to be explicitly reactivated. In contrast, perpetual triggers are automatically reactivated after firing.

5.2.13 Related concepts

There are a number of other concepts, for example, those supporting the implementation of object-oriented databases, that must remain beyond the scope of this thesis. These include the concepts of: concurrency, persistence, clustering, transaction management, memory management, versioning, integrity and constraints, and security.

Many of these are active areas of research in their own right. Articles can be obtained from a number of sources including: Proceedings of the ACM OOPSLA Conferences, the ACM OOPS Messenger, Proceedings of the ACM SIGMOD Conferences, the ACM SIGMOD Record and the Journal of Object-Oriented Programming.

Descriptions of a number of the concept described in this chapter are provided by Booch[1991], Khoshafian and Abnous [1990], Hughes [1991], Parsaye et. al. [1989], Budd[1991] and Tello[1989].

5.3 Advantages and Disadvantages for Conventional Application Development.

The object-oriented approach has a number of advantages, principally over conventional procedural programming. A number of these are also advantages when compared to other paradigms, including logic programming and rule-based programming.

One of the principal advantages of object-orientation is that it provides an immediately intuitive method for modelling problem domains. The concept of distinct, identifiable

class types and object instances, with inherent attributes and (predictable) behaviour, interacting with each other, may even be one of the first concepts that we learn. The concept of inheritance and the consequent ability to organize information into taxonomies are also likely to be familiar to many.

The development of an object-oriented model following for example Booch[1991] or Shlaer and Mellor [1988] is likely to require less abstraction, than for example an equivalent model of the same problem domain developed for the relational model [Codd, 1970].

A number of features of object-oriented systems, particularly inheritance, and polymorphism support the development of reusable code modules (e.g. class libraries), which in turn can help to reduce development times and costs.

Polymorphism, in the form of operator and function overloading, especially when combined with dynamic binding, can dramatically simplify coding and simultaneously enhance the power (and the elegance) of OOP languages.

Encapsulation and message passing help to insulate changes to the internal implementation of the behaviour and attributes of a class, from external references to it. This has the consequential impact, particularly when combined with the opportunity to separately test the internal class implementation, of both simplifying software maintenance and potentially increasing the quality of the overall implementation.

However, the object-oriented paradigm may encourage the development of larger and more complex applications than would generally be developed using a procedural language. As a consequence it may be more difficult to fully decompose and comprehend the interactive behaviour of the system. It may also be more difficult to

understand, predict and determine the impact of changes to the interface of widely used classes, particularly if they form part of an inheritance hierarchy. Again, largely as a consequence of the applications size and complexity, testing the overall behaviour of the system may also prove more to be more difficult.

5.4 Advantages and Disadvantages for KBS and Connectionist System Development.

Object-oriented applications have no in-built structures to support either logic programming or the rule-based paradigm. However, the concept of classes organized into hierarchies, and objects with attributes (and behaviour) correspond closely to the AI knowledge representations expressed using semantic networks and Frames. This has facilitated the successful development of a number of knowledge-based systems, including NexpertObject and KEE.

A number of features are shared between the object-oriented approach and connectionism. In both systems, applications function through messages being passed between potentially large numbers of distinct object or cells, and in both, individual instances (cells) have their own internal attributes and behaviour, and an external interface.

A number of researchers are investigating the simulation of connectionist networks using object-oriented software languages. Research in this area includes: Ran and Karjalainen's [1991] development of a high level object-oriented connectionist programming environment which can be used to implement feedforward back-propagation networks; Burdorf's [1993] POCON language which supports the storage and re-use of persistent neural objects; Fuentes, Aldana and Troya's [1993] development of URANO an object-oriented neural network simulation tool and Dreiseitl and Wang's code generator which automatically generates C++ code to support the simulation of neural networks.

Finally, it could be conjectured that the development of languages like Concurrent C++ by Gehani and Roome [1988], and, for example, the research into Concurrent Object-Oriented development, published in the book edited by Yonezawa and Tokoro [1987]; might eventually support the simulation of very fast, parallel connectionist networks on massively parallel processors.

5.5 Conclusions

The object-oriented approach offers a number of advantages over conventional, procedural programming. Specifically it reduces the level of abstraction, and enables more intuitive designs to be developed. It can support the development of complex software systems, promotes the extension and reuse of code, can reduce development time and ease maintenance.

The approach is also fundamentally compatible with that employed within connectionism. Object-orientation, therefore potentially provides a good environment in which to develop neural, KBS and as a consequence connectionist expert system software.

The power of neural networks lies in their ability to utilise relatively large numbers of (simple) processing units, working in parallel using only local information, to learn to compute difficult (even hard, NP complete) problems. Despite the inherently parallel architecture, many neural network implementations actually simulate the parallel behaviour of neural cells using a single sequential processor.

A number of researchers are attempting to develop massively parallel neural networks on chip. However current models of the neuron are significantly simpler than the observed behaviour of biological neurons.

A more flexible alternative could (eventually) be to use concurrent languages, and perhaps particularly, concurrent object-oriented languages to implement either simple or more sophisticated simulations of biological neurons on multi-processor and ultimately massively parallel hardware platforms. It is conjectured that this may (eventually) enable the development of powerful object-oriented applications which seamlessly integrate fundamental and conventional user-defined abstract data types with powerful artificially intelligent types, capable of exhibiting the low level processing power of connectionist systems, and the higher level reasoning capability of a knowledge-based system. However, it is probable that a significant number of problems remain to be discovered and solved before such an approach could become viable.

6.0 The Design Development and Application of a Connectionist Expert System and 'Pocket' Boltzmann machine approach to the Dynamic Customer Assignment and Vehicle Routing Problem

6.1 Introduction

This chapter describes the design, development and application of a novel connectionist architecture which can find very good or near-optimal solutions to examples of Dynamic Customer Assignment and Vehicle Routing Problems (defined in section 2.5).

The connectionist system is comprised of two types of connectionist components. The first component, is a connectionist expert system which determines the assignment of customers to vehicles. The second connectionist component type is the 'Pocket' Boltzmann machine. The 'Pocket' Boltzmann machine is a modified Boltzmann machine which can develop near-optimal solutions to examples of the TSP, whilst simultaneously satisfying variable number of additional *hard* (mandatory) constraints. The 'Pocket' Boltzmann machine was developed by applying a novel extension to Aarts and Korst's [1989a, 1989b] implementation of the Boltzmann machine.

The 'Pocket' extension is inspired by the 'Pocket' algorithm developed by Stephen Gallant [1985,1988a,1993] and applied by him to Perceptron and Back-propagation networks. Sets of 'Pocket' Boltzmann machines are dynamically generated on demand to fulfill two important functions. Firstly, they are used to determine the incremental cost of assigning a particular customer to a particular vehicle. This data is used to calculate a sub-set of the connection weights in the connectionist expert system network, and exerts a strong influence on the selection and assignment of customers. Secondly, each of the 'Pocket' Boltzmann machines attempts to develop a very good or optimal tour for a particular vehicle, with a particular set of customers assigned to it. That is, the sets of 'Pocket Boltzmann machines are used to provide cost information which supports the assignment of customers to vehicles, and in so doing they solve the vehicle routing problem component of the DCAVRP.

6.2 Overview of the Design

This section provides an overview of the design of the CONNEKT object-oriented application and introduces and places in context two of the key components, the Connectionist Expert System and the 'Pocket' Boltzmann machine. These are described in detail in sections 6.4 and 6.5

As has been stated previously, CONNEKT is a graphical-user-interface based object-oriented application, which allows the user to model, store (and retrieve), and solve examples of Dynamic Customer Assignment and Vehicle Routing Problems.

The user is able to define a particular problem example by entering customer and vehicle details using dedicated graphical data entry panels. These are used to create a set of customer objects, a set of vehicle objects and a set of location objects, each with their own attributes and behaviour. The objects are persistent and may be stored and the problem example retrieved for reuse at some later time. Many different problem examples can be simultaneously stored.

The user selects options from the system menu to control the operation of the application. These include options to store and retrieve the contents of a model, to add and edit customer and vehicle details and those concerned with the generation of solutions to the example of the problem currently being modelled.

All of the entities modelled by CONNEKT, that is customers, vehicles and locations are represented and stored as distinct objects, each with its own attributes and behaviour. Each customer entity is modelled by an instance of the class CCustomer, whilst each vehicle is represented by an object instantiated from the class CVehicle.

CCustomer objects have a number of attributes specified by the user. These include the customer's name, address and telephone number details and the coordinates of

their collection and destination locations. The CONNEKT data entry screen that supports the entry and editing of CCustomer details is described in figure 6.1

Figure 6.1 : The CONNEKT Customer Entry/Editing Screen

The screenshot shows a window titled "CNKT Windows Application - [CDCOV1.NET.2]" with a menu bar containing "File", "Edit", "View", "Tour", "Window", "Cancel", "Parameters", and "Help". Below the menu bar is a toolbar with various icons. The main area is titled "CUSTOMER DATA ENTRY FORM" and contains the following fields and buttons:

- Cust No.:** 1
- Object Identity(OID):** 3
- Name:** Mr A Smith
- Address:** 1, Cornwall St
- Town:** Birmingham
- County:** West Midlands
- Postcode:** Outer: B3, Inner: ZDT
- Collection:** 1.3
- Destination:** 2
- Cons.OID:** 0
- Time:** 0

Buttons on the right side of the form include: Add, Update, Clear, Delete, Prev, Next, Home, and End.

CVehicle objects similarly have a number of attributes into which the user-specified identification details can be stored. In addition the user is able to specify whether the vehicle is available, the vehicle's capacity and the co-ordinates of the vehicle's initial and final locations. The CONNEKT data entry screen that supports the entry and editing of CVehicle details is described overleaf in figure 6.2

The location attributes for both CCustomer and CVehicle object instances are references to CLocation objects, each of which has a pair of data members which store the 'X' and 'Y' component values of the Cartesian co-ordinate, representing the object's location.

Figure 6.2 : The CONNEKT Vehicle Entry/Editing Screen

Vehicle Data Entry Form

Registration No.	<input type="text" value="L123ABC"/>	<input type="button" value="Add"/>
Make	<input type="text" value="Leyland Def"/>	<input type="button" value="Update"/>
Model	<input type="text" value="400 Series"/>	<input type="button" value="Clear"/>
Cost Per Mile (£)	<input type="text" value="12.6"/>	<input type="button" value="Delete"/>
In Service	<input type="text" value="Y"/>	<input type="button" value="Home"/> <input type="button" value="End"/>
Capacity	Initial <input type="text" value="6"/> Max <input type="text" value="8"/>	<input type="button" value="Prev"/> <input type="button" value="Next"/>
Object Id (OID)	<input type="text" value="6"/>	
Current Position	X <input type="text" value="2"/> Y <input type="text" value="0.25"/>	

The components of CONNEKT that model and solve the particular DCAVRP example, are similarly object instances. The connectionist expert system is an instance of the class CConex and the 'Pocket' Boltzmann machines, instantiated using the class CPocketBM.

In all cases the object's class defines the attribute types and behaviour of the particular object instance. For the remainder of this chapter, the object instances and classes described thus far, will collectively be respectively described as *application objects* and *application classes*. All application object instances are assigned a unique object identity on instantiation. All application object classes are derived (and hence inherit the behaviour and attributes) from the Microsoft Foundation Class library (MFC) v3.0 class CObject. This provides all of the application objects with a set of default attributes and behaviour. In particular, this ensures that all application object instances can be relatively easily stored (serialized) and retrieved (deserialized), and can be elements of various MFC collection classes, for example *COBArray*.

The MFC library is a C++ class library developed by Microsoft [1993a] to assist in the development of object-oriented applications for the Windows v3.1 environment

and Windows95 and Windows NT operating systems. Its principal role is to provide a number of classes whose member functions act as 'wrappers', so named because their functions en-wrap (or hide) the native Windows Application Programming Interface (API) functions, written in the C programming language, from the object-oriented C++ developer. The MFC class library also provides a number of classes to assist in the development of graphical user interface applications. Many of these classes form the common constituents of any Windows application developed using Microsoft Visual C++. These graphical classes and object instances are principally used to implement the graphical user interface. They are not directly related to the functionality of the application, although obviously support it, for example by supporting the display of data within particular window view.

For the purpose of this thesis, all non-application classes and non-application object instances will be respectively denoted by the generic terms, *system class* and *system object instance*. Detailed description of the attributes and behaviour of the system classes supporting the development and operation of Windows applications using Microsoft Visual C++ are provided by Microsoft [1993a], Kruglinski [1993] and Barkakati [1993].

The application operates through the collective action and interaction of application and system object instances. In general, the latter will not be described further, since they are not directly related to the application of the connectionist approach to the DCAVRP, but merely support its object-oriented implementation in the Microsoft Windows NT v4.0 environment.

The problem solving capability of CONNEKT is modelled by the connectionist expert system application class *CConex* and the 'Pocket Boltzmann machine class *CPocketBM*. A detailed description of the design, implementation and operation of which, are the subject of the sections that immediately follow.

6.3 Operation

The steps involved in the operation of CONNEKT, to solve a particular DCAVRP example are described below.

Phase A : Initialization

<u>Step</u>	<u>Description</u>
--------------------	---------------------------

- | | |
|-----|---|
| 1.0 | The first step is for the user to either enter a set of vehicles and customers using the data entry screens displayed in figure 6.1 and 6.2, or to retrieve a previously defined problem (from hard-disk storage), using the 'Open' sub-menu option on the 'File' menu. |
| 2.0 | Once the entities from the problem domain have been entered or retrieved, the system is instructed to develop initial valid tours (see step 3.0) for each of the available vehicles by the user selecting the ' <i>Generate Initial Tours</i> ' sub-menu option from the <i>Tours</i> option on the application menu bar. |
| 3.0 | Given some (random or user-specified) assignment of customers to vehicles, a 'Pocket' Boltzmann machine (described in section 6.5) object is generated for each of the available vehicles in the fleet. |

Each 'Pocket' Boltzmann instance attempts to find an optimal tour from the vehicle depot, visiting the collection point locations, and the destination point locations for all of the customers currently assigned to its corresponding vehicle, before returning to the vehicle's depot.

Note that the 'Pocket' Boltzmann machines attempt to develop the shortest valid tour that additionally satisfies a variable number of additional *hard* or *mandatory* constraints. If they fail to satisfy these constraints they are restarted, and the operation repeated.

- 3.0 cont. In the case of the DCAVRP, the only additional constraint is that a tour is only considered to be valid, if for each customer, the tour visits the customer's collection location, before it visits the customers destination location.
- 4.0 Once an initial valid tour ($t=0$) has been generated for each of the m available vehicles, described by the set of cyclic permutations $\Pi_0 = \{\pi_{0,1}, \pi_{0,2}, \dots, \pi_{0,m}\}$, the user selects a menu option which increments the system time ($t=t+1$) and allows new stochastic, customers with random locations to be introduced into the system.
- 5.0 At time $t=1$, the system then attempts to optimally assign as many as possible of these new customers to the available vehicles in the fleet, in such a way as to minimize the value of the total cost function (TCF) given by equation 2.4.1 (reproduced below):

$$g(\Pi_t) = f(\pi_{t,1}) + f(\pi_{t,2}) + \dots + f(\pi_{t,m}) \quad [2.4.1]$$

It does this by generating a connectionist expert system object (described in section 6.4) whose structure and connectivity models the current state of the particular problem example under investigation.

- 5.1 The final step in the Initialization phase is to set the majority of connection weights in the connectionist expert system object, to their pre-determined initial values (described in section 6.4).

Phase B : Inferencing

- 6.0 The connectionist expert system implements a set of inferencing cycles in its attempt to optimally assign each of the customers to a vehicle.

6.0
cont.

A sub-set of the connectionist expert system object's connection weights are set using values calculated by generating a set of 'Pocket' Boltzmann machines. These attempt to determine the smallest incremental cost of assigning a particular customer to each of the vehicles.

If the existing tour for a particular vehicle, k , at time t is given by the cyclic permutation $\pi_{t,k}$ and if $\pi_{t+1,k}$ denotes the cyclic permutation corresponding to the best valid tour developed by a 'Pocket' Boltzmann machine, given that a new customer, c_j , has been temporarily assigned to the vehicle; then the incremental cost of temporarily assigning that customer is $I_{t+1,k}$, as defined in equation 2.5.5 (reproduced below):

$$I_{t+1,k} = f(\pi_{t+1,k}) - f(\pi_{t,k}) \quad [2.5.5]$$

where: $f(\pi_{t,k})$ is the cost of the tour for the k th vehicle at time t and
 $f(\pi_{t+1,k})$ is the cost of the tour for the k th vehicle at time $t+1$

6.1 If there are l customers that require assignment to one of m available vehicles, then the process described in step 6.0, is repeated for all of the possible assignments of customers to vehicles and the incremental cost of the assignment stored in the corresponding row and column of the $l \times m$ incremental cost matrix I .

(A potential improvement to the system would be to use simple geometry and a Branch and Bound algorithm to ensure that 'Pocket' Boltzmann machines are not generated unnecessarily, that is, for potential assignments that could not possibly have a lower incremental cost than the one with the lowest value so far.)

- 6.1 cont. In which case, where it is clear that the incremental cost of assigning a particular customer to a particular vehicle, at the closest point on the vehicle's tour, must be greater than the lowest cost so far of assigning any customer to any vehicle, the cost could be set to a predefined maximum value, I_{\max} . The value of I_{\max} is chosen such that it is greater than all possible values of $I_{t+1,k} \quad \forall t, \forall k$.)
- 7.0 As is described in detail in section 6.4, the value of a sub-set of the connection weights of the connectionist expert system are then set using the values of the elements of the incremental cost matrix, **I**.
- 8.0 Once the connection weights have been set, the inferencing cycle commences.
- 8.1 The connectionist expert system object first selects the unassigned customer that can be assigned to *any* vehicle with remaining unfilled capacity, for the least incremental cost.
- 8.2 It then assigns that customer, represented by the *j*th customer, to the vehicle (represented by the *k*th vehicle), which corresponds to the least incremental cost value.
- 8.3 The network records that the customer has been assigned and is henceforth unavailable by setting the state of the output and assignment units associated with that customer/vehicle combination to one.

8.4 Finally, the system reads the output states of the connectionist expert system network object to determine which customer has been assigned to which vehicle. The current tour for that vehicle at time t , $\pi_{t,k}$, is then replaced by the new provisional valid tour for that vehicle at time $t+1$, $\pi_{t+1,k}$. This is the valid tour generated by the 'Pocket' Boltzmann machine when it determined the incremental cost of assigning the customer to that vehicle.

Hence the new state of the problem, as modelled by the system reflects the fact that the i th customer has been assigned to the k th vehicle, and that the k th vehicle has a new very good or (near-) optimal valid tour which includes the location(s) associated with the new customer.

9.0 Before the next customer can be assigned, the connectionist expert system's connection weights must be updated to take account of the k th vehicle's new tour. The incremental cost data generated by the previous set of 'Pocket' Boltzmann machines is still valid, except for those values which correspond to the proposed assignment of customers to the k th vehicle. Hence only the incremental cost data in the k th column of the matrix **I**, and excluding the element in the j th row, need be recalculated

9.1 Steps 7.0 and 7.1 are therefore repeated but only for the provisional assignment of the remaining unassigned customers to the k th vehicle, hence updating only the elements in the k th column of the incremental cost matrix **I**.

10.0 The elements of the incremental cost matrix, **I** are used to reset a sub-set of the connection weights in the connectionist expert system object.

- 11.0 Steps 8.0 to 10.0 are then repeated and the connectionist expert system continues to select the unassigned customer that can be assigned to a vehicle, for the least increase in the value of the TCF, until either all customers are assigned or until all remaining available capacity is exhausted.

- 12.0 Once the system has assigned all of the current set of new, stochastic, un-split demand customers, the user can re-select the menu option, and the whole process can be repeated. That is, steps 3.0 to 11.0 can then be repeated until all remaining vehicle capacity is exhausted or all customers have been assigned or delivered to their destinations.

6.4 The Connectionist Expert System

6.4.1 Introduction

The connectionist expert system implemented in CONNEKT, uses customer availability and vehicle tour, assignment and capacity information to attempt to optimally assign each one of a set of new un-split demand customers to one of a fleet of available capacitated vehicles. The connectionist expert system is based on Gallant's Matrix Controlled Inference Engine (MACIE) (Gallant [1985,1988a,1993]). As in a number of the networks developed by Gallant, each of the processing units is a perceptron which represents a real world entity or concept. The topology and connectivity of the network models the current state of the example under investigation.

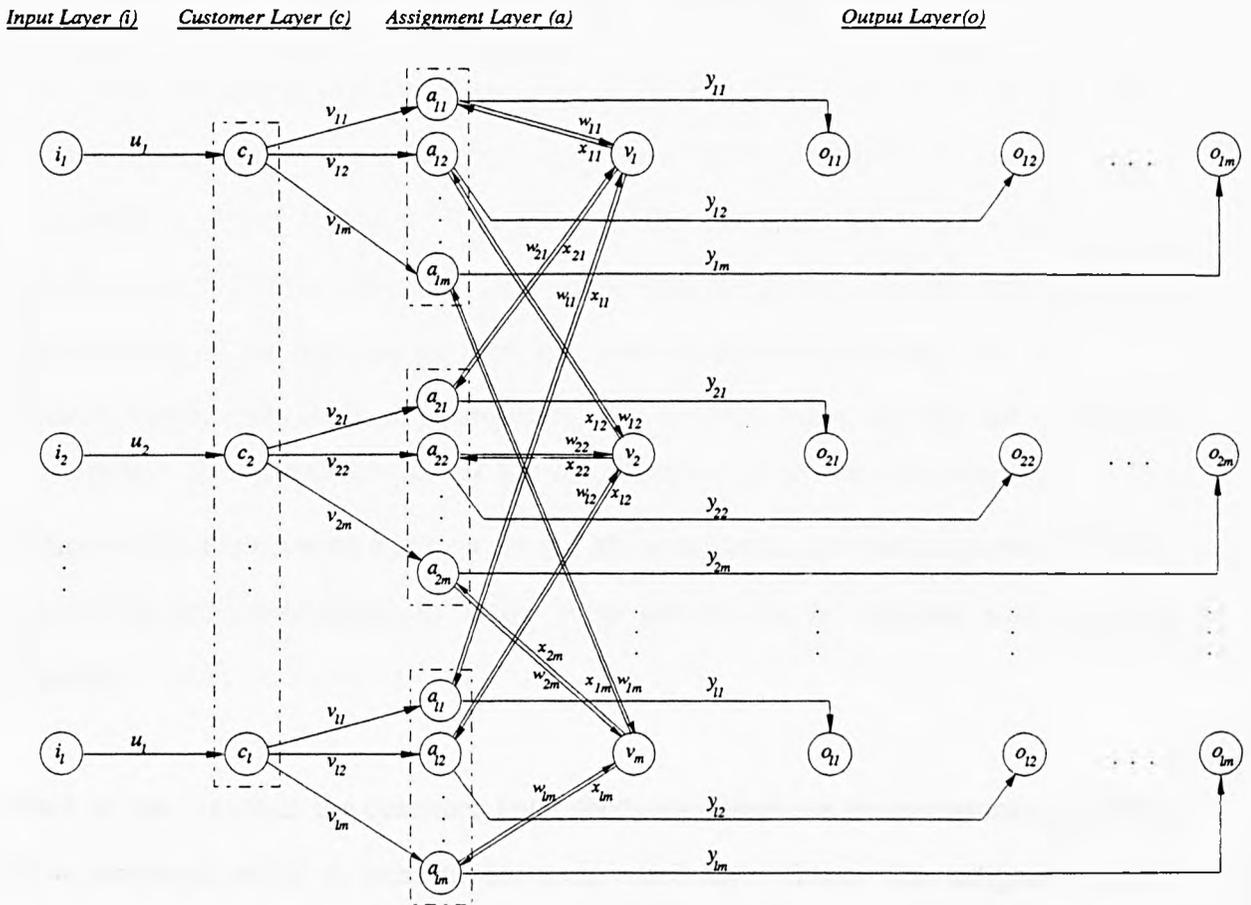
The connectionist expert system network is, in principle, able to assign any number of new customers to any number of vehicles. In practice, when implemented on a single processor machine, the system is constrained by the processing time taken to calculate the incremental cost of assigning customers to vehicles.

The sections that follow describe the topology and connectivity of the network and the network generation procedure. Subsequent sections describe the cell properties and the operation of the network.

6.4.2 Description of Network Topology and Network Generation

The novel connectionist expert system network (illustrated in figure 6.3) that infers optimal or very good assignments of multiple customers to multiple vehicles is a 5-layer, partially-connected, feedforward, perceptron-based network, which contains a set of feedback, connections between two of its layers. The network is comprised of discrete, binary valued $\{0,1\}$ units, which are arranged to form an input layer, an output layer and three intermediary or hidden layers.

Figure 6.3 : Topology of the CONNEKT Connectionist Expert System



If the number of new customers that the network is required to attempt to assign is l , and the number of vehicles with available capacity is m , the network generation procedure (CConex::GenerateNetwork(void)) creates a network with an input layer, i , comprised of l units. Each input layer unit represents a single customer. The firing state of the j th input layer unit, u_j , represents the availability of the j th customer. If the firing state, $u_j = 1$ this indicates that the j th customer is available, whilst $u_j = 0$ indicates that the j th customer has been assigned to a vehicle and hence is deemed to be unavailable.

The external environment influences the network, by mapping the values of the elements of a binary $\{0,1\}$ input vector $\vec{I} = (i_1, i_2, \dots, i_l)$ onto the corresponding units in the input layer. Each element of the input vector represents whether a particular new

customer, i_j , needs to be assigned ($i_j = 1$), or (from examination of units in the output layer), has already been assigned ($i_j = 0$). Each of the input layer units has an excitatory connection to a corresponding unit in the first of the three hidden layers, the customer layer, c . As in the input layer, each of the l units in the customer layer represents a single customer. The units in the customer layer are arranged (via inhibitory intra-layer connections) in a winner-take-all group. This ensures that given the mapping of the input vector onto the network input layer units, only one of the customer layer units is active (firing state = +1) at any time, and that all other units are inactive (firing states = 0). If the j th customer layer unit is active ($u_{e_j} = 1$) this indicates that the network has selected the j th customer as the best customer to assign, and is currently attempting to assign it to one of the m vehicles with available capacity.

Each of the l units in the customer layer feeds its output via an excitatory connection to a dedicated set of m units in the assignment layer, hence the assignment layer contains a total of $l \times m$ units. Each of the units in the assignment layer represents the assignment of a particular customer to a particular vehicle. Since a customer can only be assigned to a single vehicle, the units in the assignment layer that each customer unit is connected to are arranged into a winner-take-all group. If the j th customer layer unit is active, the m units in the j th winner-take-all group compete and only one of them becomes active. If the single unit that becomes active is the k th unit of the j th winner take all group, that is, $a_{jk} = 1$, this indicates that the j th customer has been assigned to the k th vehicle.

Since at any time, only one customer layer unit is active, and for each active customer unit, only one assignment unit can be active, only one unit in the assignment layer is active at any time.

Each of the assignment layer units, a_{jk} where $j = 1, \dots, l, k = 1, \dots, m$, feeds its output via an excitatory connection to a corresponding output layer unit, o_{jk} where $j = 1, \dots, l, k = 1, \dots, m$ which records the output from the network. Activation of a particular assignment layer unit, representing that the j th customer has been assigned to the k th vehicle automatically causes the corresponding output unit to fire (become active). The output layer units are arranged so that once they have become active they remain active, even if the associated assignment units state is reset (to an inactive state). This enables the output units to progressively record the assignment of customers to vehicles. For convenience the output units are considered to be arranged in a grid of l rows, such that the j th row in the grid represents the j th customer. Each of the rows contains m columns, and the k th unit in each row, that is, the unit in the k th column, represents the k th vehicle.

