1

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

PRECISION™ RESOLUTION TARGETS

Neuromorphic Distributed General Problem Solvers

by

Andrzej Bieszczad

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

Doctor of Philosophy

Department of Systems and Computer Engineering
Ottawa-Carleton Institute for Electrical Engineering

Carleton University

Ottawa, Ontario

October 1, 1996

© Copyright

1996, Andrzej Bieszczad

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file *Votre référence*

Our file *Notre référence*

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

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

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

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

ISBN 0-612-19356-X

Canada

Name ANDREJ BIESZCZAK

Dissertation Abstracts International and Masters Abstracts International are arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation or thesis. Enter the corresponding four-digit code in the spaces provided.

COMPUTER SCIENCE

SUBJECT TERM

0000

SUBJECT CODE

UMI

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0723
Bilingual and Multicultural 0282
Business 0688
Community College 0725
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0727
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0728
Industrial 0521
Language and Literature 0729
Mathematics 0780
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religion 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
Ancient 0289
Linguistics 0290
Modern 0291
Literature 0401
Classical 0294
Comparative 0295
Medieval 0297
Modern 0298
African 0316
American 0591
Asian 0305
Canadian (English) 0352
Canadian (French) 0355
English 0593
Germanic 0311
Latin American 0312
Middle Eastern 0315
Romance 0313
Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0318
Biblical Studies 0321
Clergy 0319
History of 0320
Philosophy of 0322
Theology 0469

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Cultural 0326
Physical 0327
Business Administration 0310
General 0310
Accounting 0272
Banking 0770
Management 0454
Marketing 0338
Canadian Studies 0385
Economics 0301
General 0301
Agricultural 0303
Commerce-Business 0305
Finance 0308
History 0309
Labor 0510
Theory 0511
Folklore 0358
Geography 0344
Gerontology 0351
History 0578
General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General 0615
International Law and 0616
Relations 0617
Public Administration 0614
Recreation 0814
Social Work 0452
Sociology 0426
General 0426
Criminology and Penology 0427
Demography 0938
Ethnic and Racial Studies 0631
Individual and Family 0428
Studies 0428
Industrial and Labor 0629
Relations 0630
Public and Social Welfare 0630
Social Structure and 0700
Development 0344
Theory and Methods 0709
Transportation 0999
Urban and Regional Planning 0453
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy 0475
Animal Culture and 0476
Nutrition 0475
Animal Pathology 0476
Food Science and 0359
Technology 0478
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Plant Pathology 0777
Range Management 0746
Wood Technology 0746

Biology

General 0306
Anatomy 0287
Biostatistics 0308
Botany 0309
Cell 0379
Ecology 0329
Entomology 0353
Genetics 0369
Limnology 0793
Microbiology 0410
Molecular 0307
Neuroscience 0317
Oceanography 0416
Physiology 0403
Radiation 0821
Veterinary Science 0778
Zoology 0472
Biophysics 0784
General 0784
Medical 0760

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geology 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleontology 0345
Paleocology 0426
Paleontology 0418
Paleontology 0985
Paleontology 0427
Physical Geography 0388
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0748
Health Sciences 0564
General 0564
Anesthesiology 0300
Chemotherapy 0992
Dentistry 0567
Education 0350
Hospital Management 0769
Human Development 0758
Immunology 0982
Medicine and Surgery 0564
Mental Health 0347
Nursing 0569
Nutrition 0570
Obstetrics and Gynecology 0380
Occupational Health and 0354
Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0440
Toxicology 0383
Economic 0384

PHYSICAL SCIENCES

Pure Sciences 0485
Chemistry 0485
General 0485
Agricultural 0749
Analytical 0486
Biochemistry 0487
Inorganic 0488
Nuclear 0738
Organic 0490
Pharmaceutical 0491
Physical 0494
Polymer 0495
Radiation 0754
Mathematics 0405
Physics 0405
General 0405
Acoustics 0984
Astronomy and 0406
Astrophysics 0406
Atmospheric Science 0408
Atomic 0748
Electronics and Electricity 0407
Elementary Particles and 0798
High Energy 0799
Fluid and Plasma 0799
Molecular 0409
Nuclear 0610
Optics 0782
Radiation 0754
Solid State 0611
Statistics 0463
Applied Sciences 0344
Applied Mechanics 0344
Computer Science 0984

Engineering 0537
General 0537
Aerospace 0538
Agricultural 0539
Automotive 0540
Biomedical 0541
Chemical 0542
Civil 0543
Electronics and Electrical 0544
Heat and Thermodynamics 0348
Hydraulic 0545
Industrial 0546
Marine 0547
Materials Science 0794
Mechanical 0548
Metallurgy 0743
Mining 0551
Nuclear 0552
Packaging 0549
Petroleum 0745
Sanitary and Municipal 0554
System Science 0790
Geotechnology 0428
Operations Research 0796
Plastic Technology 0795
Tissue Technology 0994

PSYCHOLOGY

General 0421
Behavioral 0384
Clinical 0422
Developmental 0420
Experimental 0423
Industrial 0424
Personality 0425
Physiological 0989
Psychiatry 0349
Psychometrics 0432
Social 0451

Andrzej Bieszczad
Systems and Computer Engineering, Carleton University
Room 4434 Mackenzie Building
1125 Colonel By Drive, Ottawa, Ontario K1S 5B6

To whomever it may concern,

I acknowledge that all blank pages in my Ph.D. thesis *Neuromorphic Distributed General Problem Solvers* are intentional and were assigned page numbers. The pages were inserted automatically by FrameMaker so in case of double-sided printing all chapters start on right side pages

Andrzej Bieszczad

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

NEUROMORPHIC DISTRIBUTED GENERAL PROBLEM SOLVERS

submitted by **Andrzej Bieszczad, M. Sc., M. C. S.**

in partial fulfillment of the requirements for
the degree of Doctor of Philosophy

Chair

Department

Prof. Bernard Pagurek (Thesis Supervisor)

External Examiner (Ph.D.)

CARLETON UNIVERSITY

Date* _____

*to identify date of revisions

Abstract

At this moment of time, it is justifiable to say that the great promise of symbolic Artificial Intelligence, including problem solving and planning systems, have not satisfactorily materialized. There are some areas, for example Expert Systems, that have made it through to the mainstream technology. Most beautiful dreams are still unfulfilled. That disillusionment to some extent drives and justifies the research reported in this thesis and similar efforts on applying ideas that do not have their roots in formal logic.

I propose a new approach to problem solving that employs neuromorphic methods for agents that attack a problem in a distributed and cooperative manner.

I attempt to apply these agents to the task of rearranging three different blocks in a variant of a blocks world. Blocks worlds are popular in the Artificial Intelligence community as testbeds for problem solving systems. It is a consequence of their relative simplicity, but nonetheless sufficient complexity to present ideas and verify problem solving concepts and techniques. Three types of agents are used: sensing, manipulating and solving. The sensing agents collect the state of the blocks world and communicate it to the solving agent. The solving agent employs a biologically inspired device that I call the Neurosolver. The agent uses its Neurosolver-based brain to record the detected trajectories in the state space of the blocks world. When given a rearrangement task, it uses the learned traces to perform searches and construct plans to control the movements of the blocks. The resolution plan is distributed amongst the population of manipulating agents that actually rearrange the blocks. All of the agents are mobile and communicate while visiting the same blocks world location.

An extended system that I call the Neuromorphic General Problem Solver (NGPS) may employ multiple neurosolvers. The knowledge may be distributed between the solvers in a number of ways, yet they are able to solve the posed problems cooperatively.

Acknowledgments

I would like to express gratitude to Prof. Bernard Pagurek for his support and advice throughout the years during which I conducted the research. Thank you for letting me choose my own ways.

Without the financial support from the National Science and Engineering Research Council of Canada (NSERC) and the Telecommunications Research Institute of Ontario (TRIO) I would not be able to give up my industrial career and return to school.

In the course of my studies, I used the resources of NORTEL-BNR very extensively and I am grateful for them letting me do so during my leave of absence.

I am also thankful to Prof. Wilf Lalonde who many years ago gave me a book that inspired my research.

Special thanks are due to my friend Tony who not only argued the merits of my research but also helped me to polish the final version of the thesis.

Most of all, I would like to thank for the support of the best family in the world: my son Maciek, my daughter Kasia and my wife Anna. I love you all.

Table of contents

Chapter I: Foreword	1
1.1 Quo vadis?	1
1.2 Structure of the thesis	3
Chapter II: Introduction	6
2.1 The blocks world	7
2.2 The Neurosolver	9
2.3 The agents	11
2.4 Thesis contributions	13
Chapter III: Introduction to problem solving and planning	17
3.1 Problem representation	17
3.1.1 State spaces	18
3.1.1.1 Example	19
3.1.2 Other representations	20
3.1.2.1 Problem reduction	21
3.1.2.2 Game trees	21
3.1.3 Search	21
3.1.3.1 Blind search	22
3.1.3.1.1 Breath-first search	22
3.1.3.1.2 Depth-first search	22
3.1.3.1.3 Bidirectional search	23
3.1.3.2 Heuristic search	23
3.1.3.2.1 Greedy search	23
3.1.3.2.2 A* search	24
3.1.3.3 Search with iterative improvement	24
3.1.3.3.1 Hill-climbing	24

3.1.3.3.2 Simulated annealing	25
3.1.4 GPS - General Problem Solver	28
3.2 Planning	27
3.2.1 Difference between planning and problem solving	27
3.2.2 STRIPS	29
3.2.2.1 Model	29
3.2.2.2 Operation	29

Chapter IV: Introduction to

Neural Networks	31
4.1 Introduction	31
4.1.1 A Few Words About The Nature	32
4.1.2 Neural Networks - An attempt to mimic Nature	33
4.2 Perceptron	36
4.3 Backpropagation network	40
4.4 Time-delay network	43
4.5 Recurrent network	46
4.6 Hopfield network	47
4.7 Kohonen network	49
4.7.1 On-center/off-surround architecture	49
4.7.2 Activity bubbles	50
4.7.3 Topology preserving	50
4.7.4 Kohonen Algorithm	52
4.7.5 Learning Vector Quantization	53
4.8 Weightless networks	54
4.9 Activity spreading networks	56

Chapter V: Introduction to

Markov Models	58
Chapter VI: Neurosolver. A neural network based on a cortical column	63
6.1 Introduction	63
6.2 General Architecture	65
6.2.1 The Node	70
6.2.2 Search network to the level exceeding the firing threshold	72
6.2.3 Execution network	73
6.3 Learning state transitions	74
6.4 Solving problems	77
6.5 Predicting and decision support	87
6.6 Implementation	90
6.6.1 Object-Oriented Design	90
6.6.2 Coding	91
6.6.3 Application Program Interface (API)	92
6.7 Experiments	96
6.7.1 Simple example 1	96
6.7.2 Simple example 2	97
6.7.3 More complex example	99
6.8 Dealing with noise	113
6.8.1 The Neurosolver solves a problem that STRIPS could not	117
6.8.1.1 STRIPS approach	117
6.8.1.2 Neurosolver approach	119
Chapter VII: Neurosolver in the blocks world	123
7.1 The blocks world	123
7.2 Problem state space	124
7.3 Interface between the Neurosolver and the blocks world	128
7.3.1 Input representation	128

7.3.2	Output representation	129
7.4	Experiments	130
7.5	Scaling to larger problems	130
7.5.1	Adding a state transition	137
7.5.2	Adding a trajectory - storing a plan	138
7.5.3	Adding a new state	138
7.5.4	Increasing the size of the state space	138
Chapter VIII:	The agents	141
8.1	Agents in the blocks world	142
8.1.1	Sensing agents	143
8.1.2	Solving agents	144
8.1.3	Manipulating agents	146
8.2	Agent interactions	149
8.2.1	Communication between sensing and solving agents	149
8.2.2	Communication between solving and manipulating agents	151
8.2.3	Interactions between the manipulating agents	152
8.3	Implementation	152
Chapter IX:	Multiple solving agents	155
9.1	Introduction	155
9.2	Uniform knowledge	157
9.3	Distributed knowledge	158
9.3.1	Partitioning state transition spaces	158
9.3.2	Partitioning state spaces	161
9.3.2.1	Hardwiring	162
9.3.2.2	Constraining the environment	163
9.3.2.3	Constraining the sensors	163
9.3.2.4	State elimination through interactions between the solving agents	163

9.3.2.5	Learning	164
9.3.3	Scaling to larger problems	167
Chapter X.	Directions for future research	169
10.1	The Neurosolver	169
10.1.1	Expanding the architecture	169
10.2	Implementation	170
10.2.1	Simulation	170
10.2.2	Analog implementation	171
10.3	Agents	173
10.3.1	Extensions to sensing agents	173
10.3.2	Extensions to manipulating agents	175
10.3.3	Enhancements to solving agents	176
10.3.3.1	Multiple solving agents	176
10.3.3.2	A divide et impera approach to hard problems	177
10.3.3.3	Problem presentation	177
10.3.4	Extensions concerning all agent types	179
10.3.4.1	Agent reincarnation	179
10.3.4.2	Sharing skills	180
10.3.5	A word on agent mobility	181
10.4	Further challenges	182
10.5	Network Management applications	183
10.5.1	Problem recovery	184
10.5.2	Fault detection	186
10.5.2.1	Fault detection by cascade associations	187
10.5.2.2	Model-based fault detection	188
Chapter XI	Conclusions	193

APPENDIX A: Biological roots of the Neurosolver	
An insight into the human brain	197
A 1 Evolution of the brain	197
A 2 Anatomy of the brain	199
A 2 1 Anatomy of the forebrain	200
A 2 2 Cerebral cortex	201
A 2 3 Thalamus	202
A 2 4 Basal ganglia	202
A 2 5 Limbic system	203
A 2 6 Olfactory bulbs	204
A 3 Anatomy of the cerebral cortex	204
A 3 1 A single neuron	204
A 3 2 Lateral organization	207
A.3.2.1 The cortical hemispheres	207
A 3 2 2 The cortical lobes	208
A.3.3 Laminar organization - The cortical layers	209
A.3.4 Topography of the cerebral cortex	210
A 3 4 1 Cytoarchitectonic maps	211
A 3 4 2 Projection maps	212
A 3 4 3 Functional maps	213
A 4 General function of the cortex	215
A 4 1 Control of voluntary movements	215
A 4 2 Pattern perception	217
A 4 3 Cognitive mapping	216
A 5 Functional organization of the cortex - Lurta's model	217
APPENDIX B: Biological roots of the Neurosolver	
The cortical column.....	221

B 1	Types of neurons in the cortex	221
B 1 1	Pyramidal neurons	222
B 1 2	Spiny-stellate neurons	223
B 1 3	Chandelier neurons	224
B 1 4	Basket neurons	224
B 1 5	Double bouquet neurons	225
B 1 6	Bipolar neurons	225
B 1 7	Smooth stellate neurons	226
B 2	The cortical column	226
B 3	Connectivity of the cortical column	229
B 4	The cortical column as a processing element	231
B 5	Properties of a network of cortical columns	232
References		237
Index		245

Chapter I: Foreword

1.1 Quo vadis?

In the course of my research, I have investigated neuromorphic distributed general problem solvers. *Neuromorphic*, because the research is inspired by biological neural networks, *distributed*, because a suite of agents cooperate in the resolution of a problem. In pursuing the idea of a general problem solver, I share the same dream with researchers involved in systems like a system for general intelligence SOAR (Rosenbloom, [46], Laird, [31], Cho, [17]) and its predecessors General Problem Solver (GPS) (Newell, [40]) and a robot problem solver called STRIPS (Nilsson, [41]). That goal is a domain independent, modular, adaptive problem solver that achieves a given task – solves a problem – by performing a parallel, possibly hierarchical, breadth-first search in the space of all possible states of the problem¹. My approach differs from the classical body of work, however. In contrast to symbolic approaches of the past, I use sub-symbolic, neurally-derived approach as the means for building the reasoning engines. Additionally, I investigate distributed, agent-based techniques to problem solving in contrast to centralized approach which seem prevalent in previous work. The research on agent-based technologies has been gaining popularity in recent years, but to the best of my knowledge there is little previous work in the area in which I am most interested.

¹ This space is often referred to as a problem space, state space or problem universe.

This document puts forward the following thesis *not only is a neurally inspired distributed general problem solver viable, but in many ways its behavior is, or at least presents the prospects for being, better than traditional systems of a similar kind*

To prove my thesis, a variation of the blocks world problem is used. In the past, it has proven to be a reliable workbench to explore and test the capabilities and limitations of problem solvers. The agents that are introduced into the blocks world learn the world by exploring it, learn a set of possible operations by observing the activity of the operator or another external agent interacting with the blocks, and, later, use the acquired knowledge and skills to cooperatively reconfigure the world to a desired state in a cooperative fashion.

The blocks world, as a well researched and popular problem in Artificial Intelligence (AI) circles, is a convenient means for exercising and communicating the ideas presented in my thesis. Nevertheless, its relative simplicity should not disguise the wide applicability of the presented problem solving techniques. For example, the latest findings in the management of telecommunication networks using high speed Asynchronous Transfer Mode (ATM) devices (see Wilson [67]) clearly state that it will be very difficult or impossible to manage such networks with centralized, traditional systems. The need for distributed, autonomous and adaptive control and maintenance agents becomes apparent (for example, see Goldszmidt [24], Luderer [32], Shonwalder [50] or the July 1996 issue of *The Simple Times* [57]). Many big telecommunication companies have already initiated work on agent-based network management systems. Several special purpose languages have been designed and implemented, that allow encoding of agents in such a way that they can be activated on any node in the network crossing multi-vendor boundaries. It is my hope that the research proposed in this document will contribute to the design and understanding of such, and similar, sixth-generation systems.

1.2 Structure of the thesis

I start with an overview of the research without excessive details in Chapter II *Introduction*. This summary was presented at the First International Conference on Computational Intelligence and Neurosciences (one of the three conferences in the framework of Joint Conference on Information Sciences JCIS'95) under a title *Neurosolver-Based Agents for Solving Blocks World Problems* (Bieszczađ, [7])

In Chapter III *Introduction to problem solving and planning*, I introduce the concepts of state spaces, search, problem solving and planning. These abstractions are fundamental for many areas of AI. Their understanding will help the reader to comprehend the material in the subsequent parts of the thesis, because I refer to these ideas and concepts in many places.

There are certain neural network techniques that were tried in the course of the work to interface the Neurosolver to the blocks world. I refer to them in the main thread of the thesis without further explanation. These techniques are computationally intensive. Those that were needed in the context of the Neurosolver have been tested separately from other experiments on a Stuttgart Neural Network Simulator (SNNS version 4.0; [53]). The first technique refers to the Kohonen algorithm that may be used to create problem state spaces by explorations automatically. Similarly, the time consuming backpropagation algorithm can be used to associate configurations of the blocks world to appropriate states. This technique is an example of supervised learning, i.e., with a teacher presenting pairs of input and desired output patterns. That is in contrast to unsupervised learning exemplified by the Kohonen algorithm, in which only the input patterns are presented. Chapter IV: *Introduction to Neural Networks* contains, amongst other architectures, the descriptions of the backpropagation and Kohonen networks. Other networks that are described in this chapter include recurrent architectures, weightless networks and activity spreading schema that are focal points for comparing other research to the Neurosolver.

Learning in the Neurosolver has similar characteristics to the learning in Markov Models. There are references to this similarity throughout this document, so readers not acquainted with Markov Models may want to read Chapter V, *Introduction to Markov Models*.

Chapter VI: *Neurosolver: A neural network based on a cortical column* describes the fundamental models derived from Neuroscience that have been used to design the Neurosolver. It contains details of the architecture of artificial cortical columns and their network. The functionality of this network is fundamental to the rest of the research. Understanding this chapter is critical in comprehending the remainder of the thesis. I reported this part of the research in my paper *Neurosolver: A Step Toward a Neuromorphic General Problem Solver* (Bieszczad, [9]) presented at the First IEEE World Congress on Computational Intelligence.

Chapter VII: *Neurosolver in the blocks world* describes the application of the Neurosolver in solving rearrangement problems in a version of the blocks world. A number of experimental results are presented here that illustrate the capabilities and current limitations of the Neurosolver. This material was published in two of my papers: *Neurosolver in a blocks world* (Bieszczad, [6]) presented at the International Conference on Neural Networks in 1996, and the other entitled *Solving blocks world problems with the Neurosolver* that was presented at the World Congress on Neural Networks WCNN'96 (Bieszczad, [5]). A discussion on the scaling of the Neurosolver to larger problems is included at the end of this chapter.

Chapter VIII: *The agents* examines a problem solving system with sensory, motor and control components that are based on agents. The system is capable of solving the rearrangement problems in the blocks world. This part of my research was reported in the extended version of my paper *Neurosolver-based agents for solving blocks world problems* (Bieszczad, [7]).

In Chapter IX, *Multiple solving agents*, I investigate a problem solving system with multiple neurosolvers. Of particular interest is the issue of dealing with a distributed knowledge base. I discuss the behavior and capabilities of the system in the context of simple problems as well as the prob-

lems of rearranging blocks in the blocks world

The research area in which I have conducted my studies is vast, so it would be difficult to cover everything in one Ph D thesis. In Chapter X *Directions for future research*, I present an outlook on several possible avenues of research for the future. Several thoughts on opportunities for practical application of the outcome of my research are collected in section 10.5 *Network Management applications*. Due to incredibly fast advances in technology, Telecommunications, and Network Management in particular, are exciting, dynamic areas for an application engineer. It appears that many traditional solutions are not applicable or yield poor results in this rapidly changing field. In my opinion, there is considerable potential for new ideas and approaches to prove their usefulness and become mainstream or even core technologies.

The conclusions from my research are collected in a short Chapter XI *Conclusions*.

APPENDIX A *Biological roots of the Neurosolver: An insight into the human brain* and

APPENDIX B *Biological roots of the Neurosolver: The cortical column* include a considerable amount of information on the biology of the human brain in order to give the reader some understanding of the origins of my research: namely Neuroscience. I think that the material presented in these appendices is very interesting. It may help anyone continuing this research to examine the details that may have escaped my scrutiny. Although not critical, these sections may help in understanding the main thread of this thesis.

In the reference section, I have included a collection of the publications to which I refer throughout this document.

Chapter II: Introduction

The goal of the research that led to the original introduction of the Neurosolver, as reported in (Bieszczad, [10], [8]) and [9]), was to design a neuromorphic device that would be able to tackle problems in the framework of the state space paradigm (Newell, [40]) The research was inspired by Burnod's monograph on the workings of the human brain (Burnod, [14]) The Neurosolver has been considerably revamped in the course of applying it to plan re-arrangement tasks in a blocks world (Bieszczad, [6])

The class of systems that employ state spaces to present and solve problems has its roots in the early stages of AI research derived from the studies of human information processing (e.g. General Problem Solver (Newell, [40])) This pioneering work led to very interesting problem solving (e.g., SOAR (Laird, [31])) and planning systems (e.g., STRIPS (Nilsson, [41])) NeuroSOAR (Cho, [17]) is one of the few attempts to explore similar ideas from the neural network perspective, although its main focus is on how SOAR's production system and decision procedure can be implemented in neural networks. There were other attempts to use neural networks in the context of robot path planning (see (Connolly, [19]) for a short review) Higher level tasks were addressed by rule-based connectionist models (e.g., Touretzky, [58]) Temporal difference models (Sutton, [52]), with a very successful application in playing backgammon (Tesauro [56]) are interesting in the context of state spaces, because they can be derived from Markov models (Barnard, [4]) Interesting research on merging logic and connectionist systems is reported in (Sun, [51]) Work on processing temporal sequences (a general taxonomy can be found in (Mozer, [37])) is concerned mostly with recognition of temporal patterns, but such systems can also be used to gener-

ate sequences (e.g., predict the future). A recent report on aspects of planning with weightless neural networks (Marsic-Fogel, [39]) presents an approach that has its roots in switching theory rather than in neurobiology.

The term *agent* is currently used widely across many fields (Wooldridge, [65], [66]). Agents may be classified using many criteria such as individual capabilities, social behavior, degree of cooperation, scale, architecture, application, etc. (Chaib-draa, [16]). In this research, I use mobile agents that provide aspects of distributed artificial intelligence (DAI) by cooperative attempts to resolve blocks world rearrangement problems. Without communicating with peers, a single agent is not capable of achieving that goal on its own.

In the next three sections I briefly describe the blocks world, the Neurosolver and the agents used. Further details will be presented in the later chapters of this work. An overall understanding of the components involved will make it easier to tie them up together.

FIGURE 1. A blocks world with a 3x3 workspace and three blocks: red (A), green (B) and blue (C).

2.1 The blocks world

In order to explore the capabilities of the Neurosolver and Neurosolver-based agents, I have created a version of a blocks world (Figure 1). Blocks worlds were, and still are, commonly used by many researchers (e.g., Nilsson, [41], Marsic-Fogel, [39]) as very convenient environments for

testing and presenting ideas. In the field of AI, ideas are often very complex and require relatively simple models to avoid an excess of detail and subsequent confusion.

The blocks world considered in this work consists of three colored blocks¹ that can be moved in a workspace of nine locations arranged in three vertical columns and three rows. One of the sides of the world is regarded as a table. Gravity pulls the blocks toward the table. The table and gravity introduce constraints on the feasible arrangements of the blocks and thereby determine the set of valid and invalid block configurations (the details will be discussed later in this document).

Any block configuration in the blocks world constitutes a state of the system. Changes to the configuration in time are described in terms of **state transitions**, i.e., sequences of two states corresponding to two consecutive configurations, and **trajectories**, i.e., sequences of several states that can be decomposed into a series of individual state transitions. A state transition spans two adjacent time instants, while a trajectory is a history of the system behavior over a longer time interval. In later sections of this work, the terms *state of the blocks world* and *block configuration or arrangement* will be used interchangeably.

FIGURE 2. The solution to the problem in a global sense is a sequence of configurations between the current configuration (state) and the goal.

There are many possible types of problems in my blocks world. To keep things simple, I consider the problems of rearranging the blocks from an initial configuration to a desired configuration.

¹ Due to the limited black and white presentation medium the blocks are additionally marked as A, B, and C.

Hence, a problem is posed by presenting a goal (i.e., the desired state) to the blocks world with an initial arrangement of the blocks (i.e., the current state) (Figure 2) and asking for a solution. The solution to the problem is a trajectory starting with the current state of the blocks world and ending with the goal. Only valid states can be included in the trajectory.

A block can be moved in any of the four directions: UP, RIGHT, DOWN or LEFT by an operator or by an internal agent cruising throughout the locations of the blocks world. Note that a state transition may consist of a number of elementary moves of a block. A system operator uses a graphical user interface (GUI) to perform the moves and a number of other operations. The constraints on block movements are enforced by the GUI. The agents move the blocks when a problem is being solved. To move a block, an agent has to move to the location occupied by the block, fetch the block, carry it to the goal location and drop the block off in there.

FIGURE 3. A neurosolver in action.

2.2 The Neurosolver

The Neurosolver is a device that is capable of recording the behavior of an observed system. Its architecture and functionality is based on insights into the columnar organization of the human brain (the references to relevant neurobiological literature can be found in APPENDIX A *Biolog-*

ical roots of the Neurosolver: An insight into the human brain and (Bieszczad, [10], [8] and [9]).

The system can be observed by sensors that detect its state as illustrated in Figure 3. In the case of a blocks world, a state is just a configuration of the blocks. The states and, more importantly, the patterns of their changes are input to the Neurosolver. The Neurosolver modifies its inter-nodal connections according to the adaptation rules. On the other end, each node has a determined meaning, so its output signals may be used to control manipulators to alter some aspect, or several aspects, of the observed system. In terms of the blocks world, it is another block configuration. The Neurosolver is capable of activating a sequence of intermediate states, block configurations, that lead from the current state to the goal state. The nodes that fire (i.e., their activity exceeds certain threshold value) control the manipulators that modify the system, so its behavior is the same as was observed, and recorded, in the past. Through the sequence of manipulations corresponding to the sequence of firing nodes, the desired state of the system, the desired configuration of the blocks, is achieved. The goal has been satisfied, the problem solved, hence the origin of the name of the device: neuro-solver.

The Neurosolver starts to interact with the subject system as a *tabula rasa*. It gains all its experience, and problem solving capabilities, through the interchange of sensory and manipulation control data with the system through its inputs and outputs. In spite of being used in the current model to make things simpler, in theory, a separate learning cycle is not needed. The Neurosolver may learn while servicing the system, although initially there would not be much it could do.

Various types of sensory and manipulatory interfaces between the Neurosolver and the blocks world could be designed. Two types of the interfaces have been implemented in the course of the work reported in this document. In the first interface that was implemented initially, the sensors and manipulators were directly encoded in the software. This interface was used to test the capabilities of the Neurosolver. The sensory system implemented in that way requires direct access to the state of each location. Similarly, the manipulator can modify the state of any location. While such an approach may be suitable in some cases, there are many situations where more robust

interfaces are required. For example, one can imagine a camera hanging over the workspace and digitizing the top view of the world. Such an image could be translated into a state required by the Neurosolver. However, if the top view is not available, when the locations are distant like in a telecommunication network, then such an approach is not viable. Similar problems can be raised with respect to the manipulatory system. The aforementioned reasoning led to another approach in which the interfaces are implemented as distributed systems based on agents.

2.3 The agents

There are three types of agents travelling throughout the blocks world: *sensing*, *solving* and *manipulating*. Each agent is mobile and can determine its next move. The direction of the move is generated randomly except for manipulating agents that can be controlled by the solving agent's problem resolution plan. The sensing agents are thought to be the simplest architectures, therefore, they can travel at large speeds throughout the blocks world. The solving agents are the most complex, so they may spend relatively long periods of time at each location. Although there may be any number of agents of each type in the blocks world, the experiments performed so far involved only one solving agent.

A **sensing agent** moves randomly from one location to another and records their occupants, i.e., one of the three blocks or the fact that the location is not occupied by any block. When a sensing agent meets a solving agent at any location, then the sensing agent reports the collected local view of the world to the solving agent. In order to ensure that the exchange of data is complete, neither of the agents can leave the location for the duration of the communication session. After the data is transferred, both agents continue their respective journeys.

A **solving agent** also moves randomly between the locations of the blocks world. The purpose of its trek is to meet many of the other agents and either obtain data from the sensing agents or distribute tasks to the manipulating agents. The solving agent can be presented with a problem to solve. If no such task has been assigned, then the agent attempts to determine the state of the

blocks world from the reports received from the sensing agents met in the past. If there is a high degree of consistency in the reports, then every new state is presented to the solving agent's brain, an instance of the Neurosolver, as a teaching sample. In that way, the agent learns the trajectories in the state space of the blocks world on-line. If a problem to solve is posed by presenting the goal state, then the same Neurosolver-based brain is used to generate a solution. The learning is suspended until the task is carried through. The solution path is tokenized and each of the successive state transitions is distributed in turn to the manipulating agents that are met. The solving agent waits with the distribution of the next step in the solution until reports from the sensing agents indicate that the state that is the goal of the state transition being currently distributed has been achieved. After the whole solution path has been distributed, the problem has been successfully resolved.

A **manipulating** agent moves randomly throughout the blocks world until a solving agent is met that has a block rearrangement plan to execute. The next state of the blocks world is received from the solving agents as a binary representation of the configuration of three blocks in a world with nine locations (see Figure 1). The agent discards the color and translates the configuration into a single block location space by performing a number of simple logical operations on binary vectors. The result is a pair of two locations of the block to move: the source - the location that is currently occupied by the block - and the destination - the location to which the block should be moved. The agent may use an instance of the Neurosolver to generate the path from the agent's location to the source location of the block and - after the block has been fetched - from the source to the target location. The instance of the neurosolver employed by the manipulating agents operates in the state space corresponding to the locations of the blocks world. This is in contrast to the neurosolver controlling the solving agents that operates in the state space of the blocks world. Note that any path generation technique could be used, but there are certain important advantages of using a neurosolver. Firstly, the manipulating agent learns all the time unless it is moving a block, so any permanent changes to the environment, for example emerging obstacles, could be recorded. Secondly, there might be some preferred, for example less expensive, paths that change

dynamically Thirdly, if an agent is impaired in some non-terminal way, then assuming that some mobility is left it can still learn new ways to perform tasks.

A manipulating agent that does not use a neurosolver, performs the task in a “blind search” manner. It moves randomly and if it arrives at the starting location of the block to move, then the block is fetched. Then, the agent continues its random trek and if it happens to get to the destination location, then the block is dropped off.

FIGURE 4. The solution to a reconfiguration problem in a local sense is a sequence of moving actions one block at a time. Each of the moving actions may be performed by a different manipulating agent.

There might be many manipulating agents that had received the same task from the solving agent, so there is a race to move the block. Only one agent is successful. If an agent arrives at the source location and there is no block to move, then it assumes that it has lost the race and returns to its random wandering throughout the blocks world. It may be given another task when it meets a solving agent again.

2.4 Thesis contributions

This thesis makes several contributions in the area that falls into the interest of a number of scientific fields: Cognitive Science, Artificial Intelligence, Neural Networks and Engineering

From the perspective of Cognitive Science, this thesis confirms that the biologically plausible model of higher level information processing proposed by Neuroscience (Burnod [14]) can in fact

be used to resolve practical problems that require extra dose of intelligence. This is in contrast to more primitive tasks (e.g., pattern recognition) that are traditionally tackled by neurally-derived systems and that do not exhibit the same level of intelligent behavior. The importance of inhibition is confirmed as well by using it in preventing oscillating patterns of activity.

Furthermore, the thesis proves that such a model can be successfully implemented in a form of a software simulator. Hardware implementation is pointed out as superior, because it would provide capabilities to perform truly parallel searches. Although such an implementation is not attempted in the scope of this work, the diagrams illustrating analog signalling indicate that such parallelism can, in fact, be achieved.

The model is used to solve problems that traditionally fall into the domain of Artificial Intelligence. Not only it is shown that the Neurosolver can solve such problems, but that it seems to exhibit better solving capabilities than traditional Artificial Intelligence methods. Although not mathematically proven, it is also argued that the Neurosolver scales better than other methods. That is because its ability to combine elements of learning samples providing high degree of generalization. Excellent search capabilities of the Neurosolver are the result of its parallel, breadth-first nature, aforementioned generalization and capability to synthesize bits and pieces of formerly recorded patterns. One aspect of the scaling capabilities of the Neurosolver constitutes a considerable contribution to the research concerning search: in the Neurosolver, search is not affected by the number of stored samples, assuming that the number of nodes does not increase.

Two major features make the Neurosolver very distinguishable from most of other neural networks. These are: its capability to attack higher level problems and its fast learning. Traditionally, neural networks are used to process low level tasks and take very long time to train. There are very few reports on neural networks that provide similar capabilities to solve cognitive tasks and construct plans. To the best of my knowledge, there is no other neurally derived system that would perform a similar kind of search.

From the Engineering perspective, the thesis describes a device that can be used to tackle numerous practical problems. Various aspects of real-world systems, if not complete systems, can be modeled in a framework of the state space paradigm. With such a convenient mapping, many related problems can be represented in a way suitable to be resolved by the Neurosolver.

Another important contribution is the use of multiple agents for sensing and manipulating environments. Applying a confidence factor in the algorithm for detecting a state of a system is directly applicable to applications in, for example, fault management. Similarly, researchers funded by the Department of Defence will find the concepts valuable in the context of autonomous agents in an automated battlefield. Although non-conclusive, the ideas concerning cooperative behavior of multiple solving agents should also be stimulating for further advances in related fields.

It is this author's hope that the reader not only will appreciate the contributions made by his research, but at the same time she/he will find the thesis interesting, enjoyable and stimulating.

Chapter III: Introduction to problem solving and planning

There is no doubt that the main manifestation of human intelligence is the capability to solve problems. Artificial Intelligence attempts to mimic various hypotheses that have been proposed over the years of research into the human intelligence. Inherent to the problem solving is the search for solutions using a database of premises. The presentation of the problem is a factor that directly affects the search, and at the same time indirectly the problem solving. A solution to a problem can be viewed as a plan for resolving the problem. In this chapter, I briefly introduce the concepts from symbolic problem solving predominant in Artificial Intelligence. A particular stress is put on problem representation, state spaces, search and planning. I hope that this introduction will help the reader not familiar with symbolic Artificial Intelligence to understand the later parts of the thesis in which I will refer to those concepts very often without further explanation.

3.1 Problem representation

The behavior of any system can be described by a number of observable parameters. A set of the values of those parameters at any point in time is considered a state of the system at that time. A problem can be represented in a number of ways using that basic convention.

3.1.1 State spaces

In the state-space approach (Feigenbaum [22], Nilsson [41], Winston [64], Russel [48]), each state is a point in an n -dimensional space with the n coordinates corresponding to every observable parameter of the system. At each point in time, the system is in the state that is being represented by a point, or a vector drawn from the origin to that point, in the space whose projections on each of the axes determine the values of the corresponding parameters (Figure 5). As the values of the system parameters vary over time, so does its state. If we assume that time is discrete, then we can say that the system moves from a state $s(t)$ at time t to $s(t+\Delta t)$ at the next time tick (i.e., $t+\Delta t$). That is the same as saying that the vector of system parameters, or state variables, changes from $(v_1(t), v_2(t), \dots, v_n(t))$ to $(v_1(t+\Delta t), v_2(t+\Delta t), \dots, v_n(t+\Delta t))$. Note that some, or even all, of the variables may stay unchanged.