Each of the assignment units has a *potentially* excitatory connection with a single unit in the third hidden layer, the vehicle layer, v . Each of the m units in the vehicle layer represents a particular vehicle. The assignment layer unit representing the k th vehicle in each of the l winner-takes-all groups in the assignment layer, is connected to the unit representing the k th vehicle in the vehicle layer. The connection weight is described as *potentially* excitatory because its strength is determined dynamically and chosen from the set $\{0, +1\}$. The connection weight is set to $+1$ if and only if the *destination location* of the customer associated with the assignment layer unit is the vehicles depot, and the customer has both been assigned to the vehicle and sufficient time has elapsed for the customer to have been *collected*, otherwise the connection weight is set to 0.

The threshold value of each of the units in the vehicle layer are set to a value that is just less (e.g. 0.1) than the maximum capacity of that vehicle. This means that as customers are assigned to the vehicle the assignment units representing the assignment of that customer to the vehicle will become activated. However the connection to the

vehicle unit will remain at 0 unless the customer's destination is the vehicle's depot and until the customer has been collected. Although unlikely, it is possible in principle for this to occur for a full complement of customers assigned to the vehicle. In this case, until all of the customers are collected, the vehicle remains potentially available, since the 'Pocket' Boltzmann machine may be able to assign a customer that has both its collection and destination locations occurring before the collection point(s) for one or more of the 'depot bound', previously assigned customers. Now as each of the assigned customers whose destination point is the vehicle's depot are actually collected, the connection weight from the associated assignment unit to the vehicle layer unit representing the vehicle, is switched from 0 to +1. The assignment layer unit therefore exerts an excitatory influence on the vehicle layer unit. Once the vehicle has visited the collection points for a full complement of customers whose destination point is the depot, the vehicle layer unit receives a sufficiently strong excitatory signal to just overcome the threshold value, and the vehicle layer unit becomes active.

Each vehicle layer unit feeds its output via a set of strongly inhibitory connections back to each of the assignment layer units that are able to stimulate it. The units in the vehicle layer are arranged so that once they receive sufficient stimulation to become active, they remain active. The inhibitory connections are sufficiently strong to ensure that once the vehicle unit becomes active, indicating that it already has a full complement of customers for the remainder of its tour, the assignment units for that vehicle become and are held inactive. This ensures that once the vehicle has reached its capacity, the vehicle is no longer considered for any further assignments, and the remaining vehicles compete for any available customers.

In this way the network ensures that (in the relatively unlikely situation) that a vehicle has assigned to it; and has already collected a full complement of customers whose destination locations are the vehicle's depot, then the vehicle is deemed to be unavailable and no further assignments are attempted.

Whilst the vehicle layer unit is inactive, the network may still consider assigning customers to its associated vehicle. In which case a 'Pocket' Boltzmann machine may be generated which attempts to determine the lowest cost of assigning a particular new customer to the specific vehicle. The 'Pocket' Boltzmann machine attempts to determine the shortest new tour for the vehicle, assuming that the new customer is assigned to that vehicle which in addition to satisfying the constraints of the TSP, also satisfies both of the following *hard* (mandatory) constraints. Firstly, that the tour sequence is such that, for each customer the tour visits its collection location before it visits its destination location, and secondly that at no point on the tour is the capacity of the vehicle exceeded. Tours that satisfy both the constraints imposed by the TSP and the two hard constraints described above are called *valid* tours.

Tours that satisfy the first mandatory constraint may still be invalid if the addition of a collection location at a particular point on the tour exceeds the vehicle's capacity at that point. Such a tour is rejected by the 'Pocket' algorithm, since it fails to satisfy the second of the additional hard constraints. Clearly the addition of the potential new customer's collection location at a later point on the tour, after at least one of the current customers destination locations have been visited, could satisfy both *hard* constraints, and hence constitutes a *valid* tour. It is the responsibility of the 'Pocket' Boltzmann machine to attempt to find the shortest valid tour.

6.4.3 Cell Properties, Connection Strengths and Threshold Values

In Gallant's perceptron based connectionist expert system networks [Gallant, 1993], the output (activation values of units are set in accordance with equations 4.2.1 and 4.2.2 (reproduced below):

$$u_i = f(S_i) \quad \text{where } S_i = \sum_{j \geq 0} w_{i,j} u_j \quad [4.2.1]$$

and:

$$u_i = f(S_i) = \begin{cases} +1 \text{ or } True & \text{if } S_i > 0 \\ -1 \text{ or } False & \text{if } S_i < 0 \\ 0 \text{ or } Unknown & \text{if } S_i = 0 \end{cases} \quad [4.2.2]$$

Where: $w_{i,j}$ is the strength of the connection between the j th unit and the i th unit, u_j is the output value of the j th unit and u_i is the output value of the i th unit. It is assumed, in equations 4.2.1 and 4.2.2, that the unit's threshold value is given by the connection weight w_o and that an additional unit, u_o is perpetually clamped to the value +1.

The output (activation) values of the binary $\{0,1\}$ units in the hidden and output layers are similarly calculated by applying equation 4.2.1, however the lower asymptote of the step threshold function is chosen to be 0, as opposed to -1. Hence, hidden and output layer units in the CONNEKT connectionist expert system network become active (output = 1) if the sum of the weighted inputs to the unit exceeds the units threshold value, and the output is zero otherwise.

The update rule for hidden and output units in the connectionist expert system, is therefore given by the expression:

$$u_i = f(S_i) = \begin{cases} +1 \text{ or } True & \text{if } S_i > 0 \\ 0 \text{ or } False & \text{otherwise} \end{cases} \quad \text{where } S_i = \sum_{j \geq 0} w_{i,j} u_j \quad [6.4.1]$$

At any time, t , a set of 'Pocket' Boltzmann machines compute the incremental cost of assigning each of the customers to each of the vehicles. (As stated previously, a potential improvement, reducing the number of 'Pocket' Boltzmann machines that are generated and the processing time, could be to use simple geometry and a branch and bound procedure to determine assignments that *must* be more expensive than the best so far. In this case the incremental cost would be set to some high value, I_{\max} (greater than any value that could be generated by a 'Pocket' Boltzmann machine.)

Given the incremental cost data, the values of the connections are then calculated according to equation 6.4.2.

$$\forall j = 1, \dots, l, \forall k = 1, \dots, m \quad u_j = (Inc_{\max} + 1) - \min(Inc_{jk}) \quad [6.4.2]$$

where: u_j is the weight of the connection from the j th input unit, u_j , to the j th customer unit, u_{c_j} ; Inc_{\max} is $\forall j, \forall k$ the maximum value of Inc_{jk} and $\min(Inc_{jk})$ is, $\forall k$ the minimum value of Inc_{jk} for the j th customer.

Equation 6.4.3 ensures that the connection between the input and customer units representing the customer that can be added to any of the available vehicles at the lowest incremental cost, has the largest the weight value.

The connection weights of each of the connections between the units in the customer layer and units in the assignment layer are set according to equation 6.4.3.

$$\forall j = 1, \dots, l, \forall k = 1, \dots, m \quad v_{jk} = 1 + \max(Inc_{jk}) - \min(Inc_{jk}) \quad [6.4.3]$$

where: v_{jk} is the strength of the connection between the j th customer unit, c_j , and the assignment unit, a_{jk} ; and $\max(Inc_{jk})$ is, $\forall k$ the maximum value of all Inc_{jk} for the j th customer.

Equation 6.4.4, ensures that for each customer, the largest connection weight is to the assignment unit, that represents the assignment of the customer to the vehicle which involves least incremental cost.

The strengths of the excitatory connections, w_{jk} from each of the assignment units, a_{jk} , to their associated units in the vehicle layer, v_k are simply set to unity, as

described by equation 6.4.4. This ensures that each assignment of a customer to a vehicle, increases the input to the corresponding vehicle unit by unity. As stated previously, a vehicle unit becomes active if the sum of its weighted inputs exceeds the units threshold value. Once a vehicle (or output) unit has become active it remains active until all inferencing cycles have been completed.

$$\forall j = 1, \dots, l, \forall k = 1, \dots, m \quad w_{jk} = 1 \quad [6.4.4]$$

The strengths of the inhibitory connections, x_{jk} from each of the vehicle units v_k , back to each of the assignment units, a_{jk} that influence the unit, are set to a negative value according to equation 6.4.5. The connection strength is sufficient to ensure that all of the assignment units associated with each of the vehicle units can be set to and maintained in an inactive state.

$$\forall j = 1, \dots, l, \forall k = 1, \dots, m \quad x_{jk} = -2 \quad [6.4.5]$$

The threshold value of each of the units in the vehicle layer are set to a value that is just less (by an amount $\lambda=0.1$) than the maximum capacity of that vehicle.

The threshold values of all units in the customer, assignment and output layers are set to zero. This ensures in combination with the perceptron firing rule described in equation 6.4.1 that a net excitatory stimulus (input) is sufficient to cause the unit to fire.

6.4.4 Network Operation (Inferencing)

As stated previously, given the constraints of processing power, available memory and time, the network assigns any number l of customers to any number of vehicles m , given sufficient available capacity in the vehicles. If there is insufficient capacity

for all customers to be assigned, the network assigns customers until the capacity of all of the vehicles has been reached.

In the initialisation phase, all of the units in the network are set to an inactive state. For all subsequent iterations of the inference assignment cycle, only the units in the input and customer layer are set to an inactive state. The final step in the initialisation phase involves setting all of the network connection strengths in accordance with equations 6.4.1 to 6.4.5.

The first step in the inference cycle involves setting the values of each of the l elements of the input vector $\vec{I} = (i_1, i_2, \dots, i_l)$ as described in equation 6.4.6.

$$\forall j \text{ where } j = 1, \dots, l, \quad \forall k \text{ where } k = 1, \dots, m, \quad \text{iff } o_{jk} = 0 \quad \text{then} \quad i_j = 1$$

$$\text{otherwise} \quad i_j = 0 \quad 6.4.6$$

The input vector stores a representation of the customers that are available for assignment. Equation 6.4.6 ensures that the j th input vector element is set to the value $i_j = 1$, indicating that the j th customer is available for assignment, provided that it has not been assigned to any of the m available vehicles, that is, that all of the units in the j th output vector row are inactive ($o_{jk} = 0$ for all $k = 1, \dots, m$). If one of the units in the j th row is active, then the customer has already been assigned and the value of the corresponding element in the input vector is set to zero, indicating that the customer is not available for assignment.

After the units and the connections have been set to their initial values, the network assigns customers by repeatedly performing an inference cycle which only stops when either all customers have been assigned or the capacity of all customers has been met.

The operation of the network is described overleaf.

- | <u>Step</u> | <u>Description</u> |
|--------------------|--|
| 1.0 | <u>Initialisation</u> |
| 1.1 | Set the firing states of each of the units in the network to inactive (0). |
| 1.2 | Set the values of each of the connection strengths in accordance with equations 6.4.1 to 6.4.5. |
| 2.0 | <u>Inferencing</u> - Repeat steps 2.1 to 2.9 until either of the termination criterion, described in steps 3.1 and 3.2 are satisfied. |
| 2.1 | Reset the firing states of the input and customer layer units to inactive (0). |
| 2.2 | Set the value of the elements of the input vector \vec{I} in accordance with equations 6.4.1 and 6.4.2. |
| 2.3 | Clamp the values of each of the input layer units to the value of the corresponding element in the input vector, that is set: $u_{i_j} = i_j \quad \forall j = 1, \dots, I$ |
| 2.4 | Allow the activations from the input layer units to propagate to the units in the customer layer via the set of connections, u |
| 2.5 | The units in the customer layer are arranged in a winner-takes-all group. The excitatory signals from the units in the input layer cause only the unit which has the largest connection strength to become active, all other units remain inactive. For (explanatory) convenience assume that the customer unit that becomes activated is the j th unit. |

In so doing the network has effectively selected the customer that could be added to a vehicle with the least incremental increase in the cost function.

2.6 The output from the activated customer unit, u_{c_j} , acts to stimulate each of the units in the j th winner-take-all group in the assignment layer. Setting the strength of the connections in accordance with equation 6.4.3 ensures that the greatest connection strength is for the connection to the assignment unit that represents the vehicle that this customer should be assigned to – for the smallest increase in the value of the cost function given a particular assignment strategy. The excitatory connections stimulate all of the units in the j th winner-take-all group, however only one of the units becomes activated, the one representing the vehicle that the j th customer can be assigned to for the smallest increase in the value of the cost function.

Now if the k th vehicle is the vehicle that the j th customer can be assigned to for least cost then the unit representing this assignment, that is a_{jk} becomes activated.

2.7 The activation of the assignment unit, a_{jk} , causes an excitatory signal to be passed via the connection, w_{jk} , to the output unit o_{jk} , representing the assignment of the j th customer to the k th vehicle. The threshold of all output units is zero, hence the stimulus causes the output unit, o_{jk} to become activated.

2.8 The threshold value of each of the units in the vehicle layer is set equal to the number of additional customers that the particular vehicle can accommodate. If the total excitatory input strength of the weighted connections to any of the vehicle units is (at least) equal to the threshold value of the unit, then the vehicle unit becomes activated.

2.9 The activation of any active vehicle layer units is fed back and inhibits the firing of all of the assignment units that correspond to the active vehicle layer units.

3.0 **Termination Criterion**

- 3.1 Stop performing inferencing and assignments if all of the units in the vehicle layer are activated, that is, if none of the vehicles have any unfilled capacity.
- 3.2 Stop performing inferencing and assignments if one unit in each row of the output grid is active, that is, if all customers have been assigned.

Once all of the units and connections have been set to their initial values the network repeatedly performs an inferencing assignment cycle until all of the customers are assigned or until all available capacity is exhausted.

6.5 The 'Pocket' Boltzmann machine

6.5.1 Introduction

The 'Pocket' Boltzmann machine is a novel modification of Aarts and Korst's [1989a, 1989b] implementation of the Boltzmann machine originally introduced by Ackley, Hinton and Sejnowski [1985]. The modification is inspired by Gallant's Pocket Algorithm (Gallant [1985, 1988a, 1993]) which Gallant has applied to the Perceptron and Back-propagation models.

The 'Pocket' modification stores a copy (in a 'pocket') of the 'best' solution generated by the Boltzmann machine, that is, the one that best satisfies both the *soft* constraints encoded in the Boltzmann machine network's pattern of connectivity and connection weight values; and a set of additional *hard* constraints. Each time a new solution is generated that satisfies both sets of constraints, it is compared with the solution stored in the pocket, and if superior, replaces it.

6.5.2 The 'Pocket' Boltzmann machine Algorithm

The 'Pocket' modification described below, and implemented in CONNEKT is applied to the sequential Boltzmann machine model described by Aarts and Korst [1989a, 1989b].

The 'Pocket' Boltzmann machine must attempt to find an optimal tour which visits each of the n locations in the set $L = \{l_1, \dots, l_n\}$, in such a way as to satisfy each member of a set of additional constraints $C = \{con_1, \dots, con_n\}$.

Step. Description

- 1.0 Create a Boltzmann machine with a topology and connectivity as described in section 3.4

- 2.0 Set the connection weight values using the co-ordinates of the locations in the set L , as described in section 3.4.4
- 3.0 Allow the units in the network to update their states
 - 3.1 If a unit in the network has changed state, determine if the current state of the network corresponds to a *valid* solution to the problem, that is one that satisfies both the soft constraints encoded by the network and each of the additional hard constraints in the set C .
 - 3.2 If the current state of the network corresponds to a valid solution, let $\pi_{current}$ denote the tour corresponding to the current state of the network and calculate the length of the tour $f(\pi_{current})$
 - 3.3 If the 'Pocket' is currently empty or if the length of the current tour $f(\pi_{current}) < f(\pi_{pocket})$, where π_{pocket} is the tour (if any) stored 'in the pocket', replace the tour in the pocket with the current tour.
- 4.0 Repeat steps 3.0 to 3.3 until the network has settled into a stable state.

6.5.3 Justification of the 'Pocket' Boltzmann machine Algorithm

A property of the Boltzmann machine network is that at any temperature in the annealing schedule, the network may visit and subsequently escape from the influence of many valid solutions in the solution space.

A Boltzmann machine is able to perform global optimization, provided that it is allowed to perform a sufficiently large number of transitions at each temperature, in a suitably slow annealing schedule. Hinton and Sejnowski [1986] explained this using

the analogy of a ball being shaken in a box with a bumpy surface. They stated that if the box was initially shaken with sufficient vigour to ensure that the ball can escape from any of the hollows in the bumpy surface; and that if the vigour with which the box is shaken is slowly decreased over time. Provided that before each decrease, the box is shaken for sufficient time for the probability of the ball settling in each possible hollow to be very close to 1; then eventually the box would be being shaken sufficiently vigorously for the ball to escape from all but the deepest hollow. Then the probability that the ball would settle in the deepest hollow and be unable to escape from it would be very high.

Aarts and Korst [1989b] have proven that, given the transition probability equations 3.5.8, 3.5.9 and 3.5.10, their sequential Boltzmann machine model converges asymptotically to a globally optimal state - as the control parameter, c , tends to zero, ($c \downarrow 0$), and the number of transitions (the length of the Markov chain) tends to infinity, ($k \rightarrow \infty$)

Unfortunately, the requirement that the length of the Markov chains, and hence the number of transitions at a give control parameter value (temperature) should be infinite, is clearly impractical. In practice, therefore, Aarts and Korst implement a finite-time approximation, which is able to find near-optimal solutions.

Aarts and Korst's finite time approximation is only able to find near-optimal solutions because, at each value of the control parameter (temperature) the network is allowed to make sufficient transitions for it to have a reasonably high probability of visiting states that correspond to very many of the solutions to the problem, including any optimal solutions.

Although Aarts and Korst [1989a] report that it is not proven that their asynchronous parallel Boltzmann machine model will converge, stating that:

"a rigorous proof of the asymptotic convergence of asynchronous parallel Boltzmann machines is considered an interesting open research topic."

They conjecture that the result of allowing asynchronous updates, that is erroneously calculated state transitions, does not affect the quality of the final result obtained, but may merely affect the speed of convergence. They also report that the results of computer simulations of their asynchronous parallel model fully support this conjecture, and demonstrate that the model is able to generate near-optimal solutions to examples of the TSP (Aarts and Korst, 1989a).

Aarts and Korst [1989a, 1989b] have therefore either proven, in the case of the former or demonstrated experimentally, in the case of the latter, that both their sequential and asynchronous parallel Boltzmann machine models can generate near-optimal solutions to examples of the TSP.

Now, if the set of solutions that are generated by either the sequential or asynchronous parallel Boltzmann machine models that satisfy the *soft* constraints encoded in the network are represented by the set S , and the set of solutions that also satisfy the additional *hard* constraints enforced by the 'Pocket' modification is denoted S_{All} , then clearly:

$$S_{All} \subseteq S \quad [6.5.1]$$

Now the 'Pocket' modification does not alter in any way the behaviour of the Boltzmann machine model, it simply selectively records and stores some of the results of the Boltzmann machines computation.

Since either of the Boltzmann machine models visits sufficient states and hence solutions, to make it very likely that the machine will visit and finally stabilise in a

state corresponding to a near-optimal solution; and since the set of solutions that also satisfy the additional hard constraints imposed by the 'Pocket' algorithm form a (possibly small) sub-set of these states, then provided that:

- the requirements for the Aarts and Korst's finite-time approximation are satisfied; and
- the additional hard constraints imposed by the 'Pocket' algorithm are not so stringent as to dramatically reduce the number of solutions to the point where they are so few, that it becomes unlikely that a Boltzmann machine would visit them.

Then as the 'Pocket' Boltzmann machine visits all of the same states that a Boltzmann machine without the 'Pocket' modification would visit; given a suitable annealing schedule, it is conjectured that it is likely that the 'Pocket' Boltzmann machine will fall under the influence of and visit sufficient states to obtain a very good or near-optimal solution.

Finally, since the 'Pocket' algorithm always stores the 'best ' solution to date, if a state corresponding to an optimal or near-optimal solution is visited, at any time, then the system will return this state, unless an even better one is found

Hence, for the same reasons that it is likely that either Aarts and Korst's [1989a, 1989b] sequential Boltzmann machine or an asynchronous parallel Boltzmann machine model will generate near-optimal solutions to the TSP, it is also likely, given a suitable annealing schedule, that either a sequential or asynchronous parallel 'Pocket' Boltzmann machine will generate near-optimal solutions that additionally satisfy a number of hard constraints.

6.6 Implementation

6.6.1 Selection of a suitable Development Environment

This section firstly describes the criterion used to support the selection of a suitable hardware and software development environment, and then explains and justifies the selections that were subsequently made.

Two principal criterion were developed to assist in the selection of the hardware platform, these were:

- *Suitability* including: adequate processing power, range and availability of software development tools; and the degree of similarity with the platforms used by other researchers - for ease of comparison
- *Availability*

Potentially available platforms included an IBM AS400 mid-range computer, a Motorola M6000 Unix processor and an IBM PC-AT compatible personal computer (PC).

Simulations of neural networks models could be developed on any of the platforms. However, access to the AS400 and Motorola machines could only be gained at the site of the author's employer and was only available outside of 'office hours'. The PC was therefore selected as it provided adequate processing power, and was the most flexible of the potentially available alternatives, both in terms of suitability and availability.

A number of criterion were developed to assist with the selection of a suitable software development tool. These included the following:

- The development tool should support the development of an object-oriented application, as this software paradigm shares a number of features in common with connectionism.

- The application should be developed to run on a personal computer (since this is the only readily available platform).
- The application should provide the capability to easily model and store, for subsequent retrieval, examples of dynamic customer assignment and vehicle routing problems.
- The development language should be both powerful and flexible.
- The development tool should be both affordable and be well supported.
- If possible, the development tool should support the future migration of the application onto other platforms, including those with multi-processor architectures.
- If possible, and provided that all of the other criterion can still be satisfied, an industry standard development tool should be selected since this may increase the commercial attractiveness of the research.

C++ was selected as the most suitable development language, principally because of its object-oriented character, but also because of the language's power and flexibility. Three C++ development tools were initially identified, these were: Microsoft C/C++ v7.0, Borland's Application Frameworks v3.1 and Microsoft Visual C++ v1.0. Each of the tools was carefully evaluated.

Microsoft Visual C++ was finally selected as it satisfied all of the criterion, including offering the opportunity to easily migrate the application to multi-processor platforms running the Microsoft NT multi-threaded, graphical operating system. In addition, Visual C++ includes a comprehensive set of integrated development and debugging tools which, it was judged would most effectively reduce the time taken to develop the application.

6.6.2 The Development Environment and Implementation and Operation of CONNEKT

CONNEKT is an object-oriented application, developed using Microsoft Visual C++ (v2.0) to run on a personal computer (PC) under the Microsoft Windows NT 4.0 operating system.

Visual C++ [Microsoft 1993a, 1993b, 1993c, 1993d; Shammas, 1993; Barkakati 1993; Kruglinski 1993] is a powerful and flexible *application framework* developed by Microsoft to assist programmers with the development of Windows applications. The application framework is comprised of a *Programmers Workbench* and a set of C++ class libraries, which are collectively referred to as the *Microsoft Foundation Classes*.

A complete description of Visual C++ is clearly beyond the scope of this thesis. This section provides a description of the development environment only in so far as it places in context the description of the implementation the application specific features of CONNEKT (described in section 6.5.3). The interested reader should therefore refer to the sources referenced above for a detailed description of Visual C++ and the MFC library.

The Programmers Workbench provides a number of facilities that enable the developer to quickly develop and debug, simple, executable Windows programming shells and to extend the basic shell application. The developer uses the C++ programming language, to create his or her own application specific, classes and objects instances. These typically interact with instances of MFC classes, for example, to display the attributes of an object instance in a Window or to allow the user to trigger particular behaviour by clicking on a button on the screen or by selecting an option from the application menu.

The MFC classes provide many features which assist the developer in the development of Microsoft Windows based object-oriented applications. These include [Kruglinski, 1993]:

- " • Collection classes for lists, arrays, and maps
- A useful and efficient string class
- Time, time span, and date classes
- File access classes for operating system independence
- Support for systematic object storage and retrieval to and from disk.
- A 'common root object' class hierarchy
- Streamlined Multiple Document Interface (MDI) application support
- Effective support for OLE (Object Linking and Embedding)"

and:

- " • Full support for File Open, Save, and Save As menu items with the most recently used file list
- Print preview and printer support
- Scrolling windows and splitter windows
- Toolbars and status bars
- Access to Microsoft Visual Basic controls
- Context sensitive help
- Automatic processing of data entered in a dialog box
- An easy to program interface to OLE
- DLL support " (that is support for Dynamic Link Libraries which allow functions from one application to access the functions of another application).

All standard Visual C++ applications have a single application object instance which is derived from the MFC class **CWinApp**. This class encapsulates the code needed to initialize, run and terminate the application. Applications can have one or more

document objects, derived from the MFC class **CDocument** and one or more view objects, derived from the class **CView**, each of which is attached to the document.

Document objects manage the applications data, including supporting the serialization and deserialization of document object attributes. View objects, display a documents data and allow users to interact with it. The customer and vehicle data entry and editing screens illustrated in figures 6.1 and 6.2 are instances of two view classes, **CCustView** and **CVehView** derived from the MFC class **CFormView**, itself a particular derivative of **CView**.

The user can define a particular problem example by entering customer and vehicle data into CONNEKT. The user does this by selecting either 'Customer' or 'Vehicle' from the View menu option. A member function associated with the particular menu option causes a either a **CCustView** or **CVehView** object instance to be created. This causes a data entry/editing form view to be displayed on the screen. The user can then either enter new data, and click on the *Add* button (see figures 6.1. or 6.2) or edit an existing customer or vehicle. If the user adds a new item, this causes a new **CCustomer** or **CVehicle** object to be dynamically created, A pointer to the object is stored in a **CCnktDoc** data member of type **CObList**. This is a linked list object which, in conjunction with functionality built into **CCustView** and **CVehView** allows the user to scroll through and edit or add a new entity at any point in the list.

Once the user has entered customer and vehicle details and hence defined a particular problem instance, this can be stored (on the PC's hard disk) by selecting 'Save' or 'Save As' from the File menu option. The user can also load an alternative (previously defined problem example) selecting 'Open' from the 'File' menu.

Given that a problem example has been loaded, the user instructs the system to generate an initial tour by selecting an eponymously named option from the 'Tour' menu option. The member function associated with this menu option, sequentially

creates a set of 'Pocket' Boltzmann machine objects, instantiated from class *CPocketBM*. Each *CPocketBM* object instances is passed an array of references to location objects, representing the current set of locations that the vehicle associated with the 'Pocket' Boltzmann machine instance..

Each 'Pocket' Boltzmann machine, then attempts to generate an optimal tour for one of the vehicles in the fleet. They do this by implementing the procedure described in section 6.5.

The locations of customers and the current and best tour generated so far by each of the Pocket Boltzmann machines is displayed on screen via an instance of the class *CCnktView* (derived from the MFC *CView* class).

Once the system has generated initial tours for each of the available vehicles, the user can instruct the system to introduce new, currently unassigned customers, by selecting the 'Introduce New Customers' option from the 'Tour' menu.

The member function associated with the 'Introduce New Customers' sub-menu option first creates a connectionist expert system object, instantiated from the class *CConex*. The structure and the connectivity of the network models the current state of the problem, specifically the number and identity of unassigned customers and available vehicles.

A set of 'Pocket' Boltzmann machines are then generated to determine the incremental cost of assigning each of the customers to each of the vehicles.

A side effect of calculating the incremental cost of assigning each of the customers to each of the vehicles is that a new provisional tour for each vehicle is developed, which assumes that the customer is permanently assigned to that vehicle.

The system stores the new provisional tour, developed by the 'Pocket' Boltzmann machine which returns the lowest incremental cost value. This is stored to support the subsequent replacement of the existing tour for that vehicle with its new tour value.