FIGURE 5. A 3-dimensional state space with an arrow illustrating a transition τ between state $S(t)$ and $S(t+\Delta t)$

In the state-space approach, a problem is represented by a triple $\langle S, O, G \rangle$, where

- S - a set of initial states,
- O - a set of state transition operators and
- G - a set of goal states

The solution to the problem is a sequence of state transition operators $(\tau_1, \tau_2, \dots, \tau_k)$ that transform the initial state S into the goal state G . A number of intermediate states may be involved in

the transformation, yielding a state trajectory $(S = S_0) \rightarrow S_1 \rightarrow \dots \rightarrow (S_k = G)$. The same can be annotated as $G = \tau_k(\dots(\tau_2(\tau_1(S = S_0)))\dots)$.

3.1.1.1 Example

Imagine a telecommunication network which is being polled at certain intervals and which reports such parameters as CPU and link utilization, level of service availability (for example a number of calls that were rejected due to lack of resources) and quality (for example, a number of calls that were terminated due to the system faults), etc. A set of such reports constitutes a state of the network. The network observes each of the states and decides whether the state is normal or abnormal¹. The operator is expected to undertake an action that is appropriate for any encountered abnormality.

The task for the network administrator is to force the network to change its state from the current, abnormal state to a state that is considered normal. The current abnormal state is the initial state for the problem, the desired, normal state becomes the goal and the repertoire of the network practices known to the administrator determines the state transition operators.

Note, that before any repair action can be undertaken, the network administrator has to decide in which category the current state can be classified. If the recovery action can be determined by analyzing each of the network states alone, then the state space is used just for problem classification. Every region in the state space is assigned a category². If the network state falls into such a region, then it is classified as belonging to that category. There are a number of clustering techniques that can be used to effect such a classification. Each region might be assigned a point that approximates the best all members of the corresponding category. This concept is called *Voronoi tessellation* of the state space. The borders between the *Voronoi regions* are determined by the points that have the same distance to the centers of two neigh-

¹ Note that there may be many undesired, abnormal states and many desired, normal states.

² In the simplest case, there are two categories: normal and abnormal. Quite often though, there is a whole spectrum of categories between those two extremes

boring regions. A state is classified as being of category C_i if the distance $d(c_i, S_m)$ between the center c_i of the region corresponding to the category C_i and the point S_m in the state space that corresponds to that state is smaller than to the center of any other region.

$$(\forall j \neq i) d(c_i, S_m) < d(c_j, S_m) \quad (\text{EQ 1})$$

Determining whether the network needs some attention is usually more complex than examining a single state. It is not sufficient to observe each state separately. Instead, patterns of state changes must be analyzed. Several such temporal patterns might also be undesirable and should be detected or, preferably, prevented. The prevention may involve detection of state transitions that lead to an undesired state and altering network parameters before that state is reached. Obviously, the state-space approach with its concept of state transitions is very helpful.

There are temporal patterns that are just sequences of consecutive states, but also there are others that may be sequences of states that are not consecutive; i.e., there are some intermittent states that do not contribute to the overall classification of the pattern and which differ from one case to another. Spectral analysis is very difficult, but the state-space representation may be of some help as well. It may also be possible to apply some of the compression techniques commonly used in other areas of telecommunications to detect such dispersed regularities.

3.1.2 Other representations

There are other approaches to problem representation. I present a very short description of problem reduction and game trees just to give the reader an idea what the other representations might be.

3.1.2.1 Problem reduction

In the problem reduction approach (Feigenbaum [22]), a problem is represented by a triple $\langle S, O, P \rangle$, where.

- S - a set of initial states,
- O - a set of reduction operators and
- P - a set of primitive problems that can be resolved

Using the problem reduction representation, the problem is solved when it is reduced to a number of primitive problems that are axioms or can be resolved trivially. The reduction is achieved by applying reduction operators at each stage starting with one of the initial states.

3.1.2.2 Game trees

A problem can be represented as a game tree (Feigenbaum [22], Winston [64]). The root of the tree is the initial state and the terminal nodes, the leaves, represent a win, a loss or a draw. Each node in the tree symbolizes a possible state of the game. The children nodes of a node, represent all possible moves from the situation exemplified by the parent node.

The problem is solved when a terminal node (preferably representing a win) is reached.

A game tree is constructed for two or more competing problem resolutions: one for each of the game payers. The heuristics used to determine the next move are based on favoring own wins and opponent losses.

3.1.3 Search

Now when we have a problem representation and we know what a solution is, we need a search algorithm that would find a solution and perhaps select just one if there are many possible solutions. Search implies a construction of a search tree in the state space by considering alternatives

for successive or preceding states. The search for a given problem starts at the root of the appropriate tree. The goal is expected to be on one of the leaves of the search tree

There have been many of search strategies designed. I will present some of them in the following sections. All strategies can be evaluated by the following criteria

- completeness; i.e., whether the strategy guarantees that a solution is found if one exists,
- time complexity,
- space complexity,
- optimality; i.e., whether the best solution is selected out of many possibilities

3.1.3.1 Blind search

In a blind, or uninformed search, there is no information on the cost of getting from the initial state to the goal. The system can only evaluate whether the current state is the goal state

3.1.3.1.1 Breath-first search

In breadth-first search, the search tree is constructed gradually in all possible directions at the same time. The order of the expansion of the tree is not important as long as all of the branches are expanded at the same depth before moving one hop further from the root

This strategy is complete, but very inefficient. The solution is guaranteed to be the shortest.

The uniform cost search strategy is the breath-first search with a certain cost function used to determine which branch should be expanded next. The cost function evaluates the cost of the path from the root to the current state. The node for which the function is minimal is selected as the next. This strategy finds the cheapest (as measured by the cost function) solution

3.1.3.1.2 Depth-first search

In depth-first search, each branch of the search tree is explored as far as possible. If the goal is not found after all sub-branches have been explored, then the system backtracks and the deepest next

possible branch is searched. Unfortunately, if there are cycles or infinite branches, then this strategy fails.

This strategy is not complete and might not return the best solution, but its attractiveness lies in its relatively low spatial costs.

We can limit the maximum depth of the explorations for each branch of the search tree in order to avoid inconclusive searches. If a solution cannot be found with the current depth limit, then the limit can be increased iteratively.

3.1.3.1.3 Bidirectional search

Bidirectional search is the search that involves two search trees. One is rooted in the initial state and the other is rooted in the goal state. If the two trees intersect at any state, then the solution to the problem is a concatenation of the paths from the roots of the trees to the intersection.

This strategy implies the use of another, breadth-first-like strategy to construct the trees.

3.1.3.2 Heuristic search

Heuristic searches use problem specific knowledge in order to speed up the algorithms. They employ certain evaluation functions that measure the quality of the candidates for the next state. The function orders the nodes and the best candidate is selected; therefore, the search is called best-first search.

The uniform cost search is not considered heuristic, since the evaluation of the path to the root does not inaccessibly require problem specific knowledge.

3.1.3.2.1 Greedy search

In greedy algorithm, the evaluation function measures the cost of moving from each of the candidates for the next state to the goal. The node for which the cost is minimal is chosen; therefore, the search is called *greedy*.

The greedy search is neither complete nor optimal. Nevertheless, it is relatively efficient.

3.1.3.2 A* search

A* search combines greedy search with uniform cost search. The latter measures the cost of moving from the root to the current state, while the former takes into account the cost from the next node to the goal. The combined evaluation function is a simple sum of the functions used in the greedy search and in the uniform cost search. If certain constraints are laid on the evaluation function¹, then the A* search is complete and optimal.

The greedy component makes the A* search more efficient than blind search strategies.

There are various extensions to the A* search that would not be suitable for this short introduction.

3.1.3.3 Search with iterative improvement

Iterative improvement techniques are appropriate in cases where the actual path of getting to the goal or *close* to the goal is irrelevant. Usually, a certain function is used to describe the system. Such a function is then used to guide the state transitions toward the goal.

3.1.3.3.1 Hill-climbing

In hill-climbing, or gradient decent, search, a function that measures the error that the system makes by being in a state different than the goal is used. The state transition is always in the direction that minimizes that error.

If we imagine the error function as a surface (it is easy in 2 or 3 dimensions, nevertheless, the same applies to higher dimensions), then the next state is selected in such a way that the current error moves downhill. This process is done in steps that may actually overshoot the minimum, so

¹ The requirement is that the component of the evaluation function pertaining to the greedy search (i.e. from the next node to the goal) is *admissible*. *Admissible heuristics* never overestimates the cost of reaching the goal.

the point corresponding to the current error may also move up. Going up and down the hills on the error surface is the origin of the name of this strategy.

The optimal solution is found when the error is in the global minimum of the error function. Since the error surface may contain many ridges, valleys and plateaux, the system may get stuck in a local minimum. All surrounding points are higher, so the system decides that the job has been done. That may not be so, since another minimum can be actually lower.

There are many of techniques that attempt to speed up this process and to increase the chance of hitting the global minimum.

3.1.3.3.2 Simulated annealing

Simulated Annealing gets its inspiration from ideas from Physics. The system is described by a certain function (called the system energy function) selected in such a way that the desired state is attained when the function is minimized. Then the system is allowed to change its state randomly. If the selected state yields better evaluation (decreases the function), then the system moves to that state. If the situation has worsened, then the move is made only with a certain probability that depends on how bad the change would be. The probability depends on a parameter that is called the temperature of the system. The temperature is gradually lowered, so the moves to states that make things worse are less and less likely.

It has been proven that if the temperature of the system is lowered sufficiently slowly, then the system will settle in the lowest-energy state. There is a direct analogy to annealing – a physical process of gradually lowering the temperature of a liquid until it freezes.

3.1.4 GPS - General Problem Solver

The General Problem Solver (GPS) (Newell [40]) is a precursor of many of the other systems used over the years to explore various aspects of Artificial Intelligence. GPS is based on the notion of state spaces and a heuristic technique known as means-ends analysis (see Figure 6). The

problem universe consists of all possible states with each state being a potential goal. A problem is represented by two states: the initial state and the final, desired state. Problem resolution is a sequence of state transitions that designate a path between the initial and the goal states.

The current state can be altered by a state transition operator. A number of operators can be applied, but only those that bring the system closer to the final state are considered. The best (in terms of some heuristic) operator is chosen. If the preconditions of that operator are not satisfied, then a sub-goal to achieve the preconditions is created. The resolution of the previous goal is suspended and the system switches its attention to the new goal. When the sub-goal is reached (it may involve another layer of sub-goals), then the previous goal is attained as well by the application of the operator. GPS allows also for backtracking if the path chosen so far seems to be unpromising (for example, when the distance to the top level goal increases).

There are three types of tasks:

- a *transform task* to transform one state into another,
- a *reduce task* to reduce the difference between two states and
- an *apply task* to apply an operator to a state.

The operators are accompanied by predefined methods or procedures that include the instructions on how to achieve the task. The initial task is to transform the initial state into the final state. The recursion stops when the task is trivial. A task is trivial if:

- for a transform task, there is no difference between the pre-application and post-application states;
- for a reduce task, if all relevant operators have been tried and failed. If there was a success, then recursion does not stop,
- for an apply task, if the operator is immediately applicable.

For each type of task, there is a predefined list of possible types of differences. The list is ordered by estimated difficulty. The most difficult difference is chosen for reduction. A domain specific data structure called Table of Connections defines the operators relevant to reducing each difference type. As illustrated in Figure 6, if the difference $\text{diff}(\text{CS}, \text{GS})$ between the current state (CS) and the goal state (GS) is chosen for reduction, and the operator τ is suggested, but cannot be applied to CS, then a new state is sought, called adjacent state AS, such that τ can be applied. The system switches its attention from the transition from CS to GS to the transition from CS to AS, because if the adjacent state can be reached, then simple application of the operator τ will result in GS. AS becomes a new GS. After the new sub-problem is resolved in a similar process that may involve more applications of means-ends analysis principles, then the solution is preceded with the operator τ to yield the solution of the original problem.

FIGURE 6. Applying the means-ends analysis principle in GPS.

and the goal state (GS) suggests the use of the operator τ that cannot be applied to the current state, then a new state is sought, called adjacent state AS, such that τ can be applied. The system switches its attention from the transition from CS to GS to the transition from CS to AS, because if the adjacent state can be reached, then simple application of the operator τ will result in GS. AS becomes a new GS. After the new sub-problem is resolved in a similar process that may involve more applications of means-ends analysis principles, then the solution is preceded with the operator τ to yield the solution of the original problem.

3.2 Planning

3.2.1 Difference between planning and problem solving

Planning is very closely related to problem solving. While a resolution of a problem may be perceived as a sequence of transitions between two states, the original state and the goal, a plan is commonly understood as a sequence of actions that would actually control the system so it moves through each of the states in the resolution path in order. The actual differences are more subtle

and generally have their source in the attempts to make problem solving practical. Russel in his elegant monograph on AI (Russel [48]) states the following three ideas that are differentiating factors between planning and problem solving:

- “opening up” the representation of states, goals and actions. A state is no longer a complete description of the world with every what-is and what-is-not as was the case in problem solving. The actions are based on preconditions and effects;
- the planner can add actions to the plan whenever they are needed. In problem solving, an action can be generated only if there has already been established a path from the initial state;
- independent sub-plans can be constructed separately.

Planning also implies searching, but searches are not performed in a complete state space. Instead, a space of situations or a space of partial plans is searched. Situations are similar to states, but they exclude things that are irrelevant. Searches can be progressive, from the original situation to the goal situation, or regressive, in the opposite direction.

A partial plan is a plan with missing details. The search for the solution in the space of partial plans is a process of refining the plans by the use of appropriate operators. Many modern planners use the principle of least commitment which allows for leaving certain details of the plan to remain unbound until they are actually required. Another characteristic of modern planners is the capability to deal with only a partial ordering of the steps or parts of the overall plan. Such partial order planners are able to attack a number of issues involved in the resolution of the problem, even if they are not in a chronological order.

One of the earliest, if not the earliest, planners that were developed in the framework of AI research is STRIPS (Nilsson [41]). I present a description of STRIPS in the next section. Although there are more powerful planners that cured many of the problems that STRIPS suffered from, STRIPS is still a very good illustration of the basic ideas and concepts pertaining to planning

3.2.2 STRIPS

3.2.2.1 Model

STRIPS controls a robot that can move objects between interconnected rooms. The world is represented by first-order predicate calculus. The state space representation is used to express the current state of the problem. The position of the robot and the boxes are encoded in a number of currently active formulas that constitute the state of the problem. They are accompanied by transition operators that are used to modify the state; i.e., to change the set of currently active formulas.

Each operator has a set of preconditions under which it can be applied. There are also two associated lists:

- *delete list*, with rules regulating which formulas should be deleted from the current state when the operator is applied and
- *add list*, with rules regulating which formulas should be added to the current state.

When the preconditions are satisfied, then the current state is modified by the operator in the way defined by the operator's definition.

3.2.2.2 Operation

The initial state consists of a number of well-formed-formulas (*wffs*) (Nilsson [41] or any textbook on formal logic) that reflect the initial state of the model and the desired goal expressed by another set of formulas representing the desired state of the world.

A theorem prover is used to derive (prove) the final state from the initial *wffs*. If that fails (a usual case), then the means-ends analysis is applied to transform the current state into another state from which the goal can be proven by another iteration of the theorem prover. The difference between the current state and the desired new state is based upon the formulas that are in the goal and could not have been proven so far. Relevant operators that, in their add lists, have formulas

that remove at least part of the difference are considered. If the pre-conditions cannot be met for the operator that is required, then a new sub-goal to meet those pre-condition is created.

In that way, STRIPS builds a tree of models (states) and sub-goals. The searches through the tree are controlled by a number of heuristics that I will not discuss here.

There are certain problems, sometimes quite simple that STRIPS and many other early planning systems were not able to solve. Later in this thesis, I will examine one of them, the problem of scheduling the exchange of the data contained in two registers through a third register, in the context of the solving capabilities of the Neurosolver. Least commitment, partial order planners have been shown to fare much better (Weld, [60]).

Chapter IV: Introduction to Neural Networks

4.1 Introduction

Neural Networks constitute a class of data processing systems consisting of many simple processing elements that are highly interconnected. The function of each processor is important only within the context of the whole network. The computational power of a neural network system is determined by its topology and the strength of the inter-processor connections. Neural networks are computing devices exploiting a different computing paradigm than traditional von Neuman's machines.

The speed of conventional computers increases every year. However, there are limits on the number of mappings between input and output and their efficiency when only one processor is used. An obvious solution to that problem is to increase the number of processors, but the programming of such a parallel computer is a very complex task.

Although the mapping of the input into the output in neural network systems can also be programmed, their ability to learn and adapt (i.e., to create the input-output mappings in an automatic way) makes them very distinct from conventional computers. A neural network can self-program and, if there is such need, dynamically change its programs.

In this short introduction to the Parallel Distributed Systems, first, some biological fundamentals are presented followed by an analysis of the elementary processor and the simplest neural network

architecture. Further, three main types of architectures are described using representative examples.

4.1.1 A Few Words About The Nature

Nature has provided humans with the powerful ability to learn. Through experience and practice, humans adjust to the world and acquire knowledge, which was not transferred by genes. From discoveries in Neuroscience and Psychology, it is known that the human brain is a very flexible processor of information. It is able to adapt to any change in the surrounding environment perceived by sensors. Human sensors are very well developed at birth and they are not adaptive to the same extent as other parts of the neural system.

FIGURE 7. Neurons connected by an axon, a synapse and a dendrite. This is only one of possible connections.

A neural system is composed of small cells called **neurons**. There is an example of a biological neuron shown in Figure 7. Neurons are densely interconnected with the strength of the interneuron connections believed to be the basis of the brain's processing power and storage capability. Any two neurons are connected by an **axon** which transmits a signal (the so called **train signal** or **action potential**) from the source neuron to a **synapse**, the place where the axon connects to the **dendrite**¹ of the target neuron. The synapse decides about the **connection strength** between the

pre- and post-synaptic neurons. It has been proven that via metabolic processes, the strength of the synapse, its conductivity, can be changed. In the neural network community, that change is described by the Hebb rule, so called after the name of a scientist who first proposed it. The Hebb rule states that *the strength between two neurons is increased if the neurons are excitatory for each other, and vice versa - the strength is decreased if the activation of one of the neurons forces another to decrease its activation.*

Neural systems, and particularly the human brain, are proof of how an elaborate behavior can be achieved with processing elements such as neurons. The brain processes and their mechanics are being researched by a growing part of the scientific world, but the actual discoveries are mostly limited to the microscale. At the higher level, most of the explanations of the functionality of the brain are just hypotheses.

4.1.2 Neural Networks: An attempt to mimic Nature

Gaining insight into the human ability to process and store information is a fascinating research activity, so it is not a surprise that several scientific disciplines have been involved in attempts to understand and model the brain and its functionality. Psychology (Grossberg, Anderson), Cognitive Science (Rumelhart, McClelland), Electrical Engineering (Widrow, Kohonen), Physics (Hopfield, Smolensky), Optics (Amari, Psaltis), Neuroscience (Baird, Pellionisz), and Computer Science (Hinton, Feldman, Ballard) contribute to the new - or rather newly reborn - discipline. There are several terms used to refer to it: **Artificial Neural Networks**, **Parallel Distributed Processing (PDP)**, **Connectionist** or **Neuromorphic Systems**. Each contributor has his own goals in modelling the brain architecture and functionality. The goals may be very distinct, like optimization (Physics, Computer Science), new computing paradigms (Computer Science, Engineering), autonomous controllers (Computer Science, Engineering), the explanation of the human behavior (Psychology), and many others. The general approach, ideas, architectures and algo-

1 It may also connect to other parts of the cell, but a dendrite is the most common place

gorithms are shared. Research in this field falls into a broader area of **Computational Intelligence** that encompasses, in addition to the neural systems, systems based on fuzzy and rough sets as well as evolutionary systems including genetic algorithms and genetic programming.

A fundamental building block of all neural network architectures is an **artificial neuron**. It is usually based on a biological neuron¹, but in contrast to its biological counterpart whose functionality is still not completely understood, an artificial neuron is a very simple computing device. It takes several input signals, calculates an **input function** f on them, and, depending on the result, changes its internal state A , called **activation**, and outputs a value obtained by applying an **output function** O to the state. It adaptively adjusts the input function f accordingly to certain criterion; for example, the feedback based on overall performance

FIGURE 8. Artificial Neuron.

Usually the input function f is a sum:

$$f(S) = \sum_{i=1}^m s_i \cdot w_i \quad (E.Q. 2)$$

where, $S = (s_1, s_2, \dots, s_m)$ is the **input vector** with s_i as an **ith input signal**,
 w_i - the **weight** of the **ith input signal**;
 $W = (w_1, w_2, \dots, w_m)$ is the **weight vector**.

1. Not all neural networks use a biological neuron as a prototype for a unit node. Some, in particular the networks coming from Cognitive Science and Psychology, more often referred to as connectionists rather than neural networks, use nodes that are grandmother cells (see section 7.3.1 *Input representation*). They stand in contrast to the nodes in more common neural networks that do have a meaning only as a part of larger set, in a distributed representation of concepts.

The simplest activation function follows

$$A(n_c) = f(S) = 0 \quad (\text{EQ 3})$$

where θ is a **threshold** and n_c indicates a specific neuronal cell.

In the simplest case, the output function could be.

$$O(n_c) = \begin{cases} +1 & \text{if } A(n_c) > 0 \\ -1 & \text{if } A(n_c) \leq 0 \end{cases} \quad (\text{EQ 4})$$

If a number of such simple units are interconnected, the behavior of the whole network has been proven to be very complex, and various interconnection schemes have been used to test hypotheses about the brain. Behavior is mostly determined by the type of the interconnections and their strengths, and it seems to be consistent with what is known about the nervous system.

Neural network systems are able to dynamically change their interconnections under some, externally specified, conditions. This is why we often call them **adaptive systems**, and the algorithms which perform the interconnection adjustments - **adaptive learning**.

There are several ways of adjusting the input function, which determines the type of the network interconnections. The change involves adjustments of the input weights depending on the contribution of each input to the new state of the neuron. The Hebbian rule, which has already been mentioned, is the basis for several methods. The **delta rule**, for example, adjusts the input weight vector so the erroneous behavior of the neuron or the network (**generalized delta rule**) will be eliminated. Another variation of the adaptation rule is a rule which adjusts the weights in such a way that a special function, called the **system energy** (the name and the idea are borrowed from Physics) is minimized.

In the course of this short introduction we will use four types of neural network systems:

- pattern associators.

feedforward (Perceptron, Backpropagation Network, Time-Delay Network),

- recurrent (*Jordan Network*),
- **auto-associator** (*Hopfield Network*),
- **structure mimicry** (*Kohonen Maps*) and
- **weightless** neural networks.

There are many more networks currently in use or being researched. The examples given here, however, illustrate the major trends. Before we describe the networks, we will present the precursor of many modern networks, from which almost every model borrows several ideas - the **perceptron**.

It is worthwhile noting, that all neural networks are huge approximations of the biological systems at every level of detail starting with the neuron itself.

4.2 Perceptron

The perceptron (Figure 9) is the simplest neural network system, and was introduced in 1959 by Frank Rosenblatt (Rosenblatt, [45]). It consists of two layers: the input layer (**sensors**) and the output layer (**classifiers**).

The network was thought to be a tool able to generalize and classify by processing examples. The idea was to provide the sensors with features extracted from sample data, and to receive appropriate classification on output lines. The system was supposed to automatically find the transformation function, which would produce correct classifications for each specific set of input data. The goal was achieved for a certain class of problems. It was shown later, by Minsky and Papert (Minsky, [36]), that if the input data is not *linearly separable*, then the perceptron will fail to find the categorization function. Unfortunately, most interesting problems could not be solved by the perceptron.

The system learns by adjusting its weights $\alpha(s,p_i)$, accordingly to the instructions from the teacher. Each time a training sample S is presented to the perceptron, a set of output signals O is

activated. The teacher tells the system what the correct output $T=(t(p_1), t(p_2), \dots, t(p_{nc}))$ should be, and the system, after calculating the error E , adjusts its weights as follows

$$\forall i, j \quad \alpha(s_j, p_i)_{t+1} = \alpha(s_j, p_i)_t + \eta \epsilon_i O(s_j) \quad (\text{EQ 5})$$

where, $\epsilon_i = t(p_i) - O(p_i)$ is the error for i th output, and η is a small constant (for example $\eta = 0.01$).

The threshold θ is adjusted as well:

$$\forall i \quad \theta(p_i)_{t+1} = \theta(p_i)_t + \eta \epsilon_i \quad (\text{EQ 6})$$

FIGURE 9. Perceptron.

There is the same dimensionality of the input vectors (feature vectors) and weight vectors of each output neuron. We can assume that they are in the same N -dimensional space. If we have only two sensors, we deal with a two-dimensional space spanned by $O(s_1)$ and $O(s_2)$.

The activation level of a neuron is a **linear discriminant function**, since if we expand the activation function formula we have the following:

$$A(p_i) = \alpha(s_1, p_i)O(s_1) + \alpha(s_2, p_i)O(s_2) + \dots + \alpha(s_{ns}, p_i)O(s_{ns}) + \theta(p_i) \cdot 1 \quad (\text{EQ 7})$$

We can express the activation as the inner product of the weight vector

$\alpha = (\alpha(s_1), \alpha(s_2), \dots, \alpha(s_{ns}))$ and the input vector $O = (O(s_1), O(s_1), \dots, O(s_1))$.

$$A(p_i) = \alpha(p_i)^T O \quad (\text{EQ 8})$$

The product is equivalent to the linear discriminant function, because it is proportional to the distance ρ of the feature vector O from a line normal to the weight vector $\alpha(p_i)$

$$\rho = \frac{\alpha(p_i)^T O}{\|\alpha(p_i)\|} \quad (\text{EQ 9})$$

The line divides the plane into two half-planes. The inner product is positive for input vectors falling on the half-plane containing the weight vector, and negative for feature vectors falling on the other side.

FIGURE 10. A geometric interpretation of the activation level of the perceptron in 2-dimensional space.

We can say that finding a linear discriminant function of a specific set of samples which should be classified in the same way, is equivalent to finding the weight vector normal to the line (hyper-plane in a higher dimension space) representing the function. We will find the vector using **gradient search** along the surface of a specially defined function. For the minimal value of the function, the weight vector is optimal, i.e., the discriminant function for the set of samples has been found. Figure 11 shows a simplification, which makes the definition of such a function easier. In the upper part of the figure, all positive samples of B will produce negative inner product, while for all positive samples of A , the inner product will be positive. Because we are looking for a function to minimize, all samples from B are transformed so their inner product is also positive.

This simplification ensures that all negative examples produce a negative inner product of the weight and input vectors, and vice versa - all positive examples produce a positive inner product.

FIGURE 11. Analysis of the discriminant properties of the perceptron. There are many lines discriminating between samples from the set A and B (a). In fact, there is a region that yields a proper solution (b). With a little trick (c), we have a solution region with all classified samples yielding positive activity. Only incorrectly classified patterns yield a negative response.

The following function can be used as a criterion function, since after it is minimized (equal zero), all samples classified by the final weight vector will produce a positive inner product

$$E(\alpha(p_i)) = 1 - \sum_{i \in I} \alpha(p_i) \cdot O_i = 1 - (\alpha(p_1) \cdot O_1 + \alpha(p_2) \cdot O_2 + \dots) \quad (\text{EQ 10})$$

$$E(\alpha(p_i)) = 1 - (\alpha(p_1) \cdot (O_1 + O_2 + \dots)) = 1 - \alpha(p_1) \cdot \sum_{i \in I} O_i \quad (\text{EQ 11})$$

where I is the set of incorrectly classified patterns, and the function has been multiplied by -1 to get a positive value

In the course of our search, we should move in the opposite direction to the gradient (take negation of the gradient)

$$\Delta E(\alpha(p_i)) = -\frac{\delta E(\alpha(p_i))}{\delta \alpha(p_i)} = O_1 - O_2 = \sum_{k=1}^n O_k \quad (\text{EQ 12})$$

The j^{th} weight component of the gradient is

$$\frac{\delta E(\alpha(s_j, p_i))}{\delta \alpha(s_j, p_i)} = O_1(s_j) - O_2(s_j) = \sum_{k=1}^n O_k(s_j) \quad (\text{EQ 13})$$

The last equation represents the amount by which we should adjust each weight vector in order to improve system response. Adjusting the weights can be graphically shown as gradient descent along the criterion function surface

FIGURE 12. Perceptron learning (descending a criterion surface) shown for the j^{th} weight component.

4.3 Backpropagation network

The backpropagation network was designed to overcome the problems which troubled the perceptron. Adding extra layers, called **hidden**, between the **input** and the **output** layers, transforms a non-linearly separable problems from one space into another, in which, hopefully, a linearity is

achieved. It was proven that for each nonlinear classification problem, we can design a multilayered perceptron which will produce a solution (for example, see Freeman, [23]).

FIGURE 13. Backpropagation Architecture

The activation A of the neuron n_{lj} in the layer L is as follows:

$$A(n_{lj}) = \left(\sum_{n_{(l-1)i}} O(n_{(l-1)i}) \cdot \alpha(n_{(l-1)i}, n_{lj}) \right) - \theta(n_{lj}) \quad (\text{EQ 14})$$

where $\alpha(n_{(l-1)i}, n_{lj})$ is the weight of the synapse between neurons $n_{(l-1)i}$ and n_{lj} ,
 $\theta(n_{lj})$ is the threshold value of the neuron n_{lj} .

The output function O is any semi-linear function, for example a sigmoid or sine function:

$$O(n_j) = \text{semi-linear-function}(A(n_j)) \quad (\text{EQ 15})$$

The architecture shown in Figure 13 is used for pattern matching. the input pattern $S = (s_1, s_2, \dots, s_m)$ is presented to the network and the signals from each input neuron are propagated forward, through all hidden layers to the last, the output layer. The outputs of the last layer compose the output pattern $O = (o_1, o_2, \dots, o_k)$, which is compared with the training sample, the error is calculated and propagated backward, towards the input layer. All neurons adjust their weights according to their contribution to the network error, so when the same input is presented again they will generally respond better.

FIGURE 14. Sigmoidal output function $O(n_j) = \frac{1}{1 + e^{-A(n_j)}}$.

The most popular learning scheme for the network of this type, is the backpropagation algorithm, using the generalized delta rule. The following algorithm is repeated until the weights stabilized and the error disappears or stops decreasing

- 1 Initialize all weights and thresholds to random, real-valued, numbers
- 2 Input randomly selected training sample
- 3 Calculate an activation and output for each neuron
- 4 Calculate the error in the produced output (as compared to the teaching sample $T = (t_1, t_2, \dots, t_k)$)

$$\delta(n_{Mj}) = (t_j - O(n_{Mj})) \cdot \frac{\partial}{\partial A(n_{Mj})} O(n_{Mj}) \quad (\text{EQ 16})$$

- 5 Adjust the weights of the output layer neuron (synapse strengths with each neuron of the last hidden layer)

$$\forall i \in [1, n_{(M-1)j}] \quad \alpha(n_{(M-1)j}, n_{Mj})_{k+1} = \alpha(n_{(M-1)j}, n_{Mj})_k + \eta \delta(n_{Mj}) f'(n_{(M-1)j}) \quad (\text{EQ 17})$$

If the output function is sigmoidal we will have

$$\alpha(n_{(M-1)j}, n_{Mj})_{t+1} = \alpha(n_{(M-1)j}, n_{Mj})_t + \eta(t_j - O(n_{Mj})_t)O(n_{Mj})_t(1 - O(n_{Mj})_t)O(n_{(M-1)j})_t \quad (\text{EQ 18})$$

where η is a learning step ($\eta = 0.25$)

- 6 Calculate an error contribution for each neuron in every hidden layer

$$\delta(n_{Lj}) = \frac{\partial}{\partial \Lambda(n_{Lj})} O(n_{Lj}) \cdot \sum_{k \in [1, n_{(L+1)j}]} \delta(n_{(L+1)k}) \alpha(n_{Lj}, n_{(L+1)k}) \quad (\text{EQ 19})$$

For a sigmoidal function

$$\delta(n_{Lj}) = O(n_{Lj})(1 - O(n_{Lj})) \cdot \sum_{k \in [1, n_{(L+1)j}]} \delta(n_{(L+1)k}) \alpha(n_{Lj}, n_{(L+1)k}) \quad (\text{EQ 20})$$

- 7 Adjust the weights of each hidden layer neuron

$$\forall i \in [1, n_{(L+1)i}]: \alpha(n_i, n_{Lj})_{t+1} = \alpha(n_i, n_{Lj})_t + \eta \delta(n_{Lj}) O(n_i)_t \quad (\text{EQ 21})$$

The backpropagation algorithm is also a gradient search algorithm, and as such is subject to the danger of being stuck in a local minimum. There have been many heuristic techniques designed to improve the chance of finding the global minimum, but there is no theory which would assure that backpropagation search will be successful.

The backpropagation algorithm was independently proposed by several researchers. The most common reference is given to the neural network team from San Diego: Rumelhart, Hinton and Williams (Rumelhart, [47]). The backpropagation architecture is commonly used in classifier and pattern recognition systems. Various applications in knowledge processing have also been reported.

4.4 Time-delay network

Time-delay neural networks (TDNN) are similar to backpropagation networks. They use similar learning schema, but have a different architecture. The input patterns are presented to the network in a time window; a number of input patterns at the same time. In that way, in the addition to differentiating between the patterns spatially, the network is capable of extracting features that have temporal nature. A schematic architecture of a TDNN is given in Figure 15. The input vectors enter the input grid from the top as illustrated by the arrow. In each time tick, the patterns are

shifted down the grid: the last pattern is removed from the last slot in the grid, the first slot has the newest pattern. A number of slots, a **window**, has connections to one slot in the hidden layer, as illustrated. This slot is a **receptive field** for the connected slots in the input grid. The next recep-

FIGURE 15. The architecture of a time delay neural network. The network is drawn three times to illustrate the concept of a receptive field. 8 patterns, X_{t-5} to X_{t+5} , are feeding activity into the network at the same time through three windows connected to three receptive fields in the hidden layer.

tive field, the next slot in the hidden layer, is associated with the input slots that belong to the next time window. There is another receptive field illustrated as well. The connections from the slots labelled by the same number in each window are coupled. One such coupled link corresponding

to the first slot in the window is illustrated in the figure as three arrows. Learning is similar to the backpropagation algorithm, but the modifications to the links that are coupled are the same. The modifications reflect the average of the changes that would have been made if the links were treated separately. The connections to the output layer are the same as in the case of backpropagation algorithm

It has been shown (Weibel, [59]), that time delay networks are able to learn to classify time and position invariant patterns.

Time delay networks can be trained to classify temporal patterns. To illustrate that, let us assume that if there is just one input feature; i.e., each input vector is just a single value. In such a case, the time delay network reduces to a backpropagation network (see Figure 15) The set of the val-

FIGURE 16. A temporal pattern of changes to the value of a single feature can be considered a spatial pattern in the time space.

ues of the feature over the time window (one vertical column in the input grid in the picture) can be considered a spatial vector in time space; as such, it can be fed as an input into a backpropagation network. This new vector can be associated with an output pattern by backpropagation training. The corresponding output annotates that particular pattern of changes to the input value over time. At another instant in time, another set of values would be in the temporal input vector;

another output would be associated with it. In a time delay network a similar process applies to all features simultaneously.

4.5 Recurrent network

The Jordan network (Jordan, [28]) is an example of a recurrent architecture (some call it partially recurrent). In addition to the feedforward links present in the backpropagation network, there is a **feedback loop** here. In the Jordan network, the output signals are fed back to the hidden units through some delay in, so called, **context units** (Figure 17). Therefore, the behavior of the network depends not only on the current data presented as an input vector, but also on the past data stored, in a way, in the context. In other words, a recurrent network has a **memory** that affects its future response. As a result of that feature, recurrent networks are able to store sequences of patterns. As such, they may be used to recognize patterns or to predict the future by responding with the next pattern in the sequence after the preceding pattern has been presented. There is a great deal of research on applying such networks as finite state machines.

FIGURE 17. Recurrent network.

There are a number of alternatives to the feedback connections from the output to the context. There might be feedback from the hidden units to the context (Ellman network) or the input layer with a self-feedback loop (Stornetta network) or one of the hidden layers with a similar loop (Mozer network) may act as the context.

Disregarding the feedback connections, this type of recurrent network is trained in the same way as the backpropagation network. After learning, only the first pattern in the sequence is required. Each next pattern is generated owing to the fact that the output pattern is fed back to the hidden layers through the context units.

Recurrent networks are very interesting, but at this moment suffer from a relatively small capacity to store sequences. Some of the research, or rather development, is being done for Wall Street companies interested in predicting trends in financial markets. Considerable part of this development effort is classified as confidential and, therefore, not reported through normal research channels.