The incremental cost data is stored in a matrix (an instance of the class *CMatrix*) and used to calculate a sub-set of the connection weights in the connectionist expert system object (the *CMatrix* class is based on a matrix class originally developed by Blum [1992]). The *CConex* object then implements the procedure described in section 6.4 and in so doing assigns as many customers as possible, or all customers given sufficient available capacity.

Once the customers have been assigned and the new tours for each vehicle displayed the user can re-select the 'Introduce New Customers' option from the 'Tours' menu.

This process can be repeated until all remaining capacity in each of the vehicles has been exhausted or there are no further customers (at subsequent times) requiring assignment.

6.7 Validation of the Pocket Boltzmann machine implementation

6.7.1 Introduction

The Pocket Boltzmann machine component of CONNEKT was developed by implementing the sequential Boltzmann machine model described by Aarts and Korst [1989a, 1989b] and extending it to apply a 'Pocket Algorithm', based on that developed by Stephen Gallant [1985, 1988a, 1993].

To validate the Boltzmann machine implementation, the Pocket Boltzmann machine was applied to Hopfield and Tank's 10-city problem and its performance compared with that published by Aarts and Korst [1989a] for their asynchronous parallel Boltzmann machine model.

Unfortunately, neither Hopfield and Tank [1985] nor Aarts and Korst [1989a] published the co-ordinates of the cities in the 10-city problem. The co-ordinates used here and listed in table 6.1 below, are those determined and published by Wilson and Pawley [1988].

<u>City</u>	<u>X-Coord</u>	<u>Y-Coord</u>
0	0.8732	0.6536
1	0.5171	0.9414
2	0.2293	0.7610
3	0.400	0.4439
4	0.1707	0.2293
5	0.2439	0.1463
6	0.6195	0.2634
7	0.6683	0.2536
8	0.8488	0.3609
9	0.6878	0.5219

Table 6.1: Co-ordinates of the cities in Hopfield and Tank's [1985] 10-city problem as determined by Wilson and Pawley [1988]. N.B. City labels are those attributed by Aarts and Korst [1989a].

The problem was formulated by implementing Aarts and Korst's [1989a] Linear Assignment Problem (LAP) formulation. All results were collected by running the Pocket Boltzmann machine with the 'Pocket' modification disabled and using the run-time parameter values published by Aarts and Korst [1989a], that is $\alpha=0.95$, $L=10$ and $M=100$.

6.7.2 Additional Implementation Details

To the best of the authors knowledge, Aarts and Korst [1989a, 1989b] do not specify all of the information required to fully replicate their implementation of their model.

Specifically, in equation 3.4.23 (reproduced for convenience below), it is not known how much less the inhibitory connection strength should be in comparison with the value of the expression on the right-hand-side of the equation.

$$\forall \{u_{ij}, u_{kl}\} \in C_i : s(u_{ij}, u_{kl}) < -\min\{s(u_{ij}, u_{ij}), s(u_{kl}, u_{kl})\} \quad [3.4.23]$$

In the application of the Pocket Boltzmann machine model for which results are published below, equation 3.4.23 is implemented by subtracting an additional positive quantity θ , from the expression on the right-hand-side of the equation:

$$\forall \{u_{ij}, u_{kl}\} \in C_i : s(u_{ij}, u_{kl}) = -\min\{s(u_{ij}, u_{ij}), s(u_{kl}, u_{kl})\} - \theta \quad [3.4.24]$$

6.7.3 Results

The performance of the Pocket Boltzmann machine model with the 'Pocket' modification disabled, was investigated for different values of the tuning parameter θ . For each value of θ , the network was run 100 times. The results are summarised in tables 6.2a-6.2c overleaf.

The average elapsed time required to complete a single run (for $\theta = 0.7$), on a Pentium 90Mhz Personal Computer with the 'Pocket' modification disabled was 251 seconds.

Description of Statistic \ Results	0.05	0.1	0.2	0.3	0.4
Tuning Parameter (θ)	0.05	0.1	0.2	0.3	0.4
Sample Size (m)	100	100	100	100	100
No of Solutions (n)	0	0	15	9	32
Average Tour Length ($\bar{\ell}$)	N/A	N/A	3.172	3.078	2.860
Std. Dev. Of the Tour Length (σ)	N/A	N/A	0.076	0.006	0.132
Avg. Number of Iterations ($I=m/n$)	-	-	6.667	11.111	3.125
Smallest Observed Tour Length(ℓ_v)	N/A	N/A	2.898	3.062	2.691
Largest Observed Tour Length(ℓ^*)	N/A	N/A	3.192	3.083	3.115

Table 6.2a: Performance of the Pocket Boltzmann machine (pocket disabled) for different values of θ (in range 0.05-0.4).

Description of Statistic \ Results	0.5	0.6	0.7	0.8	0.9
Tuning Parameter (θ)	0.5	0.6	0.7	0.8	0.9
Sample Size (m)	100	100	100	100	100
No of Solutions (n)	63	79	90	88	81
Average Tour Length ($\bar{\ell}$)	2.933	2.888	2.843	2.814	2.768
Std. Dev. Of the Tour Length (σ)	0.148	0.154	0.128	0.085	0.009
Avg. Number of Iterations ($I=m/n$)	1.587	1.266	1.111	1.136	1.235
Smallest Observed Tour Length(ℓ_v)	2.691	2.691	2.691	2.691	2.691
Largest Observed Tour Length(ℓ^*)	3.332	3.240	3.240	3.115	2.769

Table 6.2b: Performance of the Pocket Boltzmann machine (pocket disabled) for different values of θ (in range 0.5-0.9).

Description of Statistic \ Results	1.0	1.1	1.2	1.3	1.4
Tuning Parameter (θ)	1.0	1.1	1.2	1.3	1.4
Sample Size (m)	100	100	100	100	100
No of Solutions (n)	23	7	5	5	2
Average Tour Length ($\bar{\ell}$)	2.767	2.771	2.841	2.741	2.691
Std. Dev. Of the Tour Length (σ)	0.017	0.003	0.203	0.046	0.000
Avg. Number of Iterations ($I=m/n$)	4.348	14.286	20.000	20.000	50.000
Smallest Observed Tour Length(ℓ_v)	2.691	2.769	2.691	2.691	2.691
Largest Observed Tour Length(ℓ^*)	2.778	2.778	3.198	2.778	2.691

Table 6.2c: Performance of the Pocket Boltzmann machine (pocket disabled) for different values of θ (in range 1.0-1.4).

6.7.4 Analysis of Results

The variation in performance of the Pocket Boltzmann machine, as the value of the tuning parameter θ was varied, is illustrated graphically in figures 6.4 and 6.5 overleaf. These graphs respectively describe, as a function of the tuning parameter θ , the variation in the average tour length ($\bar{\ell}$) and the frequency (n/m) with which the final state of the network corresponded to a valid tour.

Figure 6.4: Variation in the Average Tour Length (pocket modification disabled)

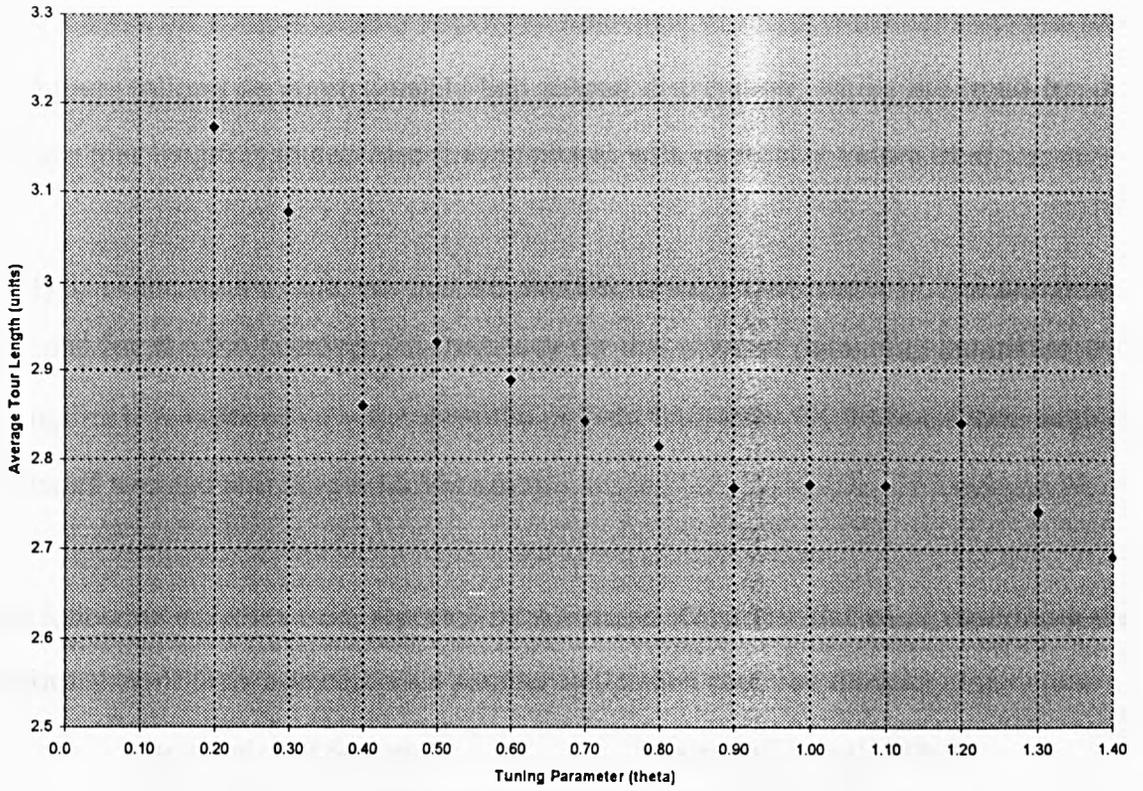
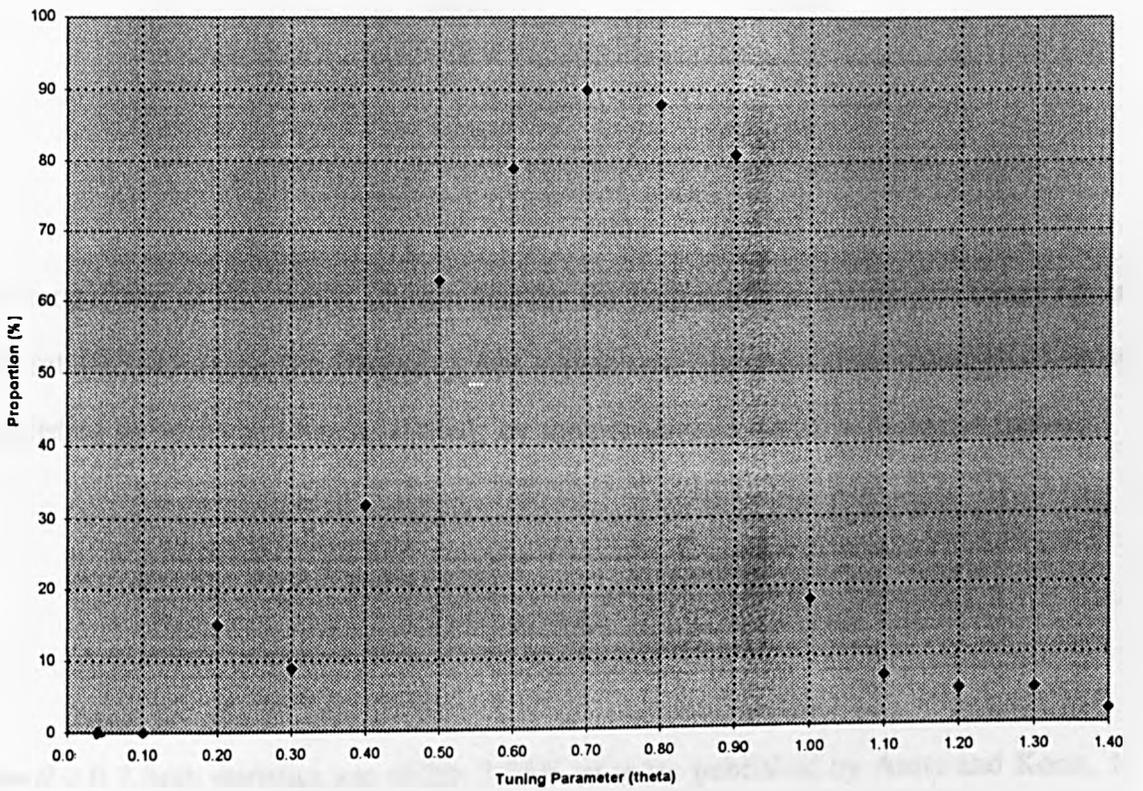


Figure 6.5: Variation in the frequency with which tours are obtained (pocket disabled)



The graphs clearly show that for values of the tuning parameter in the range $0.2 \leq \theta \leq 1.4$, the frequency with which the final state of the network corresponds to a valid tour, follows an approximately bell shaped distribution; whilst the trend for the average tour length is to decrease (i.e. improve) with increasing values of θ .

Analysis of the results indicates that the shortest average tour length (2.768 units) was obtained for $\theta = 0.9$, however the frequency for this value of the tuning parameter was not optimal. A value of $\theta = 0.7$ gave the optimal frequency (90%) but with a slightly increased average tour length (2.843 units).

The following statistics were reported by Aarts and Korst [1989a] when describing the performance of their asynchronous parallel Boltzmann machine model:

Description of Statistic	Aarts and Korst[1989a]
Tuning Parameter (θ)	UNKNOWN
Sample Size (m)	100
No of Solutions ($n=m/l$)	91
Average Tour Length (\bar{l})	2.783
Std. Dev. Of the Tour Length (σ)	0.097
Average Number of Iterations (I)	1.1
Smallest Observed Tour Length(l_{\min})	2.675
Largest Observed Tour Length(l^{\wedge})	3.060

Table 6.3: Aarts and Korst [1989a] Boltzmann machine implementing the LAP formulation

A comparison of the results reveals that for the values of the tuning parameter (θ) in the set $\{0.7,0.8,0.9\}$, the frequency and average tour length values differ from those published by Aarts and Korst [1989a], by the percentages listed in table 6.4 below.

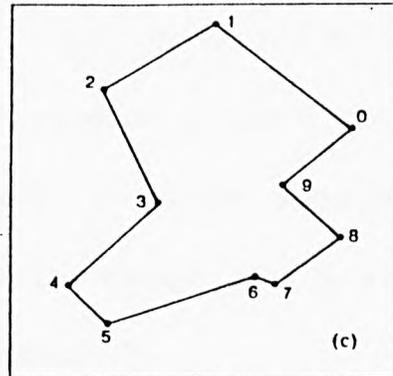
Description of Statistic	Percentage Difference		
	0.7	0.8	0.9
Average Tour Length (\bar{l})	2.15%	1.30%	-0.53%
Average Number of Iterations ($I=m/n$)	1.01%	3.31%	12.23%

Table 6.4: Percentage difference in frequency and average tour length.

For $\theta = 0.7$, both statistics are within 2.25% of those published by Aarts and Korst. It is however worth noting that there is a small discrepancy between the length of the

shortest tour (reproduced below in figure 6.6), obtained using the co-ordinates published by Wilson and Pawley [1988] and that reported by Aarts and Korst[1989a].

Figure 6.6: The optimum tour for Hopfield and Tank’s 10-city problem



Using the co-ordinates published by Wilson and Pawley [1988] the length of this tour is 2.691 units. Aarts and Korst [1989a] however report that the length of this tour is 2.675 units, that is, 0.016 units lower. It is conjectured that this difference is caused by small differences between the values of the co-ordinates used by Aarts and Korst and those published by Wilson and Pawley.

The corrected average tour length values for $\theta=0.7,0.8$ and 0.9 , and the revised percentage difference between the corrected averages and the average published by Aarts and Korst [1989a] are displayed in table 6.5.

Description of Statistic	Value		
	0.7	0.8	0.9
<i>Tuning Parameter (θ)</i>	0.7	0.8	0.9
<i>Corrected Average Tour Length ($\bar{l} - 0.016$)</i>	2.827	2.798	2.752
<i>% Difference between Corrected Average Tour Length and Aarts and Korst [1989a]</i>	1.56%	0.56%	-1.10%

Table 6.5: Corrected average tour length.

If the average tour length is corrected for this anomaly, then for $\theta=0.7$, a comparison of both the average tour length and the frequency with those published by Aarts and Korst [1989a] reveals that they differ by 1.6% and 1.0% respectively.

Finally, taking the value for the standard deviation of the population published by Aarts and Korst and reproduced in table 6.3 ($\sigma = 0.097$); the difference in average tour lengths, in terms of standard deviations is also small. Specifically, the uncorrected and corrected averages differ by less than two thirds and one third of a standard deviation respectively.

6.7.5 Conclusion

The Pocket Boltzman machine implemented in CONNEKT is based on the sequential Boltzmann machine model defined by Aarts and Korst [1989a], but incorporating their asynchronous unit-dependent cooling schedule. No results are published for the sequential Boltzmann machine model. The results that are published by Aarts and Korst [1989a] refer to an asynchronous parallel extension to their sequential Boltzmann machine model. However they do not specify in any detail how they implemented asynchronous parallelism. Specifically, they do not make clear whether they have implemented *limited* or *unlimited* parallelism [Aarts and Korst 1989b] or whether *all* state update calculations assume asynchronous parallelism.

The results generated by the Pocket Boltzmann machine are very similar in terms of the quality and frequency (both within 2.25%) of those published by Aarts and Korst [1989a]. It is therefore justifiable to state that Pocket Boltzmann machine is indeed a

Boltzmann machine - albeit one that implements sequential rather than asynchronous parallel updates - since it is based on Aarts and Korst sequential Boltzmann machine model and is able to exhibit the same behaviour as they publish for their Boltzmann machine model.

6.8 Performance of the Pocket Boltzmann machine implementation

6.8.1 Introduction

The Pocket Boltzmann machine, with the Pocket algorithm enabled, was applied to Hopfield and Tank's 10 city problem and its performance investigated for a number of discrete values of the tuning parameter θ in the range $0.05 \leq \theta \leq 100$.

For each value of θ , the system was run 100 times. For each run, the details of the best (i.e. pocket) tour obtained were recorded. In addition, where the final state of the network corresponded to a tour, the final state tour details were also recorded.

6.8.2 Results

Statistics describing the performance of the Pocket Boltzmann machine are described overleaf in tables 6.6a, 6.6b, 6.7a and 6.7b. Specifically tables 6.6a and 6.6b describe statistics relating to the generation of the best (pocket) tour, whilst tables 6.7a and 6.7b describe the final state tour details.

The variation in both the average tour length and the frequency with which tours were obtained, for different values of the tuning parameter θ , are illustrated in figures 6.7 and 6.8.

Tables 6.6a and 6.6b : Statistics describing generation of best (pocket) tours

Description of Statistic \ Results														
Tuning Parameter (θ)	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3
Sample Size (m)	100	100	100	100	100	100	100	100	100	100	100	100	100	100
No of Solutions (n)	0	0	89	69	100	100	100	100	100	100	100	100	100	100
Average Tour Length ($\bar{\ell}$)	N/A	N/A	3.089	3.018	2.785	2.742	2.695	2.695	2.716	2.705	2.720	2.730	2.744	2.748
Std. Dev. Of the Tour Length (σ)	N/A	N/A	0.161	0.205	0.088	0.042	0.017	0.017	0.037	0.030	0.038	0.069	0.052	0.082
Average Number of Iterations ($I=m/n$)			1.124	1.449	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
Smallest Observed Tour Length(ℓ_{\downarrow})	N/A	N/A	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691
Largest Observed Tour Length(ℓ^{\wedge})	N/A	N/A	3.192	3.707	3.115	2.857	2.769	2.769	2.778	2.774	2.778	3.174	3.083	3.194
$\text{Log}_{10}(\theta)$	-1.301	-1.000	-0.699	-0.523	-0.398	-0.301	-0.222	-0.155	-0.097	-0.046	0.000	0.041	0.079	0.114

Table 6.6a: Best (pocket) tour statistics generated for values of θ in range 0.05 to 1.3.

Description of Statistic \ Results														
Tuning Parameter (θ)	1.4	1.5	2.0	3.0	4.0	5.0	6.0	7.0	8.0	10.0	20.0	30.0	100.0	
Sample Size (m)	100	100	100	100	100	100	100	100	100	100	100	100	100	
No of Solutions (n)	100	100	100	99	100	99	100	99	100	100	100	100	100	
Average Tour Length ($\bar{\ell}$)	2.778	2.788	2.794	2.837	2.832	2.862	2.826	2.855	2.887	2.956	3.007	3.037	3.301	
Std. Dev. Of the Tour Length (σ)	0.125	0.134	0.150	0.241	0.250	0.290	0.203	0.370	0.357	0.447	0.447	0.465	0.602	
Average Number of Iterations ($I=m/n$)	1.000	1.000	1.000	1.010	1.000	1.010	1.000	1.010	1.000	1.000	1.000	1.000	1.000	
Smallest Observed Tour Length(ℓ_{\downarrow})	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	2.691	
Largest Observed Tour Length(ℓ^{\wedge})	3.566	3.215	3.506	4.201	4.792	4.321	4.061	5.090	4.496	5.372	4.398	4.847	5.041	
$\text{Log}_{10}(\theta)$	0.146	0.176	0.301	0.477	0.602	0.699	0.778	0.845	0.903	1.000	1.301	1.477	2.000	

Table 6.6b: Best (pocket) tours statistics generated for different values of θ in range 1.4 to 100.0.

Tables 6.7a and 6.7b: Statistics describing generation of final state tours

Description of Statistic \ Results														
Tuning Parameter (θ)	0.05	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	1.1	1.2	1.3
Sample Size (m)	100	100	100	100	100	100	100	100	100	100	100	100	100	100
No of Solutions (n)	0	0	33	3	25	52	66	84	81	83	23	13	4	1
Average Tour Length (\bar{t})	N/A	N/A	3.182	3.065	2.859	2.949	2.898	2.860	2.768	2.780	2.767	2.785	2.879	3.115
Std. Dev. Of the Tour Length (σ)	N/A	N/A	0.038	0.027	0.119	0.160	0.172	0.144	0.009	0.053	0.017	0.104	0.213	N/A
Average Number of Iterations ($I=m/n$)	-	-	3.030	33.333	4.000	1.923	1.515	1.190	1.235	1.205	4.348	7.692	25.00	100.00
Smallest Observed Tour Length(t_{\min})	N/A	N/A	3.035	3.035	2.691	2.691	2.691	2.691	2.691	2.769	2.691	2.691	2.769	3.115
Largest Observed Tour Length(t^{\wedge})	N/A	N/A	3.192	3.083	3.115	3.367	3.506	3.240	2.769	3.091	2.778	3.115	3.198	3.115
$\text{Log}_{10}(\theta)$	-1.301	-1.000	-0.699	-0.523	-0.398	-0.301	-0.222	-0.155	-0.097	-0.046	0.000	0.041	0.079	0.114

Table 6.7a: Final state tour statistics generated for values of θ in range 0.05 to 1.3.

Description of Statistic \ Results														
Tuning Parameter (θ)	1.4	1.5	2.0	3.0	4.0	5.0	6.0	7.0	8.0	10.0	20.0	30.0	100.0	
Sample Size (m)	100	100	100	100	100	100	100	100	100	100	100	100	100	
No of Solutions (n)	1	0	8	5	2	2	4	4	0	4	2	1	1	
Average Tour Length (\bar{t})	2.769	N/A	2.826	2.791	2.730	2.769	2.754	2.752	N/A	2.942	2.769	2.769	2.691	
Std. Dev. Of the Tour Length (σ)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	
Average Number of Iterations ($I=m/n$)	100.00	N/A	12.500	20.000	50.000	50.000	25.000	25.000	N/A	25.000	50.000	100.00	100.00	
Smallest Observed Tour Length(t_{\min})	2.769	N/A	2.691	2.691	2.691	2.769	2.691	2.691	N/A	2.769	2.769	2.769	2.691	
Largest Observed Tour Length(t^{\wedge})	2.769	N/A	3.115	3.115	2.769	2.769	2.778	2.857	N/A	3.115	2.769	2.769	2.691	
$\text{Log}_{10}(\theta)$	0.146	0.176	0.301	0.477	0.602	0.699	0.778	0.845	0.903	1.000	1.301	1.477	2.000	

Table 6.7b: Final state tour statistics generated for values of θ in range 1.4 to 100.0.

Figure 6.7: Variation in the Average Tour Length with the pocket modification enabled

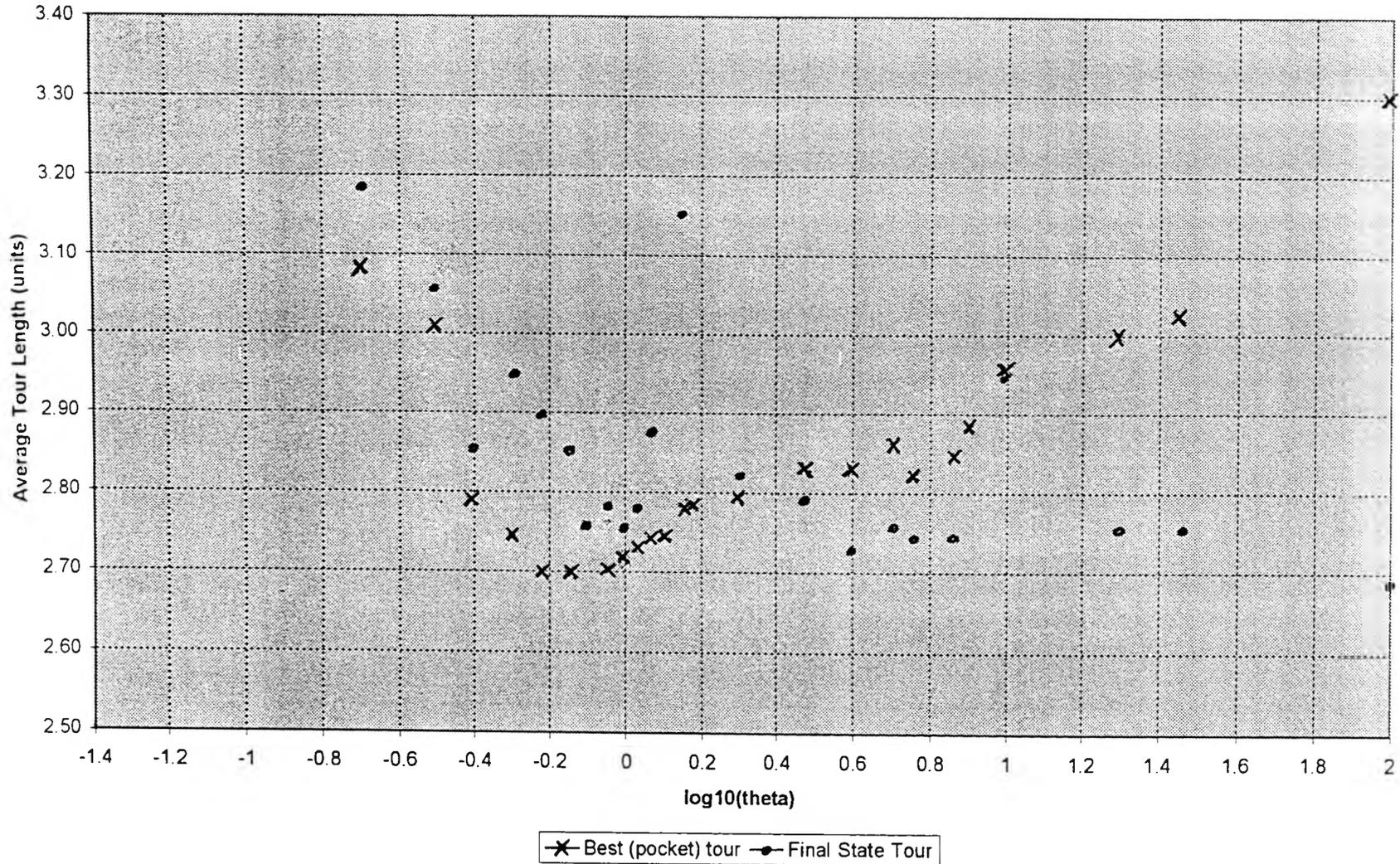
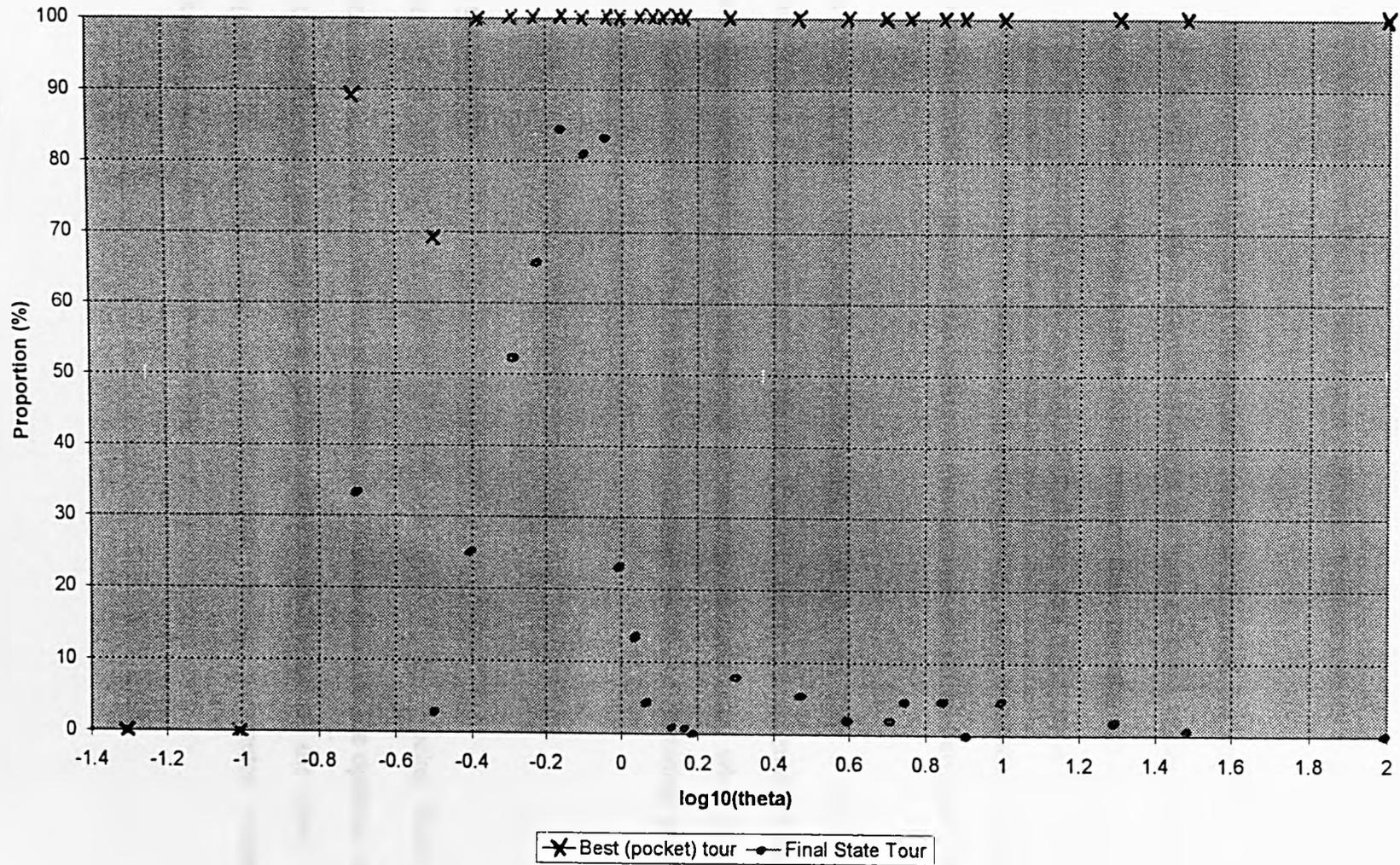


Figure 6.8: Variation in the frequency with which tours are obtained with the pocket modification enabled



6.8.3 Analysis of Results

For $\theta = 0.05$ and $\theta = 0.10$, no best (pocket) tours or final state tours were generated in each of the 100 runs.

A pocket tour was obtained in 100% of the runs for θ in the range $0.4 \leq \theta \leq 100$, that is in all 2,2000 runs the Pocket Boltzmann machine obtained a valid tour.

The smallest average tour length for pocket tours was observed for values of $\theta = 0.6$ and $\theta = 0.7$. Specifically the average tour length for both of these values of the tuning parameter was 2.695, which is only 0.1% greater than the optimal tour length of 2.691. This was because in each case the optimal tour was obtained in 96 out of 100 runs and very good tours were obtained for the remaining runs. For these values of θ the pocket algorithm improved the quality of average tour length by between 5-7%.

The average elapsed time required to complete a single run (for $\theta = 0.7$), on a Pentium 90Mhz Personal Computer with the 'Pocket' modification enabled was 335 seconds. This represents a 33% increase in the average elapsed time, when compared with the performance of the Pocket Boltzmann machine for the same tuning parameter value, with the pocket algorithm disabled.

6.8.4 Conclusion

For values of the tuning parameter $\theta = 0.6$ and $\theta = 0.7$, the Pocket Boltzmann machine with the pocket modification enabled was able to generate the optimal tour in 96 out of 100 runs and very good solutions in the remainder of the runs. As a consequence the average tour length for these tuning parameter values is approximately 0.1% less than the optimum.

Analysis of the experimental results indicates that the pocket modification can significantly improve both the average quality of solutions obtained, and the frequency with which good or even optimal solutions are generated, albeit for an approximate 33% increase in computational effort.

6.9 The Problem Examples and Results

6.9.1 Introduction

The approach described in this thesis was investigated using four problem examples. The first example was designed to demonstrate that CONNEKT does not necessarily assign a new customer to the vehicle whose tour passes closest to the collection and destination locations of that customer, but rather, it also takes account of the following additional hard mandatory constraint. That is, that the system must ensure that for each customer, the tour must visit the customer's collection location before it visits that customer's destination location.

The second problem example is designed to demonstrate that once CONNEKT has assigned a customer to a particular vehicle, if the new tour for that vehicle can accommodate a second customer for the smallest increase in the incremental cost value, and the vehicle has sufficient capacity, then the second customer is assigned to that vehicle, even if that customer would otherwise have been assigned to another vehicle.

The third problem is closely related to the second problem and shares all of the location co-ordinates. The third problem is designed to demonstrate that if the capacity of the vehicles is constrained, new customers can only be accommodated either by a vehicle with remaining capacity, or by being assigned to a vehicle which is at full capacity after that vehicle has delivered one or more of the customers assigned to it to their destination point(s).

The fourth problem examines the strategy of attempting to assign the closest customer (to any vehicle) first and then proceeds to assign as many as possible of the remaining customers by implementing the same strategy. The efficacy of this strategy is compared with one which assigns the furthest customer from any vehicle first.

Each of the problem examples is described in detail in one of the sections which follow.

6.9.2 Problem 'A'

The first problem example is very simple and is designed to demonstrate that CONNEKT attempts to assign a new customer to the vehicle which has a tour that satisfies both the constraints defined in the TSP and the *hard* mandatory constraints described above, for the least incremental cost.

The problem is comprised of two vehicles, $v1$ and $v2$, and five customers, $c1$, $c2$, $c3$, $c4$, and $c5$ each with a collection location and a destination location. At time $t=0$, the two vehicles are each assigned two customers, specifically $c1$ and $c2$ are assigned to $v1$ and $c3$ and $c4$ are assigned to vehicle $v2$. The maximum capacity of each vehicle is set to five customers.

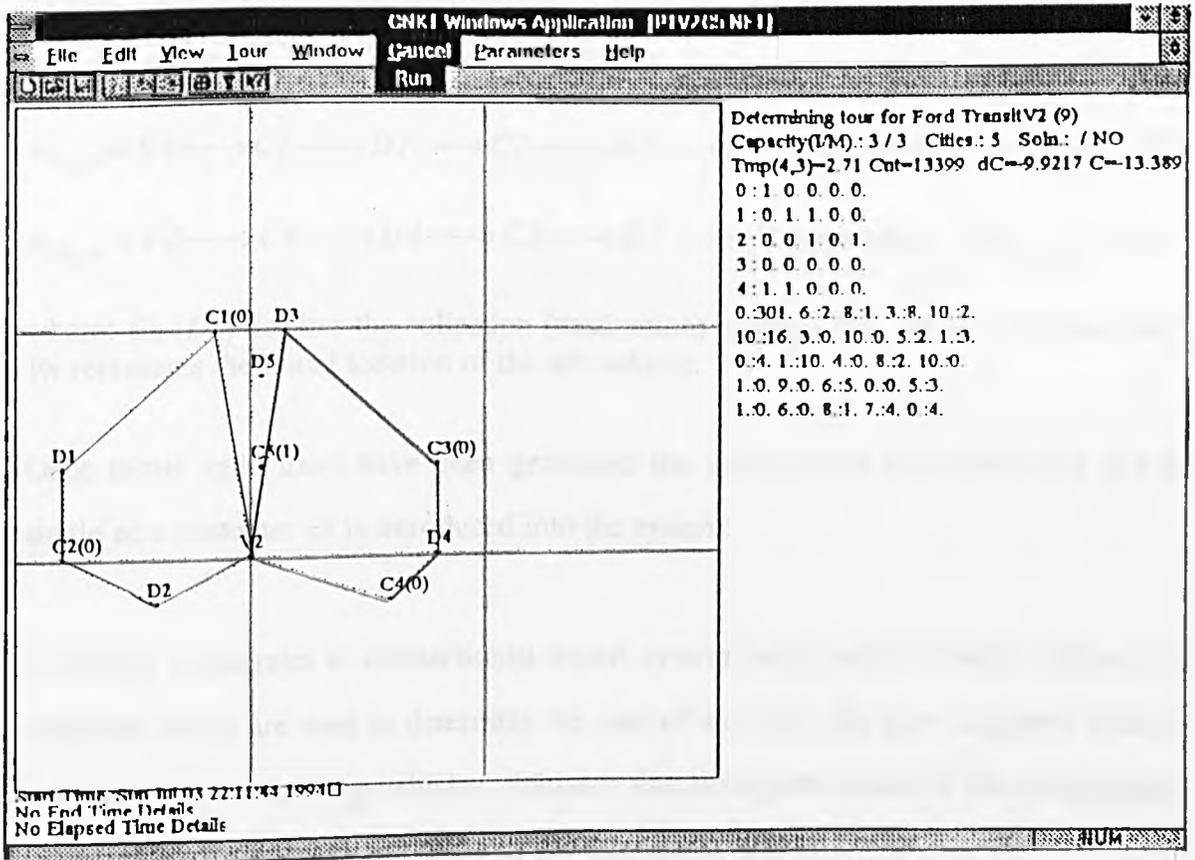
A pair of Pocket Boltzmann machines are generated (one for each vehicle), which attempt to generate optimal, initial valid tours from each of the vehicle's initial positions (i.e. the depot), through each of their customer's collection and destination points, before returning to the depot. The co-ordinates of the customer collection and destination points and the vehicle's starting and destination locations are listed overleaf in table 6.8.

The initial valid tours generated by the Pocket Boltzmann machine component of CONNEKT, are illustrated in figure 6.9 (overleaf), whilst table 6.9 (on page 245) provides a set of statistics describing the initial tours.

Entity	Collection X-Coord	Collection Y-Coord	Dest. X-Coord	Dest. Y-Coord	Initially Assigned	Available at time 't'
Customer 1 (c1)	0.85	2.0	0.2	1.4	v1	0
Customer 2 (c2)	0.2	1.0	0.6	0.8	v1	0
Customer 3 (c3)	1.8	1.4	1.15	2.0	v2	0
Customer 4 (c4)	1.6	0.8	1.8	1.0	v2	0
Customer 5 (c5)	1.05	1.4	1.05	1.8	-	1
Vehicle 1 (v1)	1.0	1.0	1.0	1.0	N/A	N/A
Vehicle 2 (v2)	1.0	1.0	1.0	1.0	N/A	N/A

Table 6.8: Assignment, availability and collection and destination locations of customers; and starting and end locations for vehicles in Problem A

Figure 6.9 : Problem A - Locations of customers and initial tours for each of the vehicles



Statistic	$v1 (t=0)$	$v2 (t=0)$
Average Tour Length ($\bar{\ell}$)	3.190	3.211
Standard Deviation of the Tour Length (σ)	0	0
Smallest observed tour length (ℓ_v)	3.190	3.211
Largest observed tour length (ℓ^{\wedge})	3.190	3.211
Sample Size (m)	10	10
Average number of iterations (I)	1	1
Smallest known value of the tour length (ℓ_{\min})	3.190	3.211

Table 6.9: Statistics describing the generation of initial tours for vehicles $v1, v2$ in problem example A

The results indicate that for this simple example, in *all* ten cases CONNEKT develops the following initial ($t=0$) valid optimal tours $\pi_{v1_{opt},0}, \pi_{v2_{opt},0}$ for vehicles $v1$, and $v2$, respectively:

$$\pi_{v1_{opt},0} = V1 \longrightarrow C1 \longrightarrow D1 \longrightarrow C2 \longrightarrow D2 \longrightarrow V1 \quad \text{for which } f(\pi_{v1_{opt},0}) = 3.190$$

$$\pi_{v2_{opt},0} = V2 \longrightarrow C4 \longrightarrow D4 \longrightarrow C3 \longrightarrow D3 \longrightarrow V2 \quad \text{for which } f(\pi_{v2_{opt},0}) = 3.211$$

where: Cn (Dn) denotes the collection (destination) location for the n th customer and Vn represents the initial location of the n th vehicle.

Once initial valid tours have been generated the system time is incremented and a single new customer $c5$ is introduced into the system.

CONNEKT generates a connectionist expert system and a pair of Pocket Boltzmann machines which are used to determine the cost of assigning the new customer to new valid tours on each of the vehicles. Statistics describing the results of assignments are displayed in Appendix A and summarised overleaf in table 6.10.

Statistic	Value
Proportion of times c5 is assigned to a vehicle with valid new tour.	100%
Proportion of times c5 optimally assigned to v1.	90%
Proportion of assignments with optimal new tour.	80%
Average length of tour with new customer ($\bar{\ell}$).	3.260
Standard Deviation of tour with new customer ($\sigma_{\ell, new}$)	0.002
Smallest observed new tour length (ℓ_v).	3.265
Largest observed new tour length (ℓ^{\wedge}).	3.553
Average Incremental Cost (\overline{Inc}_{new}).	0.129
Standard Deviation of the Incremental Cost ($\sigma_{J, new}$).	0.117
Smallest observed Incremental Cost (Inc_v).	0.07
Largest observed Incremental Cost (Inc^{\wedge}).	0.363
Sample Size (m)	10
Average number of iterations (I)	1.0

Table 6.10: Statistics describing the assignment of a single customer c5 to one of two vehicles v1, v2

For this problem, the shortest new *valid* tour, representing the optimal assignment of 'c5' to 'v1' (illustrated in figure 6.10) is:

$$\pi_{v1, opt, 1} = V1 \longrightarrow C5 \longrightarrow D5 \longrightarrow C1 \longrightarrow D1 \longrightarrow C2 \longrightarrow D2 \longrightarrow V1$$

for which the new tour length is $f(\pi_{v2, opt, 1}) = 3.265$ and the incremental distance is:

$$f(\pi_{v2, opt, 1}) - f(\pi_{v2, opt, 0}) = 3.265 - 3.190 = 0.075$$

The problem is designed so that the co-ordinates of the collection and destination points for customer 'c5' are closer to the initial tour for vehicle v2 than for vehicle v1. If no account was taken of the requirement to visit each customer, including c5's,

collection location before visiting it's destination location then, customer $c5$ could be assigned to vehicle $v2$ for the least incremental cost value of:

$$f(\pi_{v2_{opt},1}) - f(\pi_{v2_{opt},0}) = 3.227 - 3.211 = 0.016$$

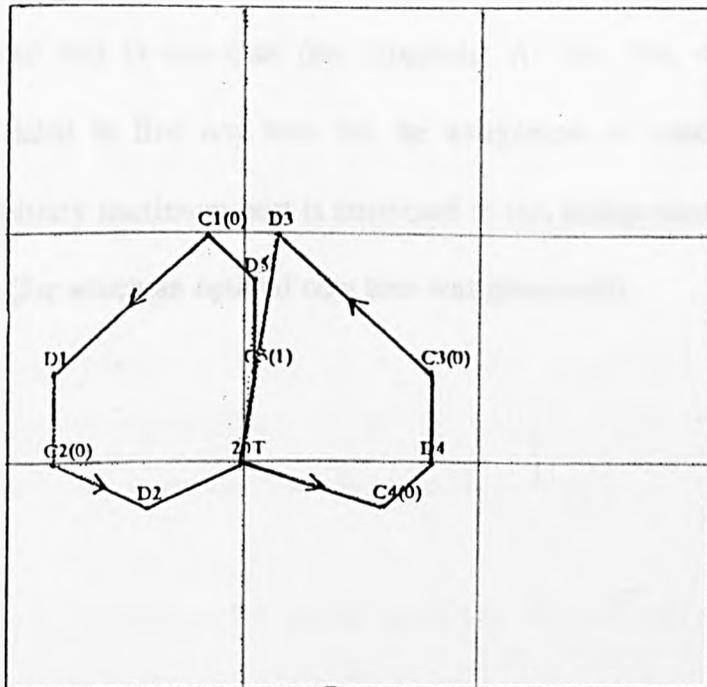
where:

$$f(\pi_{v2_{opt},1}) = f(V2 \longrightarrow C4 \longrightarrow D4 \longrightarrow C3 \longrightarrow D3 \longrightarrow D5 \longrightarrow C5 \longrightarrow V2) = 3.227$$

However, the assignment made by CONNEKT is constrained and the tour described above is rejected as an invalid tour, since it violates one of the mandatory constraints.

The tours after customer $c5$ has been optimally assigned to vehicle $v1$ and a new optimal tour generated, are displayed in figure 6.10.

Figure 6.10 : Optimal assignment of customer $c5$ to vehicle $v1$



For this simple example, the results listed in Appendix A and summarised in table 6.10 demonstrate that CONNEKT *consistently* assigns the new customer to the vehicle for which a valid tour can be generated, that satisfies the previously described *hard* mandatory constraints, for the least calculated incremental cost value.

For this simple example, the results listed in Appendix A and summarised in table 6.10 demonstrate that CONNEKT *consistently* assigns the new customer to the vehicle for which a valid tour can be generated, that satisfies the previously described *hard* mandatory constraints, for the least calculated incremental cost value.

In 9 out of 10 cases the new tour was the optimal new tour for vehicle 'v1' given the assignment of customer 'c5'. In the remaining case (see Appendix A: Run No. 2) a non-optimal but valid assignment was made to vehicle v2. This occurred because in this case the Pocket Boltzmann machine found a less than optimal tour for the assignment of customer 'c5' to vehicle 'v1'.

It should also be noted that in one case (see Appendix A: Run No. 4) the Pocket Boltzmann machine failed to find any tour for the assignment of customer 'c5' to vehicle 'v2' and an arbitrary maximum cost is attributed to this assignment. In this case 'c5' is assigned to v1 (for which an optimal new tour was generated).

6.9.3 Problem 'B'

In the second problem example two vehicles, $v1, v2$ are initially assigned two customers each from the set $C = \{c1, c2, c3, c4, c5, c6\}$. Each of the six customers, $c1, c2, c3, c4, c5$ and $c6$ has a collection location and a destination location. At time $t=0$, the two vehicles are each assigned two customers, specifically $c1$ and $c2$ are assigned to vehicle $v1$ and $c3$ and $c4$ are assigned to vehicle $v2$. The maximum capacity of both vehicles is set to five customers.

This problem example is designed to demonstrate that CONNEKT adapts dynamically as the customer assignment problem evolves. Specifically the problem is designed so that after the initial tours have been generated for both vehicles, $v1$ and $v2$, two new customers $c5$ and $c6$ require assignment. The locations of these customers are chosen (see table 6.11) so that $c5$ should be optimally assigned to $v2$, and if $c5$ was not included in the problem, $c6$ should be optimally assigned to $v1$.

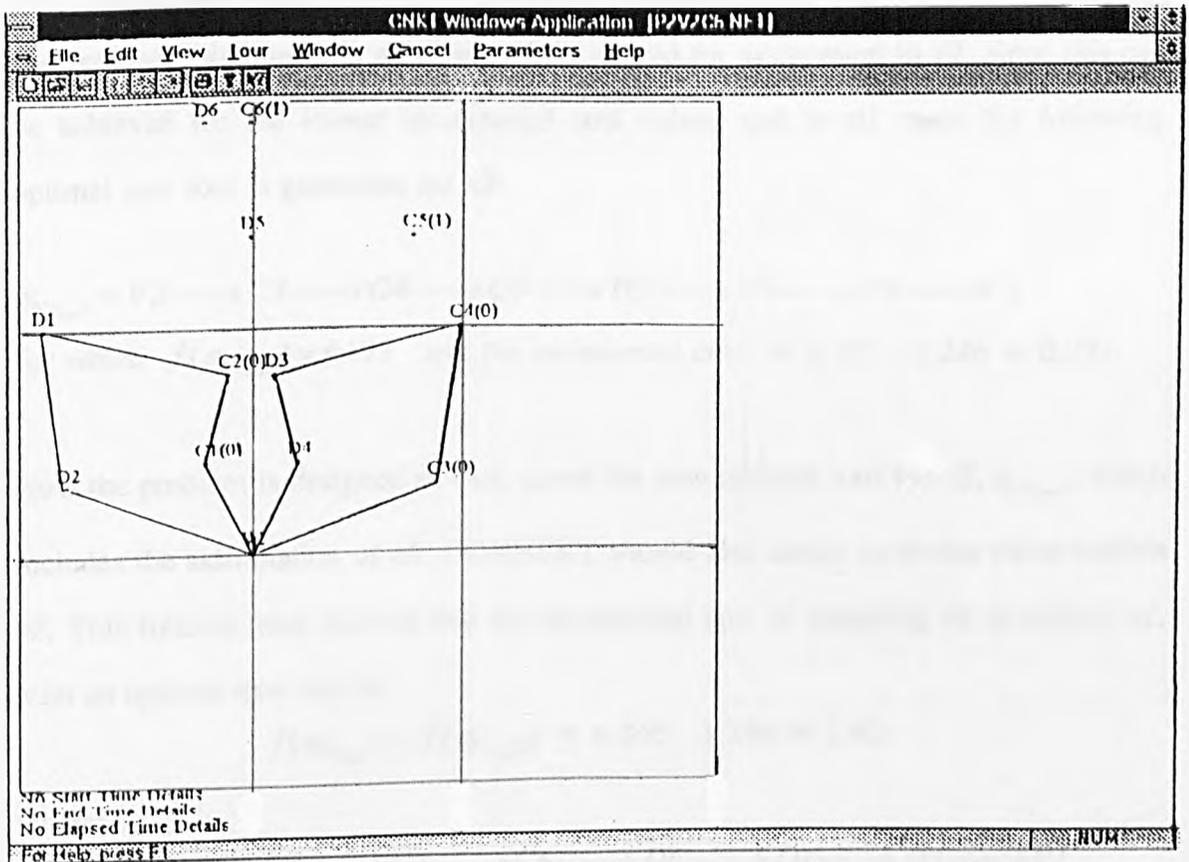
However, since both are present, and $c5$'s co-ordinates have been chosen so as to ensure that it can be assigned to $v1$ for the least incremental cost, the selection of which vehicle to then assign $c6$ to, must take account of the new tour for vehicle $v1$, which has since assimilated customer $c5$. The correct assignment for CONNEKT to make is then to also assign $c6$ to vehicle $v2$, since the incremental cost of this assignment is (designed to be) less than that of assigning $c6$ to $v1$.

The co-ordinates of the customer locations and initial tours for this problem example are displayed overleaf in figure 6.11 and table 6.11.

Entity	Collection X-Coord	Collection Y-Coord	Dest. X-Coord	Dest. Y-Coord	Initially Assigned	Available at time 't'
Customer 1 (c1)	0.8	1.4	0.1	2.0	v1	0
Customer 2 (c2)	0.9	1.8	0.2	1.3	v1	0
Customer 3 (c3)	1.8	1.3	1.1	1.8	v2	0
Customer 4 (c4)	1.9	2.0	1.2	1.4	v2	0
Customer 5 (c5)	1.7	2.4	1.0	2.4	-	1.0
Customer 6 (c6)	1.0	2.9	0.8	2.9	-	1.0
Vehicle 1 (v1)	1.0	1.0	1.0	1.0	N/A	N/A
Vehicle 2 (v2)	1.0	1.0	1.0	1.0	N/A	N/A

Table 6.11: Vehicle co-ordinates and customer assignment, availability and location co-ordinates for problem B.

Figure 6.11: Customer co-ordinates and initial tours for problem example B



When CONNEKT was applied to this problem, it was found that it was necessary to 'slow' the rate of cooling in the network in order to give the Pocket Boltzmann machine 'sufficient time' to enter states corresponding to optimal or near optimal tours. To facilitate this, the following parameter values were used for this and all remaining problem examples: $\alpha=0.99$, $L=100$, $M=500$ and $\theta= 0.7$.