4.6 Hopfield network

In 1982, John Hopfield published a theoretical study concerning the dynamics of a certain type of neural network (Hopfield, [27]). He was not interested in the learning capabilities of the system, but, instead, he was looking at the dynamic behavior of the network, given a certain set of weights. The output vector, consisting of the outputs of all neurons in a network with N interconnected units (each pair having a synaptic, symmetrical, connection), might be seen as a point in an N -dimensional space, and considered an instantaneous state of the network. As neurons compute their output functions, the state changes, and the point describing it traces a curve in the space. The curve describes the behavior of the network. The question to answer is whether the network will settle down or oscillate forever, if left unattended

The system works asynchronously; i.e., at each time quantum one neuron is chosen and the following activation function $A(n_i)$ and output function $O(n_i)$ are calculated as follows:

$$A(n_i) = \left(\sum_{j=1}^N O(n_j) \alpha(n_i, n_j) \right) - \theta(n_i) \quad (\text{EQ 22})$$

$$O(n_i) = \begin{cases} 1 & \text{if } A(n_i) > 0 \\ 0 & \text{if } A(n_i) \leq 0 \end{cases} \quad (\text{EQ 23})$$

FIGURE 18. Hopfield Network

The system output is defined as a vector of outputs of all neurons in the system

$$O = (O(n_1), O(n_2), \dots, O(n_N)) \quad (\text{EQ 24})$$

It has been discovered that, regardless of the initial values of the weight, this type of network always changes its state to minimize the following function:

$$E = -\frac{1}{2} \cdot \sum_{i=1}^N \sum_{j=1, j \neq i}^N O(n_i) \cdot O(n_j) \cdot \alpha(n_i, n_j) \quad (\text{EQ 25})$$

By analogy with equations from Physics, the function E is called **energy** of the system. Given a random starting position, the state of the network moves into direction that minimizes the system energy, and that process is similar to gradient search. There, as well, might be local minima on the

The Hopfield network is the basis of other types of networks, such as a Cauchy machine or a Boltzmann machine, for which other learning schemes have been discovered. This network has relatively poor storage capacity, but is widely used as an optimizing tool.

4.7.1 On-center/off-surround architecture

The diagram illustrates a 10x4 grid of neurons, axons, and dendrites. The columns are labeled at the top: "excitatory", "on-center", "inhibitory", and "off-surround". The rows are labeled on the left: "neurons", "axons", and "dendrites". Each neuron is represented by a circle, each axon by a square, and each dendrite by a square. Connections are shown as lines with arrows. The "excitatory" column has a central neuron with a curved arrow labeled "c" pointing to its axon. The "on-center" column has a central neuron with a curved arrow labeled "c" pointing to its axon. The "inhibitory" column has a central neuron with a curved arrow labeled "c" pointing to its axon. The "off-surround" column has a central neuron with a curved arrow labeled "c" pointing to its axon. The "excitatory" column has a central neuron with a curved arrow labeled "c" pointing to its axon. The "on-center" column has a central neuron with a curved arrow labeled "c" pointing to its axon. The "inhibitory" column has a central neuron with a curved arrow labeled "c" pointing to its axon. The "off-surround" column has a central neuron with a curved arrow labeled "c" pointing to its axon.

bors. The interactions have the nature of the, so called, *Mexican hat function*. The function divides the node's surroundings into three circular areas with the node being the center: on-center,

immediately off-surround and further off-surround. All on-center nodes are affected positively by the central cell, and the strength of the influence depends on the distance from it. Off-surround neurons are not affected at all (they are not connected) except for the small area that is adjacent to the on-center, which is affected negatively. When the network is given input signals, each active neuron excites its closest neighbors, and inhibits adjacent off-surround neurons.

FIGURE 20. Activation patterns before (a) and after (b) lateral on-center/off-surround competition. Activation *bubbles* can be recognized after the competition.

4.7.2 Activity bubbles

Such a network, initiated with a random set of nodal activities, clusters the output signals after some time. The clusters, referred to as **activity bubbles** (Figure 20), stabilize with time. By adjusting systems parameters, it is possible to control the number and size of activity bubbles. In the extreme case, there may be only one node in a bubble, called the winner of the competition.

4.7.3 Topology preserving

The input signals to the network are feature vectors; i.e., a complete set of parameters that constitutes a state of the system. Each vector is a point in the state space. The input vector is fed into the network, that is each parameter is input to every node. The weights of the connections from the input nodes form a vector as well with the same dimensionality as the input vectors. Therefore, a

weight vector may be thought of as just another vector in the input space. One of the observed fea-

FIGURE 21. Weight vector image of the network.

FIGURE 22. Illustration of the capability of the Kohonen algorithm to preserve the physical topology of the network.

tures of such a network is that the physical topology of the network is preserved in the weights to which the network maps in the input space (Figure 21). An analogy to the topology preserving capabilities of the brain are very often drawn in this context.

The order of the weight vectors is initially set randomly, but with time, as the nodes recalculate their activities and the connection strengths are adjusted, the vectors tend to organize in a way that preserves the physical topology. That process is illustrated in Figure 22. Additionally, the local point density of weight vectors in input space approximates the distribution of the training samples.

4.7.4 Kohonen Algorithm

Kohonen assumes that there is only one bubble with only one neuron, the **winner**, responding to a given input. The network consists of input neurons, called sensory neurons, and the laterally interconnected neurons, where the **competition** takes place.

FIGURE 23. Kohonen network.

Sensory cells produce real-valued outputs which are input into each neuron in the network. A set of the sensory cells' outputs and each neuron's weight vector are vectors in the input space. Each neuron calculates the distance between its weight vector and the incoming input vector. The distance becomes its activation value, and the output function is an identity of the activation.

$$A(k_j) = || O(S) - \alpha(k_j) || \quad (\text{Eq. 26})$$

$$O(k_j) = A(k_j) \quad (\text{EQ 27})$$

The neuron for which the distance is minimal wins the competition of all neurons in the network. In this sense, the Kohonen neurons act like feature detectors. However, while other feature detecting networks (perceptron-like) learn to classify as instructed by a teacher in a **supervised** way, the Kohonen network belongs to a **self-organizing (unsupervised)**, feature discovering class. Very often, it is called Self-Organizing Map (SOM).

The learning algorithm in Kohonen network follows

1. Initialize all weights to random, real-valued numbers
2. Approximately 500 times the number of neurons repeat the following
 - a. Calculate the activation and output of each neuron
 - a. Select the node with the highest activity, it is the winner of the competition
 - b. For all neurons which are closer to the winner then a certain radius, update their weights accordingly to the following rule

$$w(k_j) = w(k_j) + \gamma(t)(O(S) - w(k_j)) \quad (\text{EQ 28})$$

where $\gamma(t)$ is a function of time which decreases exponentially from 0.9 to 0.1 during the first 1,000 steps, and then linearly from 0.1 to 0 over the next 100,000 steps.

- c. Decrease the update radius by 1, unless it is already equal 1

The Kohonen network has been used in several areas: control, pattern recognition, image processing and many others, where no expert is available to provide a teaching signal and the system must discover all regularities of the input space itself.

4.7.5 Learning Vector Quantization

A variation of the Kohonen network called LVQ (Learning Vector Quantization) is a supervised learning schema. The difference between SOM and LVQ is that in the latter the set of classification nodes is given a priori, while in the former the set is discovered by the network itself. Except for that, the algorithms are basically the same.

4.8 Weightless networks

Weightless networks use a different paradigm for the basic construction block than all other networks that have been analyzed so far in this introduction, i.e., a threshold unit node with its weighted sum of inputs as presented in section 4.1.2 *Neural Networks: An attempt to mimic Nature*. It would be hard to find the origins of weightless networks in Neurobiology, they are rather rooted in Logic.

	x_{k1}	x_{k2}	...	x_{kn}	...	x_{kn}	y_k
x_1	1	0		1		0	0
x_2	0	1		0		1	1
x_3	1	0		0		0	1
x_k	1	0		1		1	0
x_M	1	1		1		0	1

FIGURE 24. A node used by weightless networks uses a truth table to compute, or rather find, the output value.

Any logical function can be expressed by a **truth table** like the one illustrated in Figure 24. For any combination of binary parameters, there is a slot with the value of the function for that combination. A node used in a weightless network is just a **lookup table**. The input vector to the node corresponds to an array of function parameters. It is used as an address in the lookup table that stores the value of the function which corresponds to the output of the node. Searching for an output value in a lookup table is similar to random access memory (RAM), so the nodes using lookup

tables are often referred to as RAMs. One very important consequence of this similarity is that RAMs can be implemented using the same technology as used in the production of memory chips.

Networks that use such nodes are obviously very accurate and fast, because they do not approximate and find the outputs by direct addressing. If all truth tables could be engineered *a priori*, then there would not be anything interesting to add to plain logical analysis. However, due to the large numbers of nodes used in the networks, a learning algorithm is needed to fill all the slots in every node. To achieve that, the tables can use more values than just 0 and 1. A node that uses an extended output alphabet is called a **probabilistic logic node (PLN)** and was introduced in (Alexander, [2]). In the simplest case, another value that is usually annotated as u represents a state that is unknown. If u stands for any y_K in the table in Figure 24, then the output for the corresponding input vector X_K is taken as 0 or 1 with equal probability. A network with such an extension can be trained by replacing all u s with 0 or 1. After all u s are replaced, the method of finding the output value for a given input is deterministic. The alphabet for output values can be larger than $\{0, 1, u\}$ with varying probabilities for replacing unknown values to either 0 or 1. The networks and learning algorithms do not differ substantially from the one presented here.

There have been a number of learning algorithms introduced to train weightless networks. The method proposed by Alexander uses a global teaching signal. The input pattern is presented to the network. There are no restrictions on the topology and connectiveness of the nodes; the network can be feedforward, recurrent, multilayered, etc. The activity is allowed to spread; i.e., at each node, an output function is found. If the output is different to u , then its value is propagated to the target nodes. If the output is u , then either 0 or 1 is propagated; the probability of propagating 0 is the same as the probability of propagating 1 (i.e., sometimes it is 0, and another time it can be 1). When the network saturates, the actual output pattern of the whole network is compared to the desired output. If they match, then for each node outputting u , the value of the function for the node's current input is set to the actual value that was propagated. If the actual output of the network differs from the desired pattern, then the propagation step is repeated a number of times.

That may lead to different behavior, because nodes with a u may propagate a different value each time (0 or 1 have the same probability). If, after a number of repeated propagation steps, the network does not produce the desired output, then the output value for the current input vector in every node is reset to u and the training is repeated for the same input to the network. This procedure ultimately leads to replacing all u s with 0 or 1 yielding the desired deterministic behavior.

An interesting extension to the basic RAM is **generalized RAM** or GRAM. There is a generalization technique that, in case of incomplete information or if the regular operation is mixed with learning, modifies all u s according to the voting by the neighbors. The set of neighbors is determined by the Hamming distance. If most of the neighbors output 0, then u is assumed to be 0, otherwise, 1 is assumed.

4.9 Activity spreading networks

Activity spreading networks look similar to neural networks, in fact, neural networks are also activity spreading networks! The difference is that activity spreading networks are not formally adaptable; i.e., learning is not essential and even if there is some adaptation, there are no standard learning algorithms. The use of activity spreading networks originated in Psychology where they were used to model various aspects of behavior. In this context, the nodes of the network encode various aspects of behavior and stimuli that affect the behavior, and the links - relationships between them. If the current state of the environment is presented to the network as an array of more or less activated stimuli, then the activity spreads along the links throughout the network. In a useful network, the process of activity spreading is finite, so the system settles down with a certain activity pattern. That pattern can be considered a response to the presented stimulus. A number of such cycles determines the behavior of the subject. For example, in Figure 25, a very simple example is presented that involves 4 nodes representing

- two stimuli: hunger and threat and
- two patterns of behavior: looking for food and running away

Hunger positively influences the desire to look for food. A threat influences the desire to run away. The node representing running away has a negative link to the node representing the desire to look for food. The network is utilized by activating the nodes that represent the stimuli. Usu-

FIGURE 25. A simple example of activity spreading network.

ally, if there is no threat and the subject is hungry it will be looking for food. The activity spreads from the node labeled by *hunger* to the node labelled by *look for food*. If activity is only applied to the node labelled by *threat*, meaning the presence of a threat, then the activity is spread to the node labelled *run away*. If both stimuli are present, then the activity in *run away* will suppress the activity in *look for food*. That happens in normal behavior, because under a threat the need to run away is stronger than the desire to eat. However, as it can easily be seen in forests during winter months, some birds that usually would not dare to come close to humans, will even jump on a hand and eat. That is because the stimulus coming from the sense of hunger appears to be higher than the fear of being preyed on. This behavior can be simulated in the network in the example by increasing the activity in the node representing hunger.

Chapter V: Introduction to Markov Models

Markov models, and more generally stochastic processes, originate in statistics and probability theory. They relate to experiments generating series of values. If each value depends on a number of preceding values, then the experiments is related to as a **stochastic process**. If the current result depends only on the latest preceding result, then we have a **Markov process**.

In fact, any system where choices are being made based on the past can be considered a stochastic process. Each choice takes the system from one state to another. There is a simple stochastic process illustrated by the **diagram** in Figure 26 with a system with three states S_1 , S_2 and S_3 . The

FIGURE 26. An example of a Markov chain.

state transitions are illustrated as arrows with the probability of a specific transitions given as a label. The transitions can be recurrent; i.e., the system may stay in the same state. In the example, the probability of state S_1 being succeeded by S_2 is 0.7, while staying in S_1 has the probability of

0.3. Of course, the sum of the probabilities of all transitions from a specific state must be equal to 1.

A Markov process that has a countable or countably infinite number of states is called a **Markov chain**. The process in the example in Figure 26 is a Markov chain.

From a computational point of view, it is more convenient to use an alternate notation for a stochastic process. Such an alternative to a diagram is a **transition matrix**. The following matrix represents the stochastic process illustrated as a diagram in Figure 26:

$$T = \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix} \quad (\text{EQ 29})$$

The matrix T is a direct reflection of the transitions shown in the diagram; for example, the probability of the transition $S_1 \rightarrow S_2$ is $T_{12} = 0.7$, and $S_2 \rightarrow S_3$ - $T_{23} = 0.25$. All elements of the table are non-negative. The sum of the values in each row must be equal to 1. The next state of the system can be computed by multiplying the current state vector by the transition matrix. A **state vector** is a vector $P_i = [p_{i1}, p_{i2}, p_{i3}]$ of probabilities of the system being in the corresponding state. For example, the **initial state vector** can be chosen as $[1, 0, 0]$; it means that the system is in state S_1 with probability equal 1; the probability of being in state S_2 or S_3 is, in this case, 0. The state vector will change to the following in the next step

$$P_0 \cdot T = [1 \ 0 \ 0] \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix} = [0.3 \ 0.7 \ 0] = P_1 \quad (\text{EQ 30})$$

If there is no direct transition between two states, then the corresponding value in the matrix is zero. For example, there is no link from S_1 to S_3 , so there is $T_{13} = 0$. As in the example, there might be an indirect link between two states that do not have a direct one. S_3 can be reached from S_1 in two steps. $S_1 \rightarrow S_2$ and $S_2 \rightarrow S_3$. The probability of that happening is a product of the probabilities of the transition; i.e., $0.7 * 0.25 = 0.175$. In fact, all transitions that could be achieved in

two steps can be computed by multiplying the transition matrix by itself, in other words raising the matrix to power of 2.

$$T^2 = \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.265 & 0.56 & 0.175 \\ 0.2 & 0.55 & 0.25 \\ 0.125 & 0.5 & 0.375 \end{bmatrix} \quad (\text{EQ 31})$$

Similarly, we can compute probabilities of transitions that can be achieved in 2, 3, 4 and more steps by raising the transition matrix to corresponding power as illustrated below

$$T^3 = \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0.3 & 0.7 & 0 \\ 0.25 & 0.5 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.2195 & 0.553 & 0.2275 \\ 0.1975 & 0.54 & 0.2625 \\ 0.1625 & 0.525 & 0.3125 \end{bmatrix} \quad (\text{EQ 32})$$

$$T^4 = \begin{bmatrix} 0.2041 & 0.5439 & 0.252 \\ 0.19425 & 0.5395 & 0.26625 \\ 0.18 & 0.5325 & 0.2875 \end{bmatrix} \quad (\text{EQ 33})$$

There is a theorem that regulates the state dependencies in a Markov chain which says that a system that is in state i will move to state j in k transitions with a probability equal to the element (i, j) of the matrix T^k where T is the transition matrix. The following are the state vectors in consecutive steps of the example:

$$P_0 \cdot T^2 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.265 & 0.2 & 0.125 \\ 0.56 & 0.55 & 0.5 \\ 0.175 & 0.25 & 0.375 \end{bmatrix} = \begin{bmatrix} 0.265 & 0.2 & 0.125 \end{bmatrix} \quad (\text{EQ 34})$$

$$P_0 \cdot T^3 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.2195 & 0.1975 & 0.1625 \\ 0.553 & 0.54 & 0.525 \\ 0.2275 & 0.2625 & 0.3125 \end{bmatrix} = \begin{bmatrix} 0.2195 & 0.1975 & 0.1625 \end{bmatrix} \quad (\text{EQ 35})$$

$$P_0 \cdot T^4 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.2041 & 0.19425 & 0.18 \\ 0.5439 & 0.5395 & 0.5325 \\ 0.252 & 0.26625 & 0.2875 \end{bmatrix} = \begin{bmatrix} 0.2041 & 0.19425 & 0.18 \end{bmatrix} \quad (\text{EQ 36})$$

If we kept computing the power matrices, we would notice that at some point the transition matrices stop changing. In our example it happens starting with the power of 22. All matrices T^k for $k \geq 22$ look the same

$$T^k = \begin{bmatrix} 0.19230769 & 0.53846154 & 0.26923077 \\ 0.19230769 & 0.53846154 & 0.26923077 \\ 0.19230769 & 0.53846154 & 0.26923077 \end{bmatrix} \quad (\text{EQ 37})$$

$$(\text{EQ 38})$$

That means that the system does not change anymore. We can say that the system reaches **equilibrium** after 22 steps. The state vector, called the **steady state vector**, at that point will be as follows:

$$P_{k+1} = P_0 \cdot T^k = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0.19230769 & 0.53846154 & 0.26923077 \\ 0.19230769 & 0.53846154 & 0.26923077 \\ 0.19230769 & 0.53846154 & 0.26923077 \end{bmatrix} = \begin{bmatrix} 0.19230769 & 0.53846154 & 0.26923077 \end{bmatrix} \quad (\text{EQ 39})$$

Note that the same result would be obtained if instead of computing $P_0 \cdot T^k$ we computed $P_1 \cdot T$, $P_2 \cdot T$, $P_3 \cdot T$ and so on, P_i annotates the vector state after i transition. Starting with the step 22, the vector state does not change, so it can be annotated just by P . Similarly, for $k \geq 22$ we can annotate T^k as T .

There is no need to perform all computations to obtain the steady state vector. More complex systems may require even larger numbers of steps to stabilize. Fortunately, there is a theorem that provides a formula for computing the steady state. First, let us notice that $P_{22} \cdot T = P_{23}$, $P_{23} \cdot T = P_{24}$ and so on. On other hand, $P_{22} = P_{23} = P_{24} = \dots = P$. If we unfold the steady state vector P into $P = [p_1, p_2, \dots, p_n]$, then the elements of the steady state vector can be found by solving the following system of equations:

$$P = P \cdot T \quad (\text{EQ 40})$$

$$\sum_{i=1}^n p_i = 1 \quad (\text{EQ 41})$$

Only regular Markov chains have a property of settling down to an equilibrium. A Markov chain is **regular** if it is irreducible and at least one entry on the main diagonal of the transition matrix is nonzero or if some power of the transition matrix has all nonzero entries. A Markov chain is **irreducible** if any state is reachable from any other state.

Higher order Markov models use a number of preceding states to determine the next state. For example, in a second order Markov model, the last two states would be used to compute the next state. The analysis of such systems is more complex and will not be presented in this introduction

Chapter VI: Neurosolver: A neural network based on a cortical column

6.1 Introduction

The cortical column, the “module-concept”, has been proposed as an anatomical entity by Szentagothai in [55]. Later, Mountcastle in [38] and Zeki and Shipp in [68] enriched the hypothesis by describing the functional context of the cortical column. Ballard in [3] was the first to propose the model that clearly identifies three distinct parts of the column: the upper, the intermediate and the lower divisions. Burnod in [14] proposed the concepts of a columnar call and action tree and the use of the trees to explain the functionality of the cortex.

A modified version of Burnod’s model of the cortical column has been used as a processing element of a neural network architecture that I call the Neurosolver. The Neurosolver is capable of computing solutions to problems in well defined domains through traversing trajectories that encode temporal relationships between the objects in the specific domain. The recollection mechanism produces solutions to presented problems in a form of sequences of patterns of network activity. The topographical relationships of the objects of the domain are used explicitly, because the problem of creating such maps has been settled by others, for example Kohonen (Kohonen, [29]).

The motivation for the Neurosolver was the perceived need to bridge the field of neural networks with the ideas well rooted in the classic symbolic Artificial Intelligence, and most notably search.

Naturally, neural networks perform searches, but these are in most cases meaningless internal number crunching exercises. Search is inseparable from problem spaces - the domains where searches take place. Of course, neural networks operate in certain domains as well. For example, n-dimensional weight spaces use some kind of computational search, like hill climbing, but these domains are rarely related to the domain from which the problems that are being solved¹ come. Search and problem spaces are prerequisites for planning, the main area in which the Neurosolver was envisioned to make a contribution.

Cho, Rosenbloom and Dolan have reported their work on Neuro-Soar that has similar objectives to the Neurosolver's (Cho, [17]). Their approach is, however, completely different from the one taken in my work. They attempt to implement elements of the symbolic architecture of the SOAR system (Rosenbloom, [46]; Laird, [31]) using neural methods. SOAR is basically a production system with knowledge distributed over a number of problem spaces. Selection of problem spaces, rules, operators and making other decisions crucial to the proper operation of the system based on the content of a working memory (i.e., the current state) require pattern matchers. A pattern matcher can be implemented efficiently as a neural network, for example, as a backpropagation network (see section 4.3: *Backpropagation network*). In contrast, the Neurosolver is a totally connectionist architecture without any kind of symbolic reasoning. There are, however, many analogies.

Other research that has a similar flavor is that reported by Mرسic-Flogel in [39]. It deals with aspects of planning with weightless neural networks (see section 4.8: *Weightless networks*). Weightless networks do not use adaptable inter-nodal connections which makes them considerably different from the Neurosolver and most of the mainstream neural networks. Their implementations are similar to lookup tables used for representing logical functions. New incarnations

¹ Sometimes, weight spaces and problem spaces can be treated as one. For example, in the Kohonen network, input vectors serve as gravity centers pulling, in the course of the learning process, weight vectors toward the closest center.

of these networks offer some degree of fuzziness introduced by the use of special undetermined values in addition to 0 and 1. These undetermined values are removed during the course of a learning process. There are also several generalization techniques that use similarity based on the Hamming distance to eliminate uncertainty. The appeal of such networks is that they can be implemented in an efficient way using VLSI technology.

In terms of neural networks, the Neurosolver belongs to a broad category of unsupervised networks (see Chapter IV: *Introduction to Neural Networks*). The fact that connections between nodes are created dynamically as needed (i.e., when a relationship is discovered) is unique. There are certain networks that use genetic algorithms to evolve, but those techniques are vastly different from the ones presented here¹. Recurrent connections are allowed. Taking into account that the nodes represent high level concepts, the Neurosolver is similar to a semantic network with uniform inter-nodal relationships, namely chronological succession. The networks that have meanings assigned to the nodes are more often referred to as connectionist rather than neural networks.

6.2 General Architecture

The Neurosolver is a network of interconnected nodes, artificial cortical columns (biological cortical columns are described in APPENDIX B: *Biological roots of the Neurosolver: The cortical column*). It comprises two subnetworks, search and execution, that will be described shortly. The internodal connections are dynamic, i.e., a connection is created only if a relationship between two nodes is discovered. Each connection has a dual nature; i.e., in fact it consists of two links in two opposite directions: forward and backward. Any two nodes can be connected.

¹ Genetic algorithms may be used to evolve a population of networks with some initial architectures. They use some kind of a fitness function with which the performance of the offspring networks can be evaluated. The networks that perform the best are used to create the next generation using mutation, crossover and other operations. Various aspects of networks can be evolved depending on details of the evolution process and fitness function.

For the Neurosolver to be functional, each of its nodes must be associated with a state of the system with which the Neurosolver interacts. For the experiments performed in the framework of my research, that association was pre-wired, because my research is focused on other aspects of the Neurosolver. However, as mentioned before, I have experimentally confirmed that any of the existing associative algorithms could be used in the general case, e.g., LVQ (Kohonen, [29]) or backpropagation (Rumelhart et al., [47])

FIGURE 27. A part of the Neurosolver trained for the blocks world with some of the associations (gray lines) and internodal links (black arrows). See the text for the explanation.

A part of the Neurosolver trained for the blocks world is shown in Figure 27. Some of the association links between nodes and block arrangements are indicated by gray lines. Some of the inter-nodal connections of node N_1 are also shown. Note, that in this particular example, each of the block arrangements associated with N_2 , N_3 and N_4 can be obtained from the block arrangement associated with N_1 by moving just one block. These arrangements are neighbors in the state space of the blocks world, because in the course of training, there were samples presented that corresponded to transitions $N_1 \rightarrow N_2$, $N_1 \rightarrow N_3$ and $N_1 \rightarrow N_4$. These relationships have been captured in the internodal links. Note that the dual nature of the connections is neglected in the figure.

Note further that to keep the figure legible neither all associations nor all internodal links for this part of the Neurosolver have been shown

FIGURE 28. A solution path to a problem of rearranging blocks generated by the Neurosolver. See the text for the explanation.

While providing different functionality, the search and execution networks cooperate in the resolution of problems. A problem (to solve) is presented to the Neurosolver by two signals: the **goal** - the **limbic input**¹ to the node associated with the desired state - and the **sensory** - the **thalamic**² - **input** to the node associated with the current state. Every node is associated with a state in the problem space, hence, a solution to the problem is a succession of firing nodes starting with the

1 - The limbic system of the human brain is believed to be the source of wishes and desires, that is the origin of the name for the limbic input.

2 - The thalamus is a relay system between the environment perceived by the sensors and the brain.

current node and ending with the goal node. This corresponds to the succession of states that lead from the current state to the desired state.

An example of a solution to a problem of rearranging blocks from the current configuration to the desired configuration generated by an instance of the Neurosolver trained for the blocks world is shown in Figure 28. The Neurosolver produces a solution path by firing a sequence of nodes $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4$. N_1 is associated with the current state, N_4 - with the desired state, and N_2 and N_3 - with the intermediate states. Node N_5 and N_6 are two sample nodes associated with other block configurations. Note, that depending on the training there may be many possible solutions to a given problem, since multiple paths between the current node and the goal node may exist.

FIGURE 29. An inquiry into the future course of events requires the attention of all nodes. The current state activity is set to force its firing. See the text for the details.

There is another type of tasks that can be carried out by the Neurosolver it can play the role of predictor or decision maker. The operation of the Neurosolver performing such activities is equivalent to building action trees that were introduced by Burnod ([14]). In any state, the network can be asked to predict the most likely succession of further states. The Neurosolver utilizes the same traces of state transitions as in the problem solving case, but the technique to come up with a prediction or advice is different. As before, the current state is indicated by providing the thalamic input to the corresponding node. All other nodes are put into anticipatory state by setting their limbic inputs at a non-firing level. Initially, all of the nodes are activated to the same degree. The meaning of that is that all states are equally probable successors to the current state. This process is illustrated in Figure 29. The figure includes some of the block configurations and association links with the corresponding nodes. In fact, all nodes have such associations, but showing them all would make the figure undecipherable. The activity that is injected into the node corresponding to the current configuration is such that it forces the node to fire. The firing activity is sent to other nodes along learned trajectories. Eventually, as the driving activity grows, another node will fire. That node is the successor of the current configuration, i.e., the most probable next configuration of the blocks. When the second node fires, similar course of action fires its successor in turn. The process is terminated when the activity in the network is too low to fire another node. An example of a prediction generated in this way is shown in Figure 30. The process of generating a prediction drawn from the node N_1 is captured at the moment when node N_4 fires. Nodes N_1 , N_2 and N_3 have already fired and are, therefore, inhibited. That is indicated by the white background which means that they are not active. The illustration includes two more nodes that would fire next: N_5 and N_6 .

Before going into the details of the functionality of the Neurosolver, let us describe the architecture of the node first. It will make it easier to explain how two composite networks of the Neurosolver are built and how they work.

FIGURE 30. A prediction of a sequence of block configurations drawn from the node N_1 . See the text for the explanation.

6.2.1 The Node

The node used in the Neurosolver is based on a biological cortical column (see APPENDIX B *Biological roots of the Neurosolver: The cortical column*, the references to the relevant neurobiological literature can be found in (Bieszczad, [8])) It consists of two divisions

- upper division and
- lower division,

as illustrated in Figure 31. The **upper division** is a device integrating internal signals from other upper divisions and from the control center providing the limbic input, i.e. a goal or, using more psychological terms, a drive or a wish. The activity of the upper division is transmitted to the

lower division where it is subsequently integrated with signals from other lower divisions and the thalamic input

The connectivity between the nodes is inspired biologically, but to increase the efficiency by limiting the number of objects to process, the internodal connections are constructed dynamically during the learning process, so only the connections with a non-zero strength are present.

FIGURE 31. Architecture of the Neurosolver

The input function of the upper division is a sum of the products of the action potentials of all afferent connections and their strengths and the limbic input. The input function to the lower division similarly integrates the incoming action potentials with the thalamic input. The lower division uses a threshold function (see Chapter IV, *Introduction to Neural Networks*) as its output function, while the upper division uses an identity function (i.e., the output is the same as the activity). If the activity of a lower division is lower than its output threshold, the high activity threshold, no action potential is generated in the efferent¹ connections. The only exception is the connection between the upper and lower divisions of the same node. The strength of that connection is 1, but the upper division alone cannot activate the lower division at a level higher than the low activity threshold. This ensures that the activity of the upper division alone cannot fire the node. If the lower division receives signals from other sources as well, then the whole signal from

¹ This term is common in Neurobiology. It indicates an outgoing flow of signals.

the upper division is integrated with the rest of the input activity. It is possible to fire a node (i.e., to activate the lower division at the level higher than the high activity threshold) with a minimal forward input activity from other lower divisions and a complementary signal from the upper division.

The output of the lower division of the node is that node's output. In the simplest case, the output is used to append the corresponding state to the resolution path. It might be used to control manipulators that would modify the system. For instance, in the blocks world example, output signals can be used to guide the rearrangement of blocks from one state to the next in the solution path.

6.2.2 Search network to the level exceeding the firing threshold

The upper divisions of the nodes and the connections between them constitute the first of the two networks mentioned earlier (Figure 31). The function of this network is to spread the activity along the connections, starting at the node associated with the goal state that receives the limbic input, the original source of activity. This is a **search network**, because the activity is spread in a hope that at some node it will be integrated to the level exceeding the firing threshold with the activity of the lower division receiving the thalamic input. At that moment, a path from the goal to the current state has been found. This is one of possibly many solution paths. A tree created by the pattern of spreading activity is called a search tree.

Figure 32 illustrates a search tree for a blocks rearrangement problem. The Neurosolver trained for the blocks world generates a search tree in response to the stimulus (thalamic input) into the goal state (node N_4). The connections that are shown indicate that in the training trajectories $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4$ and $N_6 \rightarrow N_5 \rightarrow N_4$ were presented as samples, i.e., the Neurosolver was told that N_4 can be reached from N_1 through N_2 and N_3 , or from N_6 through N_5 . The activity is spread from N_4 to both N_1 , along the subtree $N_4 \rightarrow N_3 \rightarrow N_2 \rightarrow N_1$, and N_6 , along the subtree $N_4 \rightarrow N_5 \rightarrow N_6$.

In this example, N_1 is associated with the current state, so the solution to the problem will be a path which is a reverse of the subtree $N_4 \rightarrow N_3 \rightarrow N_2 \rightarrow N_1$, i.e., a sequence $N_1 \rightarrow N_2 \rightarrow N_3 \rightarrow N_4$.

FIGURE 32. A search tree for a blocks rearrangement problem rooted in node N_4 . See the text for explanation.

6.2.3 Execution network

The second network called an **execution network** is composed of the lower divisions of the nodes and their connections (refer to Figure 31). Its purpose is to generate output signals along the solution path by firing successive nodes as illustrated in the example in Figure 28 or to generate a prediction or a course of action as illustrated in Figure 30. Only one node is allowed to fire at any time, all other nodes are suppressed. A node that fires sends action potentials through the outgoing connections of its lower division. These signals may cause another node to fire, if the expectation - attention - of that node (i.e., the activity in the upper division) is high enough. In a way,

owing to the **winner-takes-all policy**, the process of selecting the successor in the resolution path is a form of selecting the node that is the most activated

After firing, the node is inhibited for some time, in order that immediate oscillations are avoided. Inhibiting the node means that it is prevented from firing. The node still functions as usual with respect to activity transmission, because it may be a relay station for important activity links. Shutting down the node completely might cut off certain critical clues. It is still possible that the Neurosolver loops in larger oscillating circles. In such case, a counter of the firings that is maintained for every node is used to determine whether any of the nodes is involved in a cycle. If the limit is exceeded, a special recovery procedure is used that may allow progress without cycles. The description of that procedure is postponed until sufficient details are presented to make it easier to explain.

6.3 Learning state transitions

In the Neurosolver, learning has a probabilistic nature. Teaching samples are trajectories in the state space. They are temporal patterns, as they represent the state of the system over a period of time. In the blocks world example, a teaching sample is a sequence of block configurations. Examples are presented by consecutively providing thalamic input to the nodes corresponding to the states in the training pattern. During learning, thalamic signals are set to be strong enough to fire a node, i.e., the value is higher than the output threshold of the lower division of the node. As mentioned before, the number of firings, F , is recorded in each node to be used later to detect cycles. If, for any node, F exceeds certain threshold, then the Neurosolver assumes that the node is involved in an oscillating cycle of firing nodes. In such case, the search is abandoned.

For each state transition, two connections are strengthened: one, in the direction of the transition, between the lower divisions of the two nodes, and another, in the opposite direction, between the upper divisions (Figure 33). The connection between the upper and lower divisions of the same node is fixed. The number of transmissions of an action potential, T_{out} , is recorded for each con-

nection. The total number of cases when the division positively influenced other nodes, S_{out} , is collected for each division. A positive influence means that a firing node sent an action potential to the node that fired next.

FIGURE 33. Adaptation in the Neurosolver. If N_2 fires just after N_1 fired, then the connection from the upper division of N_1 and to the upper division of N_2 is strengthened. At the same time, the connection from the lower division of N_1 to the lower division of N_2 is strengthened as

Using the collected statistics, the Neurosolver determines the strength of each connection. The Neurosolver provides a choice of three ways to compute the strength. The first formula distributes the activity of the node evenly among the connections that conducted action potentials at least once during the training.

$$p = \frac{1}{S_{out}} \quad (\text{EQ 42})$$

This formula does not take into account the frequency with which the connection was transmitting action potentials relative to other connections. Therefore, there is no preference given to any outgoing link. In the second variant, the strength of the connection is taken as the probability that the connection will transmit action potential if the node fires.

$$p = \frac{I_{out}}{S_{out}} \quad (\text{EQ 43})$$

In this case, a connection that transmitted action potentials more often than another during the training is stronger. Both formulas, (EQ 42) and (EQ 43), use the statistics concerning the outgo-

ing signals from a node. For the third variant of the connection strength, we need two more pieces of statistical data that relate to incoming signals T_{in} - the number of times when the connection contributed to an activity of the node and S_{in} - the total number of times when other nodes positively influenced the node. With the new data, we can calculate the probability that an action potential incoming from a connection will fire the target as

$$p_{in} = \frac{T_{in}}{S_{in}} \quad (EQ 44)$$

The formula that is used for computing the strength of a connection in the third variant is a product of the outgoing likelihood of inducing an action potential in the connection by a firing source node and the incoming likelihood of firing the target node by a signal coming from the connection

$$p = p_{out} \cdot p_{in} = \frac{I_{out}}{S_{out}} \cdot \frac{T_{in}}{S_{in}} \quad (EQ 45)$$

The type of the connection has profound consequences on the behavior of the Neurosolver. The first variant provides the best chance for finding the shortest solution, since with it the Neurosolver equally explores all alternatives. With the other variants, the Neurosolver constructs preferred search paths that may block other paths and results in failure. If maximizing the probability of succession is desired, then the best path in that sense is found. The details of this approach will be discussed later in the context of conducted experiments.

In the course of the adaptation process, in the response to a presentation of a sequence of firing nodes (i.e., a temporal pattern) two chains are created: a **forward chain** between the nodes from the node that fired the earliest to the node that fired the latest and a **backward chain** between the same nodes but in the opposite direction (Figure 34). Such chains are memory traces of the samples presented during the learning. The backward chains are used to perform searches; they constitute the search subnetwork of the Neurosolver. The forward chains constitute the execution subnetwork of the Neurosolver; they are used to find the next node to fire in resolution mode. Further details are presented in the following sections.

FIGURE 34. Two learned chains: backward search and forward execution.

6.4 Solving problems

I have already mentioned that in order to solve a problem, the Neurosolver requires two signals: a limbic input to the node associated with the desired state indicating the goal state and a thalamic input to the node associated with the current node. The limbic input activates the upper division of the node. The node is activated, but it cannot fire, because, as explained before, in spite of being transmitted to the lower division, the activity of the upper division alone cannot activate the lower division at a level higher than the low activity threshold. That activity spreads along the backward chains originating in this node. The learned trajectories lead in directions that are opposite to the state transitions that were presented in the learning samples (Figure 35). Due to the resistance present in the connections involved in the process of spreading the activity from the source, the nodes in the search tree will usually be activated at various levels. For example, if node N_1 receives a limbic signal of value V and the strength of the connection to N_2 is 0.25, and the strength of the connection to N_3 is 0.6, then N_2 will be activated at $0.25 \cdot V$, and N_3 - at $0.6 \cdot V$; this assumes that neither of N_1 and N_2 receives any input signals from other nodes that are active.

FIGURE 35. Search along the backward chain (gray indicates search level of activity). The top view shows a search tree with many branches; the cross-section is just for one the branches.

FIGURE 36. Triggering the solution (black indicates firing level of activity).

FIGURE 37. Next step in the solution path. Note, that the node that has fired is shut down; i.e., it is not active anymore. When the node corresponding to the goal fires, it is shut down as well. In that way, the source of the search activity disappears: the goal has been achieved.

The signals are clamped onto the inputs, so the limbic input is the source of the stable activity in the search tree until the root is shut down after being fired. Shutting down a node after firing involves cleaning up the input signals, so the source of the activity in the tree is gone. The goal node will fire only if a sequence of nodes between the node associated with the current state and the goal node fire in a chain reaction¹. Firing of the goal node (i.e., the node that is receiving the limbic input) is equivalent to the goal being satisfied - the problem being solved. The chain reaction of firing nodes between the node associated with the current state and the node associated with the goal state can be **triggered** at the former when its lower division gets sufficient activity from the upper division and the thalamic input exceeds the high activity threshold of the node (i.e., the output threshold of the lower division) (Figure 36). The next node in the solution sequence fires, providing that the activity in the upper division and the input from the lower division of the node that fired immediately before also sum up to a level exceeding the high activity threshold (Figure 37).

Due to the possibility of cyclic interconnections, the search process is not guaranteed to terminate (can oscillate indefinitely) if no solution path exists. A hard cut-off is imposed on the number of simulation time ticks. In that way, the process that takes too long will be stopped, as no solution can be found. The system responds with a failure. Before the cut-off limit is used, however, there are a number of steps taken that increase the chance of success. Firstly, the limbic input of the goal node is gradually raised up to a certain hard imposed limit, i.e., a stronger desire (drive) to resolve the problem is introduced. In that way, the overall activity of the search tree is increased. As a final resort, the inhibited nodes are disinhibited (simulating the passing of time). Only if all of these techniques fail, the Neurosolver quits indicating a failure. If a node fires, the drive is reset back to its initial value. The following pseudo-code illustrates that process.

1. The node can also fire if a sufficiently strong thalamic signal is received. That could only happen if the subject system is dynamic during search. Such a capability can be useful. For example, certain search trees can be kept activated in an anticipatory mode. If the system enters any of the states involved in the tree, then it may trigger a certain course of action.

```

procedure solve(currentNode, goalNode);
begin
    while networkActive do
        begin
            while networkActive and not goalAchieved and not cycleDetected do
                propagate(currentNode, goalNode);
            networkActive := false;
            if driveLevel > UpperBoundOnDriveLevel then
                begin
                    driveLevel := enforce(driveLevel);
                    setLimbicInput(goalNode, driveLevel);
                    networkActive := true;
                end else
                    if exists(inhibitedNodes) then
                        begin
                            removeOldest(inhibitedNodes);
                            networkActive := true;
                        end
                    end
                end
            /* failure recovery algorithm */
            if (not goalAchieved and exists(partialSolution)) then
                begin
                    newCurrent := secondFrom(partialSolution);
                    if exists(newCurrent) then
                        solve(newCurrent, goal);
                    end
                end
            end
        end

```

In our experiments that will be presented in section 6.7 *Experiments*, we noticed that the procedure represented by the pseudocode triggers a solution to every problem in the blocks world domain. However, certain requests served by multiple solvers may end up with paths including cycles. Under such conditions problem resolution cannot proceed. A special recovery procedure has been implemented that yielded considerable improvements. The idea is to take the second node in the generated solution that includes cycles and make it the current state. That corresponds

to forcing the system into the next step and starting over with the new current-state/goal-state pair. A schematic version of the code that is utilized by the recovery procedure is included at the end of the `solve` procedure. The recovery process is somehow similar to the sensory guided manipulations performed by biological cortical networks that are discussed in section B.5 *Properties of a network of cortical columns*. The brain seems to initiate certain actions that can be continued only after the sensory system reports that the expected results of the action have been achieved, i.e., those results constitute the new current state of the environment. Without the new sensory input, there would not be enough activity to fire the next column.

The next problem that we noticed was the prohibitive execution time when increasing the drive gradually by adding a small constant called activity delta (e.g., 0.1)¹. Simple paths, those that do not involve too many branches, can be found quickly, but certain tasks were taking too long a time even if the activity delta is set to a relatively high value. An option to accelerate the search has been added as a remedy. In the code presented for the `solve` procedure, that is encapsulated in the call to the `continue` subroutine. Instead of increasing the drive gradually, a function is used that accelerates the increase in the value of the drive, e.g., the value of the drive can be increased by multiplying it by a constant greater than 1. In the simulator, that value is obtained by adding the activity delta to 1.

$$\text{drive} = \text{drive} * (1 + \text{activityDelta}) \quad (\text{Eq. 46})$$

where activityDelta might be, for example, 0.1, 3, 10, etc. The use of a polynomial rather than linear function to increase the drive leads to a tremendous improvement in the time required to solve problems. Unfortunately, the fine granularity of the search is sacrificed, which may lead to multiple firings. The winner-takes-all algorithm takes care of such cases. Of course, in any hardware implementation such a mechanism would not be needed, since the analog signal can be increased quickly with almost immediate propagation.

¹ This process simulates an increasing analog signal.

FIGURE 38. A simple neurosolver with four nodes for the signalling example illustrated in Figure 39. Each connection is labelled with its strength. Note that each node consists of two divisions and a link of strength 1 from the upper to the lower division.

The graphs in Figure 39 provide an explanation of the nature of the ideal activity-spreading-based search in the Neurosolver. The charts illustrate the search for a solution to a problem $N_1 \rightarrow N_4$ in the simple instance of the Neurosolver shown in Figure 38.

There are four graphs for each node illustrating changes over time to

- **U** activity of the upper division of the node
- **L** activity of the lower division of the node and
- **N** output from the lower division of the node (i.e. the output from the node)

The search starts at the node N_1 , the goal, with the application of the drive signal to the limbic input. In theory, we should start with a signal of zero and raise it gradually later. That is illustrated in the graphs. In the simulator, a certain initial value for the drive speeds up the search considerably. That signal activates the upper division of N_1 . The current state is N_1 , so another input sig

2

P4-1 3½" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

nal is applied to the thalamic input of the node N_4 . That signal activates the lower division of N_4 at the low activity level. Neither of these two activities can cause either node to fire. The activity

FIGURE 39. Illustration of propagation of activity within the four-node neurosolver from Figure 38. U stands for the value of the activity in the upper division, L - in the lower division, N - for the output of the lower division (indicates the firing node). The θ symbol is used for marking the threshold values: the high activity threshold θ_H and the low activity threshold θ_L . See the text for explanation.

of the upper division of N_1 is increased by growing the level of the drive (controlled externally). The activity is propagated to the lower division of N_1 and to the upper divisions of N_2 and N_3 . I have tried to use thresholds in the upper divisions, but I have found out that they were not neces-

sary. Nevertheless, they may speed up the search, since initially fewer nodes are involved in the propagation of activity. Removing the thresholds simplified the Neurosolver without sacrificing any of its capabilities. The lower division receives the complete signal, because the strength of the fixed connection between the upper and lower divisions is 1. The upper division of node N_2 receives two thirds of the activity of the upper division of N_1 , while N_3 - one third (due to the connection strengths indicated in Figure 38). Again, none of the nodes can fire. The activity from the upper divisions of N_2 and N_3 is sent to N_4 in addition to being propagated down to the respective lower divisions. The growth of the activity of the upper division of N_4 is on par with the activity growth in the upper division of N_1 , because both N_2 and N_3 send their output signals to N_4 . Since both connections have the strength value of 1 as indicated in Figure 38, the signals are integrated yielding the original signal that was sent from N_1 . The lower division of N_4 starts to receive the input from its upper division. That input is integrated with the activity coming from the thalamic input and at the time t_1 the high activity threshold of N_4 is exceeded. The node fires, the lower division sends the signals to the thalamic inputs of N_2 and N_3 and the node is shut down. The signals that are sent from the firing node imitate the input signal to the successive nodes and their values are set at the same level as the original thalamic signal to N_4 ¹. At the same time, t_1 , the activity in the lower division of N_1 saturates, since the activity of a lower division cannot grow more than the low activity level if only a signal from the upper division is being received. The strength of the connection from the lower division of N_4 to the lower division of N_2 is one third, while the strength of the connection to N_3 is two thirds as indicated in Figure 38. There is a visible jump in the activities of the lower divisions of N_2 and N_3 at time t_1 . Neither N_3 nor N_2 can fire immediately however, so the original search drive is allowed to grow again. N_2 and N_3 receive their share (two thirds and one third, respectively) of that growth through the incoming connections to their upper divisions and through the downward connections between the upper and lower

¹ It is a shortcut for an interaction with the environment. The firing of a node may initiate an action that will alter the environment. That change can be perceived by the sensory system and provided as the next current state to the Neurosolver.

divisions. At time t_2 , the activity of the lower division of N_2 exceeds the high activity threshold. The node fires, shuts down and sends a signal to the thalamic input of N_1 . In this case, there is sufficient surge in the activity of the lower division of N_1 to fire that node as well. After the firing, N_1 is shut down, so the source for the search activity disappears. Therefore, the activity in N_1 goes to zero as well.

In a larger search tree, firing a node could shut down a whole branch, a subtree, or, if the goal node fired, the complete tree.

Even though the graphs indicate that N_2 and N_1 fired at the same time t_2 , there is a natural delay that causes N_2 to fire first, and N_1 next. Such a delay is not restricted to the software implementation of the Neurosolver. It would also be present in an analog implementation in hardware.

The analysis of the graphs presented, lets us relatively easily extrapolate the ideas to the proof that the Neurosolver is able to find a solution to any problem if a solution exists. It is only a matter of increasing the drive to a level that activates appropriate branch containing the trigger of a solution. The only part that might not be clear, and is not directly deducible from the graphs, is activity spreading in the presence of cyclic connection paths. The graphs for more nodes become very complex, so I will present just a description of that process. Figure 40 shows a search tree of a

FIGURE 40. Spreading search activity in a cyclic search tree.

problem $N_5 \rightarrow N_1$ that includes a cycle $N_2-N_3-N_4$. The activity is fed from N_1 to N_2 to N_3 , and from there to N_4 and N_5 . The activity of N_3 is propagated back to N_2 in the cycle. Even though the

activity in N_2 may be increased, that changes nothing as far as the search is concerned. In spite of the cycle, the search activity, albeit weaker than if the cycle was not present, is propagated further to N_5 . In this example, N_5 also receives the thalamic signal, but it could have been just a root of a larger subtree. The signals in the upper and lower divisions of N_5 are integrated and the resolution process is initiated as illustrated in Figure 41. The next node to fire after N_5 is N_3 . After firing, N_3 is inhibited, so search activity is cut off from N_4 . N_2 receives the propagation signal and fires next. Although after firing N_2 sends signals to both N_1 and N_3 , only N_1 is activated; the source of the activity in the upper division of N_4 disappeared with the inhibition of N_3 after firing. This exact situation is captured in Figure 41.

FIGURE 41. An example of firing nodes along a resolution path in a search tree containing a cycle. The nodes N_5 and N_3 and the propagation links are grayed out to indicate that they are past occurrence.

It is worth noting the importance of spreading search activity in guiding the resolution toward the best possible solution path. It could appear that when the trigger fires then the drive to fire along the forward chain could be increased originating in the firing node. That, and the search drive from the direction of the goal node, would resemble a bidirectional search. Unfortunately, such an approach does not yield good results, because the succession probabilities that are independent of the location of the goal come to play and the solution may wander in undesirable directions. Increasing the search activity further, as shown in the graphs of Figure 39 (time slot t_1 - t_2 for N_2

and N_3), produces much better results. It also allows for better Neurosolver integrity, because it does not require differentiation between the search and execution modes

Activity spreading networks (see section 4.9: *Activity spreading networks*) have been used by others, e.g., (Maes, [34]), but I believe that the way I apply the ideas in the context of problem solving is novel

The low and high activity thresholds, the initial strengths of the search drive, the limit for the strength of the drive, the progress step, the length of the inhibition period, the number of allowed cycles and the acceleration flag are parameters that can be used to control the quality, speed and granularity of the resolution in the software simulation of the Neurosolver

6.5 Predicting and decision support

The prediction and decision support capabilities of the Neurosolver are analyzed under the same heading, because they are basically the same activity. Prediction applies to cases where the observed system has a nondeterministic nature that is difficult or impossible to control, for example, a stock exchange or weather. The future behavior of such systems may be predicted by the Neurosolver. Decision support is a similar activity applied to a system that can be controlled, for example, playing games, legal systems or automatic fault recovery systems. Prediction usually involves the behavior of the system in some future time window. Making a decision may involve only one step ahead, as in playing games, or more steps, as in legal systems

In contrast to the goal-oriented tasks analyzed earlier, the end effect of resolving the tasks discussed in this section is not known *a priori*. The overall objective is to provide the best course of action or the best prediction of future events. That decision is based on experience and may be influenced by additional information about any relevant conditions. Let us leave considerations of additional information for later and concentrate now on the predictive properties of the Neurosolver. Figure 42, Figure 43 and Figure 44 illustrate the production of an action

FIGURE 42. Action along the forward chain (gray indicates anticipatory activity in the upper columns) triggered by firing the current state node.

FIGURE 43. Another step in the firing chain reaction. Note, that the nodes that fire are inhibited. That prevents short (immediate in this example) cycles. The inhibited node cannot fire even if its lower division is activated as illustrated.

FIGURE 44. Next step in the action path. In this illustration, the node that fired earlier is active again, while the node that fired lately is inhibited.

The activity of upper divisions of all nodes but the node corresponding to the current state is set at some anticipatory level. Activity in the upper division cannot fire a node without contribution

from the lower division. The triggering signal comes from the current node that receives a thalamic input that activates the lower division at a level exceeding the high activity threshold. The first node is, therefore, forced to fire. It is the current state, so it is acceptable as a trigger for a prediction sequence. Activity is propagated from the lower division to other lower divisions along the paths learned during training. If the activity in one of the nodes receiving propagated signals and the anticipatory activity in the upper division sum up to a value exceeding the output threshold, then the next node fires. If none of the nodes can fire, then the anticipatory signal is increased gradually raising the anticipatory activities in all nodes. At some point one of the nodes that have received propagation signals from the firing node will fire. The node that already fired is inhibited, in order that it cannot fire again. However, since the inhibition is limited in time, it is theoretically possible to repeat firing nodes in a cycle. There is a mechanism that detects such cases, although in the blocks world domain it does not happen.

The degree of anticipation of the nodes may vary (activity at various levels). It can be the result of additional sensory links to the nodes that could provide information with finer granularity than just *on* or *off* that is used for the purpose of my work. In that way, selection of the successors could be controlled in a refined way. For example, in a legal system, the nodes that are applicable to a given case would be activated, perhaps even at various levels. The Neurosolver would consider only the cases that are relevant, disregarding any others that may have been influenced by the current case in the past, but it is not applicable in the current situation.

The following simple pseudocode illustrates the high level view of how the prediction capability is achieved in the Neurosolver. The activity is allowed to propagate until there is no further propagation or a cycle is detected. Cycles are possible if the inhibition of firing nodes is limited in time. To prevent the process from looping forever, the maximal length of an advice `maxAdviceLength` is used to terminate such behavior. I have discovered that in practice such cases are rare, but they do occur. The value of `cycleLength` is set inside the `propagate` routine.

```

procedure advise(currentNode, maxCycleLength);
begin
    while networkActive and not cycleDetected do
        begin
            setLimbicInput(allNodes, searchDriveLevel);
            setLimbicInput(currentNode, 0);
            setThalamicInput(currentNode, firingDriveLevel);

            while networkActive and (cycleLength < maxCycleLength) do
                propagate(currentNode, maxAdviseLength);

            if (searchDriveLevel < DriveLevelLimit) then
                begin
                    searchDriveLevel := enforce(searchDriveLevel);
                    networkActive := true;
                end
            end
        end
end

```

6.6 Implementation

Neurosolver was conceived as an analog device. The idea of implementing it in hardware is still alive, but software simulation had to be used to test the concepts. This section describes a software implementation of the Neurosolver. From the graphs presented in Figure 39, it is easy to imagine how the hardware implementation would perform searches. It is less clear how the learning could proceed. I will return to this question later when I discuss possible future directions for the research.

6.6.1 Object-Oriented Design

Neurosolver has been designed as an object-oriented system. Major classes are Neurosolver, Node, Division and Connection. Division is an abstract class with two derived subclasses, UpperDivision and LowerDivision. In general, Neurosolver is a collection of nodes, but it also encompasses other data required to manage the network. Similarly, each node consists of two divisions,

but there are more data members needed for a complete, working implementation, for example, a counter *F* is maintained to keep the number of firings of the node. The connections are established between divisions. The *Connection* and *Division* classes implement the management of statistical data needed for computing the strength between the nodes of the Neurosolver.

6.6.2 Coding

Neurosolver was originally implemented in Smalltalk, together with the blocks world workbench including the blocks world, testing facilities and a graphical user interface. Smalltalk was chosen because of the flexibility which it provides during coding. Owing to the excellent integrated development environment inherent to almost all Smalltalk systems, in particular very rich collection of classes, class browser and the incremental compilation, ideas that otherwise may take too long to try are quickly encoded and tested. In that way, the throughput of ideas is considerably higher than when using other languages. Additionally, a graphical user interface was built using the incorporated GUI builder. That work was done, because the results could be produced quickly. Building a user interface is, in general, a tedious task that usually does not yield significant advantage to warrant it being a mainstream activity. Using Smalltalk's GUI builder brought the comfort of having a nice UI without a huge time penalty for implementing it. In my opinion, Smalltalk-like environments provide an ideal environment for pursuing any type of research involving programming.

There is one drawback with the use of Smalltalk for certain problems. It appears that resolving a single problem even in a neurosolver with a considerable amount of knowledge (many sequential patterns recorded) takes from almost no time at all to a couple of seconds. However, if an experiment requires that all possible problems in the domain must be considered, then such a test may take a couple of hours. For example, for 60 states, it takes $3540 \cdot 60^2 \cdot 60$ trials to go through all possible state transitions (without identity transitions). The most critical aspect of the speed of the workbench is the efficiency of the Neurosolver's algorithm. Consequently, it has been ported to C++ in the hope that processing can be sped up. The new implementation proved to be much

faster. The time required to perform the tests on all possible transitions has been reduced from 7-8 hours to a couple of minutes (on a PC with 486/66MHz Intel processor).

An unexpected result of the porting to C++ was the improvement in the Neurosolver's problem solving capabilities. Certain ideas came to my mind in the process of porting. When implemented this led to a 100% success ratio in solving the problems in the blocks world domain by a single Neurosolver. One of the enhancements was the recovery procedure from detected cycles that was described earlier in this section. Another idea that worked very well was the addition of a simplified rule for the strength of connections. (originally, only a more complex formula was used that involved the outgoing as well as the incoming probabilities). Additionally, the decision making and prediction capability was added with very little actual programming.

6.6.3 Application Program Interface (API)

The Neurosolver implemented in C++ has an API that can be used by other programs to interface to the solver. The API can also be used to interact with the Neurosolver from a console.

The following is an overview of the available API commands. A prompt consisting of => is used to indicate the readiness to accept commands.

```
=> new <identifier> [<size> <base> | dynamic]
```

Generates a new instance of the Neurosolver. If **dynamic** is used, then the nodes will be added as required. Otherwise, a neurosolver with pre-selected size is created. <base> indicates the lowest identifier for the nodes.

```
=> id [<identifier> | all]
```

Selects a neurosolver to use with the following commands.

If **all** is given instead of an identifier, then the commands will be directed to all neurosolvers that have been created.

```
=> remove
```

Removes the selected neurosolver or all neurosolvers if **all** selected (with

the **id** command).

```
=> learn ( [random] | default | <file name> | | [flow] | <source> | <target> |
```

Instructs the neurosolver to learn. The learning can be random or not from a **default** set of samples or from a designated file. If random learning is selected, then each state transition from the selected training set is learned a random number of times¹. Without the **random** keyword, each transition is learned only once.

Alternatively, only a single state transition from <source> to <target> can be learned once. Of course, the learn command can be repeated many times for the same transition.

If **dynamic** creation is selected, then a node is added to the Neurosolver, if it is a part of the learning transition and it has not been created yet. If **flow** is used, then every transition between the <source> and <target> is learned. This option assumes incremental ordering of the nodes by a numerical identifier.

```
=> solve [hard] ( all | <current> | <goal> |
```

Instructs the neurosolver to resolve a problem. The request can be to solve **all** problems. The system generates problems for itself by taking all possible pairs of two different states.

Alternatively, only a single problem to find a path between the <current> state and the <goal> state can be requested.

If the **hard** keyword is used, then the neurosolver applies the failure recovery algorithm.

If the cooperative mode is on (described later), then the cooperative solving algorithm is used that employs all associated neurosolvers.

```
=> advise ( all | <current> | <range> |
```

By using the **advise** command, the neurosolver can be asked to provide a prediction or decision support capabilities. If the **all** keyword is used, then the system generates advise for all states.

Otherwise, the most probable future sequence is indicated for the

1. NOTE The **default** keyword applies to the blocks world domain and will be used in the next chapter. The **random** keyword is applicable in the context of sets of samples

<current> state up to the length indicated by <range>.

=> **list**

Lists neurosolvers with their sizes, bases, identifiers and associated neurosolvers (described later).

=> **consolidate** {on | off}

If on, then every problem that has been solved is re-learned.

=> **reset**

Resets the neurosolver. All parameters and training are lost.

=> **activityDelta** <number>

Sets the value of the activity delta used to increment the drive.

=> **acceleration** {on | off}

If off, then the search drive is increased gradually by adding the activityDelta. If on, then the search drive is increased by multiplying it by (1 + activityDelta).

=> **cycle** <number>

Sets the maximal value of allowed cycles in solutions.

=> **drive** <number>

Sets the initial value for the search drive. In complex trees, starting from 0 may lead to excessively long search times.

=> **driveLimit** <number>

Sets the upper limit on the value of the search drive.

=> **lowThreshold** <number>

=> **highThreshold** <number>

Modify the values of the thresholds.

=> **strength** {off | on | hard}

With the strength off, every outgoing connection has the same strength,

so the activity is spread evenly between a number of connections. If there is no succession/precedence relationships between nodes, then there is no connections that can be considered a connection with strength. Setting the strength **on** takes into account only the succession relationships. Setting it to be **hard** additionally takes into account the precedence relationship. See section 4.1.1 for more details and the explanation.

=> inhibit <number>

Sets the limit on the length of node inhibition after failure. It means that a node is inhibited for the next <number> steps of other nodes.

=> time (on | off)

Measures the time of command execution.

=> seed <number>

If learning randomly, the <number> is used as a seed for a random number generator.

=> cooperate (on | off)

Turns on or off the cooperative solving algorithm which uses multiple neurosolvers.

=> associate (<identifier> | all)

Associated neurosolvers cooperate in the resolution of a problem. This command associates the selected neurosolver with the neurosolver specified by the parameter. If **all** is used, then all nodes' neurosolvers will be associated.

=> slot <number>

Defines the maximum number of propagation steps for every neurosolver. If the limit is exceeded then the next neurosolver is activated.

=> dissociate (<identifier> | all)

Reverses associations.

=> debug [0 | 1 | 2 | 3]

Allows to observe the behavior of the neurosolver at various levels of detail.

=> **end**

Terminates the session.

6.7 Experiments

I present a number of results of simple experiments that I performed on the Neurosolver. The tests in this section are selected to illustrate the nature of the computation, rather than the ultimate capabilities of the Neurosolver. Therefore, a small number of states have been chosen, so the tests are easy to comprehend. A suite of more complex tests on applying the Neurosolver as a solver of rearrangement problems in the blocks world is presented in Chapter VII: *Neurosolver in the blocks world*. The interface to the Neurosolver accepts transitions as training samples. Any training sample that is a trajectory is learned as a sequence of transitions.

6.7.1 Simple example 1

The first is an example of a simple neurosolver trained with the following state transitions:

=> new 1 dynamic

=> learn 1 2

=> learn 2 3

=> learn 3 4

=> learn 4 5

=> learn 5 4

=> learn 4 3

=> learn 3 2

=> learn ? 1

=> solve 1 5

1 2 3 4 5

=> solve 5 1

5 4 3 2 1

=> advise all 10

Testing advises on all states:

```

1->2->3->4->5
2->3->4->5
3->2->1
4->3->2->1
5->4->3->2->1

```

In the case of uniform probabilities, the Neurosolver gives the first advise that it finds. For example, the advise from 2 is indicated as 2->3->4->5, although we can see that 2->1 would be equally probable. If, however, the transition 2->3->4->5 is learned again, then the advise reflects the modified connection strengths, as illustrated by the following:

```

=> learn 2 1
=> advise all 10
Testing advises on all states:
1->2->3->4->5
2->1
3->2->1
4->3->2->1
5->4->3->2->1

```

Now the advice from 2 is 2->1, because the succession probability of 2->1 is higher than the succession probability of 2->3.

6.7.2 Simple example 2

The following logs from a session with the Neurosolver illustrates how it selects preferred paths based on the frequencies of state successions. Two path, 1->2->3->6 and 1->4->5->6, are learned from 1 to 6:

```

=> new 1 dynamic
=> learn 1 2
=> learn 2 3
=> learn 3 6
=> learn 1 4

```

```
=> learn 4 5
```

```
=> learn 5 6
```

The problem of finding a path from 1 to 6 is solved as follows:

```
=> solve 1 6
```

```
1 4 5 6
```

Both paths have the same probability, 1->4->5->6 is selected due to the ordering in the simulation.

Now, if the following transition is learned:

```
=> learn 3 6
```

then another path is given as the solution:

```
=> solve 1 6
```

```
1 2 3 6
```

The following logs present further dependency of the solution on the frequencies of learning other state transition:

```
=> learn 1 4
```

```
=> solve 1 6
```

```
1 2 3 6
```

```
=> learn 4 5
```

```
=> solve 1 6
```

```
1 2 3 6
```

```
=> learn 5 6
```

```
=> solve 1 6
```

```
1 4 5 6
```

```
=> learn 3 4
```

```
=> solve 1 6
```

```
1 2 3 6
```

```
=> advise 1 10
```

```
4 5 6
```

```
=> strength hard
```

```
=> solve 1 6
```

```
1 4 5 6
```

The last four lines are interesting, because they illustrate the search guided nature of resolving problems. In spite of selecting path 1->2->3->6 as the solution to the problem 1->6, the advise request from 1 yields 1->4->5->6. That is because the advise request uses only forward execution chain to activate the nodes. If, however, the strength of the connections is used as specified by (EQ 45) by issuing the command `strength hard`, then the solution is the same as the advice. That is because 1->4 was re-learned, and 1->2 was not. If the incoming probabilities are not taken into account, then the flows of search activity from 2 and 4 are the same, otherwise, as instructed by `strength hard`, the activity flow from 4 is higher then from 2

```
=> strength hard
=> solve 1 6
1 4 5 6
```

6.7.3 More complex example

In the remaining part of this chapter, I present some of the experiments that test the response of the Neurosolver to transition problems and prediction requests in a domain of 10 states. Logs of experiments with more states would be incomprehensible and would not bring any additional value.

In each case, the Neurosolver was trained on the same file (test-10.dta in the presented logs) containing the following state transitions:

```
1->3
2->3
2->1
3->4
3->2
3->5
4->6
4->2
4->5
5->4
```


5->3
 6->10
 6->4
 6->5
 6->8
 7->9
 8->9
 8->7
 9->3
 9->7
 9->2
 10->9

In the first experiment, the strength of the connections is computed according to (EQ 42); i.e., there are no preferred paths. All branches in the search tree are fed with the same activity, so the search is uniform in all directions. The following is the log from the session:

```

=> new 1 dynamic
=> strength off
=> learn random test-10.dta
=> debug 1
=> solve all
Testing all transitions:
->1->3->2
->1->3
->1->3->4
->1->3->5
->1->3->4->6
->1->3->4->6->8->7
->1->3->4->6->8
->1->3->4->6->8->9
->1->3->4->6->10
=====
Successes: 9(100%)
Failures: 0(0%)
->2->1
  
```

->2->3

->2->3->4

->2->3->5

->2->3->4->6

->2->3->4->6->8->7

->2->3->4->6->8

->2->3->4->6->8->9

->2->3->4->6->10

=====

Successes: 18 (100%)

Failures: 0 (0%)

->3->2->1

->3->2

->3->4

->3->5

->3->4->6

->3->4->6->8->7

->3->4->6->8

->3->4->6->8->9

->3->4->6->10

=====

Successes: 27 (100%)

Failures: 0 (0%)

->4->2->1

->4->2

->4->2->3

->4->5

->4->6

->4->6->8->7

->4->6->8

->4->6->8->9

->4->6->10

=====

Successes: 36 (100%)

Failures: 0 (0%)

->5->3->2->1

->5->3->2

->5->3

->5->4

->5->4->6

->5->4->6->8->7

->5->4->6->8

->5->4->6->8->9

->5->4->6->10

=====

Successes: 45 (100%)

Failures: 0 (0%)

->6->4->2->1

->6->4->2

->6->5->3

->6->4

->6->5

->6->8->7

->6->8

->6->8->9

->6->10

=====

Successes: 54 (100%)

Failures: 0 (0%)

->7->9->2->1

->7->9->2

->7->9->2->3

->7->9->2->3->4

->7->9->2->3->4->5

->7->9->2->3->4->6

->7->9->8

->7->9

->7->9->2->3->4->6->10

=====

Successes: 63 (100%)

Failures: 0 (0%)

->8->9->2->1

->8->9->2
->8->9->2->3
->8->9->2->3->4
->8->9->2->3->4->5
->8->9->2->3->4->6
->8->7
->8->9
->8->9->2->3->4->6->10
=====

Successes: 72 (100%)

Failures: 0 (0%)

->J->2->1
->9->2
->9->2->3
->9->2->3->4
->9->2->3->4->5
->9->2->3->4->6
->9->7
->9->8
->9->2->3->4->6->10
=====

Successes: 81 (100%)

Failures: 0 (0%)

->10->9->2->1
->10->9->2
->10->9->2->3
->10->9->2->3->4
->10->9->2->3->4->5
->10->9->2->3->4->6
->10->9->7
->10->9->8
->10->9
=====

Successes: 90 (100%)

Failures: 0 (0%)

A similar experiment yields different results if the connection strength is computed according to (EQ 43); i.e., the outgoing distribution of successors is taken into account.

```
=> reset
=> strength on
=> learn random test-10.dta
=> debug 1
=> solve all
```

Testing all transitions:

```
->1->3->2
->1->3
->1->3->4
->1->3->5
->1->3->4->6
->1->3->4->6->8->7
->1->3->4->6->8
->1->3->4->6->8->9
->1->3->4->6->10
```

=====

Successes: 9(100%)

Failures: 0(0%)

```
->2->1
->2->3
->2->3->4
->2->3->5
->2->3->4->6
->2->3->4->6->8->7
->2->3->4->6->8
->2->3->4->6->8->9
->2->3->4->6->10
```

=====

Successes: 18(100%)

Failures: 0(0%)

```
->3->2->1
->3->2
->3->4
```

->3->5

->3->4->6

->3->4->6->8->7

->3->4->6->8

->3->4->6->8->9

->3->4->6->10

=====

Successes: 27(100%)

Failures: 0(0%)

->4->2->1

->4->2

->4->2->3

->4->5

->4->6

->4->6->8->7

->4->6->8

->4->6->8->9

->4->6->10

=====

Successes: 36(100%)

Failures: 0(0%)

->5->3->2->1

->5->3->2

->5->3

->5->4

->5->4->6

->5->4->6->8->7

->5->4->6->8

->5->4->6->8->9

->5->4->6->10

=====

Successes: 45(100%)

Failures: 0(0%)

->6->4->2->1

->6->4->2

->6->5->3

->6->4

->6->5

->6->8->7

->6->8

->6->8->9

->6->10

=====

Successes: 54 (100%)

Failures: 0 (0%)

->7->9->2->1

->7->9->2

->7->9->2->3

->7->9->2->3->4

->7->9->2->3->4->5

->7->9->2->3->4->6

->7->9->8

->7->9

->7->9->2->3->4->6->10

=====

Successes: 63 (100%)

Failures: 0 (0%)

->8->9->2->1

->8->9->2

->8->9->2->3

->8->9->2->3->4

->8->9->2->3->4->5

->8->9->2->3->4->6

->8->7

->8->9

->8->9->2->3->4->6->10

=====

Successes: 72 (100%)

Failures: 0 (0%)

->9->2->1

->9->2

->9->2->3

```

->9->2->3->4
->9->2->3->5
->9->2->3->4->6
->9->7
->9->8
->9->2->3->4->6->10
=====
Successes: 81 (100%)
Failures: 0 (0%)
->10->9->2->1
->10->9->2
->10->9->2->3
->10->9->2->3->4
->10->9->2->3->4->5
->10->9->2->3->4->6
->10->9->7
->10->9->8
->10->9
=====
Successes: 90 (100%)
Failures: 0 (0%)

```

With the formula (EQ 45), there is an even higher degree of preference given to repetitive path

```

Testing all transitions:
->1->3->2
->1->3
->1->3->4
->1->3->5
->1->3->4->6
->1->3->4->6->8->7
->1->3->4->6->8
->1->3->4->6->10->9
->1->3->4->6->10
=====
Successes: 9 (100%)

```


Failures: 0(0%)

->2->1

->2->3

->2->3->4

->2->3->5

->2->3->4->6

->2->3->4->6->8->7

->2->3->4->6->8

->2->3->4->6->10->9

->2->3->4->6->10

=====

Successes: 18(100%)

Failures: 0(0%)

->3->2->1

->3->2

->3->4

->3->5

->3->4->6

->3->4->6->8->7

->3->4->6->8

->3->4->6->10->9

->3->4->6->10

=====

Successes: 27(100%)

Failures: 0(0%)

->4->2->1

->4->2

->4->2->3

->4->5

->4->6

->4->6->8->7

->4->6->8

->4->6->10->9

->4->6->10

=====

Successes: 36(100%)

Failures: 0 (0%)

->5->3->2->1

->5->3->2

->5->3

->5->4

->5->4->6

->5->4->6->8->7

->5->4->6->8

->5->4->6->10->9

->5->4->6->10

=====

Successes: 45 (100%)

Failures: 0 (0%)

->6->4->2->1

->6->4->2

->6->5->3

->6->4

->6->5

->6->8->7

->6->8

->6->10->9

->6->10

=====

Successes: 54 (100%)

Failures: 0 (0%)

->7->9->2->1

->7->9->2

->7->9->2->3

->7->9->2->3->4

->7->9->2->3->5

->7->9->2->3->4->6

->7->9->8

->7->9

->7->9->2->3->4->6->10

=====

Successes: 63 (100%)

Failures: 0(0%)

->8->9->2->1

->8->9->2

->8->9->2->3

->8->9->2->3->4

->8->9->2->3->5

->8->9->2->3->4->6

->8->7

->8->9

->8->9->2->3->4->6->10

=====

Successes: 72(100%)

Failures: 0(0%)

->9->2->1

->9->2

->9->2->3

->9->2->3->4

->9->2->3->5

->9->2->3->4->6

->9->7

->9->8

->9->2->3->4->6->10

=====

Successes: 81(100%)

Failures: 0(0%)

->10->9->2->1

->10->9->2

->10->9->2->3

->10->9->2->3->4

->10->9->2->3->5

->10->9->2->3->4->6

->10->9->7

->10->9->8

->10->9

=====

Successes: 90(100%)

Failures: 0(0%)

=> end

In all cases, the Neurosolver yields very good results with slight differences in some of the resolution paths.

To get an idea of what the succession probabilities are in the network, we can use the advising capabilities of the Neurosolver. The following is the output of the system trained with the connection strength given by (EQ 42).

```
=> reset
=> strength off
=> learn random test-10.dta
=> debug 1
=> advise all 10
=> Testing advise on all states:
1->3->4->6->8->9->7
2->3->4->6->8->9->7
3->4->6->8->9->2->1
4->6->8->9->2->3->5
5->4->6->8->9->2->1->3
6->8->9->2->3->5->4
7->9->2->3->4->6->8
8->9->2->3->4->6->10
9->2->3->4->6->8->7
10->9->2->3->4->6->8->7
=>
```

The system is asked to provide advice originating at each node with a maximum length of 10 nodes. There are no loops here, so none of the paths actually exceeds that limit.

The response to the request for a prediction with the strength computed using (EQ 43) is as follows:

```

=> reset
=> strength on
=> learn random test-10.dta
=> debug 1
=> advise a 1 10
Testing advise on all states:
1->3->4->5
2->3->4->5
3->4->5
4->5->3->2->1
5->3->2->1
6->8->7->9->2->3->5->4
7->9->2->3->4->5
8->7->9->2->3->4->5
9->2->3->4->5
10->9->2->3->4->5

```

Here, the Neurosolver activates paths according to the frequencies with which connections transmit action potentials from a node.

If the formula given by (EQ 45) is used as a connection strength, then the path preference uses both outgoing and incoming signal transmission frequency statistics. It is not required to retrain the Neurosolver, because both statistics are collected in parallel. Here, the use of command `strength hard` enforces the use of both of them.

```

=> strength hard
=> advise all 10
Testing advise on all states:
1->3->4->6->10->9->8->7
2->1->3->4->6->10->9->8->7
3->4->6->10->9->8->7
4->2->1->3->5
5->3->2->1
6->10->9->2->1->3->5->4

```

7->9->2->1->3->4->6->10
 8->7->9->2->1->3->4->6->10
 9->2->1->3->4->6->8->7
 10->9->2->1->3->4->6->8->7

We can see from the examples that the type of the connection strength has a profound impact on the behavior of the Neurosolver.