Details of each of the ten runs are displayed in Appendix B. In each of runs the Pocket Boltzmann machines generated the following optimal initial tours, (illustrated in figure 6.11), for vehicles $v1$ and $v2$ respectively:

$$\pi_{v1_{opt},0} = V1 \longrightarrow C1 \longrightarrow C2 \longrightarrow D1 \longrightarrow D2 \longrightarrow V1 \quad \text{for which } f(\pi_{v1_{opt},0}) = 3.246$$

$$\pi_{v2_{opt},0} = V2 \longrightarrow C3 \longrightarrow C4 \longrightarrow D3 \longrightarrow D4 \longrightarrow V2 \quad \text{for which } f(\pi_{v2_{opt},0}) = 3.246$$

The time is then incremented and two new customers, $c5$ and $c6$, become available and require assignment. In all cases, $c5$ is selected for assignment to $v2$, since this can be achieved for the lowest incremental cost value, and in all cases the following optimal new tour is generated for $v2$:

$$\pi_{v2_{opt},1} = V2 \longrightarrow C3 \longrightarrow C4 \longrightarrow C5 \longrightarrow D5 \longrightarrow D3 \longrightarrow D4 \longrightarrow V2$$

for which $f(\pi_{v2_{opt},1}) = 4.177$ and the incremental cost = $4.177 - 3.246 = 0.931$

Now the problem is designed so that, given the new optimal tour for $v2$, $\pi_{v2_{opt},1}$, which includes the assimilation of $c5$, CONNEKT should also assign customer $c6$ to vehicle $v2$. This follows from the fact that the incremental cost of assigning $c6$ to vehicle $v1$, with an optimal new tour is:

$$f(\pi_{v1_{opt},1}) - f(\pi_{v1_{opt},0}) = 4.866 - 3.246 = 1.62$$

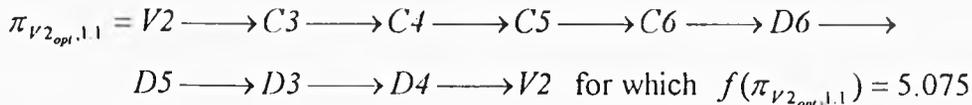
where:

$$\pi_{v1_{opt},1} = V1 \longrightarrow C1 \longrightarrow C2 \longrightarrow C6 \longrightarrow D6 \longrightarrow D1 \longrightarrow D2 \longrightarrow V1$$

and the incremental cost of assigning c_6 to v_2 , given that c_5 has already been assigned to it is:

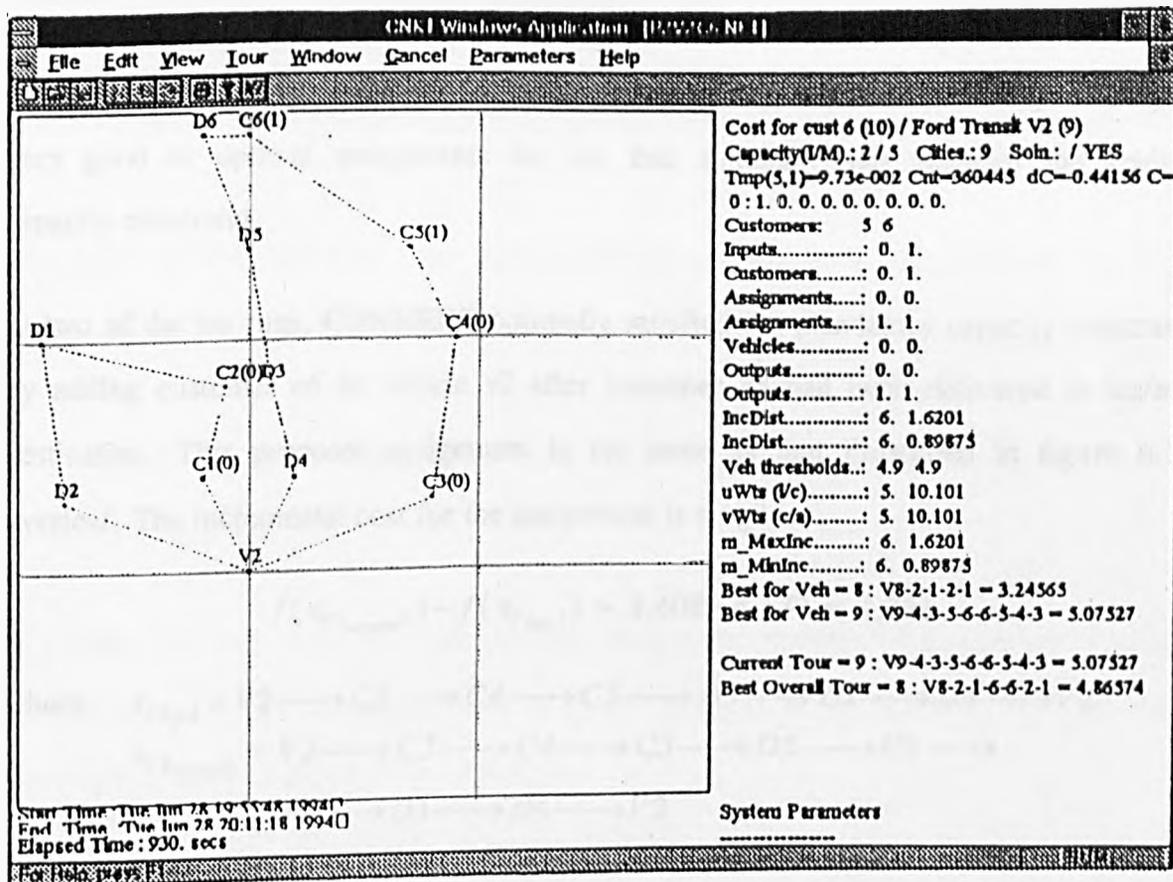
$$f(\pi_{V_2, opt, 1}) - f(\pi_{V_2, opt, 11}) = 5.075 - 4.177 = 0.898$$

where:



In six of the ten runs, CONNEKT correctly, optimally assigns customer c_6 to vehicle v_2 , with an optimal new tour generated for v_2 , as described in figure 6.12. In the four other runs, the hard mandatory constraint was satisfied by the non-optimal assignment of c_6 to v_1 . The cause in each case was the failure of the Pocket Boltzmann machine to develop a sufficiently good tour when determining the cost of the provisional assignment of customer c_6 to vehicle v_2 .

Figure 6.12: Optimal Assignment of customer c_6 to vehicle v_2 - with optimal new tour



6.9.4 Problem 'C'

The third problem example is identical to the second, except that the maximum available capacity of vehicle $v2$ is reduced from five simultaneous customers to three. The capacity of $v2$ is unchanged and remains set at five.

This example is designed to demonstrate that CONNEKT will ensure that the constraint that the vehicle's capacity is at no time exceeded, is not violated. This means that if CONNEKT first assigns customer $c5$ to $v2$, customer $c6$ must either be assigned to vehicle $v1$, which has sufficient capacity to accommodate it, or alternatively, $c6$ could be assigned to $v2$ after $v2$ has delivered one (or more) of its current assignment of customers to their destination.

Details of each of the ten runs are displayed in Appendix C. In each of the runs CONNEKT again generated optimal initial tours for both vehicles and then optimally assigned customer $c5$ to vehicle $v2$ - as described in section 6.9.3.

In attempting to assign customer $c6$, in all ten runs CONNEKT successfully developed very good or optimal assignments for $c6$, that simultaneously satisfied the vehicle capacity constraint.

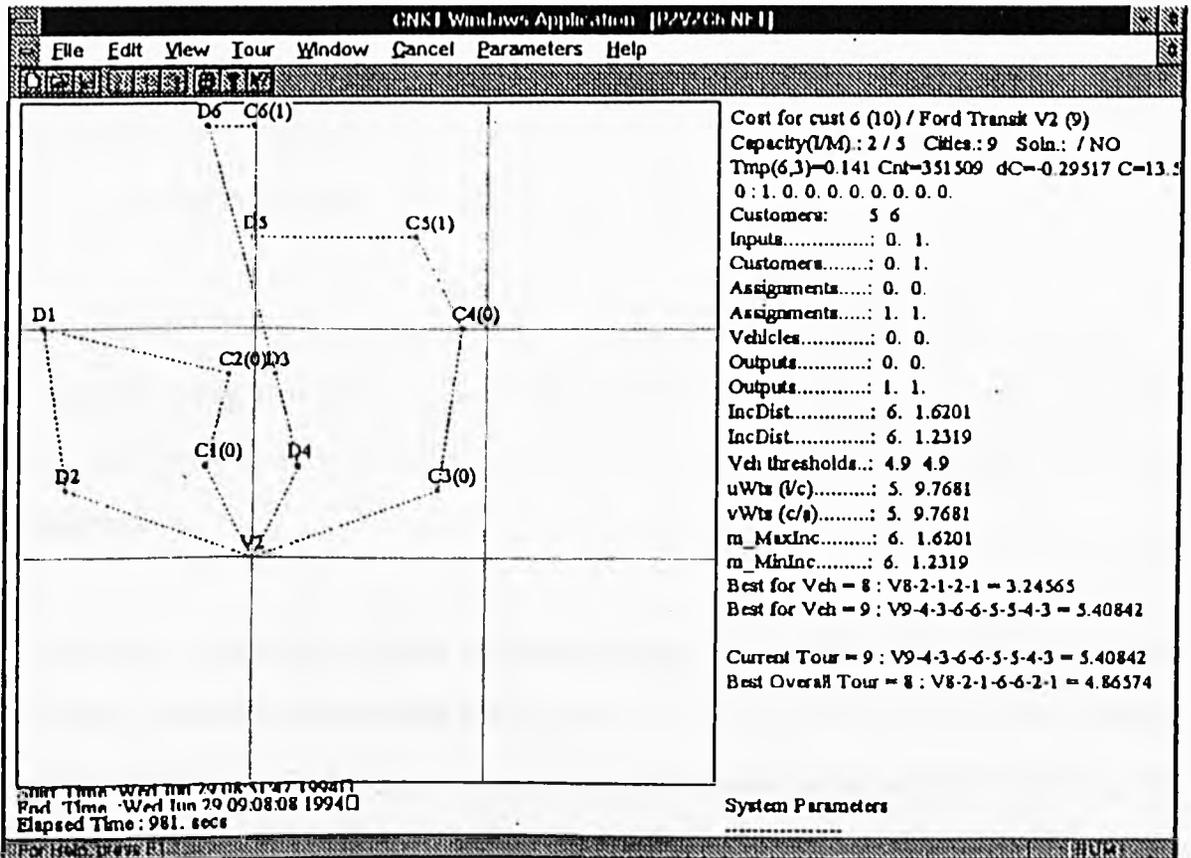
In two of the ten runs, CONNEKT optimally satisfied the mandatory capacity constraint by adding customer $c6$ to vehicle $v2$ after customer $c5$ had been delivered to his/her destination. This proposed assignment is the same as that illustrated in figure 6.13 overleaf. The incremental cost for the assignment is simply:

$$f(\pi_{V2_{nonopt},1,1}) - f(\pi_{V2_{opt},1}) = 5.408 - 4.177 = 1.231$$

where: $\pi_{V2_{opt},1} = V2 \longrightarrow C3 \longrightarrow C4 \longrightarrow C5 \longrightarrow D5 \longrightarrow D3 \longrightarrow D4 \longrightarrow V2$
 $\pi_{V2_{nonopt},1,1} = V2 \longrightarrow C3 \longrightarrow C4 \longrightarrow C5 \longrightarrow D5 \longrightarrow C6 \longrightarrow$
 $D6 \longrightarrow D3 \longrightarrow D4 \longrightarrow V2$

And $f(\pi_{V2_{opt},1}) = 4.177$, $f(\pi_{V2_{nonopt},1}) = 5.408$

Figure 6.13: Optimal Assignment of customer $c6$ to vehicle $v2$



In eight of the ten runs, the hard mandatory constraints are satisfied by the assignment of $c6$ to $v1$, for which the new optimal tour, $\pi_{V1_{opt},1}$, is generated, where:

$$\pi_{V1_{opt},1} = V1 \longrightarrow C1 \longrightarrow C2 \longrightarrow C6 \longrightarrow D6 \longrightarrow D1 \longrightarrow D2 \longrightarrow V1$$

where: $f(\pi_{V1_{opt},1}) = 1.620$

In each case this good but non-optimal assignment occurred because the Pocket Boltzmann machine developed only a good as opposed to optimal tour for the provisional assignment of $c6$ to vehicle $v2$.

6.9.5 Problem 'D'

The fourth problem is designed to investigate the assignment strategy applied in problems A to C. In each of these problems, the assignment strategy applied by CONNEKT was to first select for assignment, the customer that it determined could be assigned to *any* vehicle for the *least* incremental cost. The customer was then assigned to the vehicle for which the incremental cost was determined to be least, and this process repeated until all customers had been assigned.

An alternative would be to implement a *reverse* assignment strategy, that is to first select for assignment the customer that could be assigned to any vehicle for the *greatest* incremental cost, and to repeat this strategy until all customers had been assigned.

To examine the effect of each of these strategies, two vehicles $v1$ and $v2$ are each initially assigned two customers from the set $C = \{c1, c2, c3, c4, c5, c6\}$. The initial co-ordinates for $v1$ and $v2$ and the collection and destination locations for $c1$, $c2$, $c3$, and $c4$, are as defined for problem B (see table 6.11). The co-ordinates (x,y) of both the collection and destination locations for $c5$ and $c6$ are selected at random, but satisfy the following constraints:-

$$0 \leq x, y \leq 2$$

Twenty sets of random collection and destination location co-ordinates were generated for customers $c5$ and $c6$. Each one of the twenty sets of co-ordinates was then used to create a problem example. For each example CONNEKT was run twice. In the first run, the original strategy to assign customers in order of *least* incremental cost was implemented. In the second run, the alternate *greatest* incremental cost strategy was applied.

The randomly generated customer co-ordinates and the total incremental cost of assigning both customers - for each of the strategies in each of the twenty problem examples, are listed overleaf in table 6.12. Details of each of the twenty pairs of runs are displayed in Appendix D.

In 4 of the runs CONNEKT was unable to generate a tour, and hence determine a tour length for the provisional assignment of one of the customers c_5 or c_6 , to one of the vehicles v_1 or v_2 . Where no tour was generated the incremental cost for this provisional assignment was set to an arbitrary maximum value (of 10). This value was selected to ensure that it was greater than any incremental cost generated by experiment in any of the 40 problem runs.

In each of the twenty examples, the strategy of assigning the customer with the *least* incremental cost first, resulted in a lowest total incremental cost. It is possible, however to conceive of problems where under each of the strategies, both customers would be assigned to the same vehicle for the same total incremental cost. One such example would be where all new customers had the same collection point, equi-distant from each of the vehicle's existing tours, and a common destination point that was also the vehicle's destination (i.e. the depot).

Hence for some problem examples, the total incremental cost will be the same regardless of which of the two strategies is applied. However, the experimental results indicate that for problems of this type, the strategy of assigning customers based on least incremental cost first, is more likely to result in a lower total incremental cost, than the reverse strategy.

Run No.	Random Coordinates	Total Incremental Cost		Best Strategy
		(Nearest)	(Furthest)	
1	C5 (1.958,1.338) D5 (1.152,1.588) C6 (0.752,1.034) D6 (1.908,1.020)	1.618	3.407	Least Cost
2	C5 (1.662,1.806) D5 (1.174,1.968) C6 (1.552,1.338) D6 (1.608,1.262)	0.847	2.562	Least Cost
3	C5 (0.110,1.160) D5 (0.608,0.992) C6 (0.414,0.160) D6 (0.180,0.894)	2.289	3.415	Least Cost
4	C5 (0.560,1.138) D5 (1.258,0.094) C6 (0.534,1.496) D6 (0.198,0.056)	5.109	6.260	Least Cost
5	C5 (0.730,1.512) D5 (0.224,1.134) C6 (0.982,1.796) D6 (1.358,1.058)	0.605	2.625	Least Cost
6	C5 (0.652,0.244) D5 (1.626,0.038) C6 (0.318,1.218) D6 (1.386,1.900)	3.946	4.239	Least Cost
7	C5(0.856,1.832) D5(0.648,0.526) C6(1.814,0.438) D6(0.156,1.994)	3.384	3.996	Least Cost
8	C5(1.836,0.908) D5(0.850,1.660) C6(1.006,1.212) D6(0.656,0.684)	1.113	3.623	Least Cost
9	C5 (1.404,1.438) D5 (1.988,1.492) C6 (0.228,0.504) D6 (0.746,1.352)	2.359	6.754	Least Cost
10	C5 (1.146,0.104) D5 (0.648,0.400) C6 (1.924,0.968) D6 (0.034,1.708)	3.839	5.195	Least Cost
11	C5(1.078,1.098) D5(0.446,0.032) C6(0.796,1.918) D6(1.566,1.358)	3.465	6.754	Least Cost
12	C5(1.104,0.668) D5(0.106,0.260) C6(1.818,1.272) D6(1.688,1.406)	3.084	4.565	Least Cost
13	C5(1.916,1.838) D5(0.528,0.808) C6(1.832,1.368) D6(1.364,1.934)	1.805	3.351	Least Cost
14	C5(1.752,0.408) D5(1.234,0.138) C6(1.960,1.148) D6(0.108,0.930)	4.403	5.003	Least Cost
15	C5(1.050,0.524) D5(1.682,1.466) C6(0.962,0.970) D6(0.614,0.746)	1.322	3.249	Least Cost
16	C5(1.614,0.448) D5(1.782,0.948) C6(0.040,0.234) D6(1.994,0.346)	4.222	5.716	Least Cost
17	C5(0.872,1.126) D5(1.584,0.234) C6(0.260,0.914) D6(0.942,0.612)	2.411	4.028	Least Cost
18	C5(0.968,1.268) D5(1.428,1.348) C6(1.590,1.712) D6(0.140,1.434)	1.397	2.591	Least Cost
19	C5(0.786,0.302) D5(0.164,1.176) C6(1.032,1.814) D6(1.478,0.014)	4.647	5.433	Least Cost
20	C5(0.500,0.442) D5(1.366,1.326) C6(1.866,1.442) D6(1.322,0.802)	2.332	4.223	Least Cost

Table 6.12 Random co-ordinates and incremental costs for least and furthest strategies

7.0 APPRAISAL AND EVALUATION

7.1 Introduction

The appraisal and evaluation of the performance of the connectionist approach described in this thesis, is examined, firstly in terms of the performance of each of the principal constituent components and then secondly in terms of their collective operation, as implemented in CONNEKT.

The section that immediately follows examines the performance of the Pocket Boltzmann machine implementation. The performance of the 'Pocket' modification is then addressed, the performance of the connectionist expert system is examined and finally an appraisal and evaluation of the performance of the system as a whole is provided.

7.2 Evaluation of Results and Appraisal and Criticism of Performance

7.2.1 The Pocket Boltzmann Machine model

The results described in section 6.8 and summarised in tables 6.7a and 6.7b, clearly demonstrate that the Pocket Boltzmann machine model implemented in CONNEKT can find very good or even optimal solutions to the Hopfield and Tank's [1985] 10-city travelling salesman problem.

With the pocket modification disabled, the frequency with which solutions are obtained and the average quality of the solutions are both very close, specifically within 2.25% of those published by Aarts and Korst [1989a].

It is worth emphasizing that the convergence of the Pocket Boltzmann machine model, when simulated on a single processor machine is a very slow process. Typically, an average elapsed time of 335 seconds is required for a Personal Computer with a Pentium P90 processor, before the network converges for even relatively small 'toy' (10-city) problem examples. Even then, convergence is not guaranteed, in which case

the network computation must be restarted and the process repeated. It is conjectured that the prospects and future popularity of such techniques are heavily dependant on the successful application to and solution of, more difficult, 'real-world' problems, implemented of necessity, on arrays of parallel processors.

7.2.2 The Pocket Modification

The application of the Pocket Boltzmann machine to Hopfield and Tank's [1985] 10-city problem, indicates that when the pocket modification was enabled, the Pocket Boltzmann machine was able to generate good or very good tours for a wide variety of tuning parameter values. In addition, the average tour length was reduced by between 5-7% and the frequency increased to 100% for a wide range values of the tuning parameter θ . However, to achieve this increase in performance, the average computational run-time was correspondingly increased by 33%.

Perhaps the most significant results are those summarised in the third and fourth rows of tables 6.7a and 6.7b. These show that in *every one* of 2,200 runs with the tuning parameter θ in the range $0.4 \leq \theta \leq 100$, the 'Pocket' modification enabled the network to a generate a valid, often good or optimal tour. For values of $\theta=0.6$ and 0.7 , the optimal tour was obtained 96% of the time and the average tour length was only 0.1% greater than the optimal tour length.

The pocket modification therefore, reduces dependency on the tuning parameter and significantly increases the likelihood that the Pocket Boltzmann machine model develops a valid tour. It also makes it more likely that a higher quality, good or even optimal tour is generated. However, the improved performance does require additional computational effort with an associated increase in the average run-time.

The results for problem examples 'A', 'B', 'C' and 'D' also demonstrate that the 'Pocket' modification can extend the capabilities of conventional Boltzmann machine models. Specifically it can successfully enable additional constraints to be specified which may be difficult to encode and enforce, using patterns of connectivity and connection weight strengths. In the example problems described above, this enabled the Pocket Boltzmann machine model to develop very-good or even optimal valid tours that took account of the order of collection and destination locations of both new customers and those already assigned to the vehicle. It was also able to enforce the vehicle capacity constraints, and in so doing supported the dynamic assignment of new customers at locations on the tour after one or more other customers had been delivered to their destinations, hence ensuring that the capacity of the vehicle was never exceeded.

The results for the 10-city problem indicate that the Pocket modification consistently ensures that the network is able to store and reproduce the 'best' solution visited by the network at any point in the annealing schedule.

The benefits that follow from implementing the 'Pocket' modification are only achieved for a significant increase in the processing overhead, but as the technique only requires that the output states of the units be examined, it does not interfere in any way with the operation of the Pocket Boltzmann machine itself.

It is interesting to note that in *every* case for problem examples A, B and C (a total of 60 runs), the Pocket Boltzmann machine model developed optimal initial tours for the vehicles. However, it was necessary to select a slower annealing schedule ($\alpha=0.99$, $L=100$) and stricter termination criterion ($M=300$) to successfully obtain optimal tours for assignments to vehicles with larger numbers of locations. It is conjectured that this was necessary in order to give the network 'more time' in which to find very good or even optimal tours.

This behaviour suggests that the assertion made in the justification of the 'Pocket' modification, that:

"given a suitable annealing schedule, it is likely that the Pocket Boltzmann machine will fall under the influence of and *visit* sufficient states to obtain a near-optimal solution that satisfies both the soft network constraints and the additional hard constraints".

may not be practicable for all problem examples - since these may require a very slow annealing schedule.

It is conjectured that this may be because for some problem examples, the network may be more likely to fall into the influence of a particular solution, but escape from its influence without ever adopting the solution state. As an alternative to attempting to ameliorate this effect by selecting slower annealing schedules, it may be possible to extend the 'Pocket' algorithm to detect states that are close to solutions and to perform one or more *k*-changes [Lin and Kernighan, 1973] or some similar operation to transform the state into a (locally optimal) solution.

A determination of how the performance of the Pocket Boltzmann machine scales with problem size, both with and without the enhancement outlined above, remains a suitable topic for further research.

Finally, one of the most significant criticisms must be the large processing time required to generate a solution when the Pocket Boltzmann machine is simulated on a machine with a single serial processor. Computation times range from a few tens of seconds to several minutes for a single (10-city) run, depending on the rate of cooling applied. The processing time also scales badly as the size of the problem is increased - as Aarts and Korst [1989a] report, when running on a VAX-780 :-

"Typical running times required by the simulations are a few minutes for the 10-city problem instance upto a few hours for the 30 city problem instance."

It may however be(come) possible to significantly overcome the processing time problem by, as Aarts and Korst [1989a,1989b] suggest, implementing the Boltzmann machine model using arrays of parallel processors.

7.2.3 The Connectionist Expert System

The results from all three problem examples indicate the connectionist expert system frequently selects the customer that could be assigned to any vehicle, for the least incremental cost, that is the smallest increase in the total cost function (TCF).

As the technique took full account of new assignments and associated tours the connectionist expert system was able to repeatedly select the customer that could be assigned for the least incremental cost, until all customers were assigned or the capacity of all vehicles had been exhausted.

The results demonstrate that the novel connectionist expert system architecture is indeed able to develop very good or even optimal assignments of multiple new, un-split demand stochastic customers to a fleet of capacitated vehicles.

The behaviour of the connectionist expert system is however purely deterministic and highly dependant on the quality of the incremental cost data developed by the Pocket Boltzmann machines. In the relatively few cases where the network made a good or very good as opposed to optimal assignment, the cause is in all cases directly attributable to the Pocket Boltzmann machine developing a less than optimal provisional tour.

In contrast to the Pocket Boltzmann machine, the processing time required by the Connectionist Expert System component to perform an inference cycle *once the incremental cost data had been determined* was very small, typically < 0.2 seconds.

7.2.4 The Overall Performance of the Approach

The results demonstrate that the connectionist approach described in this thesis, at least for small problem examples can indeed develop very good or near-optimal dynamic assignments of customers to vehicles, which satisfy both the *soft* constraints imposed within the TSP and the additional (and realistic) *hard* location ordering and vehicle capacity constraints.

The results from Problem 'A', described in section 6.9.2, demonstrate that the approach can develop very good or optimal assignments of customers to vehicles, with optimal valid tours that take account of the additional constraint that for each customer, the collection location must be visited before its destination location is visited.

The results from Problem 'B' indicate that the approach can develop very good or optimal assignments and optimal valid tours, that dynamically take full account of new state information - specifically the appearance of new (stochastic) customers and the proximity of newly generated tours.

The results from problem 'C' demonstrate that the approach can enable very good or even optimal assignments of customers to vehicles, with tours that take account of the vehicle's capacity and internal structure and meaning within the tour itself. Specifically, the significance that visiting collection and destination locations has on the available capacity of a vehicle.

The experimental results from problem 'D' indicate, that for the examples investigated, the strategy of assigning the nearest customer, to any vehicle first, is more likely to result in the least increase in the total increment cost.

It is clear however, that the approach is very dependent on the performance of the Pocket Boltzmann machine model. The approach has not been proven, and because of the large processing times involved, has not been tested on larger problem examples. Hence a proof or demonstration that the approach is valid for larger problems remain open research topics.

It is also clear from the problems investigated in this thesis that the approach is very slow, requiring the generation of typically many Pocket Boltzmann machine models. In the worst case, it is clear that to assign l customers to k vehicles would require the generation of $k \sum_{i=0}^l i$ Pocket Boltzmann machine models. Whilst it may be possible to reduce this number using simple geometry and for example a Branch and Bound procedure to eliminate some possible provisional assignments, it is conjectured that the approach is in fact so computationally expensive and hence slow in operation, that it is impracticable on a purely serial machine. However it may prove to be possible to reduce the elapsed time, by as Aarts and Korst [1989a] suggest, implementing the Boltzmann machines on arrays of parallel processors.

7.3 Criticism and Comparison with Other Approaches

To the best of the authors knowledge, the precise formulation of the dynamic customer assignment and vehicle routing problem defined in this thesis and the connectionist approach taken to solve examples from this class of problems are both novel. However, it is clear that this class of problems could have been solved by replacing the connectionist expert system with a conventional expert system and the Pocket Boltzmann machine with a dedicated sequential algorithm.

It is also clear that a simulation of the approach described in this thesis is considerably slower in operation than the 'few seconds' that Aarts and Korst [1989a, 1989b] report are required to generate solutions to examples of the TSP using sequential algorithms.

No direct comparison of the performance of the components of CONNEKT with alternative techniques has been made by the author, rather this comparison largely relies on that provided by Aarts and Korst[1989a].

They report that for the 10-city problem, both the sequential simulated annealing algorithm [Kirkpatrick, Gelatt and Vecchi, 1983] and the Lin-Kernighan local search procedure [Lin and Kernighan, 1973]) produce on average solutions of a slightly higher quality than those generated by Aarts and Korst's sequential simulation of their parallel Boltzmann machine model, and in a considerably shorter time. Specifically a few seconds, compared to a few minutes.

Now, although the 'Pocket' modification can improve the quality of solutions, at best the quality of solutions obtained by the Pocket Boltzmann machine may be only comparable with those generated by the best dedicated sequential algorithms.

In addition, the performance of the connectionist expert system model described in this thesis could easily have been implemented using any one of a number of expert system shells and/or development languages.

Whilst it is recognised that some parallel procedures supporting the operation of both components of conventional expert systems and sequential optimization algorithms may be developed, one of the principal advantages of the connectionist approach outlined in this thesis is that it is based on an inherently parallel approach.

It is conjectured that, like a number of optimization problems - including np-complete problems - investigated using connectionist systems; the approach described in this thesis is only likely to outperform well established serial techniques if it can be successfully implemented on parallel hardware.

Aarts and Korst [1989a] suggest that one problem with the 0-1 programming formulation of the problem is that it typically requires the network to contain many large connection strength values with opposite signs. The consequence of this is that slow convergence is necessary if the finite-time approximation is to develop good results. It is quite difficult to set the connection weights so as to ensure that tours are strongly favoured over non-tours and to simultaneously strongly favour very good tours over poor tours.

Although given suitable conditions, which include infinitely long Markov chain lengths, the sequential Boltzmann machine model is guaranteed to converge onto an optimal solution. In practice, a finite-time approximation has to be implemented in which case the model obtains a local (possibly non-global) optimum.

Finally it is worth re-emphasising that the principal disadvantage of using a simulation of a Boltzmann machine to develop solutions to TSP's and VRP's on a single processor is the very long processing time required. For Hopfield and Tank's 10-city problem, Aarts and Korst report [1989a] that on a VAX-780 computer, several minutes were required compared to a few seconds for local search algorithms like those developed by Lin and Kernighan [1973]. This increases to a few hours for Hopfield and Tank's 30-city problem. They also report that the Lin-Kernighan algorithm found the shortest tour length for the 10-city problem (2.675) in 'approximately all cases'.

7.4 Evaluation of the Contribution

To the best of the authors knowledge this thesis includes three novel contributions.