The experiments presented in this chapter, as well as the experiments with the blocks world problems, confirm that by providing sufficient drive (the motivation signal) the Neurosolver is capable of solving all problems involving a search for the a path in a state space.

6.8 Dealing with noise

So far, I have assumed that the Neurosolver interfaces to the blocks world through an ideal sensory system that always reports the actual configuration. That may not be the case with a less than perfect sensory system. For example, in a distributed environment like a communication network, a state may be derived from the reports delivered by many local sensors. No single state detector is fully reliable due to the time needed to collect the data. Therefore, it is possible to receive an erroneous state and be exposed to an incorrect state transition.

The Neurosolver is prepared to deal with such cases. The strength formulas given by (EQ 43) and (EQ 45) are probabilistic in nature. In both cases, the preferred paths are being used. Assuming that the Neurosolver is trained for an extended period of time (i.e., it is exposed to a large number of repeating learning samples) and that the noise level does not exceed certain reasonable limits, the correct behavior will be modelled due to the statistical law of large numbers. Simply put, the probabilities of the noise transitions will be relatively low as compared to the probabilities of the valid transition. The following example illustrates such a case. Even though the Neurosolver learns an erroneous transition (underlined) and for a while gives incorrect results, repeating the correct transitions restores the correct behavior of the Neurosolver.

```
=> new 1 dynamic
=> strength on
=> acceleration off
activityDelta set to: 9e-16
=> drive 0
=> learn 1 2
=> learn 2 3
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 3 4 5
=> learn 1 2
=> learn 2 4
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 4 5
=> learn 1 2
=> learn 2 3
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 4 5
=> learn 1 2
=> learn 2 3
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 3 4 5
```

If the strength of a connection does not depend on the succession frequency, as in (EQ 42), then a single error may have severe consequences. An erroneous state transition permanently alters the search tree, because the strength of the connection does not adhere to the law of large numbers. The incorrect link may be followed in many searches, yielding incorrect solutions to the posed

problems. As illustrated in the following log, even repeated learning of the correct transitions does not fix the error in the behavior.

```
=> new 1 dynamic
=> strength off
=> acceleration off
activityDelta set to: 9e-16
=> drive 0
=> learn 1 2
=> learn 2 3
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 3 4 5
=> learn 1 2
=> learn 2 4
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 4 5
=> learn 1 2
=> learn 2 3
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 4 5
=> learn 1 2
=> learn 2 3
=> learn 3 4
=> learn 4 5
=> solve 1 5
  1 2 4 5
```

One way to attack transient and permanent flaws in the behavior of the Neurosolver is through the use of multiple solvers. A set of neurosolvers would have to use a voting mechanism in which the search or the solution can proceed only if a certain number of neurosolvers express the same

desire. For example, if one neurosolver out of the four that are currently resolving the problem attempts to generate an action by firing a node other than the others who agree on another node, then its action is prohibited. It can be achieved by enforcing the rule that the neurosolvers have to wait for the external input reporting the new state before the solution is allowed to continue. The faulty neurosolver cannot proceed with its solution, because the world moves to the state on which the other three neurosolvers agreed. The neurosolver waits for the sensors to detect the new state that never happens, because the other neurosolvers voted for something else. The chain reaction along the solution path inside the faulty neurosolver is broken. Ultimately, the faulty neurosolver will be reset during the initialization for another problem. The solution will proceed along the path selected by the voting majority of the neurosolvers. A certain minimum would apply to the voting; if not reached, the whole assembly of the neurosolvers would fail to solve the problem.

The neurosolvers may be instructed to learn each of the solutions that are made in cooperation. Allowing sufficient length of such re-training, the faulty neurosolver will be aligned with the rest. Again, we consider a society of neurosolvers as being more trustworthy than any single neurosolver. As in the real life, not always is such an assumption correct.

In the further chapters of this thesis, I will describe and analyze the use of multiple neurosolvers in the context of autonomous agents.

6.8.1 The Neurosolver solves a problem that STRIPS could not

The register exchange problem has been shown to be unsolvable by the STRIPS planner (Nilsson, [41]). The problem is to provide a plan of exchanging two values in registers X and Y using the third register Z (Figure 45).

FIGURE 45. The register exchange problem. The objective of the task is to exchange the values in registers X and Y. The arrows and numbers indicate the assignments and their order. A dash symbol in the register Z indicates that the register is empty.

6.8.1.1 STRIPS approach

Both STRIPS and the register problem are analyzed in (Nilsson, [41]). I will not repeat all of the details, but I present the essence of the STRIPS' deficiency in solving the register problem. STRIPS uses a two-element database: the state of the world and the current goal. The former is a flat set of well-formed formulas (wffs); the latter - a stack of goals and their subgoals. If all elements of the goal can be satisfied (in the terms of formal logic) with the facts from the current state, then the goal is achieved. Until that happens (or all ways to achieve it have failed), STRIPS attempts to satisfy the elements of the compound goal by putting them on the stack and switching its attention to them. If all of the subgoals have been achieved, then the compound goal (that might be a subgoal of another compound goal) is attained. If there is a single literal on the top of the goal stack, then the collection of possible actions is consulted to find whether any actions are applicable. Each action consists of three elements: a precondition, a delete list and an add list. If the literal on the top of the goal stack matches one of the elements of the add list, then the action is added to the goal. When the preconditions are satisfied, then the action is executed by deleting

the literals (ground instances) in the delete list from the current state, adding the literals in the add list to the state and removing the action from the stack.

A single action realizing an assignment of a value from one register to another is needed for the register problem:

```

assign(u, r, t, s)
  precondition: contains(r, s) and contains(u, t)
  delete:      contains(u, t)
  add:         contains(u, s)

```

There are three registers used: X, Y and Z. The following table illustrates the way in which STRIPS attempts to solve the problem:

Step	Current state	Goal stack
1	contains(X, A) contains(Y, B) contains(Z, -)	contains(X, B) and contains(Y, A)
2	contains(X, A) contains(Y, B) contains(Z, -)	contains(X, B) contains(Y, A) contains(X, B) and contains(Y, A)
3	contains(X, A) contains(Y, B) contains(Z, -)	contains(r, B) and contains(X, t) assign(X, r, t, B) contains(Y, A) contains(X, B) and contains(Y, A)
4	contains(X, B) contains(Y, B) contains(Z, -)	contains(Y, A) contains(X, B) and contains(Y, A)

In step 3, STRIPS matches the top subgoal with the current state with a unifier {Y/r, A/t}. The precondition for the action is satisfied, so the register X is assigned B. Unfortunately, that overwrites A which is lost. The next subgoal on the stack (step 4) is not satisfiable with the current state. STRIPS fails because no further subgoals can be created.

The problem seems to be in the STRIPS eagerness to satisfy the final goal. Unfortunately, while looking at the details, STRIPS loses the global picture. It focuses on the assignment of B to X, but it does not take into account that the value that is currently stored in X, A, will be needed in the

future. STRIPS fails because it is greedy, unwilling to perform an action which, at first sight, appears to move it away from its goal

6.8.1.2 Neurosolver approach

The Neurosolver does not suffer from a local-bias syndrome, because it operates on global states. Assuming as before that the values in the registers are A and B, there are 6 possible states of the system:

State	X	Y	Z
1	A	B	-
2	A	-	B
3	-	A	B
4	B	A	-
5	B	-	A
6	-	B	A

FIGURE 46. State diagram for the register problem.

Figure 46 shows the state diagram of the system. The diagram is the basis for the knowledge to solve the problem that the Neurosolver has to either be given or acquire. The former can be viewed as a declarative approach in which the knowledge is encoded, and the latter as an algorithmic approach in which the Neurosolver learns skills from teaching samples. STRIPS is based on formal logic, so its knowledge was declarative.

The following session with the Neurosolver illustrates that the problem is easily solved after learning. The file `strips.dta` contains all valid state transitions as illustrated in Figure 46.

```
=> new 1 dynamic
=> learn strips.dta
=> solve 1 4
1 2 3 4
```

The sequence represents the solution to the problem that is illustrated in Figure 45 by arrows and ordering number in black circles.

The assumption that there are just two values, A and B, in the registers requires some commentary. A and B represent any value that can be contained in the registers X and Y; they are variables. In a symbolic world, a variable is a well understood concept. In the sub-symbolic world, the use of symbols is not present, so a variable must represent something else. One of the hypotheses, drawn from Neuroscience (for example the Burnod's monograph [14]), is that assemblies of nodes (cells) that interchangeably occupy certain slots in various contexts become associated through cells acquired especially for that purpose. Such a cell that relays signals between assemblies can be considered a variable that represents all of the assemblies. Each of the assemblies is a specific value of the variable. For example, thinking about an apple activates images of the apples that were seen in the past; 'apple' is a concept that was created to represent all of the objects of the same nature to which the observer was exposed over a period of time. Of course, with time such a concept starts to live its own life and can be used to create other concepts; for example, in the case of 'an apple' a concept of 'a fruit' may emerge. I hope that I have made the point, so let us return to the register problem. In this case, A and B represent the concepts of 'the contents of a register at certain time t_0 ' referring to the registers X and Y respectively. In that way, we do not need to deal with all possible values that can be stored in the registers.

The register problem is not the only problem that was 'too hard' for the STRIPS and solvable by the Neurosolver. In the next chapter, I present research on the application of Neurosolver in a blocks world. The Neurosolver is able to solve any problem in the blocks world chosen for this

work. Sussman's anomaly that cannot be solved by STRIPS (nor by Sussman's own planner, HACKER) is just one of the reconfiguration problems that can be solved with Neurosolver.

It should be mentioned that next generation symbolic planners, particularly least-commitment, partial-order planners, were shown to solve the register exchange, Sussman and other problems that were proven to be intractable by the STRIPS system.

All symbolic planners suffer from long execution times required to solve complex problems. Although the Neurosolver is computationally intensive as well, the potential for a hardware implementation that would provide a true parallel breadth-first search is very appealing. Similar implementations of symbolic planners are hard to imagine. Parallel processors certainly do improve the efficiency of symbolic algorithms, but experience indicates that parallel implementations of symbolic AI languages have not fulfilled the expectations.

Chapter VII:Neurosolver in the blocks world

7.1 The blocks world

The blocks world, engineered especially for exploring the capabilities of the Neurosolver, was introduced in section 2.1: *The blocks world*. Remember that the blocks world is composed of 9 locations, arranged in 3 columns and 3 rows. There are three different blocks in the world in specific configurations. Three colors and labels are used to differentiate the blocks: red - A, green - B - and blue - C. For the purpose of this research, problems of rearranging the blocks are considered. In this section, a more detailed analysis of such problems is presented.

FIGURE 47. A workbench to explore the blocks world by neuromorphic agents.

The front end to the blocks world includes a graphical user interface that allows the operator to move the blocks, compose sequences of moves, save and restore sets of teaching samples and control the learning and behavior of the Neurosolver. Additional dialogs are available to observe

activity patterns, input vectors and to control state changes by pointing and clicking on nodes. A blocks world workbench including the graphical view of the blocks world is illustrated in Figure 47.

The Neurosolver interacts with the blocks world through the sensory and manipulatory interfaces discussed in section 2.2: *The Neurosolver*. Although a graphical user interface is a nice feature of any system, it is very often inconvenient for many tasks. In the case of the blocks world, it would be very tedious to present all teaching samples manually or, for that matter, perform a comprehensive test on the quality of the solver attacking large numbers of rearrangement problems. Therefore, in such cases a textual interface was used that omits the time consuming graphics.

7.2 Problem state space

There are many types of problems in the blocks world. To keep things simple, I consider the problem of rearranging the blocks from an initial configuration to a desired configuration. Any block configuration in the blocks world constitutes a state of the system. Changes in time to the configuration are described in terms of state transitions and trajectories that were defined in section 2.1: *The blocks world*. A problem is posed by a pair of states: the goal and the current state. The solution to the problem is a trajectory starting with the current state and ending with the goal (Figure 48). Only valid state transitions can be included in the solution.

FIGURE 48. The solution to the problem is a sequence of configurations between the current configuration (state) and the goal.

Let us return to the constraints placed on the configuration of the blocks that were described in Chapter II: *Introduction*. The set of valid block configurations is determined by the following constraints:

- a block can occupy a position that is not taken by another block
- a block rests either on the table or on another block.

Other configurations are considered invalid unless some transient configurations are also allowed. For example, a configuration with a single block violating the second constraint, but remaining adjacent to any of the remaining two blocks can be considered temporarily valid; for example, for the duration of the movement of the violating block.

It should be noted that the constraints presented here are chosen arbitrarily. The blocks world could have functioned with another set of constraints or without any constraints at all. By use of constraints two goals are attained. Firstly, the problem becomes more realistic, as there are few systems that operate without any kind of constraints. Secondly, the state space for the problem is limited, so it is more manageable for the research as well as presentation purposes. Later on, several interesting questions about automatic mechanisms for limiting the state space based on the teaching samples that satisfy the constraints will be addressed.

FIGURE 49. With strong constraints (see the text for the explanation), there are 10 possible arrangements of blocks if the color is disregarded. Each three blocks can be arranged in 6 ways, so there are $10 \times 6 = 60$ possible configurations taking into account the colors.

Taking into account the constraints on the blocks and disregarding colors, there are 10 arrangements of the blocks, as illustrated in Figure 49. Each three blocks can be permuted in 6 ways, so if the color is again taken into account, there are 10×6 possible configurations. Therefore, the cardinality of the problem state space is 60. We will regard the constraints that allow only such a number of states as strong.

If the constraints are relaxed in order to allow certain transient states, then the state space is larger. For example, with the additional block arrangements illustrated in Figure 50 the size of the state space for the blocks world increases to 126 (60 original and 66 additional states).

FIGURE 50. With weaker constraints (see the text for the explanation), there are 11 additional arrangements of blocks with the color being disregarded. Each three blocks can be arranged in 6 ways, so there are $11 \times 6 = 66$ additional configurations taking the color into account again.

In the case of relatively simple system like the blocks world, it is possible to determine the size of the problem state space by straightforward analysis. With more complex problems, however, it may be difficult or not possible at all to predict which states should be accounted for. In such cases, Kohonen self-organizing maps (Kohonen, [29]; see also Chapter IV: *Introduction to Neural Networks*) can be used to eliminate states that are not valid, because the states that are not used (and invalid states are not used) are not present in the topology map. The Kohonen algorithm is a clustering technique that, in its extremal version, results in a one-to-one mapping of input into output. One can start with a large number of available nodes and end up only with some of them being actually used for storing states, because certain inputs may never be activated. The unused nodes can be pruned to limit the size of the network. Alternatively, some of the unused nodes may

be left for future considerations; they can be used to accommodate new input patterns. A supervised algorithm, for example the backpropagation or Kohonen's LVQ algorithms, could also be used, but unsupervised algorithms are superior in the sense that they do not need any a priori knowledge and can adapt autonomously; i.e., without a teacher. In the course of this research, I used the Kohonen and backpropagation methods in order to confirm that a state space can be obtained automatically. The strong constraints have been used that yield 60 states. The results confirmed the above claims. In the Kohonen case, a larger number of nodes than the cardinality of the state space had to be used. From 225 (15x15 grid) original nodes, after a number of training epochs (in the range of hundreds of thousands), 60 distinct nodes started to respond to 60 input vectors encoding the configuration of the blocks world. Using only 60 nodes did not yield desired effects - one node would respond to more than one input pattern with certain nodes remaining unused. In spite of trying various parameters that impact the learning in the Kohonen network (as allowed under the SNNS simulator; see [53]) and running the training for very long time, the desired one-to-one mapping has not been obtained.

The backpropagation method required that input-output pairs were presented at the same time. The elements of each of the training samples have been associated in the way described in Chapter IV: *Introduction to Neural Networks*. After training, a distinct output node responded to each input vector. 15 hidden units were used in the network. The output layer was an array of 60 nodes. The input vector used for both methods is described in the next section.

Providing that the input vectors are linearly independent (i.e. the state space is linearly decomposable), the simpler Perceptron algorithm, that does not require hidden units, might have been used.

7.3 Interface between the Neurosolver and the blocks world

7.3.1 Input representation

The state of the blocks world is encoded using a binary vector that is presented to the Neurosolver as input. Each of the locations of the blocks world is represented by two bits in the vector:

- 00 - empty
- 01 - red (A)
- 10 - green (B)
- 11 - blue (C).

Hence, the dimension of the input vector is $(3 \times 3) \times 2 = 18$. Any configuration of the blocks can be encoded using that number of nodes. An example is illustrated in Figure 51.

FIGURE 51. Encoding the configuration of the blocks in the blocks world into an input vector to the Neurosolver.

The input nodes are connected to state nodes in such a way that one state node is activated for each block configuration. The connections that are hard-wired, could have been determined by one of the techniques described in the previous section. In the case of the blocks world, the con-

nections may also be engineered manually by the analysis of the state space. In all but the backpropagation case, the resulting wiring is of the type illustrated in Figure 52. The only difference in the network obtained by the backpropagation method would be the presence of intermediate hidden units between the input and output nodes. In every case, one and only one output node is set in response to any pattern. The units that exclusively respond to input signals are commonly called **grandmother cells** by researchers in the research areas related to Cognitive Science, including Neural Networks.

It should be noted, that although there are efferent connections going from each input node to every output node, some of the connections (those that have strength equal to 0) are, in fact, not present. They only need to be considered during the learning process.

FIGURE 52. Every input cell is connected to all grandmother cells. With a matrix of the connection weights W , each combination of input signals activates just one of the grandmother cells representing the meaning (concept) C_i of the input vector I .

7.3.2 Output representation

Whenever one of the Neurosolver's nodes fires, the state associated with that node is translated into an input vector that is a binary representation of the block configuration (Figure 53). The blocks in the workspace are re-arranged accordingly by the system of manipulators.

The translation of a state into a configuration can be achieved by a network analogous to the architecture presented in Figure 52. The direction of the flow of data is reversed, i.e., each grand-

mother cell has connections of strength fixed to 1 to every node in the input vector (which now becomes an output vector) that belongs to the binary representation of the configuration corresponding to that cell (Figure 53).

FIGURE 53. Decoding the state of the Neurosolver into a configuration of the blocks in the blocks world.

7.4 Experiments

In the previous chapter I presented several experiments on simple instances of the Neurosolver. Those neurosolvers were simple, because their basic objective was to test the major ideas. In this section, I describe the capabilities of the Neurosolver in the context of the blocks world. The objective here is not to assess the ideas behind the development of the Neurosolver, but to test its capabilities in solving actual problems. In spite of a belief that both objectives are not that dissimilar, the actual practical gap was considerable. Some things that worked in simple cases, required implementation refinements to the simulator before yielding the same degree of quality as that

observed in the original experiments. The reason for that lies in the fact that the Neurosolver is in its nature an analog device that needed to be simulated in software. For example, in simple cases, the activity in the Neurosolver rarely increased to large levels. With many branches and cycles in the search trees, the maximum allowed activity must have been set much higher than originally engineered. Another thing, already described in the previous section, was the need for accelerated increases in the search drive; in complex trees, the activity growth rate was simply too slow for practical use. Another heuristic was designed regarding the length of inhibition. For problems that fail, the inhibited nodes are gradually disinhibited, because they probably block important branches in the search tree.

The experiments with the Neurosolver were directed into solving problems in the blocks world with both strong and weak constraints. Both cases can be considered as successive stages of testing the practical scaling capabilities of the Neurosolver.

I performed a large number of tests with the Neurosolver with various sets of parameters (specified in section 6.6.3: *Application Program Interface (API)*). I recognized several trends, but none of them was critical to the problem solving capabilities of the Neurosolver. Most of them would affect the time required for the solution, while some would improve or worsen the chance for a solution in certain cases. For example, allowing the Neurosolver to increase the drive gradually (simulating the analog ideal), increases the time required to solve certain problems; sometimes, the time is unreasonably long, yielding practical failures:

```
=> new 1 dynamic
=> learn random nbwtraj1.dta
=> acceleration off
=> lowThreshold 0.45
=> highThreshold 0.9
=> activityDelta 1
=> time on
Execution time: 0 seconds
=> solve 1 2
```



```

1 22 30 12 39 33 28 24 2
Execution time: 0.33 seconds
=> activityDelta 0.1
Execution time: 0 seconds
=> solve 1 2
1 22 30 12 39 33 28 24 2
Execution time: 1.26 seconds
=> activityDelta 0.01
Execution time: 0 seconds
=> solve 1 2
1 22 30 12 39 33 28 24 2
Execution time: 5.55 seconds
=> activityDelta 0.001
Execution time: 0 seconds
=> solve 1 2
^C

```

In the presented example, the problem 1→2 is not solved because it takes too long. ^C indicates that the Neurosolver process was interrupted¹. Applying the accelerated increase in the drive improves the chances that the Neurosolver will find a solution:

```

=> new 1 dynamic
=> learn random nbwtraj1.dta
=> acceleration on
=> lowThreshold 0.45
=> highThreshold 0.9
=> time on
Execution time: 0 seconds
=> activityDelta 10
Execution time: 0 seconds
=> solve 1 2
1 22 30 35 41 10 28 24 2
Execution time: 0.06 seconds

```

¹ A more graceful solution with the use of exceptions could have been implemented; I considered it outside of the scope of this work.

```

=> activityDelta 1
Execution time: 0 seconds
=> solve 1 2
  1 22 30 12 39 33 28 24 2
Execution time: 0.11 seconds
=> activityDelta 0.1
Execution time: 0 seconds
=> solve 1 2
  1 22 30 35 41 10 28 24 2
Execution time: 0.38 seconds
=> activityDelta 0.01
Execution time: 0 seconds
=> solve 1 2
  1 22 30 35 41 10 28 24 2
Execution time: 1.43 seconds
=> activityDelta 0.001
Execution time: 0 seconds
=> solve 1 2
  1 22 30 12 39 33 28 24 2
Execution time: 7.09 seconds

```

We can see that there is a significant difference in time¹ required to find the solution depending on the activity delta by which the search drive is increased (by the acceleration formula (EQ 46)). An interesting observation is that the solutions are not the same in all cases in this example. That is because the activity patterns in each case are different due to variations in the granularity of the search. The best solution path would be obtained in a true analog implementation. Turning off the acceleration and decreasing the activity delta approximates that, but unfortunately, increases the simulation time at the same time.

In the previous chapter, I have already stated that the behavior of the Neurosolver is affected by the choice of the strength formula for the connections. Not only various solution paths may be

1. All times were measured on a PC 486/66MHz with 32MB RAM under Windows 95

found, but in general the time required to find a solution increases with the complexity of the connection strength formula. The following is a log of a session with the Neurosolver using the formula (EQ 42):

```
=> new 1 dynamic
=> strength off
=> learn random nbwtraj1.dta
=> lowThreshold 0.45
=> highThreshold 0.9
=> time on
Execution time: 0 seconds
=> activityDelta 1
Execution time: 0 seconds
=> solve 1 2
  1 22 30 35 41 10 28 24 2
Execution time: 0.11 seconds
=> activityDelta 0.01
Execution time: 0 seconds
=> solve 1 2
  1 22 30 35 41 10 28 24 2
Execution time: 1.64 seconds
=> activityDelta 0.001
Execution time: 0 seconds
=> solve 1 2
  1 22 30 35 41 10 28 24 2
Execution time: 9.39 seconds
=> activityDelta 0.0001
Execution time: 0 seconds
=> solve 1 2
  1 22 30 35 41 10 28 24 2
Execution time: 465.93 seconds
```

The following session sets the strength according to (EQ 45):

```
=> new 1 dynamic
=> strength hard
```

```
=> learn random nbwtraj1.dta
=> acceleration on
=> lowThreshold 0.45
=> highThreshold 0.9
=> time on
Execution time: 0 seconds
=> activityDelta 1
Execution time: 0 seconds
=> solve 1 2
  1 22 30 12 39 33 28 24 2
Execution time: 0.27 seconds
=> activityDelta 0.1
Execution time: 0 seconds
=> solve 1 2
  1 22 30 12 39 33 28 24 2
Execution time: 0.99 seconds
=> activityDelta 0.01
Execution time: 0 seconds
=> solve 1 2
  1 22 30 12 39 33 28 24 2
Execution time: 4.67 seconds
=> activityDelta 0.001
Execution time: 0 seconds
=> solve 1 2
  1 22 30 12 39 33 28 24 2
Execution time: 45.7 seconds
```

The main source of the differences in the length of the search is the fact that the process of spreading activity is simulated in software. However, even if the Neurosolver was implemented as an analog device, so the search process was truly parallel (i.e., all of the branches in the search tree were activated simultaneously), there would be differences in search times due to the varying resistance of the connections to the flow of the search activity.

The most important measurement for the quality of the Neurosolver in context of the blocks rearrangement problems in the blocks world was its performance ratio measured as a number of

solved cases against the failed ones. The Neurosolver was trained on all valid trajectories with each trajectory presented random number of times. All types of strength formulas for the connections were applied each yielding the following results

Experiment:	testing the capacity of the Neurosolver in the blocks world with strong constraints (60 states)
Learning sample:	all 144 valid state transitions; each transition presented randomly in 10000 iterations
Problem:	all tasks of rearranging the blocks ($60 \times 60 - 60 = 3540$ in total)
Results:	Successes: 3540(100%) Failures: 0(0%) Only valid state transition in each solution

7.5 Scaling to larger problems

In an attempt to better understand the scaling capabilities of the Neurosolver, I performed similar tests in a larger domain of the blocks world with the weaker configuration constraints. The time required to find a solution to a problem in this problem space was, in general, longer. That is because a larger state space had to be covered with plenty of iterations simulating the increase in the search drive. The allowed range for the drive growth had to be adjusted (to allow for coverage of larger search trees), but the problem solving capabilities were preserved. The behavior of the Neurosolver is the same as in the smaller domain, and although these experiments are insufficient to generalize, the least I can say is that there is nothing intrinsic that would prevent the Neurosolver from scaling up well.

For the record, the results of the larger test of the solving capabilities and the capacity of the Neurosolver are as the following

Experiment:	testing the capacity of the Neurosolver in the blocks world with weak constraints (126 states)
Learning sample:	all 268 valid state transitions; each transition presented randomly in 10000 iterations
Problem:	all tasks of rearranging the blocks

(126*126-126 = 15750 in total)

Results: Successes: 15750 (100%) Failures: 0 (0%)
Only valid state transition in each solution

In many statements of the following analysis, I will refer more to an ideal, analog, implementation of the Neurosolver rather than the software simulator used in the experiments. I have already indicated that a number of heuristics had to be used to speed up the search in the simulated environment. The software has to simulate the gradual increase in the search drive through many iterations of an increase in the activity by a small value (activity delta). To simulate the parallelism, the simulator has to propagate an increase in the activity to all neighbors in turn in a large number of cycles. It is obvious that such implementation is far from an ideal, analog spreading of activity.

In the following sections, I analyze an impact on the Neurosolver of a number of factors that may potentially increase complexity of the problem.

7.5.1 Adding a state transition

If a transition is learned that was presented in the past, then the Neurosolver merely increases the strength of the connection between the divisions of two involved nodes (in the simulator it is achieved by updating the statistics in the divisions and the forward and backward connections). There is no impact on the search, since the strength of the corresponding connections is as well established as before the learning (in the simulator it is calculated dynamically using the statistical data). In an analog implementation, learning a new transition would be the same. In the simulator, a new connection object must be instantiated. That does not have any major bearing on the performance of the Neurosolver.

See section 7.5.3: *Adding a new state* for the analysis of acquiring a transition involving a new state.

7.5.2 Adding a trajectory - storing a plan

A trajectory is a sequence of state transitions and is learned as such. Therefore, the impact of recording a trajectory on the complexity of the Neurosolver is the same as when learning a number of separate transitions.

A trajectory can also be considered a plan. Adding such a plan to the Neurosolver has only a modest impact on the space requirements involving instantiating connections for the elements of the plan that have not been presented previously. All other parts merely increase the strength of the corresponding connections.

Adding new states is analyzed in the next section.

7.5.3 Adding a new state

There is no major impact on the Neurosolver of increasing the state space by a single point. In the software simulator, a new object has to be created. That is only possible if the simulator is in the dynamic mode. A new state is presented as a part of a state transition, so there are also 4 connection objects that have to be instantiated.

An analog implementation would most likely have nodes and zero-valued connections to all other nodes available by design.

7.5.4 Increasing the size of the state space

It is a well established fact that a growth in the number of parameters and the range of their values increases the size of the state space of the system dramatically. No system can escape from the resulting *exponential growth* in the complexity of the problems. Nevertheless, neural methods are well suited to deal with such issues through their capability to generalize. The following brief discussion illustrates such capabilities in the context of the Neurosolver and the blocks world.

There was a significant reduction in the number of valid states in the overall state space of the blocks world with 9 positions. If only 3 locations are occupied by blocks, then we have

$$\binom{9}{3} = \frac{9!}{3!(9-3)!} = 84 \quad (\text{EQ 47})$$

possible configurations. It yields $84 * 3! = 84 * 6 = 504$ states if the color is taken into account. As we saw earlier, only 10 configurations were considered valid under the strong constraints. That yielded $10 * 3! = 60$ states that were used in the described experiments. The reduction ratio is $504 : 60 = 8.4$.

Let us consider a larger blocks world with 16 locations (4 rows and 4 columns) and 4 color blocks. The number of possible configurations in such a world is:

$$\binom{16}{4} = \frac{16!}{4!(16-4)!} = 1820 \quad (\text{EQ 48})$$

Taking into account that 4 colors can be arranged in $4! = 24$ ways, we have a potential state space of $1820 * 24 = 43680$ states. However, only 34 configurations are valid under constraints that are similar to the strong constraints that were used with the smaller blocks world. That yields $34 * 24 = 816$ valid states which is considerably higher than 60 states needed in the previous case. However, the ratio of the potential number of states in the space to the number of states that are actually used, which is $43680 : 816 \sim 53.53$, is considerably higher than 8.4 for the smaller case.

If we analyzed larger blocks worlds and kept the constraints on their configuration, then the reduction ratio would be higher.

Although this analysis is not Neurosolver-specific, nevertheless the Neurosolver provides a natural capability for reducing problem complexity. As pointed out before, such a reduction can be achieved by applying a Kohonen (or another) self-organization algorithm to construct a task-specific neurosolver up front or by allowing the Neurosolver to grow dynamically. In an analog implementation, the Neurosolver could have a large repertoire of available interconnected nodes

that would be used as new states are detected. In the simplest case, a node could be used for each new combination of system parameters. To deal with potential input noise an adaptive method would have to be applied. An ultimate goal would be to combine an adaptive construction of the state space with the Neurosolver learning.

The resource requirements for encoding large state spaces might be tremendous. However, even as of this writing, it is possible to place millions of gates on a silicon chip. A Neurosolver's node is not much more complex than a gate, so there is already a potential of dealing with many real-world problems. Furthermore, with the inevitability of incoming nanotechnology, it will be possible to put more processing power than that of the human brain in a 1 cubic centimeter device (Merkle, [35]).

Chapter VIII: The agents

From its conception, the Neurosolver was destined to be a prototype for an autonomous, adaptive agent. If such an agent was provided with sensory and manipulatory systems and injected in any environment, it would acquire the behavior of the domain and use that knowledge to modify the environment to fulfill certain objectives. Such a vision of the Neurosolver was the genesis for my interests in autonomous agents.

The early implementations of the Neurosolver did not provide capabilities that would guarantee resolution of every rearrangement problem. That deficiency has been changed with each refinement added, ultimately yielding perfect scores on the blocks world problems (see the results in Chapter VII: *Neurosolver in the blocks world*). Prior to determining perfect experimental behavior, the problems contributed to the idea of applying multiple neurosolvers. The argument was that if a number of imperfect instances of the Neurosolver are applied, then perhaps *en masse* they might make up for their individual deficiencies and cooperatively resolve any rearrangement problem in the blocks world. Each instance of the Neurosolver might have had only a partial knowledge of the system behavior, but together the neurosolvers could cover the knowledge completely. The transient and permanent incompleteness or faults that were mentioned in Chapter VII: *Neurosolver in the blocks world* also added fuel to my interest in agents.

This and the succeeding chapters present the use of the Neurosolver in the context of autonomous, mobile agents. In this chapter, I concentrate on the use of a single solving agent based on the Neurosolver and a community of agents constituting the sensory and manipulatory systems. In the

next chapter, I will present the research and thoughts on issues related to the use of multiple neurosolvers.

Research on agents have been thriving for several years. It is partly due to the generality of the term *agent* itself. Webster's Ninth Collegiate Dictionary gives a number of meanings for the word. Each of them could be used in the context of computation. "...*something that produces or is capable of producing an effect...*" can certainly be applied to anything that personifies a drive to yield an effect. The largest body of current research employs the term in its another meaning: "...*one who acts for or in place of another by authority from him as [...] a representative, emissary...*". This encompasses all of the work related to Internet agents and information filtering agents (for example, see Etzioni [21]). Another meaning of the term given by Webster is again very general: "...*a means or instrument by which a guiding intelligence achieves a result...*". Most of the research into autonomous robots would probably best adhere to such a definition of agent.

In the context of this work, agents are media that cooperatively resolve externally imposed problems through interaction with other individuals of the same or different species. In the context of this work, contrary to the agents used in other research that possess higher level of organizational and individual skills, a lone agent is not capable of accomplishing anything. Such use of agents lets us classify this part of the research as belonging to the field of Distributed Artificial Intelligence (DAI) (Chaib-draa, [16]). Multi-Agent Systems (DAS), Distributed Problem Solving (DPS) and Cooperative Problem Solving (CPS) would also be appropriate tags for the classification as, basically, their meaning is the same.

8.1 Agents in the blocks world

Three types of agents are used in the context of this work. The blocks world is populated with **sensing**, **manipulating** and **solving** agents. Each agent is mobile; i.e., can move from one location of the blocks world to another. An agent determines the next move on its own. The direction

of the move is generated randomly except for manipulating agents that can be controlled by solving agents. Amongst the three, the sensing agents are meant to be the simplest architectures, so they can travel at high speeds throughout the blocks world. The solving agents are the most complex. They may spend relatively long periods of time at each location. In the simulation environment, the time spent at a blocks world location is controlled by a modifiable delay parameter of the loop generating the next move of an agent.

The society of sensing agents constitutes a sensory system, and the manipulating agents - the manipulatory system. Both provide an interface between the problem solver consisting of a number of solving agents and the blocks world.

8.1.1 Sensing agents

The principal task of a sensing agent is collection of information about the state of the blocks world. A sensing agent moves randomly from one location to another and, for each location, records that location's occupant; i.e., one of the three blocks or the fact that the location is not occupied by any block. When a sensing agent meets a solving agent at the same location, then the sensing agent reports the collected local view of the world to the solving agent. The collected information is encoded as an 18-bit vector with 2 bits for each position as described in section 7.3.1: *Input representation*. The location is frozen for the duration of the communication; i.e., neither agent can move to another location. That ensures a successful completion of the data exchange without disruption. After the state of the world is transferred, both agents continue their respective journeys. The solving agent stays longer at each location due to a longer delay in the generation of the next move, so the chance of meeting many sensing agents is increased.

The blocks world is a dynamic system, so the blocks may change their locations between two consecutive visits of a sensing agent. Therefore, the view of the world as recorded by a sensing agent may not be the world's actual state at that moment. This situation, unacceptable with only one slow agent, is satisfactorily handled by multiple agents that travel quickly relative to the changes

that occur in the system. That is a case with, for example, a telecommunication network where the data is being transferred usually much faster than the changes to the network state may occur. A similar assumption is made with respect to the blocks world. The agents travel much faster than the speed with which the operator can rearrange the blocks. In spite of that, there are still moments, during changes to the state of the blocks world, when an agent has a distorted view of the world for some time. It would have to travel through all locations that were involved in the state change to accumulate the correct new state. In the meantime, the agent may report the incorrect state to the solving agents met on its way. That requires certain intelligence on the part of the solving agent to deduce the correct state from the reports brought by many sensing agents. Details of such a mechanism will be described in section 8.2.1: *Communication between sensing and solving agents*.

8.1.2 Solving agents

A solving agent also moves randomly between the locations of the blocks world. The purpose of its trek is to meet as many other agents as possible and either obtain data from the sensing agents or distribute rearrangement tasks to the manipulating agents. The solving agent is capable of resolving blocks rearrangement problems, because it is equipped with a Neurosolver-based brain operating in the state space of rearrangement problems.

When no rearrangement task is assigned to an agent, the agent is in learning mode. The agent determines the state of the blocks world from the reports received from the sensing agents met in the past. This state is used as a learning sample by the agent's brain in the way described in Chapter VI: *Neurosolver: A neural network based on a cortical column*. The difference is in the interface to the blocks world. As indicated in section 7.3.1: *Input representation*, the Neurosolver is connected to a state perception mechanism that, disregarding the particular way in which it was created, is a network of grandmother cells. So far, it has been assumed that the input layer is somehow set by software that has a global view of the blocks world and therefore can determine its state. Very often this approach is difficult or impossible to realize (for example in telecommu-

nication networks). Now, the Neurosolver residing on a solving agent interfaces indirectly to the blocks world through sensing agents. The sensing agents provide the input patterns which can be applied to the network of grandmother cells. It is not important whether the network resides on the sensing agent or on the solving agent. The result is that the solving agent receives the output from that network; i.e., a state vector consisting of all grandmother cells with only one cell being on.

When an actual state of the blocks world is determined (see section 8.2.1: *Communication between sensing and solving agents* for details), then it is input to the agent's brain - an instance of the Neurosolver - as a teaching sample. The Neurosolver behaves in the same way as described in the earlier chapters; i.e., it learns trajectories in the state space of the blocks world on-line. Consequently, we can say that the agent learns as well, because the neurosolver is its brain.