Firstly, although a number of variants of the General Assignment Problem (GAP) and Vehicle Routing Problems (VRP) have been developed, to the best of the authors

knowledge a novel approach has been taken to the precise form of the problem investigated in this thesis.

Secondly, the implementation required the development of a novel connectionist expert system architecture which was successfully applied to (albeit simple) examples from this class of problem.

Finally, a novel extension to the Boltzmann machine model, originally developed by Ackley, Hinton and Sejnowski [1985] and implemented by Aarts and Korst [1989a,1989b] called the Pocket Boltzmann machine model is described. The 'Pocket' modification is inspired by the 'Pocket' algorithm developed by Stephen Gallant [1985] and applied by him to perceptron and back-propagation networks.

The thesis describes how the 'Pocket' algorithm can be applied to develop tours for vehicles satisfying the VRP component of DCAVRP examples and in so doing calculate incremental cost data, which is used by the novel connectionist expert system architecture to determine very-good or even optimal dynamic assignments of new stochastic, un-split demand customers to members of a fleet of available capacitated vehicles.

7.5 Suggestions for Further Research

Suggestions for further research include the investigation of the performance of the approach when applied to larger, more realistic, problem examples. To support such an application, it may for example, prove to be necessary, or simply beneficial to combine elements of the Lin-Kernighan local search algorithm with the 'Pocket' algorithm.

Theoretical topics for further research include attempts to develop:-

- Mathematical model(s) to predict the performance and scalability of the Pocket Boltzmann machine model as a function of the number of locations in the tour.
- Alternative formulations of the TSP, which enable relatively large differences in consensus to be developed between short and long tours, compared with tour and non-tours (than are achievable using Aarts and Korst's [1989a,1989b] 0-1 QAP and LAP programming formulations).

An avenue of research that the author hopes to pursue is the development of hybrid applications that combine the requirements of conventional business applications with the connectionist optimization capability presented in this thesis. It may, for example be(come) possible to develop object-oriented applications running on a single processor machine, - for example a PC running Microsoft Windows NT - which interfaces with an array of parallel processors installed on a PC expansion card, which adequately performs the processor intensive Pocket Boltzmann computations.

8.0 SUMMARY and CONCLUSIONS

This thesis describes a novel connectionist approach which can determine very good or near optimal solutions to at least simple examples of dynamic customer assignment and vehicle routing problems.

To the best of the author's knowledge, the approach described in this thesis had not previously been applied to the precise formulation of the class of Dynamic customer Assignment and Vehicle Routing Problems, as defined in this thesis.

The approach is implemented using a combined connectionist-expert system and Pocket Boltzmann machine architecture, simulated using the object-oriented Visual C++ language and implemented on a personal computer (PC) running the Microsoft Windows NT4 operating system.

The thesis introduces the Pocket Boltzmann machine, which utilises a novel extension to the Boltzmann machine model originally developed by Ackley, Hinton and Sejnowski [1985] and implemented by Aarts and Korst [1989a, 1989b] and hence introduces a new class of neural network models, referred to as Pocket Boltzmann machines.

The 'Pocket' extension was inspired by, and is based on, the 'Pocket' algorithm developed by Gallant [1985] and applied by him to perceptron and backpropagation networks. The 'Pocket' modification does not affect the operation of the Boltzmann machine model and can equally well be applied to sequential, synchronous and asynchronous Boltzmann machines and other similar models. For an additional computational cost, the 'Pocket' modification enables the 'best' solution obtained by a Boltzmann machine to be stored, for subsequent retrieval.

The experimental results indicate that the technique can be used to significantly improve the probability with which a solution is found and to increase the quality of solutions

developed by existing Boltzmann machine models. It is conjectured that the 'Pocket' modification may also alternatively be used to allow models of this type to develop good-solutions to more restricted problems, specifically those that require that a variable number of additional *hard* mandatory constraints are also satisfied.

The performance of the Pocket Boltzmann machine has only been tested on problems with up to 10-cities (locations). It must also be remembered that although the justification for the operation of the algorithm that has been presented, is based on the same argument that Ackley, Hinton and Sejnowski [1983] use to justify and explain the global optimization capability of the Boltzmann machine model; the justification is not proven. However, it has been confirmed experimentally, at least for small problem examples.

However, the issue of scalability remains an open question. There is some experimental evidence to suggest that for larger problems, the Boltzmann machine may not visit a large number of states corresponding to solutions, unless a suitably slow cooling schedule is applied. In contrast for smaller problems (up to five cities), in particular the generation of initial tours for problems A, B and C, the network was able to generate optimal tours in each one of 60 consecutive cases.

If further research reveals that for larger problems the Pocket Boltzmann machine model does not often fully adopt states corresponding to solution, but simply falls within their influence, then it may be both possible and beneficial to combine the 'Pocket' algorithm with a local search algorithm, like the Lin-Kernighan heuristic (Lin and Kernighan, 1973) to generate a locally optimal solution from the current state of the network.

The performance of the Pocket Boltzmann machine is directly determined by the operational behaviour of the underlying Boltzmann machine. Korst and Aarts [1989a,

1989b] acknowledge that there are inherent weaknesses in the 0-1 programming formulations used to model particular instances of the TSP on a Boltzmann machine. In particular it is difficult to set the network connection weights so that both the difference in consensus between good and bad tours is not small in comparison with the difference in consensus for tours and non-tours. In addition, in many cases a number of transitions are required for the network to move from one state that corresponds to a solution (a feasible state) to another. The steps between the two states often represent infeasible states and have a low consensus value. This increases the chance (at lower) temperatures that the network becomes trapped in a local minimum.

Although given suitable conditions, (which include infinitely long Markov chain lengths), the sequential Boltzmann machine model is guaranteed to converge onto an optimal solution. In practice, a finite-time approximation has to be implemented in which case the model obtains a local (possibly non-global) optimum.

In spite of this limitation, the results clearly demonstrate that the 'Pocket' modification can, both improve the average quality of results obtained by Boltzmann machine models, and increase the likelihood that either a very good or even optimal solution is found.

The 'Pocket' modification is also independent of any particular implementation, since the 'Pocket algorithm simply 'monitors' the results generated by the network and stores the best result to date.

The Pocket Boltzmann machine instances described in this thesis have largely been used to compute the incremental costs of assigning customers to vehicles, and in so doing they attempt to develop optimal tours that satisfy the *soft* constraints of the TSP encoded in the network, and a set of *hard* mandatory constraints. The incremental cost data was used for four different problem examples to determine a sub-set of the connection weights of a novel connectionist expert system.

The connectionist expert system is a five layer perceptron based network, based on the Matrix Controlled Inference Engine (MACIE) developed by Stephen Gallant. The results have demonstrated that the connectionist expert system and Pocket Boltzmann machine model presented in this thesis, and collectively known as CONNEKT, can determine very good or even optimal assignments of customers to vehicles and develop very good or optimal tours for the vehicles. It has been demonstrated that the architecture is able to take account of semantic information associated with locations in the tours, specifically that certain locations were either customer *collection* or *destination* points.

The results demonstrate that CONNEKT was able to modify its behaviour and attempt different assignments depending on the relationships between collection and destination locations in the tour, and the current and maximum capacity values of the vehicles. This included breaking existing tours and optimally assigning new customers after the point at which the vehicle had taken an existing customer to his or her specified destination.

In addition to the open question on scalability, perhaps the most significant criticism that can be levied at the approach is the fact that it is computationally very expensive.

Aarts and Korst [1989a,1989b] report that serial techniques like the Lin-Kernighan heuristic (Lin and Kernighan, 1973) and sequential simulated annealing (Kirkpatrick, Gelatt and Vecchi [1983]) take only a few seconds (on a VAX-780) to generate solutions to travelling salesman problems that are of a slightly higher quality than those generated in a few minutes by their asynchronous parallel Boltzmann machine model.

However, Aarts and Korst [1989a, 1989b] also suggest that their Boltzmann machine model may run many orders of magnitude more quickly than sequential algorithms if it is

implemented on an array of parallel processors. They state that given this type of hardware optimization (Aarts and Korst 1989a):

"typical running times of a Boltzmann machine would be in the region of a few milliseconds".

A similar argument can be made for improving the processing speed of the Pocket Boltzmann machine, to that suggested by Aarts and Korst[1989a, 1989b]. This follows because the 'Pocket' modification does not interfere with the operation the Boltzmann machine, and if implemented in parallel, need not directly interfere with attempts to significantly increase the processing speed.

This could be achieved by the addition of a set of connections dedicated to reading the output state of each of the units in the network. In practice, it is possible that the processing speed of a parallel Pocket Boltzmann machine may have to be reduced to the rate at which state data can be read from the parallel arrays, however, since the state information is in a binary $\{0,1\}$ form, this process, although serial, could still be very fast.

In the simulations described in this thesis, a single instance of the 'Pocket' algorithm has been used to examine the states of all of the units in the network, and determine if they correspond to a higher quality solution after *any* of the units changed its state. However a processing 'bottleneck' could occur if a single 'Pocket algorithm instance attempted to evaluate the stream of binary data generated by the states of the network. This processing 'bottleneck' could be relieved by implementing a set of 'Pocket' algorithms in parallel. Each algorithm running on a separate processor, could examine a portion of the binary data stream generated by the outputs from the Boltzmann machine model, comparing any current solution with a single common copy of the best solution so far. The single common copy of the best solution could either be stored at a single common location, which can be read by all of the processors running instances of the 'Pocket'

algorithm, or the best copy could be communicated (broadcast) between the processors and stored locally in each 'Pocket' algorithm processor.

Finally, whilst it is readily accepted that the large processing times associated with sequential simulations of models like the Pocket Boltzmann machine model, are unlikely to be commercially acceptable, it is conjectured that, provided that the approach is scalable and that the increase in processing speed that has been suggested proves to be feasible; connectionist optimization approaches of the type described in this thesis could become viable.

Appendices

Appendix A: Results for Problem A

Run No.	Step	Tour	Tour Length	Incremental Cost
1	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C5-C4-C3-D4-D5-D3	4.581	1.370
	Assign new Customer 5 to vehicle V1			
2	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C2-D2-C5-D5-C1-D1	4.459	1.269
	Determine cost of assigning customer 5 to V2	V2-C5-C4-D4-C3-D3-D5	3.810	0.599
	Assign new Customer 5 to vehicle V2			
3	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C4-D4-C3-C5-D5-D3	3.700	0.489
	Assign new Customer 5 to vehicle V1			
4	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	No tour	10.000	6.789
	Assign new Customer 5 to vehicle V1			
5	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C4-D4-C3-C5-D3-D5	3.699	0.488
	Assign new Customer 5 to vehicle V1			
6	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C4-D4-C5-C3-D3-D5	4.425	1.214
	Assign new Customer 5 to vehicle V1			
7	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D4-D3	4.772	1.561
	Assign new Customer 5 to vehicle V1			
8	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C4-C3-D4-C5-D5-D3	4.150	0.939
	Assign new Customer 5 to vehicle V1			

Appendix A: Results for Problem A (cont.)

Run No.	Step	Tour	Tour Length	Incremental Cost
9	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C4-D4-C3-C5-D3-D5	3.699	0.488
	Assign new Customer 5 to vehicle V1			
10	Develop initial tour for V1	V1-C1-D1-C2-D2	3.190	
	Develop initial tour for V2	V2-C4-D4-C3-D3	3.211	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-D1-C2-D2	3.265	0.075
	Determine cost of assigning customer 5 to V2	V2-C3-C4-D4-C5-D5-D3	4.295	1.083
	Assign new Customer 5 to vehicle V1			

Appendix B: Results for Problem B

Run No.	Step	Tour	Tour Length	Incremental Cost
1	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-D1-C2-C6-D6-D2	6.062	2.816
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-C6-D6-D5-D3-D4	5.075	0.899
	Assign new Customer 6 to vehicle V2			
2	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-C6-D6-D5-D3-D4	5.075	0.899
	Assign new Customer 6 to vehicle V2			
3	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-C6-D6-D5-D3-D4	5.075	0.899
	Assign new Customer 6 to vehicle V2			
4	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-C6-D6-D2	5.720	2.474
	Determine cost of assigning customer 6 to V2	V2-C3-C4-D4-D3-C5-D5-C6-D6	7.055	2.878
	Assign new Customer 6 to vehicle V1			
5	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C4-C5-C3-D3-D5-C6-D6-D4	7.065	2.889
	Assign new Customer 6 to vehicle V1			

Appendix B: Results for Problem B (cont.)

Run No.	Step	Tour	Tour Length	Incremental Cost
6	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C5-D5-C2-C1-D2-D1	5.947	2.701
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-C6-D6-D5-D3-D4	5.075	0.899
	Assign new Customer 6 to vehicle V2			
7	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-D1-C6-D6-C2-D2	5.661	2.415
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-C6-D5-D6-D3-D4	5.907	1.731
	Assign new Customer 6 to vehicle V1			
8	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C2-C6-D6-C1-D1-D2	6.094	2.849
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-C6-D6-D5-D3-D4	5.075	0.899
	Assign new Customer 6 to vehicle V2			
9	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-D3-C4-D4-C6-D6	7.085	3.839
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-C6-D6-D5-D3-D4	5.075	0.899
	Assign new Customer 6 to vehicle V2			
10	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C4-C5-C6-D6-D5-C3-D3-D4	6.471	2.295
	Assign new Customer 6 to vehicle V1			

Appendix C: Results for Problem C

Run No.	Step	Tour	Tour Length	Incremental Cost
1	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C2-C1-D2-D1-C6-D6	5.917	2.672
	Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-C6-D6-D4	5.690	2.445
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-D5-C6-D6-D3-D4	5.408	1.232
	Assign new Customer 6 to vehicle V2			
2	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-D1-C6-D6-C2-D2	5.661	2.415
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-C5-D5-D3-D4	6.232	2.055
	Assign new Customer 6 to vehicle V1			
3	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C2-C6-D6-C1-D2-D1	6.272	3.026
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D4-D3	5.805	2.560
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C4-C5-C6-D6-D5-C3-D3-D4	6.471	2.295
	Assign new Customer 6 to vehicle V1			
4	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D4-D3	5.805	2.560
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C4-C5-C6-D6-D5-C3-D3-D4	6.471	2.295
	Assign new Customer 6 to vehicle V1			
5	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-D3-C4-C6-D6-D4	6.012	2.766
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C4-C5-C6-D6-D5-C3-D3-D4	6.471	2.295
	Assign new Customer 6 to vehicle V1			

Appendix C: Results for Problem C (cont.)

Run No.	Step	Tour	Tour Length	Incremental Cost
6	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-C6-D6-D4	5.690	2.445
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C5-C6-D6-D5-C3-D3-C4-D4	7.578	3.402
	Assign new Customer 6 to vehicle V1			
7	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C4-C5-C6-D6-D5-C3-D3-D4	6.471	2.295
	Assign new Customer 6 to vehicle V1			
8	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C4-C5-C6-D6-D5-C3-D3-D4	6.471	2.295
	Assign new Customer 6 to vehicle V1			
9	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D6-D3-D4	5.034	1.788
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-D3-C4-C5-C6-D6-D5-D4	6.052	1.876
	Assign new Customer 6 to vehicle V1			
10	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	5.106	1.860
	Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	4.177	0.931
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-C6-D6-D4	5.690	2.445
	Assign new Customer 5 to vehicle V2			
	Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D6-D1-D2	4.866	1.620
	Determine cost of assigning customer 6 to V2	V2-C3-C4-C5-D5-C6-D6-D3-D4	5.408	1.232
	Assign new Customer 6 to vehicle V2			

Appendix D: Results for Reverse, Random Assignment

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost	
1	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (1.958,1.338) D5 (1.152,1.588) C6 (0.752,1.034) D6 (1.908,1.020)			
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-C2-D1-D2	5.057	1.812	
		Determine cost of assigning customer 5 to V2	V2-C3-C5-C4-D3-D5-D4	3.366	0.120	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D2-D1-C6-D6	5.657	2.411	
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-C4-D3-D4	4.098	0.852	
		Assign new Customer 5 to vehicle V2			0.120	
		Determine cost of assigning customer 6 to V1	V1-C1-D1-C2-D2-C6-D6	5.731	2.485	
		Determine cost of assigning customer 6 to V2	V2-C6-C4-C5-D6-C3-D4-D5-D3	4.864	1.498	
		Assign new Customer 6 to vehicle V2			1.498	
		Total Incremental Cost			1.618	
		Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
			Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C5-D5-C2-C1-D1-D2	5.085	1.839	
	Determine cost of assigning customer 5 to V2		V2-C3-C5-C4-D3-D5-D4	3.366	0.120	
	Determine cost of assigning customer 6 to V1		V1-C1-C2-D2-D1-C6-D6	5.657	2.411	
	Determine cost of assigning customer 6 to V2		V2-C6-D6-C3-C4-D3-D4	4.098	0.852	
	Assign new Customer 6 to vehicle V1				2.411	
	Determine cost of assigning customer 5 to V1		V1-C1-C2-C6-D6-C5-D5-D1-D2	6.653	0.996	
Determine cost of assigning customer 5 to V2	V2-C3-C5-C4-D3-D5-D4	3.366	0.120			
Assign new Customer 5 to vehicle V1			0.996			
Total Incremental Cost			3.407			
2	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (1.662,1.806) D5 (1.174,1.968) C6 (1.552,1.338) D6 (1.608,1.262)			
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	4.772	1.526	
		Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4	3.426	0.180	
		Determine cost of assigning customer 6 to V1	V1-C6-D6-C1-C2-D1-D2	4.360	1.114	
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-C4-D3-D4	3.329	0.083	
		Assign new Customer 6 to vehicle V2			0.083	
		Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D5-D1-D2	4.772	1.526	
		Determine cost of assigning customer 5 to V2	V2-C3-C4-C5-D5-D3-D4-C6-D6	4.093	0.764	
		Assign new Customer 5 to vehicle V2			0.764	
		Total Incremental Cost			0.847	
		Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
			Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C1-C5-D5-C2-D1-D2	4.622	1.376	
	Determine cost of assigning customer 5 to V2		V2-C3-C4-C5-D5-D3-D4	3.426	0.180	
	Determine cost of assigning customer 6 to V1		V1-C6-D6-C1-C2-D1-D2	4.360	1.114	
	Determine cost of assigning customer 6 to V2		V2-C6-D6-C3-C4-D3-D4	3.329	0.083	
	Assign new Customer 5 to vehicle V1				1.376	
	Determine cost of assigning customer 6 to V1		V1-C5-C6-D6-C1-C2-D5-D1-D2	5.808	1.186	
Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-C4-D3-D4	3.329	0.083			
Assign new Customer 6 to vehicle V1			1.186			
Total Incremental Cost			2.562			

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost	
3	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (0.110,1.160) D5 (0.608,0.992) C6 (0.414,0.160) D6 (0.180,0.894)			
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5	3.475	0.230	
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	4.832	1.587	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-D6	5.148	1.903	
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-C4-D3-D4	5.856	2.610	
		Assign new Customer 5 to vehicle V1			0.230	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C5-D5-C6-D6	5.535	2.059	
		Determine cost of assigning customer 6 to V2	V2-C4-C3-D3-D4-C6-D6	6.390	3.145	
	Assign new Customer 6 to vehicle V1			2.059		
	Total Incremental Cost			2.289		
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5	3.475	0.230	
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	4.832	1.587	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-D6	5.148	1.903	
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-D4-C6-D6	5.864	2.618	
		Assign new Customer 6 to vehicle V2			2.618	
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5	3.475	0.230	
Determine cost of assigning customer 5 to V2		V2-C6-D6-C5-D5-C4-C3-D3-D4	6.661	0.797		
Assign new Customer 5 to vehicle V2				0.797		
Total Incremental Cost			3.415			
4	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (0.560,1.138) D5 (1.258,0.094) C6 (0.534,1.496) D6 (0.198,0.056)			
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5	4.984	1.738	
		Determine cost of assigning customer 5 to V2	V2-C5-C3-C4-D3-D4-D5	5.905	2.659	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D1-D2-D6	5.190	1.945	
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D4-D3-C6-D6	6.256	3.010	
		Assign new Customer 5 to vehicle V1			1.738	
		Determine cost of assigning customer 6 to V1	No Tour	10.000	5.016	
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D4-D3-C6-D4-D6	6.617	3.371	
	Assign new Customer 6 to vehicle V2			3.371		
	Total Incremental Cost			5.109		
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-C2-D1-D2	5.899	2.654	
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D4-D3-C5-D5	5.948	2.702	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D1-D2-D6	5.190	1.945	
		Determine cost of assigning customer 6 to V2	V2-C3-D3-C4-D4-C6-D6	6.851	3.606	
		Assign new Customer 6 to vehicle V2			3.606	
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-C2-D1-D2	5.899	2.654	
Determine cost of assigning customer 5 to V2		V2-C5-C4-D4-C3-D3-C6-D6-D5	8.569	1.717		
Assign new Customer 5 to vehicle V1				2.654		
Total Incremental Cost			6.260			

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost	
5	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (0.730,1.512) D5 (0.224,1.134) C6 (0.982,1.796) D6 (1.358,1.058)			
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-C5-C2-D1-D2-D5	3.401	0.155	
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	4.701	1.455	
		Determine cost of assigning customer 6 to V1	V1-C1-C6-C2-D1-D2-D6	4.043	0.797	
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-C6-D4-D6	3.696	0.450	
		Assign new Customer 5 to vehicle V1			0.155	
		Determine cost of assigning customer 6 to V1	V1-C1-C5-C2-C6-D6-D5-D2-D1	5.181	1.780	
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-C6-D4-D6	3.696	0.450	
		Assign new Customer 6 to vehicle V2			0.450	
		Total Incremental Cost			0.605	
		Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
			Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C1-C5-C2-D1-D2-D5	3.401	0.155	
	Determine cost of assigning customer 5 to V2		V2-C3-C4-D3-D4-C5-D5	4.701	1.455	
	Determine cost of assigning customer 6 to V1		V1-C1-C2-C6-D1-D2-D6	4.100	0.854	
	Determine cost of assigning customer 6 to V2		V2-C3-C4-D3-C6-D4-D6	3.696	0.450	
	Assign new Customer 5 to vehicle V2				1.455	
	Determine cost of assigning customer 6 to V1		V1-C1-C6-C2-D1-D2-D6	4.043	0.797	
	Determine cost of assigning customer 6 to V2		V2-C3-C4-D4-C5-D5-C6-D3-D6	5.871	1.170	
	Assign new Customer 6 to vehicle V2				1.170	
	Total Incremental Cost				2.625	
	6		Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (0.652,0.244) D5 (1.626,0.038) C6 (0.318,1.218) D6 (1.386,1.900)	
		Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
Determine cost of assigning customer 5 to V1		V1-C1-C2-D1-D2-C5-D5		5.683	2.438	
Determine cost of assigning customer 5 to V2		V2-C5-D5-C3-C4-D3-D4		5.493	2.247	
Determine cost of assigning customer 6 to V1		V1-C1-C2-D2-C6-D1-D6		4.944	1.699	
Determine cost of assigning customer 6 to V2		V2-C6-D6-C3-C4-D3-D4		5.103	1.858	
Assign new Customer 6 to vehicle V1					1.699	
Determine cost of assigning customer 5 to V1		No Tour		10.000	5.056	
Determine cost of assigning customer 5 to V2		V2-C5-D5-C3-C4-D3-D4		5.493	2.247	
Assign new Customer 5 to vehicle V2					2.247	
Total Incremental Cost					3.946	
Greatest		Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5	5.683	2.438	
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D4-D3	5.949	2.704	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D2-C6-D1-D6	4.944	1.699	
		Determine cost of assigning customer 6 to V2	V2-C6-C3-C4-D3-D6-D4	5.016	1.770	
		Assign new Customer 5 to vehicle V2			2.704	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-D6	4.781	1.536	
		Determine cost of assigning customer 6 to V2	V2-C6-C5-D5-C3-D3-C4-D6-D4	7.204	1.255	
		Assign new Customer 6 to vehicle V1			1.536	
		Total Incremental Cost			4.239	

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost
7	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(0.856,1.832) D5(0.648,0.526) C6(1.814,0.438) D6(0.156,1.994)		
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D1-D2-D5	3.880	0.635
		Determine cost of assigning customer 5 to V2	V2-C4-C3-D3-C5-D4-D5	5.335	2.090
		Determine cost of assigning customer 6 to V1	V1-C6-C1-C2-D1-D6-D2	5.230	1.984
		Determine cost of assigning customer 6 to V2	V2-C6-C3-C4-D6-D3-D4	6.126	2.880
		Assign new Customer 5 to vehicle V1			0.635
		Determine cost of assigning customer 6 to V1	No Tour	10.000	6.120
		Determine cost of assigning customer 6 to V2	V2-C6-C3-C4-D3-D6-D4	5.995	2.749
		Assign new Customer 6 to vehicle V2			2.749
		Total Incremental Cost			3.384
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D1-D2-D5	3.880	0.635
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-C5-D5-D4	5.436	2.190
		Determine cost of assigning customer 6 to V1	V1-C6-C1-C2-D2-D1-D6	5.727	2.481
		Determine cost of assigning customer 6 to V2	V2-C6-C3-C4-D4-D3-D6	6.160	2.915
		Assign new Customer 6 to vehicle V2			2.915
		Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D1-D2-D5	3.880	0.635
		Determine cost of assigning customer 5 to V2	V2-C6-C3-C4-D3-D6-C5-D4-D5	7.242	1.081
		Assign new Customer 5 to vehicle V2			1.081
		Total Incremental Cost			3.996
		8	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(1.836,0.908) D5(0.850,1.660) C6(1.006,1.212) D6(0.656,0.684)
Develop initial tour for V1	V1-C1-C2-D1-D2			3.246	
Develop initial tour for V2	V2-C3-C4-D3-D4			3.246	
Determine cost of assigning customer 5 to V1	V1-C5-C2-D5-C1-D1-D2			5.031	1.785
Determine cost of assigning customer 5 to V2	V2-C5-C3-C4-D3-D5-D4			3.936	0.691
Determine cost of assigning customer 6 to V1	V1-C6-C1-C2-D1-D2-D6			3.669	0.423
Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-D4-C6-D6			4.169	0.924
Assign new Customer 6 to vehicle V1					0.423
Determine cost of assigning customer 5 to V1	V1-C6-C5-D6-D5-C2-C1-D2-D1			6.514	2.845
Determine cost of assigning customer 5 to V2	V2-C5-C3-C4-D3-D5-D4			3.936	0.691
Assign new Customer 5 to vehicle V2					0.691
Total Incremental Cost					1.113
Greatest	Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C5-D5-C2-C1-D2-D1	5.303	2.057
	Determine cost of assigning customer 5 to V2		V2-C5-C3-C4-D3-D5-D4	3.936	0.691
	Determine cost of assigning customer 6 to V1		V1-C6-C1-C2-D1-D2-D6	3.669	0.423
	Determine cost of assigning customer 6 to V2		V2-C3-C4-D3-D4-C6-D6	4.169	0.924
	Assign new Customer 5 to vehicle V1				2.057
	Determine cost of assigning customer 6 to V1		V1-C2-C6-C5-D6-C1-D5-D1-D2	6.869	1.566
	Determine cost of assigning customer 6 to V2		V2-C3-C4-D3-D4-C6-D6	4.169	0.924
	Assign new Customer 6 to vehicle V1				1.566
	Total Incremental Cost				3.623