If a solving agent is given a problem to solve, then it switches its mode to execution. A problem is presented, as in the case of the Neurosolver (see Chapter VI: *Neurosolver: A neural network based on a cortical column*), by a current-state/goal-state pair. This pair is passed to the Neurosolver. Learning is suspended until the task is carried out; that is more of a convenience than a necessity. The Neurosolver attempts to solve the rearrangement problem, that is to generate a solution path transforming the current state to the goal state. The mechanisms governing that activity have already been described in section 6.4: *Solving problems*. The solution - a rearrangement plan - is distributed to the manipulating agents for execution. This process is described in section 8.2.2: *Communication between solving and manipulating agents*.

The presentation of a problem to solve constitutes another logistic problem. For the purpose of this work a direct communication link to the solving agent was assumed. Such a link might be difficult to establish in the real world because of the mobility of the solving agent. This can be overcome by implementing an indirect communication link. There are a number of options and they are discussed in section 10.3: *Agents*.

In this chapter, I analyze problem solving with only one solving agent. Multiple solvers will be discussed in Chapter IX: *Multiple solving agents*.

8.1.3 Manipulating agents

While discussing the application of the Neurosolver to solve the rearrangement problem in the blocks world, we assumed that there is a manipulatory system in place which moves the blocks to the locations ascribed by the desired configuration. With the introduction of manipulating agents, we can build an appropriate manipulatory system, because the manipulating agents are capable of moving the blocks between the locations of the blocks world. A manipulating agent can grab a block and carry it to another location. Such an agent moves randomly throughout the blocks world until a solving agent is met that has a block rearrangement plan to execute. In the course of the communication session (see section 8.2.2: *Communication between solving and manipulating agents* for details), the solving agent orders the manipulating agent to move a block by indicating the desired course of action. The action is determined from the portion of the solving agent's rearrangement plan that has not been distributed yet. The manipulating agent uses the information obtained to pick up the block from its current position and carry it to the desired location.

The next state of the blocks world is received from the solving agents as an 18-bit vector corresponding to the desired configuration of the blocks. Each movement of any block is recorded by the Neurosolver, so that the solution contains only transitions that involve a movement of one block. That makes the task that is assigned to a manipulating agent easy, as it involves just one block. The agent discards the color and translates the configuration into a single block location space by performing a number of simple logical operations on binary vectors. The result is a pair of two locations of the block to move: the source, the location that is currently occupied by the block, and the destination, the location to which the block should be moved (Figure 54). The configuration of three blocks constitutes the global configuration. From the perspective of a manipulating agent, only one block is important, the block to be moved. That block is the agent's local focus. The local current positions and the local goal that are computed as illustrated in Figure 54

are just reflections of that point of view. Owing to the manipulations shown in the figure, the agent can deal with a much smaller domain of 9 possible positions of the block. The whole operation is just a calculation of the difference between the current state and the goal

FIGURE 54. Translating a global state transition into a local movement of only one of the blocks. $\oplus$ indicates operation XOR (exclusive OR). $\wedge$ represents AND.

Two types of agents have been tried: random and goal-oriented. A random manipulating agent picks up a block if the agent happens to visit the starting location and the block at this location happens to be the block to move. The block is grabbed and carried until the agent, in its random journey, visits the goal location. Then the block is dropped off and the agent continues its random movement. In a small domain this option is quite competitive. In a larger workspace, there would be just too much time wasted on unproductive movements.

Higher efficiencies can be achieved with the use of navigating manipulating agents. One of the options here is to use an instance of the Neurosolver to resolve orientation problems in the blocks world; i.e., how to move quickly from one location to another. Note that the state space in which this instance of the Neurosolver operates is different from the state space of the Neurosolver controlling the solving agent: the instance of the Neurosolver employed by the manipulating agents

operates on the state space corresponding to locations of the blocks world. That is in contrast to the Neurosolver controlling a solving agent that operates in the state space of the blocks world. Except for that, in both cases it is the same Neurosolver. The training, determined by the explorations of the host agent, decides on the particular use of the Neurosolver. It is worth noting that after learning, the neurosolvers employed by different manipulating agents do not have to be the same, because each agent learns independently. That can lead to differences in handling of the same task by various agents.

FIGURE 55. A transition (a) is translated into four moves of an agent (A) without a block and three moves of an agent carrying a block. To move a block, an agent has to arrive to the location occupied by the block, pick the block up, carry it to the goal location and drop it off in there

The navigating manipulating agent uses its Neurosolver-based brain to generate two paths. First, it finds the path from the agent's location to the source location of the block to move. After that location is reached and the block is picked up, the Neurosolver searches for a path from the source to the target location, so the block can be carried along that path to its new location. An example is given in Figure 55.

There might be many manipulating agents that have received the same task from the solving agent, so there is a race to move the block. Only one agent is successful. If an agent arrives at the source location and there is no block to move, then that agent has lost the race. It returns to random wandering throughout the blocks world. The agent may be given another chance when a solving agent is distributing rearrangement plans at some later time.

I have already indicated that there are several important advantages of using the Neurosolver over other techniques that could be used to generate paths. Let us recall these advantages:

- the manipulating agent learns continuously unless it is moving a block, so it can record any permanent changes to the environment, for example stationary obstacles,
- the agent may acquire some preferred (e.g. less expensive) paths that change dynamically,
- if an agent is non-terminally impaired, then it can adapt and still use its remaining ability to move and learn to perform tasks in novel ways,
- the use of a neurosolver might yield additional benefits if the manipulating agents were to execute larger parts of re-arrangement plans,
- the Neurosolver - as a neurally-based device - may relatively easily accommodate noise factors (for example, stationary or dynamic obstacles) by accepting additional dynamic inputs corresponding to the noise in the execution mode.

8.2 Agent interactions

8.2.1 Communication between sensing and solving agents

Due to the danger of distortion in the view of the blocks world by a sensing agent (that was discussed earlier), the solving agent has to determine the actual state using appropriate heuristics. Therefore, it builds a histogram of the states reported by the sensing agents as illustrated in Figure 56.

FIGURE S6. The solving agent uses a histogram for state detection.

To determine when is the right time to determine the state, a confidence factor, γ , is computed using the ratio of the two topmost values in the histogram

$$\gamma = 1 - \frac{h_{\max 2}}{h_{\max 1}} \quad (\text{EQ 49})$$

where $h_{\max 1} = \max\{h_i, i \in [1, N]\}$,
 $h_{\max 2} = \max\{h_i, i \in [1, N], i \neq i_{\max}\}$,
 N is the cardinality of the state space,
 $\forall i: h_i < h_{\max}$

If the factor is sufficiently high (i.e., if a certain threshold is exceeded, for example 0.95), then the state with the maximum reports is accepted as the actual state of the blocks world

$$S = \begin{cases} S_{\max} & \text{if } (\gamma > 0) \\ \text{none} & \text{otherwise} \end{cases} \quad (\text{EQ 50})$$

To speed up the determination of the state, momentum is introduced in accumulating values for reported states in the histogram. The momentum changes according to an increasing function f for each state that is reported consecutively, it is reset to its initial value m if a different state is obtained from any of the sensing agents. In that way, under stable conditions, the confidence factor grows faster, because one of the slots in the histogram accumulates increasing values, so the ratio between the second top value and the top value decreases

$$h_i(t+1) = h_i(t) + \delta(t) \quad \text{where} \quad \delta(t) = \begin{cases} m & \text{if } S(t) \neq S(t-1) \\ t(\delta(t)) & \text{if } S(t) = S(t-1) \end{cases} \quad (E-Q51)$$

With a high threshold on the certainty factor it is unlikely (but still possible) that a state is deduced that is different from the actual state. Neurosolver learning is probabilistic, so even if occasionally a real state is mistaken, the knowledge should not change by much. If an incorrect state is inferred while the Neurosolver is distributing a solution, then it may lead to an earlier distribution of further steps in the plan.

8.2.2 Communication between solving and manipulating agents

The solution path generated by the solving agent is tokenized by the same agent and submitted for execution to the manipulating agents. The solving agent distributes the rearrangement plan step by step. Each of the successive portions of the plan is distributed in turn to the manipulating agents that are met. The manipulating agents that have been applied so far can handle only one state transition at a time. The solving agent suspends distribution of the next step in the solution path until reports from the sensing agents indicate that the state that is the target state of the state transition that is being currently distributed has been attained. This is similar to sensory guided manipulations known from Neuroscience, for example, hand-eye coordination. The brain seems to set the next target position for the movement of a hand only after the eye detected the previous position¹. With only one sub-goal being executed at a time, the issue of coordinating the manipulations by different agents that may attempt to perform contradictory actions is absent from the system.

The manipulating agents that have been applied in the course of my research are relatively simple. There is a much wider range of possibilities involving more complex methods. They are discussed in Chapter X: *Directions for future research*. For example, manipulating agents could deal with

¹ The position may be detected by the sensors in the joints as well, but the eye is needed for precision.

multi-step rearrangement plans. There is potential of significant improvements in performance by executing the rearrangement plans in parallel rather than sequentially.

8.2.3 Interactions between the manipulating agents

There are usually many manipulating agents engaged to perform the same task by the solving agent. That is because, as already explained, the solving agent waits with the distribution of the next task until the previous task has been carried out. Distributing the same task to many agents is beneficial taking into account the possibility of some of the agents failing due to unforeseen conditions. The solving agent distributes a task (i.e., a state transition) until it is performed (i.e., the target configuration of the blocks is set), so there is no danger of a certain part of the plan not being executed. Due to the constraints on the distribution of rearrangement plans, there is no need for the manipulating agents to communicate between themselves. Nevertheless, in a sense, the solving agent distributing a plan is a communication medium for the manipulating agents carrying out the current step of the plan. More sophisticated communication would be required if complex techniques of rearranging the blocks in the blocks world were used.

8.3 Implementation

The agents are encoded as Smalltalk classes. An agent is an object, an instance of the Agent class, that is injected into the blocks world by enlisting it as a visiting agent at a certain location through a registration function. The basic behavior of an agent is encoded as a block closure (a fragment of code) running as a Smalltalk process; i.e., it executes in its own thread of control. In pseudocode it can be expressed in the following way:

```
while true do
  begin
    freezeTheLocation;
    communicateWithOtherAgentsAtTheLocation;
    defrostTheLocation;
    performAnyAction;
```

```
    deregisterFromTheLocation;  
    moveToNextPosition;  
    registerAtTheNewLocation;  
    wait(visitDelay);  
end
```

The basic behavior of an abstract agent class is inherited by each of the agent types by proper subclassing. The behavior is modified by each subclass as required by overloading `communicateWithOtherAgentsAtTheLocation`, `performAnyAction` and `moveToNextPosition`. No agent can register/deregister if a location is frozen for communication.

Some alternatives for implementing agents in general are presented in section 10.4 *Further challenges*.

Chapter IX: Multiple solving agents

9.1 Introduction

So far, I have discussed solving block rearrangement problems with multiple sensing and manipulating agents, but only one solving agent. Such an arrangement may be prone to failure in environments hostile to agents. An example of such an environment might be a telecommunication network; for example, with failures in hardware. Even if the solving agent survives, it can be impaired in some way thereby providing an inferior quality of service. If its solving capabilities remain intact, there is still an issue of service availability. The solver might not be where it is needed and it may take some time before the request is directed to the solver. There are probably more good reasons for using multiple solvers in a distributed environment, but the ones given here should be convincing enough to justify the use of multiple solving agents. From a theoretical point of view, multiple solvers provide a basis for an attempt to deal with generally significant growth of state spaces through the distribution of the knowledge.

In Chapter VIII: *The agents*, I discussed the interactions between a solving agent and other types of agents. We will map those interactions onto the multi-solver case and deal rather with the inter-solver interactions in this chapter. Our focus is on cooperative solving capabilities of a society of solving agents. I called the system with multiple neurosolvers NGPS which stands for Neuromorphic General Problem Solver (GPS is described in Chapter III *Introduction to problem solving and planning*).

There are a number of scenarios when multiple neurosolvers would prove beneficial. The most obvious is the somewhat chaotic behavior of uniform solving agents that are injected into the environment with little or no knowledge. The agents acquire knowledge of the environment on their own pace. Assuming that noise levels are acceptable, after some time, each of the neurosolvers constructs statistically identical models of the environment. This occurs because they all interact with identical views of the world communicated to them by the sensory system (for example, the sensing agents). Both, the solving and the sensing agents are mobile, so, allowing sufficient time, all aspects of the environment will be accommodated by the knowledge of every solving agent. Of course, each of the agents may be exposed to various, not necessarily identical errors. Due to the statistical nature of learning, however, such errors will sink in the ocean of true data. The end result is a number of identical solving agents cruising throughout the world.

In this work, I will consider two other cases. Both cases involve solving agents that do not possess the same knowledge and require a certain degree of proper engineering. First, we may imagine agents that are equipped with neurosolvers with less nodes than states in the problem domain. As before, the agents can be injected in the same environment. Through the same type of interactions with the sensing agents, the solving agents acquire only the knowledge that relates to the states that are represented in their respective neurosolvers. If a request for resolving a problem is handed to them, the neurosolvers, in general, will have to cooperate. The only exception is when a single agent has a complete resolution path in its neurosolver.

The second case with non-uniform agents is when their neurosolvers have the same nodal representation of the state space, but differ in the encoded state transitions. To achieve such a distribution of knowledge, the agents would have to be trained on different partial models of the world. Alternatively, the agents might be constrained in same way, for example through the use of an input filter, so that only certain state transitions are passed to the neurosolver as learning samples. Again, in most cases, the solvers have to cooperate to solve a problem.

There is a general heuristic that can be used in any multi-solver environment if the agents fail to solve any given problem. This technique resembles means-end analysis (Nilsson, [41]) and implies that the state of the world is changed to some other state that is attainable from the current state. The hope is that the path from the next state can be found. That path, concatenated with the pre-requisite state transition, yields the solution to the problem. It does not have to be blind search. The state transition can be guided by an objective function. In the context of the blocks world, in a case of a failure, the next state in a partial resolution path is taken¹. If no partial resolution can be generated, then advice is requested from the current state and the most probable state is selected as the next state. Of course, this heuristic is not bullet proof; it may still end in failure due to a cycle. In such cases, we could try to select the next state randomly in order that any cycle is sooner or later broken.

In the following sections, I provide details of the techniques introduced in the context of the blocks world.

9.2 Uniform knowledge

In a stationary world, all agents equipped with identical neurosolvers will ultimately acquire the same knowledge. Conflicting solutions should not be generated, unless there are faulty agents that misbehave in the course of learning or later on during the resolution of a problem. Some kind of voting mechanism similar to that described in 8.2.1 *Communication between sensing and solving agents* is needed to deal with such cases. A voting mechanism with a confidence factor would be a part of each manipulating agent or special purpose solution collecting agents

The voting would also be beneficial in cases when knowledge is still being acquired or in a dynamic system where learning never saturates. Of course, if the subject system is more dynamic than the control system can accommodate, then neither voting nor other techniques can be of any

1. Most failures are due to the detection of cycles in resolution paths

assistance. We have to assume that changes to the world are slower than the acquisition process. A similar assumption was made in 8.2.1 *Communication between sensing and solving agents*.

Whatever the task delivery mechanism, it is highly probable that multiple solvers will attempt to solve the problem posed. Consequently, many agents will attempt to distribute their solution to the manipulating agents. In the procedure described in 8.2.2 *Communication between solving and manipulating agents*, we saw that the solving agent distributes the next step in the solution only if the source state in the corresponding state transition is the current state. Therefore, all solving agents distribute the same step. Distribution of the solution can be blocked for short transitory periods when the new state of the world is being still determined. The manipulating agents compete for execution of that step; one succeeds; the rest give up. A solving agent detects the next state after the transition independently of the fact of whether the manipulating agent that carried out the task was instructed by this solving agent or another. Hence, the execution of the plan proceeds in every solving agent.

9.3 Distributed knowledge

As I have already stated, through a proper engineering of the agents, the knowledge that the associated neurosolvers acquire may vary from agent to agent. There are two cases considered here: distributing the states and distributing the state transitions. The former corresponds to partitioning the state space between the neurosolvers; the latter - to partitioning the space of state transition. Both cases limit the size of the involved neurosolvers measured either in the number of required nodes or inter-nodal connections. In the following two sections I examine systems of multiple solvers built on those two approaches.

9.3.1 Partitioning state transition spaces

Even though the neurosolvers used in the solving agents may have identical sets of nodes, in the real world situation, it is possible that they learn only certain of the state transitions if we consider that the agents may be geographically and time constrained. In a simulated environment, we may

train such solvers on various data samples that define different aspects of the behavior of the subject system. If the environment can be controlled, then each of the neurosolvers can be trained in the real world that is forced to exhibit only limited behavior, different for each neurosolver. The end result is identical in all such cases. We end up with a number of neurosolvers which encode different parts of the system behavior, i.e., a different set of state transitions

FIGURE 57. Three neurosolvers, N_1 , N_2 and N_3 , with the same nodes but different connections. The neurosolvers exchange their nodal activities through the communication channels shown as grayed out vertically oriented arrows.

The neurosolvers must communicate in some way in order to solve a problem within a search tree that spans more than a single neurosolver. In the case of a distributed space of state transitions, we may allow for exchange of activity between the neurosolvers. A case involving three neurosolvers, N_1 , N_2 and N_3 , is illustrated in Figure 57. Each of the neurosolvers has a number of state transitions encoded in its inter-nodal connection. The grayed out vertically-oriented arrows indicate that the neurosolvers exchange activities for each node. If the goal is injected in N_1 as illustrated

in the figure, then the activity spreads not only along the backward chains inside that neurosolver, but is also transmitted to the corresponding nodes in other neurosolvers. A similar process controls the spread of activity in every other neurosolver. Therefore, at some point, assuming that the search drive is sufficient, the search activity arrives at the neurosolver that has a node receiving the triggering, thalamic input corresponding to the current state of the subject system. That node fires and its activity is propagated along its forward chains. Again, the activity is shared with all other neurosolvers. The next node to fire might be in the same or in another neurosolver. That process proceeds until the goal state fires; i.e., the problem is solved. The search activity is fed into the goal node as the limbic input. It may be fed into just one neurosolver or into all; one way or another, every goal node in each neurosolver gets the search activity. If there is a backward link from that node in at least one neurosolver, then the activity spreads into the search tree.

The following is a log from a session with the NGPS illustrated in Figure 57. All neurosolvers have 9 nodes, but they are trained with different sets of state trajectories. After training, the neurosolvers are associated and instructed to cooperate with a keyword `broad` that enforces the exchange of nodal activity. When asked for a solution to a problem that spans all neurosolvers, the NGPS returns a composite resolution path.

```
=> new 1 1 9
=> new 2 1 9
=> new 3 1 9
=> id 1
From now on using neurosolver 1
=> learn 1 2
=> learn 2 3
=> id 2
From now on using neurosolver 2
=> learn 3 4
=> learn 4 5
=> id 3
From now on using neurosolver 3
=> learn 5 6
```

```
=> learn 6 7
=> id 1
From now on using neurosolver 1
=> learn 7 8
=> id 2
From now on using neurosolver 2
=> learn 8 9
=> associate all
=> cooperate broad
=> solve 1 9
1 2 3 4 5 6 7 8 9
```

We can set the debug level to a level that forces the output of the partial solutions with corresponding neurosolvers. The following log illustrates how the solution to the problem has been constructed. The numbers in parentheses indicate neurosolvers; others are identifiers of nodes.

```
=> debug 2
=> solve 1 9
Solving 1->9
->1(1)->2(1)->3(1)
Neurosolver 1==> 1 2 3
->3(2)->4(2)->5(2)
Neurosolver 2==> 3 4 5
->5(3)->6(3)->7(3)
Neurosolver 3==> 5 6 7
->7(1)->8(1)
Neurosolver 1==> 7 8
->8(2)->9(2)
Neurosolver 2==> 8 9
1 2 3 4 5 6 7 8 9
```

9.3.2 Partitioning state spaces

In order to partition the state space, each of the neurosolvers is constructed from a set of nodes that are associated with a subset of states from the state space. This is a more difficult case than

partitioning the space of state transitions, since it takes us back to our discussions of encoding state spaces in neurosolvers. I stated previously that such issues are not the main concern of my thesis, but I feel obliged to review some possibilities here. Some mechanisms for achieving the partitioning of a state space into subspaces encoded into individual neurosolvers are as follows:

- hardwiring,
- constraining the environment,
- constraining the sensory system and
- state elimination through interactions between the solving agents.

Note that we are talking about distributing the states/nodes between the neurosolvers. Learning state transitions may be performed at the same time (it is described separately in section 9.3.2.5: *Learning*), but it does not need to. Ideally, both processes would be simultaneous, but as I explained in the earlier parts of this thesis, acquiring the state space automatically is a much longer process than obtaining the state transitions. It may be a waste of time to learn state transitions when the associations between states and nodes are not well established. The mechanism for learning state transitions may be left inactive until the error in the state assignment falls below a certain threshold (ideally, when it reaches zero).

9.3.2.1 Hardwiring

If the system is well understood, the state space can be partitioned by hand. The states from each partition are assigned to one of the neurosolvers. After distribution of the states, all neurosolvers can be trained on the same model (same set of state transitions). They acquire only the state transitions that involve the states represented by one of their nodes.

For the purpose of testing the solving capabilities of a society of neurosolvers presented in this thesis the neurosolvers were hard-wired.

9.3.2.2 Constraining the environment

Each of the neurosolvers might be exposed to an environment that exhibits only a partial model of the overall behavior. That can be achieved both on-line and off-line. In the on-line case, an agent is injected into the world that has been constrained in some way, so only certain states can occur. As with a single neurosolver, some kind of self-organization can be used to acquire the states. In the off-line case, the neurosolvers are trained on the data samples that have been hand-tailored or collected from constrained environments.

9.3.2.3 Constraining the sensors

The third option considered here is associating the states with the nodes by constraining the sensory input to a neurosolver. Such a constrained sensory system delivers only partial models to the neurosolvers. Each neurosolver should have a different input filter, so it acquires a different part of the state space. A possible realization of such a filtering strategy could be allotting time slots to the solving agents for communication with the sensing agents. A solving agent would be allowed to interact with sensors only during the allotted time slot. If each state was presented only once and each state change was synchronized with the allotted time slots, then it would be associated with a node in only one neurosolver. However, if the behavior of the subject system cannot be controlled, then assigning the appropriate filters might be difficult.

A filtering strategy can also be used to attack problems in a hierarchical way. In this case, the filters deliver to the neurosolvers views of the world at various levels of abstraction.

9.3.2.4 State elimination through interactions between the solving agents

In this alternative to partitioning state spaces, each solving agent is equipped with a capability to exchange information with another solving agent. If two solving agents meet, they exchange data about the texture of their neurosolvers. Any state that is represented in each of the agents is removed from the one whose cardinality is higher. This rule ensures an even distribution of states between the agents. After some time, each state is represented only once and the distribution of

the states amongst the agents is statistically even (i.e., there might be some differences in sizes depending on the statistical patterns of the inter-agent contacts).

FIGURE 58. Three neurosolvers, N_1 , N_2 and N_3 , with different nodes interconnected through external ports.

9.3.2.5 Learning

As we saw earlier in this thesis, the process of learning state transitions is efficient after the distribution of states is stable. We can consider it to be the second phase of knowledge acquisition, and it has been our main focus. The state transitions between two states that are represented in the same neurosolver are learned in the same way as in the case of a single neurosolver. If the successor of a state transition is not present in some neurosolver, then a forward external port is created for the output of the lower division of the predecessor to the lower division of the successor. In the opposite direction, if the predecessor happens to be in another neurosolver, then an external port is created for the output of the upper division of the predecessor. The external ports are shown as single circles in Figure 58.

There are three neurosolvers, N_1 , N_2 and N_3 , in Figure 58. They have disjoint sets of nodes. The internal connections are shown as horizontal arrows. The state transitions between the nodes of different neurosolvers are shown as dashed arrows with a vertical orientation. They are not actual connections, but just ports (windows). The signal is passed through such connections when a neurosolver contacts another neurosolver that contains the node associated with the target state of the connection. There might be a delay in the spread of activity between neurosolvers, but sooner or later the agents will meet and the signal will be exported. The communication that propagates the search activity from the source in N_1 to N_2 may look something like the following

H1, I am N_1 . I have a message for node X.

H1, I am N_1 . I do not have node X.

Bye, then.

Bye.

H1, I am N_1 . I have a message for node X.

H1, I am N_1 . I do have node X. Give me the message.

Here it is: <message>.

Thank you. I will pass it to node X.

Thank you. Bye for now.

Bye.

Except for the change in the schema of spreading activity, the neurosolvers behave exactly as a single neurosolver.

The following is a log from a session with the NGPS that illustrates the ideas on a simple example. There are three neurosolvers created. Each has three nodes, but all of the nodal identifiers are different. The second number in the new command indicates the base for the identifiers in the neurosolver (see section 6.6.3 *Application Program Interface (API)*). The same state transitions as in Figure 58 are learned. All neurosolvers are associated, i.e., they are forced to exchange information (we are not simulating agents here). Then the command to cooperate through external ports is

issued (`cooperate deep`) and the system is asked to resolve a problem $1 \rightarrow 9$. The response is a correct solution path spanning three neurosolvers.

```
=> new 1 1 3
=> new 2 4 3
=> new 3 7 3
=> learn 1 2
=> learn 2 3
=> learn 3 4
=> learn 4 5
=> learn 5 6
=> learn 6 7
=> learn 7 8
=> learn 8 9
=> associate all
=> cooperate deep
=> solve 1 9
WARNING: Using ALL neurosolvers
1 2 3 4 5 6 7 8 9
```

Again we may request the distribution of the partial solution paths amongst the neurosolvers by setting an appropriate level of debugging as illustrated in the following

```
=> debug 2
WARNING: Using ALL neurosolvers
=> solve 1 9
WARNING: Using ALL neurosolvers
Solving 1->9
->1(1)->2(1)->3(1)
Neurosolver 1==> 1 2 3
->4(2)->5(2)->6(2)
Neurosolver 2==> 4 5 6
->7(3)->8(3)->9(3)
Neurosolver 3==> 7 8 9
1 2 3 4 5 6 7 8 9
```

9.3.3 Scaling to larger problems

Although I have not found any theoretical limitations to the scaling of the techniques of associating neurosolvers to deal with space partitioning, the solving capabilities did deteriorate in the simulated environment, particularly with an increasing number of the solvers. For example, many blocks world problems were not solved.

There are a number of reasons. In an ideal case, the association between the neurosolvers should be direct, with immediate propagation of activity. This is difficult to simulate in software, as each of the neurosolvers has to go through its own computing loop. I experimented with a number of tightness levels for the associations, and I have come to the conclusion that the tighter the association, the better the solving capabilities of the society of neurosolvers. Taking into account that the neurosolvers are thought of as the brains for the agents, tight associations would be difficult to maintain and be very inefficient. This would make the execution of the activity spreading algorithms depend on the frequency of inter-agent contacts thereby lengthening the process that is already quite time consuming in the software simulation. Without very tight associations, some neurosolvers may make mistakes with only partial information (incomplete propagation of activity). Others may get into loops that indicate failures before they get more details of the environment. Lack of true parallelism in the simulator can also contribute to construction of false search trees.

In the context of the blocks world, the simulation parameters have to be adjusted. In general, if the delta with which the drive grows is increased, then the solution is found faster or a failure is indicated sooner (that is hardly surprising; we already observed similar behavior in a single neurosolver in section 7.4: *Experiments*). There is a danger, however, of overstepping the search drive level that could trigger the solution if the step size (activityDelta) is too large. There are certain limits in the simulator, like the range for double precision numbers, that prevent sufficient propagation of activity between neurosolvers in some cases. In general, neurosolvers tend to give up sooner. That is, because they attempt to perform computations on their own in between the

exchanges of the activity with others and without a proper flow of activity into certain parts of the search tree it is more likely to get into cycles. The use of MEA-like heuristics (means-end analysis) operating behind the solve hard command (see section 6.6.3: *Application Program Interface (API)*) helps in many cases.

All problems that became evident during my tests with multiple neurosolvers systems within the blocks world domain were inherent to the simulation and to the acceleration techniques used to speed up computations. The lack of evident theoretical limitations to the solving capabilities of the society of multiple neurosolvers leads me to believe that a hardware implementation of the NAGPS with tight (hard-wired) inter-nodal, inter-solver associations would be as efficient as a single neurosolver. I will shortly examine such opportunities in section 10.2: *Implementation*.

Chapter X: Directions for future research

In this chapter, I survey research that could be worth further explorations. The following sections examine each of the elements of the system that I have built.

10.1 The Neurosolver

10.1.1 Expanding the architecture

The current version of the Neurosolver is analogous to first order Markov chains (see Chapter V: *Introduction to Markov Models*). The analogy yields one research thread which is to expand the model by adding second (and possibly higher) order connections that would make it similar to second (or higher) order Markov models.

Figure 59 illustrates the difference in functionality of the Neurosolver with first order and second order connections. The second order connections add an element of momentum to the movement along the resolution path, so that at any intersection of statistically equivalent directions of the further search (at node X in the example) the right direction is chosen correctly (see the figure). The purpose of this extension would be to make the search more directed by the past, but not necessarily immediately preceding states.

FIGURE 59. If two sequences are crossed during the learning (a), then with only first order connections (b) the search is split into two equally probable branches. With secondary connections (c) the search branch that is a part of the originally learned sequence will be more activated.

10.2 Implementation

10.2.1 Simulation

Smalltalk proved to be a convenient language to encode a simulation environment involving agents because it provides internal processes. There is no limit on the number of processes, except for the machine resource limitations. The C++ implementation made the neurosolver code very efficient increasing the performance significantly. Emerging new technology, Java, seems to put together the flexibility of the Smalltalk with many of the virtues of C++. However, it is code mobility that is inherent to Java technology that is the most important concept in the context of distributed, multi-agent environments.

If consider locations visited by agents as independent computing units, and that is the case with the nodes in a telecommunication network, then Java is probably the best currently available technology agent encoding. Agents can be implemented as Java applets and transmitted over the network using the HTTP protocol. They are executed by a local Java virtual machine that can be implemented in software or installed as a hardware chip. It is not my intention to follow this thread any further, because this is more a technological than a scientific issue. Nevertheless, I mention it here as a very interesting path to pursue in, perhaps, somebody else's Master's thesis or project. It naturally falls directly into the area that will be considered in section 10.5 *Network Management applications* concerning the outlook for possible applications of my research in Telecommunications.

10.2.2 Analog implementation

A detailed consideration of the possibilities for implementation of the Neurosolver on an analog computer (in hardware) is beyond the scope of this research. Nevertheless, it would be interesting to see autonomous robots with such hardware units in the future. An example of a possible hardware architecture for an implementation of the basic Neurosolver is illustrated in Figure 60. In the

FIGURE 60. A possible hardware architecture for the Neurosolver. The connection at the line crossings annotated as α and β represent, correspondingly, forward and backward connections that are affected by the transition $B \rightarrow D$. α and β represent the connection for the opposite transition, $D \rightarrow B$.

example, each node has vertical and horizontal double links that are connected to the links of all other nodes. The connections are adaptable and assigned in the way suggested in the figure. The rows correspond to source nodes in transitions; the columns - to targets. For example, the connections α and β are modified for the sequence $B \rightarrow D$. α corresponds to the forward connection between the lower divisions of the node B and D, and β - to the backward connection between the upper divisions of D and B. The case for the reversed transition, $D \rightarrow B$, is also shown.

The architecture proposed in Figure 60 implements a completely interconnected neurosolver; i.e., every two nodes have to make a complete set of two contacts. Not all of those contacts must be full-blown connections, though. If there is a succession relationship between the two nodes, then a connection would be made out of something that otherwise merely provides an opportunity for a connection. It is a prototype for an analog device, so there are no simulation cycles. Every node is updated immediately. The order of firing nodes would be guaranteed due to the natural delays in the spread of electricity caused by the resistance of the conducting media. It was a challenge to build a such parallelism into a software simulator. It would be equally challenging to provide an adaptation mechanism that could be used to implement the Neurosolver's connections (i.e., α , β , γ and δ) in hardware. In my opinion, an efficient mechanism for spreading search activity should, in fact, be easier to achieve on an analog computer. It should not be difficult to select the winner of the competition by hardware means by implementing a MAX function.

In section 9.3.3: *Scaling to larger problems* of Chapter IX, *Multiple solving agents*, I expressed the opinion that NGPS using multiple neurosolvers would benefit considerably from a hardware implementation. Exchanging data in the case where all neurosolvers have the same nodes is not a problem. Direct links can be put in place between the corresponding nodes in each of the neurosolvers. The other case - the partition the state space - would pose more of a challenge due to the need to address the destination of the external connection through an external port. A broadcasting scheme that would put the data on a universally accessible bus could be a potential solution. All neurosolvers would receive the broadcasted message, but only the one with the addressee

would actually consume the data. This scheme is remarkably similar to a blackboard system: the neurosolvers act as long as they can without communicating with others. If the help of an outside party is needed, then a request is posted on the blackboard. Assistance can come from any of the neurosolvers that have access to the blackboard. Of course, in the case of the Neurosolver, all what is communicated is the activity.

10.3 Agents

In the main part of my thesis, I focused on the analysis of the solving capabilities of the Neurosolver and a community of several neurosolvers. I also addressed some of the issues concerning the use of agents in a distributed environment. It is a very rich domain that naturally leaves considerable scope for future exploration. In the following sections, I will describe some future possibilities in that area of research. I will explore at the issues concerning each type of the agents sensing, solving and manipulating as well as the issues concerning all agents considered in my research.

10.3.1 Extensions to sensing agents

In relation to other types of agents studied in this work, the sensing agents are the simplest entities. There are correspondingly fewer avenues for future explorations. Still, there are a number of issues to address.

The sensing agents could be allowed to share their respective views of the world. An argument against it is that the sensing agents should probably remain small, and therefore efficient data collectors, so they can best fulfill their mission. As the solving agents are the evaluators of the overall state of the block world from the data collected from many independent sources, the need to keep the knowledge uniform inside sensing agents by exchanging the missing chunks is not obvious and perhaps even undesired.

At present, a solving agent has an interface to represent a rearrangement problem directly. This is an easy technique, but hard to implement without agent tracking capabilities. Even with a tracking system in place that would detect the location that is being visited by an agent, it may be difficult to communicate with the agent directly. There are a number of options to consider. Here are some of them that relate to the sensing agents. I will also return to this subject in section 10.3.3:

Enhancements to solving agents

The idea of using sensing agents in goal presentation is that they could discover not only the state of the world, but also the desired state that would be distributed throughout the world. For example, in the context of the blocks world, that would require another set of blocks to indicate the desired configuration. This is just a representational issue. The agents would just be modified to accept a new representation (3 bits vs. current 2 bits) and the solving agent would be able to evaluate the current state and the goal (if there is one). To present the current and the desired block arrangements, an additional bit (the first bit in the representations shown below) could be used to indicate a desired or actual configuration, so the three bits would have the following meaning:

- 000 for an empty location,
- 001, 010 and 011 for one of the three colors that are desired in the location, and
- 101, 110 and 111 for one of the three possible colors of the block that actually occupies the location.

Another approach, retaining just one set of blocks, is to tell either the sensing agents or the solving agents what the current configuration of the blocks means. The communication with the solving agents must be worked out in any case, so it seems reasonable to concentrate on the solving agents. The role fulfilled by the sensing agents would not then change, but the solving agents would decide how to interpret the data reported by the sensing agents

10.3.2 Extensions to manipulating agents

It is very important to emphasize that the essential feature of rearranging blocks in the blocks world is that all configurations involved are valid and the order of the configurations is exactly as seen in the past. Without this constraint, the blocks could be moved in a chaotic way in parallel by a bunch of manipulating agents focusing just on the goal state without consideration for what the intermediate configurations might be. Such a scheme would certainly include configurations that are invalid or not related by state transitions. The proper way of rearranging the blocks is controlled by the solving agent that coordinates the distribution of the resolution plan. Therefore, in the context of the blocks world, we cannot allow the solving agent to lose its grip on plan distribution, unless some other mechanisms are put in place that could enforce configuration constraints. It certainly would be interesting to attempt to carry out the reconfiguration plan in parallel by multiple manipulating agents executing various parts of the reconfiguration plan. Due to some fuzziness in the agent environment, there would be many serious questions about plan coordination and resolution of arising conflicts.

In the course of my studies, I have looked at random and navigating agents as explained in Chapter VIII: *The agents*. The random agents move randomly all of the time. When such an agent is assigned a task by a solving agent, then it continues to move randomly. When it hits the source location it picks up the block and carries it in, again, a random direction. In the context of a blocks world it is reasonable to enforce a policy that an agent carrying a block cannot visit the locations occupied by other blocks. When the target location is visited, the block is dropped off and the agent returns to its random wandering. If more complex tasks, multistep subplans, were distributed, then the random agents could repeat the block movement loop for each step in the plan without any complex coordination scheme.

Navigating agents use a neurosolver to solve navigation problems of first getting to the source location and next to the destination in the shortest possible time. With the weaker constraints on block configurations, the plans produced by the solving agents are detailed with each step involv-

ing just one location. If the option to use stronger movement constraints are used, then the plans include fewer details. In such a case, the agent has to deal with more difficult tasks of execution subplans involving the movement of a single block through a number of locations.

The implementation of agents and the blocks world assumes that there are no obstacles in the workspace that could prevent a block from being moved through certain locations. A capability taking that into account could be implemented for all agents. Neither sensing nor solving agents carry blocks, but they may require such a capability in order to detect whether a location is reachable. An obstacle can be stable or dynamic. A manipulating agent using a neurosolver could adapt its navigating abilities to a stable obstacle. However a dynamic obstacle does not leave permanent traces in the agent's brain, because the traces are based on the statistical laws of large numbers. A moving obstacle causes the distortion of statistics for only a short moment. An interesting question to answer is whether an agent would avoid an area that is visited by a dynamic obstacle most often. For that purpose, we could confine the obstacle to a certain number of locations that can be visited, so that sometimes an agent could go through a location in this area, but another time it could not.

10.3.3 Enhancements to solving agents

10.3.3.1 Multiple solving agents

In Chapter IX, *Multiple solving agents*, I examined the NGPS, a solver with multiple neurosolvers. In the context of agents, there are several issues that have not been addressed. They are not directly related to the problem solving which is the major theme of this work and, as such, are left for the future. These issues were already discussed briefly in section 9.3.3, *Scaling to larger problems* and concern assuring proper levels of communication between the agents carrying various neurosolvers that need to exchange activity. As we saw, the strength of coupling between the agents is critical to the problem solving capabilities of the whole society.

Another issue is the distribution of the overall plan. In the simulator of the NGPS, the final touch to the solution, like removing cycles, is done by the controller of the neurosolvers. It is the same controller that manages the exchange of data between the neurosolvers. With agents, such a controller disappears, so that the solution is either distributed as is by the neurosolvers firing their nodes or the firings are collected by some medium that would process and then distribute it to the manipulating agents. Such a medium could be arbitrarily complex with respect to the final resolution path as well as its distribution (for example, in parallel chunks as briefly discussed in section 10.3.2: *Extensions to manipulating agents*).

10.3.3.2 A divide et impera approach to hard problems

An alternate approach to address hard problems (i.e., ones that cannot be resolved) is to divide them into subproblems and then solving the overall problem by solving the subproblems and combining the partial solutions. In the blocks world context, if a problem fails, the *divide et impera* (divide and conquer) approach is to instruct the manipulating agents to move the blocks to any valid configuration and attack the two subproblems constructed by splitting the problem current $\rightarrow$ goal into current $\rightarrow$ new and new $\rightarrow$ goal. The agent would attack the first subproblem and return to the second part later or it would assign the second subproblem to another solving agent (in a multi-agent case). The division scheme would proceed as long as required. There is still a danger of a complete failure, but the chances for a success may be increased. If the division point is reachable from the current state by a single state transition, then this technique resembles means-ends analysis (Nilsson, [41]). In general, any division point could be chosen, but the better the selection, the better chance for a success. The divide and conquer methodology, as any technique based on heuristics, does not guarantee that the problem will be solved. Finding a proper heuristic might not be easy, however.

10.3.3.3 Problem presentation

In section 10.3.1, *Extensions to sensing agents* I analyzed the use of a sensing agent as an indirect means of problem presentation. I indicated that there must be a more realistic way of communica-

tion with the solving agents. If a communication link is established, then the presentation problem is easier to address. Two options come immediately to mind:

- communication posts (a.k.a. mailboxes)
- communication agents.

A communication post would be a location that has a repository of messages for the agents that visit the site. The agents are mobile, so assuming their random movements they would visit such centers from time to time. The frequency of such visits depends on the density of the posts. The message could be directed to the solving agents. It could be a complete problem to solve, or just the information on what it is that the sensing agents are reporting at that specific time. It may be a current or a desired configuration. In any case, the task of the sensing agents would not change, but the solving agents would know what they are collecting.

Alternatively, the sensing agents could read the posted messages, retain the messages for themselves and use them to determine what they are collecting. Each state reported to the solving agents would include the information on whether it is a current or a desired state. Taking into account that the number of the sensing agents is considerably higher than the solving agents and that the sensing agents are faster than the solving agents, this approach might be superior.

It is not certain at this moment whether the use of communication agents in place of the stationary communication posts would be beneficial. The communication agents could be injected into the world carrying a message to the solving agents. Alternatively, the sensing agents could play the role of communication agents. As explained before, they would collect messages from the communication posts, but instead of using that information themselves, they would pass it onto the solving agents instead. An obvious problem with injecting new agents into the world each time there is a need to pass some information is that it requires a deterministic method for removing them. When should they be removed: after meeting just one solving agent, a number of them, after a timeout or some other criterion? There could be a number of communication agents always

present in the world and communicate with them using the communication posts, but then why not extend the behavior of the sensing agents as suggested before. An argument for the introduction of another type of agent could be that efficiency is higher if the sensing and communication agents are kept as simple as possible.

The preceding discussion also brings an interesting idea of a moving target – a situation when the goal is not stationary but changes constantly. The society of agents could act as a target tracking system. This thread seems to be reaching too far for the scope of this work, so I do not intend to follow it any further.

10.3.4 Extensions concerning all agent types

10.3.4.1 Agent reincarnation

In the real world, agents could decrease, so there should be some way of reincarnating them. The solving agent could have some way of evaluating such a need and inject sensing or manipulating agents into the world as needed. Should it work in the other direction as well, i.e. if there are too many agents, should they be eliminated?

What about the solving agents? Should they be able to clone themselves if there are too few of their kin (remember, knowledge diversification)? There is an immediate question as to whether we would want the solving agents to share their knowledge. In cases of diversified knowledge, we obviously do not desire such a capability. It seems reasonable, however, that if a new agent is introduced to the world, it should learn faster by being told rather than from its own experience.

One way of discovering the density of agents in the world is to engineer checkpoints that could measure the frequency of contact with various types of agents. A checkpoint can be stationary, for example at a certain location, or it could be an agent (for example a solving agent). If we adhere to the desire of keeping the agents simple by limiting their tasks, then perhaps a better idea would be to introduce another type of agent, administration agents, whose sole task would be to keep the numbers of all agents at certain levels. Such agents do not have to be fast and efficient, so they

3

OF/DE

3

PM-1 3 1/2" x 4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

could assume more administrative tasks as needed. Going back to the density checkpoints, if the frequency of meeting a specific species of agents falls below a certain predetermined threshold, then new agents of that type are generated. A serious problem requiring a solution is that injecting new solving agents without any previous knowledge requires them to learn. It could be done rather inefficiently *from examples*, starting from scratch with many logistic issues to resolve, or by *being told* by other solving agents. The second case is addressed in the next section.

10.3.4.2 Sharing skills

There are two hypothetical goals of sharing skills by agents: to improve knowledge or to pass on knowledge. The first goal cannot be pursued in the context of solving agents, since the whole idea of having agents in the first place is to distribute knowledge. If we allowed for knowledge sharing, that would lead to many solving agents with the same knowledge base. Similar arguments can be used against sharing knowledge by the manipulating agents operating in large workspaces. It is critical to preserve the variety of manipulation agents, so that the navigating knowledge is completely covered. Otherwise, after some time, the skills of all manipulating agents would be the same and there could be certain paths in the workspace not known to any of the manipulating agents. No block could be moved between the endpoints of such paths.

In the case where agent cloning is allowed, the issue to address is avoiding cloning of the same agent of the community of multiple solving agents. Assuming that the desired density of agents is given as a window rather than a single threshold and taking into account the random nature of possible contacts of a new agent, over time the number of solving agents with each part of the knowledge base should stay balanced. The danger of extinction of a certain species is still there and should be addressed or at least measured. One possibility that seems a little inflexible, but that would certainly work, is to create a library of prototypes for various types of solving agents. Each kind of a solving agent with a specific part of the knowledge could be labelled appropriately, so the density of each particular type would be measured rather than the density of any solving agent.

Cloning is not particularly important for manipulating agents, because they can learn the orientation in the world relatively easy (they do not need the reports from the sensing agents and they do not deduce the states on which they operate). In addition, due to the possibility of dynamic obstacles, it is better if the manipulating agents have the latest possible knowledge of the world.

As we already discussed in section 10.3.1 *Extensions to sensing agents*, sharing the locally collected views of the world by sensing agents is questionable, because the accuracy of the state evaluation by the solving agents depends on the timely arrival of reports. The reports should be as recent as possible. Averaging the local views of the sensing agents is done inside of the solving agent.

The rearrangement plans that the solving agents distribute to the manipulating agents can be shared between the latter. That may lead to a faster execution, because de facto there would be more distribution channels. Consequently, more manipulating agents would be aware of the plan, so the chances of one of them starting the rearrangement would increase. On the other hand, the communication between the manipulating agents requires certain overhead that, in realistic applications, would slow down the agents.

The agents carrying messages (sensing, communication or administration) could distribute the messages not only to their ultimate destinations - the solving agents - but also to other agents capable of carrying messages. That would considerably increase the speed of the communication channels.

10.3.5 A word on agent mobility

Throughout this document I assumed the mobility of agents. In today's literature, mobile agents are often called *worms*, and a set of agents that exhibit some sort of intelligence only in a group - *swarm intelligence*. There are certain important questions to answer regarding mobility. An agent may actually be mobile, i.e., the body of the agent can be moved to another location. Alternatively, the body can be stationary, but cloned to every possible location. Instead of transmitting

the whole agent, only control is passed that activates its local copy. The second option seems to save bandwidth, but would it be sufficient in a case of agents similar to the sensing agents described in this document? If the body of the agent is encoded by some highly expressive language, than its size might be negligible if related to the size of the data that would nevertheless have to be transmitted.

In any case, this thread seems like a fertile ground for interesting research, particularly in the context of telecommunication networks.

10.4 Further challenges

There are many doors that have gate opened for possible research directions spinning off from this project. Many of them were highlighted in the preceding sections. The areas that require distributed, autonomous and adaptive approaches to problem solving provide opportunities for applying the ideas from this thesis. A number of examples of prospective practical applications of the results of my studies will be addressed in section 10.5 *Network Management applications*. The following are some additional thoughts on the general extensions.

It would be interesting to include additional actions that an agent has to perform on a block before the block can be moved. Instead of using simple blocks, complex units could be required, so before the unit is moved, another problem space must be employed to actually construct what is required. The solution path could span through two, or more, state spaces and would imply hierarchical searches. For example, in the telecommunications context, an agent could be required to test the sanity of the visited node and attempt to repair the node before moving to the next node, carrying, for example, a chunk of communication data like a cell or packet. In another example, in a job scheduling task, a Neurosolver-based agent can be asked to move an object from the inventory to the production floor. However, if the requested object is not available, then another request to make the desired component must be issued that would be handled by another type of the Neurosolver. With such ideas we are entering the world of hierarchically trained neurosolvers.

Obtaining a timely response from the system has not been taken into account so far, but it certainly would be challenging to introduce time as a constraint on providing a solution. An agent would have to make a trade-off between the accuracy of the solution and the speed with which a solution is found. It implies the use of more complex solving agents that can do more than just indicate their failure. The heuristics introduced earlier to deal with failures would be expanded to accommodate time constraints. There is a parameter that controls the maximum length of the propagation of activity, but it is not certain whether it could be successfully used for that purpose.

10.5 Network Management applications

The potential for practical applications of the Neurosolver has always motivated my research. In this chapter, I take a look at some of the opportunities in Telecommunications. Network Management is of particular concern because of my, and my supervisor's, interests and the experience in the area that I have gained over a number of years in the industry. I hope that ultimately some of the ideas presented in this section will be put to work.

Let me start with a claim that the concepts from the blocks world can be easily mapped into concepts in Telecommunications. The locations of the blocks world can correspond to nodes of a telecommunications network. A configuration of the blocks can be considered to be a state of the network. Reconfiguring the blocks in the blocks world may correspond to reconfiguring a telecommunication network. That may involve actions like reprovisioning the components or modifying the topology. Certain actions that are not possible to be performed automatically or that require confirmation involve issuing external requests to the network operators.

In this section, I will concentrate on two areas that seem to be critical in managing networks.

- self-repairing networks using problem recovery schema and
- fault prevention and detection.

The agent paradigm is particularly well suited for a distributed environment such as a telecommunication network. Most of the agent-related material presented in this thesis is directly applicable in the Network Management context and will not be repeated in this section. For example, obtaining the real state of a network has, historically, been one of the hardest problems in Network Management. The use of agents similar to the sensing agents analyzed in this work may bring a reasonably acceptable solution. Solving agents can be prototypes for some type of network doctors cruising throughout the network and healing whatever they can by using manipulating agents. That would postpone the need to involve a human operator, decreasing the costs of maintaining the network by doing so. Assuming the adaptability of the agents, little or no programming would be needed to accommodate the dynamics that are inherent in any telecommunications network.

10.5.1 Problem recovery

One of the most critical aspects of Network Management is the provision of an efficient mechanism for resolving network problems. Traditionally, network operators observe the reports generated by the network and undertake appropriate actions upon detecting faults. Usually, there are huge amounts of various types of data generated by the network, so manual monitoring is an onerous task. There have been many techniques used to limit the flow of data to the operator. Some of them involve methods from Artificial Intelligence: expert systems, neural networks, case-based reasoning, etc. A successful example of applying a rule-based system for detecting network problems is described in (Bieszczad, [11]) and (White, Bieszczad, [62], [63]). Unfortunately, even partially self-repairing networks are remain unrealized. Below, I present a possibility for realizing such systems using the results of my research.

Certain states of the network correspond to network failures. Others indicate healthy conditions. In this context, moving from a state indicating an error to another state that is healthy corresponds to performing a problem recovery routine. A single neurosolver or a number of neurosolvers can be used to acquire problem resolution methods by observing and recording actual resolution processes performed by the network operators. Of course there are certain conditions that require

hardware repairs that are not suitable for automatic recovery. Nevertheless, many others involve just issuing certain commands to appropriate elements in the network. We can use such commands for training and further adaptation. If we allow the Neurosolver to observe these recovery routines, then if a similar fault occurs in the future, it can be repaired automatically. The Neurosolver applied in such a practical application can have a certain degree of generalization added, so it discards the information about specific components of the network. In that way, a similar fault involving any component can be treated in a similar manner. Figure 61 illustrates that technique.

Learning a problem recovery routine

Problem recovery

FIGURE 61. Acquiring, recalling and applying a problem recovery routine by the Neurosolver.

on a simple example. Each node represents a state of the network¹. A transition from one state to another may be associated with a certain action. A set of such actions corresponding to the transitions in the trajectory from the erroneous state to a correct state constitutes a resolution of the problem. Initially, the system starts as a *tabula rasa* and is incapable of resolving any problems.

¹ This state space would be very large if all network parameters are taken into account. However, the size of the space could be limited by working with various views through the use of appropriate filters. It will be discussed with more detail later in this chapter.

By observing the state transitions and associating them with the activities performed in the course of actual recovery from the problem, a trace, a temporal pattern, is built between the erroneous state and the correct state. The routine (path) that is being learned at this moment is drawn in black in the picture, there are two other earlier acquired traces drawn in gray. If the fault occurs again, then the system can perform the same routine automatically. To do so, the activity is applied to the desired state or a number of such states. Search trees are built, and one of them may be triggered by the current erroneous state. If that happens then a recovery procedure can be activated. Otherwise, another learning session will be given by the network operator.

A state transition can be associated with an action that forces the transition in an adaptive way (e.g., by the use of a backpropagation or time-delay neural network (see section 4.4 *Time-delay network*)). A network that takes the source and target states as its inputs can be taught to associate them with an action represented as a certain output. After the action is applied, the system state changes. The new state with the next state in the recovery routine will in turn activate another action. Similar sequences can be generated by recurrent networks (see section 4.5 *Recurrent network*), although such networks are not able to perform searches similar to the Neurosolver's.

Note, that the network can be considered as a set of subsystems (hierarchical, functional, topographical, etc.). Each of such subsystems could be managed by a different neurosolver or set of neurosolvers. Resolution of problems at a higher level of the hierarchy can be translated into lower level tasks that are presented to the neurosolvers capable of solving them. Training such neurosolvers would just involve proper engineering of the sensory system. The sensory system would be like a net that catches the states only at a certain level of abstraction.

10.5.2 Fault detection

Fault recovery implies a mechanism for fault detection, a recovery routine can only be activated if a problem has been detected. Detection of network problems involves pattern recognition. A very wide variety of pattern recognition methods is used in fault detection, that includes neural net-

work techniques. While recognition of spatial, stationary patterns is fairly well understood, it is not so simple with complex temporal patterns. Time-delay networks and recurrent networks are primary architectures used in the classification of temporal patterns. Using a state-space approach might also be an alternative.

Below, I present a number of ideas on possible uses of the Neurosolver for fault detection. These are thought experiments, so there are many practical as well as conceptual problems left unanswered. For example, there is a serious problem of detecting temporal patterns that are composed of non-sequential states of the system.

10.5.2.1 Fault detection by cascade associations

Let us analyze the system in Figure 62. The parameters of an element of the managed system that may be a network or just a component are monitored by the managing system. At each point in time, a vector of system parameters constitutes a state of the system. As before, that state is represented by a node in the state space of the system and when the system changes its states, a trace of those changes, a temporal pattern, can be recorded in a neurosolver. At some point in time, a human observer, data analysis, mathematical model, etc., may indicate that the system's behavior is abnormal (a *bad* state has been reached). The current state is associated with the fault, as are the nodes in the trace with the strengths dependent on the distance from the last node (i.e., the erroneous state). If the same scenario is repeated without supervision, the temporal pattern is repeated, then the associations contribute to the activity of the node representing the fault. It may be possible to find such a firing threshold, that if a complete pattern, or very similar pattern, is repeated, then the fault node fires indicating a fault.

FIGURE 62. A neural-based state space system for fault recognition.

10.5.2.2 Model-based fault detection

Another approach to fault detection involves unsupervised learning. The idea is to let the monitoring system interact with a perfect model of the network or its component for some time, so the correct behavior is learned. The model might be just a number of system parameters, network states, that are collected from a network whose behavior was normal. Alternatively, an actual model of the network can be used. The monitoring system observes the changes to the network state and uses a neurosolver to record them. All memorized trajectories represent the perfect behavior of the network. Any trajectory not recorded during the interactions with the perfect model is considered abnormal, because it was not observed in the normal behavior of the model.

FIGURE 63. A normal (a) behavioral pattern activates the checkpoint node through gating the connections from the successors to that node. An abnormal (b) patterns is detected if there are firing nodes that are not parts of the learned trajectory.

To detect abnormal trajectories in the state space of the system we need to add some functionality to the Neurosolver as illustrated in Figure 63. There could be a special behavior checkpoint node created with connections from all nodes in a sequence. The checkpoint node is on only if a sequence of state changes occurring in the network is matched with one of the learned temporal patterns. That may be achieved by a requirement that some number of nodes in the path describing the recent behavior of the system send activity signals to the checkpoint node (see Figure 63). The connections of any learned sequence are gated by the firing predecessors in the sequence. For example, in Figure 63, the learned pattern is ABCD. If the nodes fire in that order, then A enables the connection from B to the checkpoint node, B enables the connection from C, and so forth. Each sig-

nal received by the checkup node is sustained for some time (3 time ticks in the example). The checkup node is on only if all of the nodes in the observation timeframe are on (Figure 63a). If any of the signals from the nodes is missing, then the checkup node's activity is decreased below a certain threshold - the node is off (Figure 63b). That indicates an abnormal situation. In the example in Figure 63b, if the sequence of firing nodes is ABEF and E does not follow AB in any of the recorded patterns, then B does not gate the connection from E to the checkup node. The high activity of E is not transmitted, so the checkup node indicates that an abnormal pattern AB $\bar{E}$ has been detected. Even if EF is a normal pattern, there is a missing contributor to the high activity of the checkup node. That is sufficient to detect the problem.

The observation timeframe of the checkup node is like a window sliding through the pattern of network state changes. Every node in the window must belong to the same temporal pattern that has been memorized. We may want to relax the last requirement and insist that only a minimal number of nodes in the window must be active. In that way, the system could be allowed to wander a bit if only the main direction of state transitions is maintained. On the other hand, we may elect to limit the size of the window to one. Such a limit would cause a detection of any state transition that was not recorded in the past.

One important advantage of this technique presented here over other fault management techniques is that a trend leading to a fault might be detected before the actual fault occurs. Therefore, an attempt to modify the abnormal behavior might be undertaken before it leads to the faulty condition.

Recurrent networks again could be used instead, although at this moment the capacity of these networks might not be practically sufficient.

An alternate approach would be to assign a grandmother cell for each pattern of behavior (i.e. for each recorded trajectory). The action potential on each of the connections from the members of that pattern to the grandmother cell could be sustained for some time. If a certain threshold num-

ber of nodes in one pattern fired (ideally all, that could be difficult to achieve however), then the behavior would be recognized as normal (i.e., one of those recorded before). Otherwise, an abnormal behavior would be indicated. A network similar to the one presented in Figure 52 on page 129 could be used with all input nodes being the members of the input vector. There is a serious drawback in this approach however: the order of the nodes is lost.

The biggest problem with learning the behavior of a perfect network is the enormous size of the state space which would be required. A solution to that could be specialized detectors that operate in state spaces limited by various constraints. A good illustration of this idea is an analogy to a treatment of a seriously ill person who is taken care of by several doctors. Each of the doctors is a specialist in a specific area, so he ensures that the person's condition is normal from that perspective. Of course, there is a negotiation process needed, so the person does not die from an over-treatment or contradictions in the treatments ordered by different doctors. A negotiating mechanism would be useful in the case of an unhealthy network, but fortunately, in spite of their complexity, networks are easier to take care of and are less sensitive to potential errors in the diagnosis and treatment than human beings.

An example of a potential application of behavior modelling could be problem detection by the analysis of routing patterns and trends. In a usually healthy network, for example an ATM network, the cells (or packets) are routed according to provisioning data. The model of the pattern and trends that a neurosolver was allowed to acquire over an extended period of time could be considered ideal. If there are problems, then very often the built-in backup utilities of the network will take care of maintaining the required level of service quality. Nevertheless, the network administrator would like to be informed about problems even if they have been already fixed. In case of a problem, routing patterns and trends change. That change reflects modifications to the network that have been done by the automatic recovery routines. The change could be detected by the neurosolver by comparing the ideal model with currently observed in a way as described above. The network operator could be informed about the detection of a problem, so any danger

of long term repercussions can be prevented, for example by exchanging faulty equipment or re-engineering the network

I realize that the ideas presented in this chapter are just ideas. Some of them seem possible to implement through the use of the ideas explored in my research, others need a lot of additional thought even in theory. I hope that I will have plenty of opportunities in my further professional life to deal with these and other challenging problems successfully.

Chapter XI: Conclusions

The most general concluding remark is that this thesis constitutes a proof that ideas and methods derived from neuroscience and neurobiology can be successfully applied to high level tasks like planning and problem solving. It represents one of very few attempts to look for such a proof because of the complexity involved in attacking such problems. Furthermore, the thesis shows how to apply the ideas to engineer a device that can be used to tackle problems to which many real-world problems can be mapped.

The aforementioned general conclusion of this work is drawn from several facts established in the course of my research. I will briefly discuss each of them.

The problems of rearranging blocks in the blocks world used as a testbed in the course of my research can be successfully tackled by a community of agents with the solving agents based on the Neurosolver, a neural network based on a cortical column. A single Neurosolver is capable of resolving any rearrangement problem in a blocks world with 3 color blocks and 9 locations with strong and weak constraints laid on the blocks configurations.

The Neurosolver learns by being told. It is a fast learner; it does not require iterative presentations of teaching samples. That is in contrast to most neural networks that require considerable time to train.

The Neurosolver is a problem solving system, but with the difference between a planning system and a problem solving system being so subtle as it is, the Neurosolver can be considered a planner

as well. A sequence of two firing nodes, a state transition, can be associated with an action. As a planner, the Neurosolver is a respectful generalizer. It combines the same elements of a plan and can use them to construct novel plans. It does so with the same ease as when recovering a plan that was presented as a learning sample. With its generalizing capabilities that built upon the statistical nature of the connections, the Neurosolver can adapt to new or changing environments.

The Neurosolver uses node inhibition to prevent cycles of firing nodes in the search tree. This is a biologically plausible and critical characteristic for the problem solving capabilities. The length of the inhibition after a firing determines the length of cycles that can be prevented.

The Neurosolver represents a fair potential for scaling to larger problems. Notably, the search is not affected by learning samples as long as there are no new states introduced. Adding a new state is not a big burden on the search either, because it potentially increases the search tree only by one node. The Neurosolver may deal with exponential growth in problem complexity by using only that part of the state space that is relevant. In many cases, such an approach reduces the complexity considerably.

The Neurosolver exhibits prediction capabilities based on the past experience and as such can be used to estimate the next state of the subject system.

Agents bring an additional value to the problem solver. The use of mobility improves availability of the solving agent, because its multiple instances travel in the workspace. It is an important issue in vast distributed systems characterized by large coverage areas and long distances. The presence of multiple agents is critical in hostile environments.

The sensing agents provide a relatively fast and comprehensive sensory interface. That is important in systems where indirect perception is a necessity (e.g., a telecommunication network, a battlefield, etc.).

The manipulating agents provide a reliable manipulatory system That, in turn, is important in environments that might be hostile to the agents. Telecommunication networks and battlefields are again good examples of such environments.

In spite of being an add-on value with respect to the solver availability, the results of the simulation indicate that the use of multiple solving agents poses a serious synchronization problem. The synchronization is probably the most important issue to address in any further research concerning the use of multiple solving agents.

There are a number of other areas that could be explored in the future. Some of them have been discussed in Chapter X: *Directions for future research*.

APPENDIX A: Biological roots of the Neurosolver:

An insight into the human brain

A.1 Evolution of the brain

The brain of a man, one of the most complex systems known, is the result of thousands or even millions years of evolution. Simple vertebrates had only a simple nervous tube, or spinal cord, that was sufficient for survival of the species. That tube evolved to receive sensory information through various nerve fibers and send motor signals to contract the animal's primitive muscles. It is shown in Figure 64.

FIGURE 64. A primitive brain - a spinal cord (Kolb, [30]).

Soon, it became apparent that it is easier to move in one direction rather than in many. It was more important now to know what was happening in the direction of movement than in any other, so together with evolution of the body there was also a specialization of the control system. The frontal, or anterior, with respect to the preferred direction of the movements, part of the spinal cord received sensory signals that became more important than signals from other parts of the

body. That part refined its sensory capabilities and the brainstem, a precursor of the human brain, appeared. The brainstem is illustrated in Figure 65 with its three parts marked

FIGURE 65. The brainstem - a precursor of the human brain (Kolb, [30]).

New challenges from the changing environment induced further development in the motor and sensory systems. The brainstem grew, so that the new functionality could be controlled in a better, more direct manner. The frontal part of the brainstem was enlarged by addition of two cerebral hemispheres. The back, or posterior, part evolved into the cerebellum. Those new additions are shown in Figure 66. The cerebral hemispheres acquired functionality for data analysis and decision making; like movement origination. The function of the cerebellum was to coordinate the execution of movement commands. All mammals have brains like that shown in Figure 66, although the development of certain functions, and the parts of the brain supporting those functions, varies amongst species.

FIGURE 66. Mammalian brain (Kolb, [30]).

Humans have the best developed brain among all mammals. The biggest change in the relative size and functionality came to the cerebral hemispheres. The cerebral cortex evolved into the control center with the most sophisticated functionality. It is no longer just a place where facts are analyzed and actions are generated. In the course of phylogenetical development, the relatively simple tube of nerves evolved into the origin of intelligence.

A.2 Anatomy of the brain

Usually one thinks about the brain as being completely enclosed in the skull. In fact, however, the brain can be divided into three main parts (Figure 67), not necessarily located in the head. These are:

- spinal cord,
- brainstem and
- forebrain

FIGURE 67. Three basic parts of the human brain (Buchsbaum, [15]).

The spinal cord is the lowest element of the brain that is also the oldest phylogenetically and still fulfills the same tasks of receiving sensory signals from various body parts and sending motor commands to muscles as in primitive vertebrates. It is divided into a number of segments with

each segment servicing more or less the part of the body at the same height as the segment. The sensory nerves that enter the spinal cord from the back (or top in animals, therefore called dorsal or superior) provide sensory information. They make synapses with the neurons that will carry that information to the thalamus and further to the cortex in the forebrain (see below). The tracts carrying the motor signal connect to the muscle's neurons in the frontal part of the spinal cord (or bottom in the animals, therefore called ventral or inferior).

The functionality of many parts of the brainstem has not been completely determined. Generally speaking, the brainstem handles the basic life-supporting functions. In many animals, the brainstem provides the highest level of functionality, that in more highly evolved species were overtaken by the forebrain. There are several entities that are relatively well understood. The hypothalamus is involved in almost all aspects of behavior, like feeding, sleeping, sexual behavior, temperature regulation, emotion control and movement. The cerebellum is regarded as the center for equilibrium, postural reflexes and coordination of movements. Another interesting part of the brainstem is the reticular formation that has been assigned the responsibility for consciousness, general arousal and the control of basic functions like breathing, heartbeat, body temperature, chemistry of the blood, etc.

The newest component in the evolution of the brain, the forebrain, and particularly the cerebral cortex, is the location of what we consider to constitute intelligence. Parts of the forebrain and cerebral cortex will be described in the next sections in more detail, since they are more important as seen from the perspective of the work described in this thesis.

A.2.1 Anatomy of the forebrain

The forebrain consists of the following five anatomical parts

- neocortex or cerebral cortex,
- thalamus,

- basal ganglia,
- limbic system and
- olfactory bulbs

A.2.2 Cerebral cortex

The cerebral cortex is the youngest brain structure and the relation of its size to the size of whole brain is the most visible difference between the brains of humans and animals. Because an average human is evidently more intelligent than an average animal, the neocortex has been thought to be the location of that value-added functionality. Many important experiments with animals and with humans have confirmed that the cortex is the highest level center of information processing. Almost all sensory data is transmitted to the cortex where it is analyzed. Response to stimuli that depends heavily on past experience, memory, is calculated, and signals initiating proper action are sent to the lower level brain structures that carry out the commands.

FIGURE 68. The thalamus and the basal ganglia: the input and the output of the cortex.

The nature of information processing that goes on in the cortex is the main interest of this thesis. Therefore, the cerebral cortex will be discussed in detail in several further sections of this work.

A.2.3 Thalamus

Sensory information, with the exception of smell data (see below), is sent to the cerebral cortex for analysis. The pathway, however, is not a direct one. The thalamus is the input switching system that passes the signals detected by senses to appropriate areas of the brain. Principally, the signals are directed to the primary somatosensory areas, the visual areas, the auditory areas, etc. In addition, there are pathways to secondary and higher areas, but they have relatively small throughput. The primary, secondary and tertiary areas will be discussed later in this appendix.

The thalamus is not just a simple relay station. There are many reciprocal connections from the cortex that carry signals that are used to modulate incoming input data. The data that is not important in a given context may be suppressed, and vice versa, a weak signal that the cortex indicates as essential may be amplified. The thalamus may even be instructed to expect certain input, so that the proper action can be undertaken to diminish the difference between the expected and actual sensory data.

A.2.4 Basal ganglia

There are many basic functions of the motor system that are controlled by the spinal cord, like the spinal reflex which ensures that a group of muscles contracts as one, the opposing ones relax. More complex movements are the responsibility of higher and higher levels in the motor system. The basal ganglia is a part of the motor system that is, in lower vertebrates, the highest level center for motor control. In humans and higher vertebrates, the basal ganglia is the source of multiple signals initiating and terminating partial movements. It uses inhibition and modulation in order to control lower level motor subsystems in the brainstem and spinal cord, ensuring that movements are smooth. It closely cooperates with the cerebellum, that is responsible for coordination of the movements.

The basal ganglia is the main relay station for nerve fibers going from the motor cortex to subordinate subsystems

A.2.5 Limbic system

Initially, the limbic lobe was thought to provide olfactory functionality, because of its connection to the olfactory bulbs. That theory has been abandoned as was the hypothesis that there is a single function that the system realizes. There is much work to be done before all of the parts of the limbic lobe can be assigned a proper function. There is enough evidence, however, to state that the hippocampus - a part of the limbic lobe - is critical for memory storage, spatial organization, organization of movements, inhibition and learning.

FIGURE 69. The limbic system and the olfactory bulbs (Hamilton, [25]).

The hippocampus has abundant connections with other parts of the limbic system. Many experiments have shown that there are reward and punishment centers in the limbic lobe. Consequently, data processed by the brain is being weighed in the hippocampus and information considered worthy is stored in memory. Data that is not important is discarded or stored only as long as it is needed for the current task to complete. There are no details, however, about the underlying mechanisms of that functionality.

Another part of the limbic lobe, the septum, seems to be responsible for emotions, something that usually is attributed to whole limbic system

A.2.6 Olfactory bulbs

The sense of smell is considered to be the oldest sensation that is perceived and processed by the brain. That sense was very important in the evolution of vertebrates, since it could guide the animal to food and, as a result, increase its chances for survival. The olfactory bulbs that are part of the limbic system are responsible for the processing of the sense of smell. It is very interesting, that the sense of smell is the only sense that is not handled by the cerebral cortex. It is probable that these judgment centers developed in the limbic system because of the proximity to the smell center. Something that smelled good was worth remembering, while the opposite was best forgotten. That might be the reason why the limbic system is so important to memory.

A.3 Anatomy of the cerebral cortex

The cortex is a thin (a few millimeters) layer of convoluted tissue that covers all lower level brain structures located inside the skull. It is a layer of gray matter, so called because of the high concentration of grayish cell bodies, and white matter, that is located below the former, again so called because of the high concentration of white neuronal fibers connecting neurons in various parts of the brain. The space between these neurons and associated fibers is filled by glial cells. The function of glial cells is not definitely determined, but most of the researchers assign them some support functions, like nutrition and waste disposal. Neurons, interconnected by a network of axons and dendrites, are the fundamental information processing units in the brain.

A.3.1 A single neuron

There are many types of neurons in the central nervous system in general, and in the cerebral cortex in particular. The anatomy and functionality of all of them is, however, similar. One of the most important neurons in the cortex, the pyramidal cell, is shown in Figure 70. The pyramidal

cell consists, as other neurons do, of the cell body or **soma**, an **axon** and **dendrites**. The axon and the dendrites are fibers that are attached to the soma. Usually, the axon is longer than the dendrites, because they carry signals from the neuron to other, sometimes very distant, areas of the nervous system. Axons make connections with somas of other neurons, with their dendrites and sometimes with other axons. The task of the dendrites is to capture signals from the axons in the neighborhood of the neuron and bring them to the soma. All signals are integrated in the neuron and the resulting integral determines the level of excitation or activity of the neuron. This activity is, in turn, transmitted over the axon to another set of neurons where the activity recalculation scheme is repeated.

FIGURE 70. A neuron (Kolb, [30]).

The location where an axon makes contact with another entity is called a **synapse**. There are two types of synapses: symmetric, commonly viewed as inhibitory, and asymmetric, being excitatory in nature. If an axon makes an inhibitory synapse, the activity carried over the axon will have a negative effect on the activity of the recipient. The excitatory synapse has a reverse effect on the receiving cell.

The topology of the connections that a neuron makes with other cells is determined by the genetic code. However, synapses are highly adaptive devices, since their conductivity, i.e., the ability to

transmit signals, may be modified. The extent of that modification depends on the pattern of activity in the pre-synaptic and post-synaptic entities. If there are certain regular changes in the patterns of activity of a number of neurons, and consequently, their synapses, then these changes may be encoded in the pattern of conductivity or strengths of the synapses transmitting signals between the cells involved. Later, that pattern of activity can be recalled even if only a part of the original pattern is known. The adaptivity of synapses is the fundamental mechanism that the brain uses to store information and perform calculations. All artificial neural networks that attempt to mimic the circuitry of biological nervous systems make use of some kind of adaptive synapse. This computational paradigm is very often called connectionism.

The details of communication between the cells, the transmission of signals, the integration of the inputs and all related processes at the cellular and inter-cellular level are fascinating. They are however not directly related to the content of this work. One of the best publications on the subject is the book by Shepard [49].

A.3.2 Lateral organization

A.3.2.1 The cortical hemispheres

The cerebrum is a major mass of the brain composed of many millions of nerve fibers and covered with the cerebral cortex. The cerebrum consists of two hemispheres, as shown in Figure 71,

FIGURE 71. The cerebral hemispheres (Kolb, [30]).

divided by the medial longitudinal fissure¹. The hemispheres are connected by the corpus callosum, a tract of many nerve fibers. These fibers are used to exchange information between both halves of the cerebrum. The low level functionality of the hemispheres is similar, but they control opposite sides of the body: the left hemisphere controls the right side of the body, and vice versa. That arrangement is as surprising now as it was for Erasistratus. Ramón y Cajal [44] attempts to explain that phenomenon by the need for maintaining continuity of the visual image created in the brain from the signals sent by the sensors in the retinas. Coghill [18] suggests that this anatomical arrangement developed in primitive vertebrates (he studied *Amblystoma*) that used a coil as their basic movement. The sensory neurons on one side of the body excite the motor neurons on the

1. Fissure - a cleft in the surface of the cortex that is deep enough to indent the brain ventricles

other side, so the animal can move away from any noxious stimulus. The olfactory bulbs are not crossed, the only such system in the brain, because the animal wants to move in the direction of the food that it smells. There are other hypothetical explanations, but the issue is far from being settled, though.

Psychologists believe that the hemispheres differ in high level functionality that they provide. Although it may vary from one individual to another, usually the left hemisphere is the logical brain. The processing of information in the left hemisphere has a symbolic nature. Given a set of clues, the brain follows the logical paths that ultimately lead to the solution of the problem. On the contrary, the right hemisphere is an artistic brain that uses images rather than symbols. The processing of information in the right hemisphere has a global, associative nature. The sensations that are delivered to the right hemisphere are correlated, resulting in a recognition of shapes, colors, sounds, etc. The right hemisphere, however, has to contact the left side of the brain to attach proper symbolic labels to the recognized objects. The same happens in the opposite direction. If there is no obvious logical pattern to the resolution of the problem, the right hemisphere might be helpful by associating facts or sensations that are not linked by any logical reasoning.