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost	
9	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (1.404,1.438) D5 (1.988,1.492) C6 (0.228,0.504) D6 (0.746,1.352)			
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-C5-D5-C2-D1-D2	5.156	1.910	
		Determine cost of assigning customer 5 to V2	V2-C5-C3-D5-C4-D3-D4	3.484	0.238	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-D6	4.616	1.370	
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C4-C3-D3-D4	5.662	2.416	
		Assign new Customer 5 to vehicle V2			0.238	
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D2-C6-D1-D6	5.367	2.121	
		Determine cost of assigning customer 6 to V2	No Tour	10.000	6.516	
		Assign new Customer 6 to vehicle V1			2.121	
		Total Incremental Cost			2.359	
		Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
			Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C1-C5-D5-C2-D1-D2	5.156	1.910	
	Determine cost of assigning customer 5 to V2		V2-C5-D5-C3-C4-D3-D4	3.842	0.597	
	Determine cost of assigning customer 6 to V1		V1-C1-C2-D1-D2-C6-D6	4.616	1.370	
	Determine cost of assigning customer 6 to V2		V2-C6-D6-C3-C4-D3-D4	5.358	2.112	
	Assign new Customer 6 to vehicle V2				2.112	
	Determine cost of assigning customer 5 to V1		V1-C5-D5-C1-C2-D1-D2	5.172	1.927	
	Determine cost of assigning customer 5 to V2		No Tour	10.000	4.642	
	Assign new Customer 5 to vehicle V2				4.642	
	Total Incremental Cost				6.754	
	10		Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5 (1.146,0.104) D5 (0.648,0.400) C6 (1.924,0.968) D6 (0.034,1.708)	
		Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
Determine cost of assigning customer 5 to V1		V1-C5-D5-C1-C2-D2-D1		5.824	2.578	
Determine cost of assigning customer 5 to V2		V2-C3-C4-D3-D4-C5-D5		5.371	2.125	
Determine cost of assigning customer 6 to V1		V1-C6-C1-C2-D1-D6-D2		4.960	1.714	
Determine cost of assigning customer 6 to V2		V2-C6-C3-C4-D4-D3-D6		5.588	2.342	
Assign new Customer 6 to vehicle V1					1.714	
Determine cost of assigning customer 5 to V1		V1-C5-C6-C1-C2-D6-D1-D2-D5		7.265	2.305	
Determine cost of assigning customer 5 to V2		V2-C3-C4-D3-D4-C5-D5		5.371	2.125	
Assign new Customer 5 to vehicle V2				2.125		
Total Incremental Cost				3.839		
Greatest		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-D1-C2-D2-C5-D5	5.854	2.608	
		Determine cost of assigning customer 5 to V2	V2-C4-C3-D3-D4-C5-D5	5.897	2.651	
		Determine cost of assigning customer 6 to V1	V1-C6-C1-C2-D1-D6-D2	4.960	1.714	
		Determine cost of assigning customer 6 to V2	V2-C6-C3-C4-D3-D6-D4	5.534	2.288	
		Assign new Customer 5 to vehicle V2			2.651	
		Determine cost of assigning customer 6 to V1	V1-C6-C1-C2-D1-D6-D2	4.960	1.714	
	Determine cost of assigning customer 6 to V2	V2-C4-C6-C3-D3-D6-D4-C5-D5	8.440	2.543		
	Assign new Customer 6 to vehicle V2			2.543		
Total Incremental Cost			5.195			

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost
11	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(1.078,1.098) D5(0.446,0.032) C6(0.796,1.918) D6(1.566,1.358)		
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-C1-D1-C2-D2-D5	5.550	2.304
		Determine cost of assigning customer 5 to V2	V2-C3-D3-C4-D4-C5-D5	6.142	2.896
		Determine cost of assigning customer 6 to V1	V1-C1-C2-C6-D1-D2-D6	4.462	1.216
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-C6-D4-D6	4.407	1.162
		Assign new Customer 6 to vehicle V2			1.162
		Determine cost of assigning customer 5 to V1	V1-C5-C1-D1-C2-D2-D5	5.550	2.304
		Determine cost of assigning customer 5 to V2	No Tour	10.000	5.593
	Assign new Customer 5 to vehicle V1			2.304	
	Total Incremental Cost			3.465	
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-C1-D1-C2-D2-D5	5.550	2.304
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	5.479	2.233
		Determine cost of assigning customer 6 to V1	V1-C2-C1-D2-D1-C6-D6	4.857	1.611
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-C6-D4-D6	4.407	1.162
		Assign new Customer 5 to vehicle V1			2.304
		Determine cost of assigning customer 6 to V1	No tour	10.000	4.451
Determine cost of assigning customer 6 to V2		V2-C3-C4-D3-C6-D4-D6	4.407	1.162	
Assign new Customer 6 to vehicle V1				4.451	
Total Incremental Cost			6.754		
12	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(1.104,0.668) D5(0.106,0.260) C6(1.818,1.272) D6(1.688,1.406)		
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-C1-D1-C2-D2-D5	5.952	2.706
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	5.775	2.530
		Determine cost of assigning customer 6 to V1	V1-C6-D6-C2-C1-D1-D2	4.826	1.580
		Determine cost of assigning customer 6 to V2	V2-C3-C6-D6-C4-D3-D4	3.389	0.144
		Assign new Customer 6 to vehicle V2			0.144
		Determine cost of assigning customer 5 to V1	V1-C2-C1-C5-D5-D2-D1	6.186	2.940
		Determine cost of assigning customer 5 to V2	V2-C4-C6-D6-C3-D4-C5-D5-D3	7.483	4.094
	Assign new Customer 5 to vehicle V1			2.940	
	Total Incremental Cost			3.084	
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C2-C1-C5-D5-D2-D1	6.186	2.940
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	5.775	2.530
		Determine cost of assigning customer 6 to V1	V1-C6-D6-C2-C1-D1-D2	4.826	1.580
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-C4-D3-D4	3.594	0.349
		Assign new Customer 5 to vehicle V1			2.940
		Determine cost of assigning customer 6 to V1	V1-C6-D6-C1-D1-C2-C5-D5-D2	7.810	1.624
Determine cost of assigning customer 6 to V2		V2-C6-C3-D6-C4-D3-D4	3.364	0.119	
Assign new Customer 6 to vehicle V1				1.624	
Total Incremental Cost			4.565		

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost	
13	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(1.916,1.838) D5(0.528,0.808) C6(1.832,1.368) D6(1.364,1.934)			
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246		
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246		
		Determine cost of assigning customer 5 to V1	V1-C1-C2-C5-D1-D2-D5	5.507	2.262	
		Determine cost of assigning customer 5 to V2	V2-C3-C5-C4-D3-D4-D5	4.210	0.964	
		Determine cost of assigning customer 6 to V1	V1-C6-D6-C2-C1-D1-D2	5.023	1.777	
		Determine cost of assigning customer 6 to V2	V2-C3-C6-C4-D6-D3-D4	3.261	0.015	
		Assign new Customer 6 to vehicle V2			0.015	
		Determine cost of assigning customer 5 to V1	V1-C1-C5-C2-D1-D2-D5	5.295	2.050	
		Determine cost of assigning customer 5 to V2	V2-C3-C5-C4-C6-D6-D3-D4-D5	5.051	1.790	
		Assign new Customer 5 to vehicle V2			1.790	
		Total Incremental Cost			1.805	
		Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
			Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C2-C5-C1-D1-D2-D5	5.752	2.506	
	Determine cost of assigning customer 5 to V2		V2-C3-C5-C4-D3-D4-D5	4.210	0.964	
	Determine cost of assigning customer 6 to V1		V1-C6-D6-C2-C1-D1-D2	5.023	1.777	
	Determine cost of assigning customer 6 to V2		V2-C3-C6-C4-D6-D3-D4	3.261	0.015	
	Assign new Customer 5 to vehicle V1				2.506	
	Determine cost of assigning customer 6 to V1		V1-C6-C5-D6-C2-D2-C1-D1-D5	6.597	0.845	
	Determine cost of assigning customer 6 to V2		V2-C3-C6-C4-D6-D3-D4	3.261	0.015	
	Assign new Customer 6 to vehicle V1				0.845	
	Total Incremental Cost				3.351	
	14		Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(1.752,0.408) D5(1.234,0.138) C6(1.960,1.148) D6(0.108,0.930)	
		Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
Determine cost of assigning customer 5 to V1		V1-C2-C1-D2-D1-C5-D5		6.306	3.060	
Determine cost of assigning customer 5 to V2		V2-C5-D5-C3-C4-D3-D4		5.225	1.979	
Determine cost of assigning customer 6 to V1		V1-C6-C2-C1-D1-D2-D6		5.533	2.288	
Determine cost of assigning customer 6 to V2		V2-C6-C3-C4-D3-D4-D6		5.220	1.974	
Assign new Customer 6 to vehicle V2					1.974	
Determine cost of assigning customer 5 to V1		V1-C5-D5-C1-C2-D1-D2		5.674	2.429	
Determine cost of assigning customer 5 to V2		V2-C3-D3-C4-D4-C6-C5-D5-D6		7.886	2.666	
Assign new Customer 5 to vehicle V1					2.429	
Total Incremental Cost					4.403	
Greatest		Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-C2-D2-D1	6.201	2.955	
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D3-D4	5.225	1.979	
		Determine cost of assigning customer 6 to V1	V1-C6-C2-C1-D1-D2-D6	5.533	2.288	
		Determine cost of assigning customer 6 to V2	V2-C3-C6-C4-D3-D4-D6	5.250	2.004	
		Assign new Customer 5 to vehicle V1			2.955	
		Determine cost of assigning customer 6 to V1	V1-C5-C6-D5-C1-C2-D2-D1-D6	8.249	2.048	
		Determine cost of assigning customer 6 to V2	V2-C6-C3-C4-D3-D4-D6	5.220	1.974	
		Assign new Customer 6 to vehicle V1			2.048	
		Total Incremental Cost			5.003	

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost
15	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(1.050,0.524) D5(1.682,1.466) C6(0.962,0.970) D6(0.614,0.746)		
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C2-C1-D1-D2	5.359	2.113
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C4-C3-D4-D3	4.724	1.478
		Determine cost of assigning customer 6 to V1	V1-C6-C1-C2-D1-D2-D6	3.606	0.360
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-C4-D3-D4	4.163	0.917
		Assign new Customer 6 to vehicle V1			0.360
		Determine cost of assigning customer 5 to V1	V1-C5-C6-D5-C2-C1-D1-D2-D6	5.853	2.247
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D3-D4	4.208	0.962
		Assign new Customer 5 to vehicle V2			0.962
	Total Incremental Cost			1.322	
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C2-C1-D1-D2	5.359	2.113
		Determine cost of assigning customer 5 to V2	V2-C5-C3-D5-C4-D3-D4	4.022	0.777
		Determine cost of assigning customer 6 to V1	V1-C6-C1-C2-D1-D2-D6	3.606	0.360
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-D4-C6-D6	4.166	0.920
		Assign new Customer 5 to vehicle V1			2.113
		Determine cost of assigning customer 6 to V1	V1-C5-C6-D5-C2-C1-D6-D2-D1	6.494	1.135
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-D4-C6-D6	4.166	0.920
		Assign new Customer 6 to vehicle V1			1.135
Total Incremental Cost				3.249	
16	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(1.614,0.448) D5(1.782,0.948) C6(0.040,0.234) D6(1.994,0.346)		
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5	5.353	2.108
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D3-D4	4.097	0.851
		Determine cost of assigning customer 6 to V1	V1-C2-D2-C6-D6-C1-D1	8.562	5.316
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-C4-D3-D4	6.550	3.304
		Assign new Customer 5 to vehicle V2			0.851
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-D6	6.616	3.371
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C4-C3-D3-D4-C5-D5	9.171	5.074
		Assign new Customer 6 to vehicle V1			3.371
	Total Incremental Cost			4.222	
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5	5.353	2.108
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D3-D4	4.097	0.851
		Determine cost of assigning customer 6 to V1	V1-C2-D2-C1-D1-C6-D6	8.111	4.865
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C4-C3-D3-D4	7.269	4.023
		Assign new Customer 6 to vehicle V1			4.865
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C6-C5-D6-D5	6.873	-1.238
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D3-D4	4.097	0.851
		Assign new Customer 5 to vehicle V2			0.851
Total Incremental Cost				5.716	

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost
17	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(0.872,1.126) D5(1.584,0.234) C6(0.260,0.914) D6(0.942,0.612)		
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-C2-D1-D2	5.524	2.279
		Determine cost of assigning customer 5 to V2	V2-C4-C3-D4-D3-C5-D5	5.889	2.643
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-D6	3.920	0.674
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-D4-C6-D6	4.995	1.749
		Assign new Customer 6 to vehicle V1			0.674
		Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C6-C5-D6-D5	5.657	1.736
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-C5-D4-D5	5.716	2.470
		Assign new Customer 5 to vehicle V1			1.736
		Total Incremental Cost			2.411
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-D5-C1-C2-D1-D2	5.524	2.279
		Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D3-D4	4.800	1.554
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-D6	3.920	0.674
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D4-D3-C6-D6	5.255	2.009
		Assign new Customer 5 to vehicle V1			2.279
		Determine cost of assigning customer 6 to V1	V1-C1-C2-D1-D2-C6-C5-D6-D5	5.657	0.132
		Determine cost of assigning customer 6 to V2	V2-C3-C4-D3-D4-C6-D6	4.995	1.749
		Assign new Customer 6 to vehicle V2			1.749
		Total Incremental Cost			4.028
		18	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(0.968,1.268) D5(1.428,1.348) C6(1.590,1.712) D6(0.140,1.434)
Develop initial tour for V1	V1-C1-C2-D1-D2			3.246	
Develop initial tour for V2	V2-C3-C4-D3-D4			3.246	
Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5			4.178	0.933
Determine cost of assigning customer 5 to V2	V2-C5-D5-C3-C4-D3-D4			3.503	0.257
Determine cost of assigning customer 6 to V1	V1-C1-C6-C2-D1-D6-D2			4.385	1.140
Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D4-D3-D6			4.887	1.641
Assign new Customer 5 to vehicle V2					0.257
Determine cost of assigning customer 6 to V1	V1-C1-C6-C2-D1-D6-D2			4.385	1.140
Determine cost of assigning customer 6 to V2	V2-C4-C6-C3-D4-D6-D3-C5-D5			6.494	2.991
Assign new Customer 6 to vehicle V1					1.140
Total Incremental Cost					1.397
Greatest	Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C5-C1-D5-C2-D1-D2	4.195	0.949
	Determine cost of assigning customer 5 to V2		V2-C5-D5-C3-C4-D3-D4	3.503	0.257
	Determine cost of assigning customer 6 to V1		V1-C6-C2-C1-D1-D6-D2	4.523	1.278
	Determine cost of assigning customer 6 to V2		V2-C3-C4-C6-D4-D3-D6	4.887	1.641
	Assign new Customer 6 to vehicle V2				1.641
	Determine cost of assigning customer 5 to V1		V1-C5-C1-D5-C2-D1-D2	4.195	0.949
	Determine cost of assigning customer 5 to V2		V2-C5-D5-C3-D3-C4-C6-D4-D6	5.743	0.856
	Assign new Customer 5 to vehicle V1				0.949
	Total Incremental Cost				2.591

Appendix D: Results for Reverse, Random Assignment (cont.)

Run No.	Strategy	Step	Tour	Tour Length	Incremental Cost
19	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(0.786,0.302) D5(0.164,1.176) C6(1.032,1.814) D6(1.478,0.014)		
		Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-C1-C2-D1-D2-D5	4.756	1.510
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	5.899	2.653
		Determine cost of assigning customer 6 to V1	V1-C2-C6-C1-D2-D1-D6	6.242	2.996
		Determine cost of assigning customer 6 to V2	V2-C3-C4-C6-D3-D4-D6	5.440	2.195
		Assign new Customer 5 to vehicle V1			1.510
		Determine cost of assigning customer 6 to V1	V1-C1-C6-D1-C2-D2-C5-D6-D5	8.072	3.317
		Determine cost of assigning customer 6 to V2	V2-C3-D3-C4-C6-D4-D6	6.383	3.137
		Assign new Customer 6 to vehicle V2			3.137
		Total Incremental Cost			4.647
	Greatest	Develop initial tour for V1	V1-C1-C2-D1-D2	3.246	
		Develop initial tour for V2	V2-C3-C4-D3-D4	3.246	
		Determine cost of assigning customer 5 to V1	V1-C5-C1-C2-D1-D2-D5	4.756	1.510
		Determine cost of assigning customer 5 to V2	V2-C3-C4-D3-D4-C5-D5	5.899	2.653
		Determine cost of assigning customer 6 to V1	V1-C6-C2-C1-D2-D1-D6	6.188	2.942
		Determine cost of assigning customer 6 to V2	V2-C6-D6-C3-D3-C4-D4	7.049	3.803
		Assign new Customer 6 to vehicle V2			3.803
		Determine cost of assigning customer 5 to V1	V1-C5-C1-C2-D1-D5-D2	4.875	1.629
		Determine cost of assigning customer 5 to V2	V2-C3-C6-D3-C4-D4-D6-C5-D5	7.685	0.636
		Assign new Customer 5 to vehicle V1			1.629
		Total Incremental Cost			5.433
		20	Least	New random collection 'C' and destination 'D' coordinates for new customers '5' and '6'	C5(0.500,0.442) D5(1.366,1.326) C6(1.866,1.442) D6(1.322,0.802)
Develop initial tour for V1	V1-C1-C2-D1-D2			3.246	
Develop initial tour for V2	V2-C3-C4-D3-D4			3.246	
Determine cost of assigning customer 5 to V1	V1-C5-C1-D5-C2-D1-D2			5.375	2.129
Determine cost of assigning customer 5 to V2	V2-C5-C3-C4-D3-D4-D5			4.923	1.677
Determine cost of assigning customer 6 to V1	V1-C2-C1-D2-D1-C6-D6			5.604	2.358
Determine cost of assigning customer 6 to V2	V2-C3-C6-C4-D3-D4-D6			3.795	0.550
Assign new Customer 6 to vehicle V2					0.550
Determine cost of assigning customer 5 to V1	V1-C1-C2-D1-D2-C5-D5			5.028	1.782
Determine cost of assigning customer 5 to V2	V2-C5-C4-C3-C6-D6-D5-D3-D4			6.476	2.681
Assign new Customer 5 to vehicle V1					1.782
Total Incremental Cost					2.332
Greatest	Develop initial tour for V1		V1-C1-C2-D1-D2	3.246	
	Develop initial tour for V2		V2-C3-C4-D3-D4	3.246	
	Determine cost of assigning customer 5 to V1		V1-C5-C1-D5-C2-D1-D2	5.375	2.129
	Determine cost of assigning customer 5 to V2		V2-C5-C3-C4-D3-D4-D5	4.923	1.677
	Determine cost of assigning customer 6 to V1		V1-C6-D6-C2-C1-D1-D2	5.792	2.546
	Determine cost of assigning customer 6 to V2		V2-C3-C6-C4-D3-D4-D6	3.795	0.550
	Assign new Customer 6 to vehicle V1				2.546
	Determine cost of assigning customer 5 to V1		V1-C6-C2-C1-D1-D2-C5-D6-D5	6.866	1.075
	Determine cost of assigning customer 5 to V2		V2-C5-C3-C4-D3-D4-D5	4.923	1.677
	Assign new Customer 5 to vehicle V2				1.677
	Total Incremental Cost				4.223

Bibliography

- Aarts and Korst, 1989b** Aarts, E.H.L and Korst, J.H.M., *Simulated Annealing and Boltzmann Machines : A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley and Sons, Chichester, 1989.
- Aleksander and Morton, 1990** Aleksander, I., and Morton, H., *An Introduction to Neural Computing*, Chapman and Hall, London, 1990.
- Barkakati, 1993** Barkakati, N. (and revised by Hipson, P.), *Visual C++ Developers Guide*, 1993, Sams Publishing, Prentice-Hall, Carmel, Indiana, USA.
- Blum, 1992** Blum, A., *Neural Networks in C++*, 1992, John Wiley & Sons Inc.
- Booch, 1991** Booch, G., *Object Oriented Design With Applications*, Benjamin/Cummings, Redwood City, California, 1991.
- Budd, 1991** Budd, T., *An Introduction to Object-Oriented Programming*, Addison-Wesley, Reading, MA, 1991.
- Cox, 1986** Cox, B., *Object-Oriented Programming: An Evolutionary Approach*, Addison-Wesley, Reading, MA, 1986.
- Duncan, 1987** Duncan, T. "Advanced Physics Volume II: Fields, Waves and Atoms (3rd edition)", John Murray Ltd., London, 1987.
- Parsaye and Chignell, 1988** Parsaye, K. and Chignell, M.I., *Expert Systems for Experts*, John Wiley and Sons Inc., Chichester, UK, 1988.
- Parsaye et al., 1989** Parsaye, K., Chignell, M., Khoshafian, S. and Wong, H., *Intelligent Databases*, 1989, John Wiley, New York.
- Gallant, 1993** Gallant, S.I., "Neural Network learning and expert systems", MIT Press, 1993.
- Garey and Johnson, 1979** Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., San Francisco, 1979.
- Giarratano and Riley, 1989** Giarratano, J.C. and Riley, G., *Expert Systems : Principles and Programming*, PWS-Kent Publishing Co., Boston, MA., 1989.
- Hebb, 1949** Hebb, D., *Organization of Behaviour*, John Wiley, New York, 1949.
- Hoare, 1972** Hoare, C.A.R., "Notes on Data Structuring", *Structured Programming*, Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R (eds.), Academic Press, New York, pp.83-174.
- Hughes, 1991** Hughes, J.G., *Object-Oriented Databases*, 1991, Prentice-Hall International, Hemel Hempstead, 1991.

- Khoshafian and Abnous, 1990** Khoshafian, S. and Abnous, R., *Object Orientation: Concepts, Languages, Databases, User Interfaces*, 1990, John Wiley & Sons Inc.
- Kohonen, 1988** Kohonen, T., *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1988.
- Kruglinski, 1993** Kruglinski, D.J., 1993, *Inside Visual C++*, Microsoft Press, dist. by Penguin Books Ltd, Middlesex, 1993.
- Lafore, 1991** Lafore, R., "Object-Oriented Programming in Turbo C++", Waite Group, 1991.
- Lippman, 1991** Lippman, S.B., *C++ Primer (second edition)*, Addison-Wesley, Reading, Massachusetts, 1991.
- Meyer, 1988** Meyer, B., *Object-Oriented Software Construction*, Prentice-Hall Publishing, Englewood Cliffs, NJ, 1988.
- Microsoft, 1991** Microsoft Corporation, *Microsoft C/C++ V7.0 Reference Manuals*, Microsoft Corporation, Widdersh Triangle, Reading, U.K., 1993.
- Microsoft, 1993a** Microsoft Corporation, *Microsoft Visual C++ (v1.0) C/C++ Version 7.0 Update*, Microsoft Corporation, Widdersh Triangle, Reading, U.K., 1993.
- Microsoft, 1993b** Microsoft Corporation, *Microsoft Visual C++ (v1.0) Class Library User's Guide*, Microsoft Corporation, Widdersh Triangle, Widdersh, Reading, U.K., 1993.
- Microsoft, 1993c** Microsoft Corporation, *Microsoft Visual C++ (v1.0) Reference Volume I*, Microsoft Corporation, Widdersh Triangle, Widdersh, Reading, U.K., 1993.
- Microsoft, 1993d** Microsoft Corporation, *Microsoft Visual C++ (v1.0) User's Guides : Visual Workbench User's Guide; App Studio User's Guide*, Microsoft Corporation, Widdersh, Reading, U.K., 1993.
- Minsky and Papert, 1969** Minsky, M.L. and Papert, S.A., "Perceptrons - An Introduction to Computational Geometry", MIT Press, Cambridge, MA., 1969.
- Rich and Knight, 1991** Rich, E. and Knight, K., *Artificial Intelligence (Second Edition)*, McGraw-Hill, London, 1991.
- Rosenblatt, 1962** Rosenblatt, F., *The Principles of Neurodynamics*, Washington, Spartan Books, 1962.
- Schildt, 1987** Schildt, H., "C - The Complete Reference", Osborne McGraw-Hill, Berkeley, California, 1987.
- Shlaer and Mellor, 1988** Shlaer, S., and Mellor, S.J., *Object-Oriented Systems Analysis Modelling the World in Data*, Yourdon Press, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

- Simpson, 1990** Simpson, P.K., *Artificial Neural Systems: Foundations, Paradigms, Applications and Implementations*, Pergamon Press Inc., Oxford, 1990.
- Stroustrup, 1986** Stroustrup, B., *The C++ Programming Language*, Addison-Wesley, Reading, MA, 1986.
- Tello, 1989** Tello, E.R., *Object-oriented Programming for Artificial Intelligence*. 1989, Addison-Wesley, Reading, MA.
- Thro, 1991** Thro, E., *The Artificial Intelligence Dictionary*, 1991, Microtrend Books, Slawson Communications Inc., San Marcos, CA.
- University of Warwick, 1993** University of Warwick, "*Guide to Examinations for Higher Degrees by Research*", Graduate School, University of Warwick, Sep. 1993.
- Winston, 1992** Winston, P.H., *Artificial Intelligence (Third Edition)*, Addison-Wesley Publishing Company, Reading, MA., 1992

References

- Aarts and Korst, 1989a Aarts, E.H.L and Korst, J.H.M., "Boltzmann machines for travelling salesman problems", *European Journal of Operational Research* **39**, 1989, pp.79-95.
- Ackley, Hinton and Sejnowski, 1985 Ackley, D.H., Hinton, G.E. and Sejnowski, T.J., "A Learning Algorithm for Boltzmann Machines", *Cognitive Science* **9**, 1985, pp.147-169.
- Agrawal and Gehani, 1989 Agrawal, R. and Gehani, N.H., "ODE (Object database and environment): The Language and the Data Model", *ACM SIGMOD Conference on the Management of Data*, Portland, OR, 1989, pp.36-45.
- Akiyama et al.,1989 Akiyama,Y., Yamashita ,A., Kajiura, M. and Aiso, H., "Combinatorial Optimization with Gaussian Machines", *1989 International Joint Conference on Neural Networks*, Washington, D.C., 18-22 June 1989, Vol I, pp.533-540.
- Amari, 1972a Amari, S-I, "Characteristics of random nets of analog neuron-like elements", *IEEE Transactions on Systems, Man and Cybernetics*, SMC-2, pp.643-657.
- Amari, 1972b Amari, S-I, "Learning patterns and pattern sequences by self-organizing nets of threshold elements, *IEEE Transactions on Computers*, C-21, pp.1197-1206.
- Amari, 1974 Amari, S-I, "A method of statistical neurodynamics", *Kybernetik*, **14**, 1974, pp.201-215.
- Amari, 1977a Amari, S-I, "Dynamics of pattern formation in lateral-inhibition type neural fields", *Biological Cybernetics*, **27**, 1977, pp.77-87.
- Amari, 1977b Amari, S-I, "Neural theory of association and concept formation, *Biological Cybernetics*, **26**, 1977, pp.175-185.
- Alspector and Allen,1987 Alspector, J. and Allen, R.B., "A neuromorphic VLSI system", in Losleben, P., (Ed.), *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*, MIT Press, Cambridge, MA, p313.
- Amari and Maginu, 1988 Amari, S-I and Maginu, K., "Statistical Neurodynamics of associative memory", *Neural Networks*, **1**,1988, pp.63-74.
- Appoloni and de Falco, 1991 "Learning by Parallel Boltzmann Machines", *IEEE Transactions on Information Theory*, Vol. 37, No.4, July 1991, pp.1162-1165.
- Angeniol et al., 1988 Angeniol, B., Croix, V.G, and Le Texier J., "Self-Organizing feature maps and the Travelling Salesman Problem", *Neural Networks* **1**, pp.289-293.
- Balakrishnan, 1993 Balakrishnan, N., "Simple Heuristics for the Vehicle Routing Problem with Soft Time Windows", *Journal of the Operational Research Society*, Vol. 44, No. 3, pp.279-287.
- Bertsimas, 1992 Bertsimas, D.J, "A Vehicle Routing Problem with Stochastic Demand", *Operations Research*, Vol. 40 No. 3, May-June 1992, pp.574-585.

- Bertsimas and Van Ryzin, 1993** Bertsimas, D.J and Van Ryzin,G., "Stochastic and Dynamic Vehicle Routing in the Euclidean Plane with Multiple Capacitated Vehicles", *Operations Research*, Vol. 41 No.1, January - February 1993, pp.60-76.
- Bhide et al., 1993** Bhide, S., John, N. and Kabuka, M.R., "A Real Time solution for the Traveling Salesman Problem using a Boolean Neural Network", *1993 International Joint Conference on Neural Networks*, San Francisco, pp.1096-1103.
- Bobrow et al., 1986** Bobrow, D.G. et al. "CommonLoops Merging Lisp and Object-oriented programming", *Proceedings of OOPSLA 86*, September 1986, pp.17-29.
- Bochereau and Bourguine, 1990** Bochereau, L. and Bourguine, P., "Extraction of Semantic Features and Logical Rules from a Multilayer Neural Network", *1990 International Joint Conference on Neural Networks - IJCNN 90* pp.97-100.
- Boeres and de Carvalho, 1992** Boeres, M.C.S., de Carvalho, L.A.V. and Barbosa, V.C., "A Faster Elastic-Net Alogarithm for the Traveling Salesman Problem", *1992 IEEE International Joint Conference on Neural Networks*, 7-11 June 1992, Baltimore, Maryland, Vol II, pp.215-220.
- Bramel et al., 1992** Bramel, J., Coffman Jr., E.G., Shor, P.W. and Simchi-Levi, D., "Probabilistic Analysis of the Capacitated Vehicle Routing Problem with Unsplit Demands", *Operations Research*, Vol. 40 No. 6, November-December 1992, pp.1095-1106.
- Burdorf, 1993** Burdorf, C., "Language Supported Storage and Reuse of Persistent Neural Network Objects", *IWANN 1993*, pp.376-381.
- Carpenter and Grossberg, 1987a** Carpenter, G. and Grossberg, S. "A Massively Parallel Architecture for a Self Organizing Neural Pattern Recognition Machine", *Computer Vision, Graphics and Image Processing*, **37**, 1987, pp.54-115.
- Carpenter and Grossberg, 1987b** Carpenter, G. and Grossberg, S. "ART2: Self Organization of Stable Category Recognition Codesfor Analog Input Patterns", *Applied Optics*, **26**, 1987, pp.4919-4930.
- Codd, 1970** Codd, E.F., "A Relational Model of Data for Large Shared Databanks", *Communications of the ACM*, **13**, No.6, June 1970.
- Cohen and Grossberg, 1983** Cohen, M.A. and Grossberg, S., "Absolute stability of global pattern formation and parallel memory storage by competitive neural networks", *IEEE Transactions on Systems, Man and Cybernetics*, **13**, pp.815-825.
- Cohen and Hudson, 1990** Cohen, M.E. and Hudson, D.L., "A Medical Decision Aid Based on a Neural Network Model", *Uncertainty in Knowledgebases. 3rd International Conference on Information Processing and Management of Uncertainty in Knowledge-based systems, IPMU 90*, pp.588-597.
- DeGloria et al., 1993** DeGloria, A., Faraboschi, P. and Olivieri, M., "Efficient implementation of the Boltzmann machine algorithm", *IEEE Transactions on Neural Networks*, Vol: 4 Iss: 1, Jan. 1993, pp.159-63.

- Desrochers et al., 1992** Desrochers, M., Desrosiers, J., Solomon, M., "A New Optimization Algorithm for the Vehicle Routing Problem With Time Windows", *Operations Research*, Vol. 40, No.2, March-April 1992, pp.342-354.
- Dreiseitl and Wang, 1993** Dreiseitl, S. and Wang, D., "Automatic Generation of C++ code for Neural Network Simulation", IWANN 1993, pp.358-363.
- Duda and Reboh, 1983** Duda, R.O. and Reboh, "AI and Decision Making: The PROSPECTOR experience", in *Artificial Intelligence Applications for Business.*, Reitman, W. (ed.), Ablex Publishing Co., Norwood, N.J., 1983.
- Dumas et al., 1991** Dumas, Y., Desrosiers, J., and Soumis, F., "The pickup and delivery problem with time windows", *European Journal of Operational Research* 54, 1991, pp.7-22.
- Durbin and Willshaw, 1987** Durbin, R. and Willshaw, D., "An analogue approach to the travelling salesman problem using an elastic net method", *Nature* 326, April 1987, pp.689-691.
- El Ghaziri, 1991** El Ghaziri, H., "Solving Routing Problems by a Self-Organizing Map", *Artificial Neural Networks* *Artificial Neural Networks*, Kohonen, K., Makisara, K., Simula, O., and Kangas, J., (Eds.), 1991, Elsevier Science Publishers B.V. (North Holland).
- Fisher, 1981** Fisher, M.L., "The Lagrangian Method for Solving Integer Programming Problems", *Management Science* 27, 1, 1981, pp.1-18.
- Fisher, Jaikumar and Van Wassenhove, 1986** Fisher, M.L., Jaikumar, R. and Van Wassenhove, L.K., "A Multiplier Adjustment Method for the Generalised Assignment Problem", *Management Science* 32, 9, 1986, pp.1094-1103
- Federgruen and Zipkin, 1984** Federgruen, A. and Zipkin, P., "A Combined Vehicle Routing and Inventory Allocation Problem", *Operations Research*, Vol. 32, No. 5, September-October 1984, pp.1019-1037.
- Fu, 1989** Fu, L.M., "Building Expert Systems on Neural Architecture", *IEEE Artificial Neural Networks*, 16-18 October 1989, London, pp.221-225.
- Fu, 1992** Fu, L.M., "A Neural Network Model for Learning Rule-Based Systems", 1992 *IEEE International Joint Conference on Neural Networks*, 7-11 1992, Baltimore, Maryland, Vol I, pp.343-348.
- Fuentes et al., 1993** Fuentes, I., Aldana, J.F. and Troya J.M. "URANO: an Object-Oriented Artificial Neural Network Simulation Tool", IWANN 1993, pp.364-369.
- Gallant, 1985** Gallant, S.I., "Automatic Generation of Expert Systems from Examples", *Proceedings of the Second International Conference on Artificial Intelligence Applications*, sponsored by IEEE Computer Society, Miami Beach, FL, Dec. 11-13, 1985, pp.313-319.
- Gallant, 1986a** Gallant, S.I., "Three Constructive Algorithms for Network Learning", *Proc. Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, Aug. 15-17, 1986, pp.652-660.

- Gallant, 1986b** Gallant, S.I., "Optimal Linear Discriminants", *Proc. Eighth International Conference on Pattern Recognition*, Paris, France, Oct. 28-31, 1986, pp.849-852.
- Gallant, 1987** Gallant, S.I., "Automated Generation of Expert Systems for Problems Involving Noise and Redundancy", *AAAI Workshop on Uncertainty in Artificial Intelligence*, Seattle, WA, July 10-12, 1987, pp.212-221.
- Gallant, 1988a** Gallant, S.I., "Connectionist Expert Systems", *Communications of the ACM*, 31, no. 2, Feb. 1988, pp.152-169.
- Gallant, 1988b** Gallant, S.I., "Example-Based Knowledge Engineering with Connectionist Expert Systems", *Proc. IEEE MIDCON*, Dallas, Texas, Aug.30th-Sep.1st, 1988, pp.32-37.
- Gallant, 1990a** Gallant, S.I., "A Connectionist Learning Algorithm with Provable Generalization and Scaling Bounds", *Neural Networks* 3, 1990, pp.191-201.
- Gallant, 1990b** Gallant, S.I., "Perceptron-Based Learning Algorithms", *IEEE Transactions on Neural Networks* 1, no. 2, June 1990, pp.179-192.
- Gallant and Hayashi, 1991** Gallant, S.I. and Hayashi, Y.A., "A Neural Network Expert System with Confidence Measurements", in Bouchon-Meunier, B., Yager, R. and Zadeh, L.A. (eds.), *Uncertainty in Knowledge Bases*, Lecture Notes in Computer Science 521, Springer-Verlag, Berlin, 1991, pp.562-567.
- Gehani and Roome, 1988** Gehani, N.H., Roome, W.D. "Concurrent C++ : Concurrent Programming with Class(es)", *Software- Practice and Experience* 18(12), 1988, pp.1157-1177.
- Geman and Geman, 1984** Geman, S. and Geman D., "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6, 1984, pp.721-741.
- Giles and Omlin, 1993** Giles, L.C. and Omlin, C.W., "Rule Refinement with Recurrent Neural Networks", *1993 IEEE International Joint Conference on Neural Networks*, San Francisco, pp.801-806.
- Gindri, Gimitro and Parsatharasay, 1988** Gindri, G., Gimitro, A. and Parsatharasay, K., "Hopfield Model Associative memory with non-zero diagonal terms in the memory matrix", *Applied Optics*, 27, 1988, pp.129-134.
- Goldberg and Robson, 1983** Goldberg, A. and Robson, D., *Smalltalk-80: The Language and its Implementation*, 1983, Addison-Wesley, Reading, MA.
- Golden and Assad, 1986** Golden, B.L. and Assad, A.A., "Perspectives on Vehicle Routing: Exciting New Developments", *Operations Research*, Vol. 34, No.5, September-October 1986, pp.803-810.
- Golden and Wasil, 1987** Golden, B.L. and Wasil E.A., "Computerised Vehicle Routing in the Soft Drink Industry", *Operations Research*, Vol. 35, No.1, January-February 1987, pp.7-17.

- Green and Noakes, 1989** Green, A.D.P. and Noakes, P.D., "Linked Assembly of Neural Networks to Solve the Interconnection Problem", *IEEE ART NN*, 1989, London, pp.216-220.
- Grossberg, 1988** Grossberg, S., "Nonlinear Neural Networks: Principles, Mechanisms and Architectures", *Neural Networks*, Vol. 1, 1988, pp.17-61.
- Healy, 1989** Healey, M.J., "The elements of adaptive neural systems", *SPIE Vol. 1095 Applications of Artificial Intelligence VII*, 1989, pp.830-837
- Heuter, 1991** Heuter, G., "Solution of the Travelling Salesman Problem with an Adaptive Ring", *1991 IEEE International Joint Conference on Neural Networks*, Nov. 18-21st, 1991, Singapore, Vol I, pp.85-92.
- Hillman, 1991** Hillman, D., "Systems on Cascading Neural Nets", *AI Expert*, December 1991, pp.47-53.
- Hinton, 1990a** Hinton,G.E., "Mapping Part-Whole Heirarchies into Connectionist Networks", *Artificial Intelligence* 46, 1990, pp.47-75.
- Hinton, 1990b** Hinton,G.E., "Preface to the Special Issue on Connectionist Symbol Processing", *Artificial Intelligence* 46, 1990, pp.1-4.
- Hinton and Sejnowski, 1986** Hinton, G.E. and Sejnowski, T.J., "Learning and Relearning in Boltzmann Machines", *Parallel Distributed Processing: Exploration in the Microstructure of Cognition, Vol. 1*, Rumelhart, D.E. and McClelland, J.L. (Eds.), 1986, MIT Press, Cambridge, Massachusetts.
- Hinton, Ackley and Sejnowski, 1984** Hinton, G., Ackley, D., and Sejnowski, T., "Boltzmann machines: Constraint satisfaction networks that learn", Carnegie Mellon University, Department of Computer Science Technical Report, CMU-CS-84-119.
- Hopfield, 1982** Hopfield, J. J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proceedings of the National Academy of Sciences* 79, April 1982, pp.2554-2558.
- Hopfield, 1984** Hopfield, J. J., "Neurons with graded response have collective computational properties like those of two-state neurons", *Proceedings of the National Academy of Sciences* 81, May 1984, pp.3088-3092.
- Hopfield and Tank, 1985** Hopfield, J. J. and Tank, D.W., "Neural Computation of Decisions in Optimization Problems", *Biological Cybernetics* 52, 1985, pp.141-152.
- Ichbiah et al., 1979** Ichbiah, J. et al., "Rationale for the design of the programming language ADA", *ACM SIGPLAN NOTICES*, 14, No. 6.
- Jaikumar, 1981** Jaikumar, R., "A Generalised Assignment Heuristic for Vehicle Routing", *Networks* 11, 1981, pp.109-124.
- Kamgar-Parsi and Kamgar-Parsi, 1992a** Kamgar-Parsi, B. and Kamgar-Parsi, B., "Dynamic Stability and Parameter Selection in Neural Optimization", *1992 International Joint Conference on Neural Networks*, Vol IV, pp.566-571.

- Kamgar-Parsi and Kamgar-Parsi, 1992b** Kamgar-Parsi, B. and Kamgar-Parsi, B., "Hopfield Model and Optimization Problems", *Neural Networks for Perception Volume 2: Computation, Learning, and Architectures*, Weschsler, H (ed.), Academic Press Inc., 1992, pp.94-110.
- Kane and Milgram, 1993** Kane, R. and Milgram, M., "Extraction of Semantic Rules from Trained Multilayer Networks", *IEEE International Conference on Neural Networks*, San Francisco, 1993.
- Kanal and Raghavan, 1992** Kanal, L. and Raghavav, S., "HYBRID SYSTEMS - A Key to Intelligent Pattern Recognition", *1992 International Joint Conference on Neural Networks*, 7-11 June 1992, Baltimore, Maryland, Vol IV, pp.177-183.
- Kirkpatrick, Gelatt and Vechi, 1983** Kirkpatrick, S., Gelatt Jr., C.D. and Vecchi, M.P., "Optimization by Simulated Annealing", *Science* **220**, No. 4598, May 1983, pp.671-680.
- Kolen et al., 1987** Kolen, A.W.J., Rinnooy Kan, A.H.G., Treinekens, H.W.J.M., "Vehicle Routing with Time Windows", *Operations Research*, Vol.35, No. 2, March-April 1987, pp.266-273.
- Korst and Aarts, 1989** Korst, J.H.M. and Aarts, E.H.L, "Combinatorial Optimization on a Boltzmann Machine", *Journal of Parallel and Distributed Computing* **6**, 1989, pp.331-357.
- Liang and Jin, 1993** Liang, P. and Jin, K., "Solving Search Problems with Subgoals Using an Artificial Neural Network", *1993 IEEE International Joint Conference on Neural Networks*, San Francisco, pp.81-85.
- Laporte, Nobert and Desrochers, 1985** Laporte, G., Nobert, Y., and Desrochers, M., "Optimal Routing under Capacity and Distance Restrictions" *Operations Research*, Vol.33, No. 5, September-October 1985, pp.1050-1073.
- Laporte, 1992** Laporte, G., "The Vehicle Routing Problem: An overview of exact and approximate algorithms", *European Journal of Operational Research* **59**, 1992, pp. 345-358.
- Khosla and Dillon, 1993** Khosla, R., and Dillon, T., "Task Decomposition and Competing Expert System-Artificial Neural Net Objects for Reliable and Real Time Inference", *IEEE International Conference on Neural Networks*, San Francisco, 1993.
- Li et al., 1992** Li, C-L., Schimi-Levi, D., Desrochers, M., "On the Distance Constrained Vehicle Routing Problem", *Operations Research*, Vol. 40, No. 4, July-August 1992, pp.790-799.
- Lieberman, 1986** Lieberman, H., "Using Prototypical objects to implement shared behaviour in object-oriented systems", *Proceedings of the OOPSLA-86*, Portland, Oregon, 1986.
- Lin, 1965** Lin, S., "Computer Solutions of the Travelling Salesman Problem", *Bell System Technical Journal* **44**, pp.2245-2269.
- Lin and Kernighan, 1973** Lin, S., and Kernighan, B.W., "An Effective Heuristic Algorithm for the Travelling Salesman Problem", *Operations Research* **21**, pp.498-516.

- Liow and Vidal, 1991** Liow, R. and Vidal J.J., "A Dual Network Expert System", *1991 IEEE International Joint Conference on Neural Networks*, Nov. 18-21st, 1991, Singapore, pp.1670-1674.
- Liskov and Zilles, 1974** Liskov, B. and Zilles, S.N., "Programming with Abstract Data Types", *ACM SIGPLAN Notices*, **9**, pp.50-59.
- Little, 1974** Little, W., "The existence of persistent states in the brain", *Mathematical Biosciences*, **19**, 1974, pp.101-120.
- Little and Shaw, 1978** Little, W. and Shaw, G., "Analytical study of the memory storage capacity of a neural network", *Mathematical Biosciences*, **39**, 1978, pp.281-290.
- Looney, 1993** Looney, C.G., "Neural Networks as Expert Systems", *Expert Systems with Applications*, 1993, Vol. 6, pp.129-136.
- Madrid et al., 1992** Madrid, R., Williams, B., and Holland, J., "Artificial Intelligence for Ordnance Disposal System (AI-EOD)", *International Joint Conference on Neural Networks, 1992*, Vol I (0-7803-0559-0/92), pp.378-383.
- Martello and Toth, 1981** Martello, S. and Toth, P., "An Algorithm for the Generalized Assignment Problem", in Brans, J.P (Ed.), *Operational Research*, 1981, pp.589-603
- McEliece et al., 1987** McEliece, R., Posner, E., Rodemich, E. and Venkatesh, S., "The capacity of the Hopfield associative memory", *IEEE Transactions on Information Theory*, IT-33, 1987, pp.461-482
- McCulloch and Pitts, 1943** McCulloch, W., and Pitts, W., "A logical calculus of the ideas immanent in nervous activity", *Bulletin of Mathematical Biophysics*, **5**, pp.115-133.
- Meng, 1993** Meng, M., "A Neural Production System and its Application in Vision-Guided Mobile Robot Navigation", *1993 IEEE International Joint Conference on Neural Networks*, San Francisco, pp.807-812.
- Metropolis et al., 1953** Metropolis, N, Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., and Teller, E., "Equation of State Calculations by Fast Computing Machines", *Journal of Chemical Physics* **21**, 1953, pp.1087-1092.
- Minsky, 1981** Minsky, M., "A Framework for Representing Knowledge", In *Mind Design*, edited by Haugeland, J., MIT Press, Cambridge, Massachusetts, 1981.
- Moon, 1986** Moon, D.A., "Object-Oriented Programming with Flavors", *Proceedings of OOPSLA 86*, Portland, Oregon.
- Myllymaki et al., 1990** Myllymaki, P., Tirri, P., Floreen and Orponen, P., "Compiling High-Level specifications into Neural Networks", *AI Communications* Vol: 3 Iss: 4 p. 172-83, Date: Dec. 1990.
- Nang et al., 1991** Nang, J.H., Oh, D.H., Yoon, H. and Maeng, S.R., "A Parallel Boltzmann machine on Distributed -Memory Multiprocessors", *1991 IEEE International Joint Conference on Neural Networks*, Nov. 18-21st, 1991, Singapore, pp.608-613.

- Nonaka and Kobayashi, 1992** Nonaka, F. and Kobayashi, Y., "Sub Optimal Solution Screening in Optimization by Neural Networks", *1992 International Joint Conference on Neural Networks*, Vol. IV, pp.606-611.
- Nottola, Condamin and Naim, 1991** Nottola, C., Condamin, L. and Naim, P., "On the use of Hard Neural Networks for Symbolic Learning Application to Company Evaluation", *International Joint Conference on Neural Networks*, Singapore, 1991.
- Oka, 1992** Oka, N., "Hybrid Cognitive Model of Conscious Level Processing and Unconscious Level Processing", *1991 IEEE International Joint Conference on Neural Networks*, Nov. 18-21st, 1991, Singapore, pp.485-490.
- Omlin, Giles and Miller, 1992** Omlin, C.W., Giles, C.L. and Miller, C.B., "Heuristics for the Extraction of Rules from Discrete-Time Recurrent Neural Networks", *International Joint Conference on Neural Networks IJCNN-1992*, 7-11 June 1992, Baltimore, Maryland Vol. I, pp.33-37.
- Pensini et al., 1989** Pensini, M.P., Mauri, G. and Gardin, F., "Flowshop and TSP" *Parallelism, Learning, Evolution. Workshop on Evolutionary Models and Strategies, and Workshop on Parallel Processing: Logic, Organization and Technology - WOPPLOT 89*, p. 157-82, Becker, J.D., Eisele, I. and Mundemann, F.W. (Eds.), Springer-Verlag, Berlin, Germany, 1991.
- Pham and Degoulet, 1988** Pham, K.M. and Degoulet, P., "MOSAIC: A Macro-connectionist Organization System for Artificial Intelligence Computation", *IEEE International Conference on Neural Networks* Vol. II, 1988, pp.533-540.
- Potvin and Rousseau, 1993** Potvin, J-Y. and Rousseau, J-M., "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows", *European Journal of Operational Research* 66, 1993, pp.331-340.
- Potovin and Shen, 1991** Potovin, J-Y. and Shen, Y., "A Neural Network Approach to the Vehicle Dispatching Problem", *1991 IEEE International Joint Conference on Neural Networks*, Nov. 18-21st, 1991, Singapore, pp.1230-1235.
- Quillian, 1968** Quillian, M.R., "Semantic memory", 1968, in *Semantic Information Processing*, Minsky, M. (Ed.), MIT Press, Cambridge, MA.
- Ran and Karjalainen, 1991** Ran, A., and Karjalainen, M., "High Level Object Oriented Programming Environment for Efficient Neural Computations", *Artificial Neural Networks - Proceedings of the 1991 International Conference ICANN-91*, Vol. 1, pp.596-602.
- Ray, 1991** Ray, A.K., "Equipment Fault Diagnosis - A Neural Network Approach", *Computers in Industry* 16, 1991, pp.169-177.
- Rumelhart, Hinton and Williams, 1986** Rumelhart, D.E., Hinton, G.E. and Williams, R.J., "Learning Internal Representations by Error Propagation", *Parallel Distributed Processing : Explorations in the Microstructure of Cognition*, Vol. 1, Rumelhart, D.E. and McClelland, J.L. (Eds.), Bradford Books, Cambridge, MA, pp.318-362.

- Saito and Nakano, 1988** Saito, K., and Nakano, R., "Medical Diagnostic Expert System Based on the PDP Model", *IEEE International Conference on Neural Networks*, San Diego, CA, 24-27 July 1988.
- Saito and Nakano, 1990** Saito, K., and Nakano, R., "Automatic Extraction of Classification Rules", *International Neural Networks Conference*, July 9-13, 1990, Paris, France.
- Samad, 1988** Samad, T., "Towards Connectionist Rule-Based Systems", *IEEE International Conference on Neural Networks*, San Diego, 24-27 July 1988, Vol II, pp.525-532.
- Savelsbergh, 1990** Savelsbergh, M.W.P., "An efficient implementation of local search algorithms for constrained routing problems", *European Journal of Operational Research* 47, 1990, pp.75-85.
- Schreinmakers and Touretzky, 1988** Schreinmakers, J.F. and Touretzky, D.S., "Interfacing a Neural Network with a Rule-Based Reasoner for Diagnosing Mastitis", Vol.2, pp.487-490. 1988
- Shinozawa et al., 1991** Shinozawa, K., Uchiyama, T., and Shimohara, K., "An Approach for Solving Dynamic TSPs using Neural Networks", *1991 IEEE International Joint Conference on Neural Networks*, Nov. 18-21st, 1991, Singapore, Vol 3, pp.2450-2454.
- Smieja, 1991** Smieja, F.J., "Multiple Network Systems (MINOS) modules: Task Division and Module Discrimination", *AISB91 Proceedings of the Eighth Conference for the Study of Artificial Intelligence and Behaviour*, 16-19 April 1991, Leeds, U.K. pp.13-25.
- Solomon, 1987** Solomon, M.M., "Algorithms for Vehicle Routing and Scheduling Problems with Time Window Constraints", *Operations Research*, Vol.35, No. 2, March-April 1987, pp.254-265.
- Soumis et al., 1991** Soumis, F., Sauve, M., and Le Beau, L., "The Simultaneous Origin-Destination Assignment and Vehicle Routing Problem", *Transportation Science*, Vol. 25, No.3, August 1991, pp.188-200.
- Stein, 1987** Stein, L.A., "Delegation is inheritance", 1987, *Proceedings of OOPSLA-87*.
- Stein, Lieberman and Ungar, 1989** Stein, L.A., Lieberman, H. and Ungar, D., "A Shared View of Sharing: The Treaty of Orlando", *Object-oriented Concepts, Databases and Applications*, Kim, W. and Lochovsky F.H. (eds.), ACM Press, New York, 1989.
- Stottler and Henke, 1992** Stottler R.H., and Henke, A.L., "Automatic Translation from an Expert System to a Neural Network Representation" *1992 International Joint Conference on Neural Networks*, Vol I, 1992, pp.13-20.
- Suddarth and Holden, 1991** Suddarth, S.C. and Holden, A.D., "Symbolic-neural systems and the use of hints for developing complex systems", *Int. J. Man-Machine Studies*, 35, 1991, pp.231-311.
- Sun, 1991** Sun, R., "Neural Network Models for Rule-Based Reasoning", *International Joint Conference on Neural Networks IJCNN-91*, 18-21 November 1991, pp.503-508.

- Szu, 1991** Szu, H., "Fast TSP Algorithm based on binary neuron output and analog neuron input using the Zero-Diagonal Interconnect Matrix and Necessary and Sufficient Constraints of the Permutation Matrix", *1991 IEEE International Joint Conference on Neural Networks*, Nov. 18-21st, Singapore, Vol II, pp.606-611.
- Tesar et al., 1989** Tesar, B.B., Kapenga, J. and Trenary, R., "A Boltzmann machine solution of the traveling salesperson problem: a study for parallel implementation", *TENCON '89. Fourth IEEE Region 10 International Conference. 'Information Technologies for the 90's'*, pp.142-144, Publisher: IEEE, New York, NY, USA, 1989.
- Ticknor and Barrett, 1987** Ticknor, A.J. and Barrett, H.H., "Optical implementations in Boltzmann machines", *Optical Engineering* 26, 1987, 16.
- Touretzky, 1990** Touretzky, D.S., BoltzCONS: "Dynamic Symbol Structures in a Connectionist Network", *Artificial Intelligence* 46, 1990, pp.5-46.
- Touretzky and Hinton, 1988** Touretzky, D.S. and Hinton, G.E., "Symbols Among the Neurons: Details of a Connectionist Inference Architecture", at Computer-Science Department, Carnegie-Mellon University, Pittsburgh, 1988.
- Tsai et al., 1990** Tsai, W.K., Parlos, A., and Fernandez, B., "A Novel Associative Memory for High Level Control Functions", *Proceedings of the 29th Conference on Decision and Control, Honolulu, Hawaii, December 1990*, pp.2374-2379.
- Wang and el Ayeb, 1992** Wang, S. and el Ayeb, B., "Diagnosis: Hypothetical Reasoning With a Competition - Based Neural Architecture" *1992 IEEE International Joint Conference on Neural Networks*, 7-11 June 1992, Baltimore, Maryland, Vol I, pp.7-12.
- Widrow, 1962** Widrow, B., "Generalization and information storage in networks of adaline `neurons'", in *Self-Organizing Systems*, Yovits, M., Jacobi, G. and Goldstein, G. (Eds.), 1962, pp.435-461, Spartan Books, Washington.
- Wilson and Pawley, 1988** Wilson, G.V. and Pawley, G.S., "On the stability of the Travelling Salesman Problem of Hopfield and Tank", *Biological Cybernetics*, 58, 1988, pp.63-70.
- Yonezawa and Tokoro, 1987** Yonezawa, A., and Tokoro, M., (eds.), *Object-Oriented Concurrent Programming*, MIT Press, Cambridge, Massachusetts, 1987.
- Yoon et al., 1990** Yoon, Y., Brobst, R.W., Bergstresser, P.R., and Peterson, L.L., "Automatic Generation of a Knowledge-Base for a Dermatology Expert System", *Third Annual IEEE Symposium on Computer-Based Medical Systems, 1990*, pp.306-312.
- Yu and Lee, 1992** Yu C., S. and Lee W., D., "Parallel Mean Field Annealing Neural Network for Solving Traveling Salesman Problem", *1992 International Joint Conference on Neural Networks*, Vol IV, pp.532-536.