A.3.2.2 The cortical lobes

Each of the two hemispheres of the cortex is divided into four¹ anatomical regions called **lobes** that are illustrated in Figure 72. Although these are anatomically defined areas, they are often used in conjunction with the functionality observed in the specific area. That is, however, far from accurate. A much better reflection of the functionality of various regions of the cortex are the topographic maps described in further sections. The lobes are, however, convenient orientation terms that are used commonly by neuroscientists.

1. Sometime anatomists count the limbic system as another lobe, bringing the number of lobes to five

FIGURE 72. The cerebral lobes (Albus, [1]).

A.3.3 Laminar organization - The cortical layers

Neuroscientists divide the cortex into six layers, depending on how the neurons are arranged. Despite such a generalization, there are certain variations in the anatomy of different parts of the cortex. In some areas of the brain certain layers might be thinner, almost non-existent, while others are thicker than average. Some researchers argue that there are more layers than six, and actually in the literature there are letters added to the layer numbers, like IVa, IIIb, etc., to indicate sublayers.

There are certain functional attributes that can be associated with the cortical layers. For example, layer IV receives signals from afferents¹ coming mostly from the thalamus or olfactory system, and is therefore considered to be an input layer. This layer is very thick in primary sensory areas (see topographical maps below) and very thin in the motor regions of the brain. The layers below

¹ Afferents - the tracts that bring signals from somewhere else; efferents - the tracts that carry signals to somewhere else

layer IV, send efferents to the lower brain structures and are considered to be output layers. That functionality explains why they are thick in the motor areas: the signals that are generated there must be carried to motor neurons that ultimately will cause contraction of muscles. The layers above layer IV make connections mostly with other cortical areas.

FIGURE 73. Cortical layers (Broadman, [12]).

The detailed description of those interactions will be presented in APPENDIX B *Biological roots of the Neurosolver: The cortical column*.

A.3.4 Topography of the cerebral cortex

Since the times of Gall and his phrenology theory¹, many scientists have performed extensive studies of various aspects of the cortex, trying to divide it into more or less meaningful regions. In the course of these explorations, studies of the brain were published that described various characteristics of the cerebral cortex. There are three topographical maps of the brain that are widely

1. In short, phrenology was a theory that certain physical characteristics of the skull (like bumps) and indirectly the brain, can be the basis for assigning functions to the regions of the brain. Although today that theory sounds amusing, it was the first attempt to create a functional map of the brain, an endeavor that was attempted by many scientists later on.

used in the field today: functional, projection and cytoarchitectonic maps. Although the maps are based on the different research methods used, the areas that were defined are quite similar. That seems to imply that there is a close relationship between the anatomy, connectivity and functionality of the brain areas.

FIGURE 74. Cytoarchitectonic map of the brain (Elliott, [20]).

A.3.4.1 Cytoarchitectonic maps

Cytoarchitectonic maps are constructed by examining the cells in every region of the cortex. Their type, density, arrangement, etc., is taken into account. The map that is used most often was created by Brodmann at the beginning of this century and one of its versions is shown in Figure 74. Brodmann examined various parts of the cortex without any overall plan, so the numbers that he assigned to different areas are a little chaotic. The numbers are used very widely to describe the

location of a function realized by the specific part of the brain. For example, area 17 is commonly used as a reference to the primary visual cortex, while area 41 is a reference to the primary auditory cortex (see next section).

A.3.4.2 Projection maps

Projections maps are created by tracing the efferents that leave the cortex and make connections with various subsystems in the lower structures of the brain and by looking for the source of the afferents carrying signals to the cortex. It has been noted that for each type of sense, there is an area in the cortex that receives most of the sensation signals carried by the axons of the neurons in the sensory system. Those areas are called primary sensory areas. The neurons from those areas send axons to secondary areas and from there to tertiary areas. That regularity led to the hypothesis that information processing in the cortex is hierarchical. Now, however, there is a clear evidence, that although there are obviously hierarchies in the processing of sensory data, there are also many additional pathways that contribute to the parallel nature of the overall functionality of the brain. Figure 75 illustrates the main projection maps and the pathways between the primary,

FIGURE 75. Projection map of the brain (Luria, [33]).

secondary and tertiary areas. The primary areas are black and the secondary areas are gray. The arrows from or to the secondary areas indicate the tertiary regions. It is worth noting, that while primary areas are highly uniform in the character of the incoming signals, the other areas are more

and more inclined to process signals with diversified origins. They are called associative areas (particularly the tertiary areas), since they correlate signals from many sources.

A.3.4.3 Functional maps

The functional maps are drawn taking into account the data obtained by one or more of the following experimental techniques:

- stimulating cortex areas and registering what behavior is induced,
- stimulating cortex areas and registering what sensations are reported by the subjects,
- recording the activity of the cortex while performing certain behavioral tasks,
- observing the changes in the behavior after cortical damage.

Some neuroscientists use animal lobotomy to register changes in the animal behavior. That technique can hardly be classified as scientific, since it is as cruel as the experiments of Erasistratus.

The best known functional map of the cortex is the one prepared by Penfield [42] and his coworkers at the Montreal Neurological Institute. They were created by examining cortices of patients awaiting brain surgery under local anesthetic. They noticed that each part of the body responds to a stimulus applied to a specific location of the cortex. What is more, the topology of the body is preserved in that mapping, although relative proportions of the areas corresponding to parts of the body vary from the differences between their actual physical sizes. For example, there is a relatively large area of the cortex devoted to hands and fingers as compared to the area controlling the trunk. That is easy to explain. There is a much higher degree of precision required from the hands and fingers than from the torso, so that more resources must be devoted to hand control.

Similar mapping of the body parts has been discovered by recording the sensations that were reported by subjects upon a stimulus being applied to the cortex. The areas of the first mapping (stimuli movement) are localized in the anterior¹ part of the parietal lobe more or less along the

postcentral gyrus¹. The areas of the second mapping (stimuli-somatic sensation) are located in the posterior² part of the frontal lobe, along the precentral gyrus.

FIGURE 76. Homunculi - functional maps of the somatosensory and motor areas of the brain (Penfield, [43]).

Figure 76 shows a popular illustration of Penfield's observations in a form of homunculi, little men, that are more or less symmetrical with respect to the central sulcus³. That symmetry is as

1. =frontal

1. Gyrus - a ridge on the surface of the cortex

2. =back

3. Sulcus - a cleft in the surface of the cortex that is shallower than a fissure

important to the control of movement as the topology preserving nature of the mappings. Both areas, motor and somatosensory areas, exchange information during voluntary movements, so the part controlling the movement (motor areas) is aware of progress in the movement (as reported by the sensors sending signals to the somatosensory areas).

The region of the cortex responsible for speech was discovered in the previous century by Broca [13]. That area is today called by his name. Low level speech understanding was discovered by Wernicke [61] and is also called by the name of the scientist. Those areas are located in the medial¹ superior² part of the temporal lobe (Broadmann areas 41 and 42). The occipital lobe is the part of the brain where visual information processing takes place (Broadmann areas 17, 18 and 19). That function was assigned for the first time by Holmes [26] who was treating soldiers with head injuries in the World War I and noticed that injuries to different parts of the occipital lobe caused blindness in different parts of the eye.

A.4 General function of the cortex

The functionality of the cortex is still not well researched. However, the following three types of computation is performed by the cortex and comprises most of its functionality:

- control of voluntary movements,
- pattern perception and
- cognitive mapping.

A.4.1 Control of voluntary movements

The cortex receives sensory signals from thalamic afferents. That information is the basis for determining responses to the observed state of the environment. Any complex action that is a part of that response is planned in the cortex. Every item in the sequence, the schedule of the response,

1 =middle

2 =top

is sent to lower brain structures for execution. The functionality of some of those underlying structures was discussed briefly earlier in this appendix. The cycle repeats itself when the sensors notice the change caused by the response, and again report that state to the cortex. Through that feedback, the cortex ensures that not only is a proper action undertaken, but also that it is carried out accurately.

In tests on animals with the cortex removed, the ability to generate movements, as complex as walking, eating, drinking, mating, is not lost. However, they are not able to perform any of those movements as a part of an overall goal. They may carry food, but they will not hoard it. They may perform some elements of grooming, but they do not really groom.

A.4.2 Pattern perception

The sensory system is bombarded with a variety of bits and pieces of information about the environment. That information is correlated in the cortex. The cortex is the place where the nature of the observed changes is comprehended. Without that comprehension it would not be possible to react properly to the incoming data, since the same singular symptom can be a part of many, sometime quite different, syndromes.

Animals without the cortex recognize distinct signals like place and intensity of light, but are not able to differentiate between the patterns that use those signals.

A.4.3 Cognitive mapping

The question about the localization of the memory has not been answered yet. The fact is that many things can be learned without the cortex. However, the cortex provides additional functionality. Cognitive maps that are constructed in the cortex during learning, can later be used to perform actions that were not learned.

Decorticated animals perform as well, or almost as well, as healthy ones on tasks like classical conditioning, approach learning, cue learning, etc. However, they are not able to learn dependen-

cies that are distant, and not as part of the learned sequence. If a healthy rat learns to run from the south side of a maze to the north side, it will reach the north side even if positioned in the west. Clearly, it has built a map during learning that it uses later on. That map is located in the cortex, since that ability is absent in rats without the cortex, although they can learn the path from the south side to the north side.

Cognitive maps are constructed very quickly. Sometimes it takes only one or two trials. That learning speed suggests that built-in neural connections are utilized.

A.5 Functional organization of the cortex - Luria's model

There has been a lot of evidence collected since the nineteenth century that the posterior cortex receives signals from the sensory systems and is generally more concerned with sensory function than the anterior cortex. It has also been proven through various experiments and observations of humans with cortex lesions, that the anterior cortex is responsible for the motor system, and much less responsible for sensory processing. The interactions with the sensory systems are rather indirect through the modulating influence that is procured by the cortex efferents connected to the thalamus and other sub-cortical structures. A hierarchical model of cortex function has been built that involves three types of cortical areas:

- 1 Primary areas, sensory and motor
2. Secondary areas, sensory and motor.
- 3 Tertiary areas that are called associative.

Many researchers have proposed theories based on this basic, hierarchical model. Luria's model seems to be a good example. Although it has been proved since then that a purely hierarchical theory of the cortex function is incorrect, Luria's model is a good top level description of the processes and pathways that constitute overall cortex functionality.

In Luria's model, the cortex is divided into two basic parts with respect to the performed functionality. The posterior part, that includes the occipital, parietal and temporal lobes, is the sensory unit. The anterior part of the cortex is the motor unit. It comprises only one lobe - the frontal one.

Figure 77 illustrates the sensory cortex with sub-regions representing the primary, secondary and tertiary sensory units (graded shadowing). The signals from the thalamus arrive at the primary areas (most dark) where they are analyzed with respect to their location, intensity and patterns of activation. To fulfill that task properly, the primary cortices are organized in arrays or maps representing those characteristics. The homunculus that was presented earlier is an example of such a map in the somatosensory area. The results of primary analysis are transmitted to the secondary sensory areas (medium gray) where sensation is consolidated, but still retains its modality. The integration of various types of sensations occurs in the tertiary, associative, areas (white). Luria believed that sensations are assigned their symbolic meanings in those areas and, consequently, is where the abstract thinking begins.

FIGURE 77. Luria's model - The sensory cortex (Luria, [33]).

The motor cortex is illustrated in Figure 78. The direction in which signals are passed from one area to another are reversed here; i.e., the tertiary area sends efferents to the secondary area and

the secondary area, in turn, sends the signals to the primary area. The tertiary motor area is the highest level structure in the brain. It is the area in which intentions are created. These intentions are translated into complex behavioral patterns that are divided into singular actions in the secondary areas (pre-motor cortex). The primary motor cortex has an organization similar to the primary sensory cortex. It similarly contains topographical maps (like that illustrated by the motor homunculus). These maps are used to direct actions toward the appropriate locations with the correct intensity.

FIGURE 78. Luria's model - The motor cortex (Luria, [33]).

The problem with the Luria's model is not that it is wrong, but rather that it is incomplete. As new research techniques and tools were designed, more and more data was discovered that contradicted the pure hierarchical model. There are many cortical connections that do not fall into any hierarchical pathways. That mixture of the traditional hierarchical view and new parallel extensions to that model underline the complexity of the functions performed by the cortex.

APPENDIX B: Biological roots of the Neurosolver:

The cortical column

In Appendix A, the laminar organization of the cortex was presented. This organization is based on the types of neurons that can be found in the cortex and their arrangement. That same criterion was used by Broadmann in order to divide the cortex into many distinct regions. In this appendix, a closer look at the cytoarchitectonic structure of the cortex is presented. Researchers discovered that although various areas of the cortex can be distinguished, there is a certain characteristic microarrangement of a number of neurons that is common to all of them. That arrangement is called a cortical column. In the course of many experiments, it became obvious that the cortical column constitutes a second fundamental level¹ information processing unit. The architecture, behavior and computing capabilities of the cortical column and a network of cortical columns are discussed in this appendix.

B.1 Types of neurons in the cortex

Many types of neurons have been found in the cortex. Some of them are included in the illustration in Figure 79. The most characteristic ones are **pyramidal cells**, easy to recognize by, and called after, the pyramidal shape of their somas. Other neurons are more difficult to classify, and they are often referred to by several common terms: **stellate cells**, interneurons or non-pyramidal cells. They might be bipolar, multipolar or bitufted with respect to the shape of the tree of their dendrites. According to the concentration of spines on its dendrites, an interneuron may be spiny

¹ A neuron is the first level organizational unit

cortical areas. Signals can be transmitted directly by synapses with the afferents or axons on the somas and dendrites of the pyramidal cells or indirectly through other cortical neurons

It has been demonstrated that the axons of pyramidal neurons constitute the main output of the cortex. Using various tracing techniques, many researchers found that the axons of the infragranular pyramidal cells usually project to other, lower, neural structures. They carry activation signals for actions that are carried out by the appropriate subsystem. For example, they can activate a movement by sending signals to the basal ganglia, or even further to the spinal cord. The supragranular pyramidal neurons have been found to send their axons (usually) to proximal¹ or distal² pyramidal neurons in the cortex. These axons constitute main cortical pathways that distribute activation throughout various areas of the cortex in order to integrate and correlate different aspects of the perceived phenomenon. That columnar activity is internal to the cortex and represents abstract thinking; i.e., reasoning without the involvement of any physical resources

The cortex grows starting with the lower layers, so it may explain why the pyramidal cells in the lower layers fulfill more basic functions providing control for the lower level subsystems. When higher layers are added to the cortex, the extra resources can be used on more sophisticated functionality. That is why the supragranular pyramidal neurons can migrate toward other cortical regions in that way creating a framework for thinking

B.1.2 Spiny-stellate neurons

It has been demonstrated through tracing experiments that layer 4 of the cortex is the location where most of the thalamic efferents terminates. Layer 4, a layer that is said to be granular because of the high density of various interneurons, is commonly considered to be the input layer of the cortex. That layer is very thick in the primary sensory areas and very thin in other regions that do not have direct input from the thalamus

1. =close

2. =far

or smooth. Usually, interneurons do not send axons outside the area in which their body is located. Many of the interneurons have been given names after the shape of their dendritic or axonal trees.

The pyramidal and non-pyramidal neurons will be described in greater detail in the following few sections.

FIGURE 79. The most important cortical neurons (Szentagothai, [54]).

B.1.1 Pyramidal neurons

Pyramidal neurons usually have their somas in layers 2 and 3 or 5 and 6. Characteristic for a pyramidal cell is not only its soma, but also its apical¹ dendrite. The apical dendrite usually crosses several cortical layers. That design gives the pyramidal neuron the capability to integrate a variety of inputs specific to different cortical layers. The cells that have their somas in layers 2 and 3 have shorter apical dendrites than the cells with their somas in layers 5 and 6. Both types of pyramidal neurons, supragranular - those with their somas above layer 4 that is called granular - and infragranular - with somas in layers 5 or 6 - receive only excitatory synapses. The main sources of input signals to pyramidal neurons are afferents incoming from other neural structures or other

¹ That is, oriented perpendicularly to the surface of the cortex.

Spiny stellate cells (cell labeled S_1 in Figure 79) are found exclusively in the middle layers (mostly in layer 4) of the cortex. Many neuroscientists believe that they are, together with the pyramidal neurons, the main targets of thalamic signals. Spiny stellate cells, however, span their dendritic trees locally with respect to the laminae and the cortical area. The axons of spiny stellate neurons have similar, local, range, but in addition to layer 4 they may project to adjacent layers 3 or 5. Due to the property of limited localization, spiny stellate interneurons are not integrating units.

Spiny stellate cells have an excitatory nature. They project to the somas or apical dendrites of pyramidal neurons and other interneurons. This organization is conducive to processing input data.

Cortico-cortical tracts may also terminate in layer 4. The term "input", therefore, includes both types of input: external - thalamic - and internal - cortical.

B.1.3 Chandelier neurons

A chandelier neuron, whose dense pattern of axonal ramifications resembles candles of a chandelier, is mostly found in layers 2 or 5. The cells with their somas in layer 5 send their axons up, and vice versa. The cells with their somas in the supragranular layers send their axons down.

The behavior of chandelier cells is inhibitory. They connect to other neurons, mostly pyramidal and spiny stellate, usually in the vicinity of the initial segment of the output axon. For that reason, they are thought to be gating the output from the post-synaptic neuron.

Chandelier neurons can be found mostly in sensory areas.

B.1.4 Basket neurons

Basket interneurons (cell labeled S_2 , S_3 in Figure 79) are so called because they have axons shaped in the form of a basket around the cell bodies and proximal dendrites belonging to pyrami-

dal cells. Their somas are usually in layers 3 and 5. They project horizontally, but also cross several layers before they reach their destination. Basket cells receive signals from collateral axons of pyramidal cells

Basket cells exhibit an inhibitory behavior. They may, therefore, suppress the activity in pyramidal neurons in the neighborhood while the activity in the region is high. Such behavior enhances the contrast between neighboring groups of cells.

The density of basket neurons is relatively high in the motor areas of the cortex.

B.1.5 Double bouquet neurons

The somas of double bouquet interneurons (cell labeled S_5 in Figure 79) are found mostly in supragranular or granular layers. Their axons form dense vertical branches directed down and up. That gave them their name - the axonal branches look like bouquets.

Double bouquet neurons are inhibitory, but because the preferred targets of their axons are other inhibitory cells, their overall role is dis-inhibitory. Their role might be modulation of cortical activity across the layers.

B.1.6 Bipolar neurons

Bipolar cells are similar to double bouquet neurons, because of the dendritic and axonal trees that do also have preferential vertical orientation. They receive signals from supragranular pyramidal cells.

Bipolar neurons are excitatory. Hence, their task might be to move high activation from the supragranular to the infragranular layers.

B.1.7 Smooth stellate neurons

Smooth stellate neurons are inhibitory. Their somas have been detected in many layers of the cortex, but mainly in layer 4. These interneurons are multipolar and often are called short axon or local circuit cells, because they usually project to the same or neighboring cortical layers.

Smooth stellate neurons are considered general processing elements, because they synapse with a wide variety of other neurons. They can connect to somas, dendrites, spines and axons of pyramidal and non-pyramidal cells.

B.2 The cortical column

In the course of their experiments, neuroscientists noticed that reaction to a stimulus is highly localized in the cortex. They observed that the response to a sensory input is independent of the depth of the placement of the electrodes that detect and measure the activity of the cortex. Slight differences in the promptness of the response have been observed. When the electrode is placed in the granular layer 4, there is the shortest delay between the stimulus and the response. That is in agreement with the earlier observations that layer 4 is the main input to the cortex. Efferents connect directly to the somas and dendrites of the receiving cells. The supragranular layers get activated next. There are fewer direct connections between afferents and the cells in those layers, so the activity is carried over the interneurons. The additional synapses that are required are the reason for the delay. The infragranular layers are activated last. One reason is that they have only their apical dendrites in the proximity of the afferents, while the supragranular cells have their somas and basal dendrites closer to the terminals of the axons of the input tracts with their large surfaces more inclined to make connections. Additionally, spiny stellate cells have their axons projecting upward usually, and that favors the upper cells as the targets of the propagation of the activity. The fact that the infragranular layers are activated last in the cortex, underlines their role as output zones. Axons of the pyramidal cells carry the integrated signals to other regions of the cortex or to other neural structures.

Another important observation is that if the electrode is moved a little bit, it detects activation that is the response to possibly completely different stimulus. It has been proven that neighboring groups of neurons that span the area of the cortex with a diameter of 0.5 to 1.0 mm respond best to different stimulus. Little activity is spread laterally; i.e., high activity in one mini-circuit does not cause a change in the activity of its neighbors.

These findings suggested that there is a functional unit that is vertically oriented crossing all cortical layers. Szentagothai was the first to put forward this idea. He called the mini-circuit of interconnected cortical neurons that respond in a uniform way to a stimulus a cortical column. Since then, there have been many followers, and currently the existence of a functional unit that retained the name given it by Szentagothai [55] is a commonly approved fact.

In section A.3.4: *Topography of the cerebral cortex*, the topology preserving mapping that occurs in the cortex was described. The columns are an integral part of that phenomenon. The columns of a specific region responsive to a particular sensation may represent different aspects of the perception. For example, there is a region in the somatosensory cortex responsive to stimuli applied to a hand. The columns in that region are activated by the sensory information coming from sensors of various types. There are columns that respond to tactile data, temperature, position of the joints, etc. Such distant sensations do not fit very well with the isomorphic mapping scheme that represents the whole body as one image in the cortex. However, recent studies show that there are many representations of the body. Some projections can be repeated many times in various regions, implying that the particular sensation space that they map is a component of many more complex perception spaces. Such a two-tier organization can be explained by the need to represent a multidimensional world on a twodimensional plane, the cortex. There have been, unfortunately, only a few areas in which spaces of the first and second level indices are clearly defined.

FIGURE 80. Two-tier organization of the topology preserving maps in the cortex (Ballard, [3]).

Figure 80 illustrates the two-tier organization of the topology preserving maps of the cortex (after Ballard [3]). Two regions with the first level index spaces A and B contain several regions with the second level index spaces. Some of those sub-regions, namely Y and Z, are repeated in A and B. This implies that a sensation that can be indexed by Y is a component of two different compound perceptions. As illustrated by A in Figure 80, an entity that maps to a higher tier in one location may be mapped to a lower level tier in another region

From the perspective of this thesis, there are two important things worth noting

- there are uniform functional units - cortical columns - in the cortex and
- a functional map, in fact a network of a number of interconnected columns, may be used for various purposes.

The composition of cortical columns in various parts of the cortex, particularly between its motor and sensory parts, might differ. There are, however, enough similarities that allow for the degree of generalization that has been presented here

B.3 Connectivity of the cortical column

The role of the basket cells enhancing the contrast between neighboring columns through inhibitory connections with pyramidal cells belonging to other columns was described previously. That aspect of the connectivity pattern of the cortical column is a vital part of the capability of the cortex to self-organize itself into feature maps

FIGURE 81. Types of efferents and afferents of a cortical column by layer (Shepherd, [49]).

The remaining patterns of connectivity of a column with more distal entities are summarized in Figure 81. This figure shows a cross-section of the cortex with images that are generated by three types of staining, i.e., marking the elements of the cortex so they can be photographed. There are certain differences in the connectivity of the cortical columns in the motor and sensory areas that are disregarded here. The same degree of generalization is made as was done with the cortical column itself

The afferents arriving at a cortical column usually have their terminals in layer 4. The medial section of the cortical column is, therefore, considered to be an input interface. There are two sources of input signals incoming to the interface: cortical - from other regions of the cortex, and thalamic - from the thalamus. The cortico-cortical afferents can interconnect proximal or distal areas of the

cortex. They include connections between the two hemispheres of the brain, called callosal afferents.

Activity is distributed across the whole column through internal interneuron connections. Activity is easily transmitted to the supragranular pyramidal neurons in the upper part of the column because of the favorable location of their somas and the dendritic trees. Higher levels of input activity are required to excite the infragranular pyramidal neurons in the lower part of the column.

It is important to indicate that the input activity of a column is a sum of the cortico-cortical and thalamo-cortical afferents. Consequently, the activity of a column depends on the sensations induced by the external stimuli as well as on the internal processes occurring in the cortex.

The pyramidal neurons were described as the output cells of the cortex. They fulfill the same role with respect to the cortical column to which they belong. Generally, the distance to which a pyramidal cell projects depends on the laminar position of its soma in the cortex. The deeper the soma is located, the further the axon transmits the neuron's activity. That arrangement can be explained by analyzing the ontogeny of the brain and the cortex. The growth of the cortex begins with the lower layers and the neurons in those layers are used to accomplish the most basic functionality. That requires long distance connections that can reach as far as the spinal cord. The level of the functionality that is still unrealized increases when the higher layers of the cortex start to develop. The more complex the functionality provided by a specific neural system is, the closer to the cortex that subsystem is located and, as a consequence, the shorter the length of the efferents. Finally, connections between cortical areas are sprouted from the higher cortical layers.

Hence, a cortical column has two output interfaces

- cortico-cortical efferents, realized by the axons of the supragranular pyramidal neurons in the higher section of the column and
- cortico-sub-cortical efferents, realized by the axons of the infragranular pyramidal cells in the lower sections of the column.

The cortico-cortical connections are considered to be the basis for higher level functionality that is usually associated with abstract thinking and problem solving. Spreading activity across the cortex can be considered to be an attempt to correlate asynchronous input data with the information stored in long term memory. If the activity is low, it might be sufficient to activate only the supragranular pyramidal neurons of the receiving column, so the information is processed without any external output.

The corticosub-cortical efferents carry the signals that represent commands to be executed by the lower level neural subsystems. There are, for example, axons that terminate in the basal ganglia or the spinal cord and the signals carried over those connections trigger more or less complex motor actions. The level of excitation of the infragranular pyramidal neurons represent an overall state of the column, because of their integrating characteristics described before.

The corticosub-cortical efferents also include the cortico-thalamic efferents that are used to modulate the sensory data that is being transmitted via the thalamocortical afferents to the cortex. The cortex may control the level of input activity by enhancing the sensitivity to sensations that are being considered important or suppressing the signals that are regarded as being noise.

B.4 The cortical column as a processing element

The cortical column is an elementary information processing unit of the cerebral cortex. It receives an input that arrives at the granular layer and is distributed vertically across the column: first to the supragranular, later to the infragranular layer. The level of input activity that is required to activate the upper pyramidal neurons is lower than the level required to activate the infragranular cells. The activity in the higher sections of the column represents a concept or feature with which the column is associated. When the pyramidal cells in layers 2 or 3 are activated, they can excite other columns, also with some well defined representation, via the cortico-cortical efferents. In that way, various features, concepts, aspects of the sensation, etc., can be correlated. If several columns project to the same column, the integrated input activity might be sufficient not

only to activate the upper pyramidal neurons, but also the lower pyramidal cells. That might trigger an external action that will, in turn, change the overall pattern of activity in the cortex through the feedback mechanism.

The following three states of a cortical column can be defined on the basis of the observations presented so far:

- no, or very low, activity, suggesting that the feature or concept associated with the column has neither been detected by the sensors nor recalled from the memory (definitive NO),
- activity only in the higher sections of the column, suggesting that the feature or concept has been recalled from memory due to some internal processes (MAYBE), and
- activity in the whole column, suggesting that the feature or concept has been confirmed as present in the current context (definitive YES)

In each of those states, the column may receive input signals from other cortical regions or the thalamus. The columnar output is completely passive in the NO state. While in the MAYBE state, the column sends signals only through its cortico-cortical efferents, because the activity is too low in the lower section of the column. The YES state results in the firing of the infragranular pyramidal neurons. In addition to the possibility of inducing an external action, high activity in the lower layers may activate inhibitory interneurons that suppress the activity in the column. If a partial activation of a column is considered as an anticipation of the presence of a feature or concept, or as a goal, the phenomenon that shuts down the activity in the column after the column is completely activated can be treated as a neural implementation of goal satisfaction.

B.5 Properties of a network of cortical columns

The cortical columns, through their efferents and afferents, constitute a network of interconnected processing units. Some of the properties of the cortex were described in Appendix A. Those properties were related to the capability of the cortex to organize into topographical maps. The maps in

the sensory cortex reflect the physical world as perceived by human sensors and communicated to the cortex via the thalamo-cortical connections. On the other hand, the maps that are created in the motor cortex are organized according to the layout of the body parts that they control. The motor cortex includes the frontal areas that are considered to be the location in the cortex where the most abstract processing occurs. There are less efferents leaving the cortex from those areas - most of them terminate somewhere else in the cortex, possibly closer to regions directly controlling the motor neurons.

The capability to generate feature maps comes from the competitive nature of lateral interconnections between columns that was described earlier. It is a very interesting aspect of the overall functionality of the cortex that has been studied by many researchers. There have been several artificial networks implemented, for example Kohonen's network, that behave in a similar way; i.e., where local specialization develops when sample data is fed into the network. Ultimately, certain features or concepts will activate a single unit or a well defined set of units. The relationships between the images in the network, as represented by the patterns of the connectivity, are a reflection of the relationships in the training sample data. There is strong evidence that the same process occurs in the cortex.

From the perspective of this work, other capabilities of a network of cortical units are more interesting. It has been suggested in the previous section that a lightly activated (i.e., only the pyramidal neurons in the upper sections of the column are active) column can represent an anticipation of a certain feature or concept. That is equivalent to saying that the column represents a certain goal. If the anticipation is satisfied, a conclusion is that the goal has been reached. The initial anticipation is probably the result of a desire or need born somewhere else in the brain. The limbic system may translate such needs into signals that are carried over the cortex afferents to appropriate columns. Many columns are probably involved in such a complex task. Each of the involved columns represents a sub-goal that must be satisfied to achieve the global goal. The activity that represents a sub-goal, that from the local perspective is just another goal, is spread across the cor-

tex through the cortico-cortical efferents. None of the receiving columns get highly activated, unless they are also anticipating or getting external inputs from the sensors. In such a case, the anticipation of the receiving columns is fulfilled (i.e., the goal is satisfied) and the columns are shut down by the inhibitory interaction between the lower and upper layers.

Although that is only a hypothesis, there is research data that seems to support the described behavior. It has been demonstrated that before a hand can be moved, for example, to a certain desired position, the column in the sensory cortex that represents the hand in that position is activated slightly before the actual movement takes place. The hand is in another position at that time, so the activity cannot be caused by a sensation. It seems that the cortex is presented with the goal that is personified by the activity in the column. The activity that spreads throughout the cortex finally reaches the column that represents the current position of the hand. Because the hand is actually in that position, the sensors send signals to the column's input. The result is that the column now becomes highly activated. That activity can easily be transmitted to the primary motor cortex that is more or less symmetrical, with respect to the topographical maps, to the somatosensory cortex. That may trigger a motor action that will move the hand into another position that is closer to the goal position. The next sensory column once again gets two inputs: one from the sensors and another from the anticipating column, so the same process occurs. Finally, the hand is guided to the desired position, so the first column in the chain gets highly activated and shuts down. The goal has been achieved. The hand is in the desired position.

The process of moving a hand has been dramatically simplified here to illustrate the capability of the cortex to perform searches. The search trees in the cortex might be very complex with many branches at many levels. The basic mechanism, however, seems to be the same at every point.

There are certain activities that are being performed routinely, without any degree of uncertainty. That may be due to the fact, that certain cortical paths are repeated many times, so the influence of certain columns on others becomes so strong that the activity is transmitted without any loss of its intensity between the columns. Therefore, if a column gets activated with a sensory input, it may

activate the one that is tightly coupled even if the other cell is not in the anticipating state. If that happens in the somato-sensory cortex, the activity can be transmitted to the motor cortex and the movements can be generated as described earlier

References

- [1] Albus, J. S. (1981), *Brains, behaviors and robotics*, Byte Publications Inc.
- [2] Alexander, I. (1989), *The logic of connectionist systems*, in *Neural computing architectures: The design of brain-like machines*, Alexander, I., Ed., MIT Press, Cambridge, MA
- [3] Ballard, D. H. (1986), *Cortical connections and parallel processing: structure and function*, Behavioral and Brain Sciences.
- [4] Barnard, E. (1993), *Temporal-difference methods and markov models*, IEEE Transactions on Systems, Man and Cybernetics, vol. 23, no. 2, pp. 357-365.
- [5] Bieszczad, A. and Pagurek, B. (1996), *Planning rearrangements in a blocks world with the Neurosolver*, Proceedings of the World Congress on Neural Networks WCNN'96, pp. 856-859, San Diego, CA, INNS Press, Lawrence Erlbaum Associates, New York.
- [6] Bieszczad, A. and Pagurek, B. (1996), *Neurosolver in a blocks world*, Proceedings of IEEE International Conference on Neural Networks ICNN'96, Vol. II, pp. 1215-1220, Washington, DC, IEEE Press.
- [7] Bieszczad, A. and Pagurek, B. (1995), *Neurosolver-based agents for solving blocks world problems*, Joint Conference on Information Sciences JCIS'95, Plan C: First International Conference on Computational Intelligence and Neurosciences, Wrightsville Beach, NC
- [8] Bieszczad, A. (1994a), *Neurosolver: a neural network based on a cortical column*, Proceeding of the World Congress on Neural Networks WCNN'94, Vol. II, pp. 756-761, San Diego, CA, INNS Press, Lawrence Erlbaum Associates, New York

- [9] Bieszczad, A. (1994b), *Neurosolver: A step toward a neuromorphic general problem solver*, Proceedings of IEEE World Congress on Computational Intelligence WCCI'94, Vol. III, pp. 1313-1318, Orlando, FL, IEEE Press
- [10] Bieszczad, A. (1992), *Neurosolver: A neural network based on a cortical column*, Master's Thesis, Carleton University, Ottawa, Canada.
- [11] Bieszczad, A., White, T. (1992), *Customization of network surveillance expert systems*, Proceedings of GLOBCOM'92, Orlando.
- [12] Broadman, K. (1909), *Vergleichende lokalisationlehre der grosshirnrinde in ihren prinzipien auf grund des zellenbaues*, J. A. Barth, Leipzig
- [13] Broca, P. (1861), *Perte de la parole*, Bulletin de la Societe d'Anthropologie, Vol 2, pp. 235-237.
- [14] Burnod, Y. (1988), *An adaptive neural network: The cerebral cortex*, Masson, Paris
- [15] Buchsbaum R., (1948), *Animals without backbones*, University of Chicago Press
- [16] Chaib-draa, B. (1995), *Distributed artificial intelligence: An overview*, Technical Report, Department d'Informatique, Faculte des Sciences et de Genie, Universite Laval, Saninte-Foy, Quebec.
- [17] Cho, B., Rosenbloom, P. S. and Dolan, C. P. (1991), "*Neuro-SOAR: A neural network architecture for goal-oriented behavior*", Proceedings of Thirteenth Annual Conference of the Cognitive Science Society, Lawrence Erlbaum Associates, Chicago, IL.
- [18] Coghill, G. E. (1929), *Anatomy and the problem behavior*, Cambridge University Press, Cambridge, MA.
- [19] Connolly, C. L. (1993), *A robotics perspective on motor programs and path planning*, Behavioral and Brain Sciences, vol. 15, no 4, pp. 728-729
- [20] Elliot, H. (1969), *Textbook of neuroanatomy*, Lippincott, Philadelphia and Toronto

- [21] Etzioni, O. and Weld, D. S. (1995), *Intelligent agents on the Internet: Facts, fiction and forecast*, IEEE Expert, August, pp. 44- 49.
- [22] Feigenbaum, E. A. et al. (1987), *Handbook of Artificial Intelligence*, Morgan Kaufman, Los Altos, CA
- [23] Freeman, J. A. and Skapura, D. M. (1991), *Neural Network: Algorithms, applications and programming techniques*, Addison-Wesley, Reading, MA.
- [24] Goldszmidt, G. and Yemini, Y. (1995), *Distributed management by delegation*, Proceedings of the 15th International Conference on Distributed Computing Systems; available online from Computer Science Department, Columbia University.
- [25] Hamilton, L. W. (1976), *Basic limbic system anatomy of the rat*, Plenum, New York and London.
- [26] Holmes, G. M. (1956), *Selected papers*, Walshe, F. M. (Ed.).
- [27] Hopfield, J. J. and Tank, D. W. (1985), *Neural computation of decisions in optimization problems*, Biological Cybernetics, vol. 52, pp. 141-152
- [28] Jordan, M. I. (1989), *Serial order: A parallel, distributed processing approach*, in Advances in Connectionist Theory: Speech
- [29] Kohonen, T. (1988), *Self-organization and associative memory*, Springer-Verlag, Berlin, 2nd edition
- [30] Kolb, B. and Whishaw, I. Q. (1990), *Fundamentals of human neuropsychology*, W. H. Freeman & Co., New York
- [31] Laird, J. E., Newell, A. and Rosenbloom, P. S. (1987), *SOAR: An architecture for general intelligence*, Artificial Intelligence, Vol. 33, pp. 1-64.

- [32] Luderer, G., Ku, H., Subbiach, B. and Narayan, A (1996), *Network Management agents supported by a Java environment*, Technical Report, Arizona State University, Tempe (<ftp://www.eas.asu.edu/pub/centers/asufc/isinm96ps.ps>) Submitted to ISINM'97, SanDiego CA.
- [33] Luria, A. R. (1973), *The working brain*, Penguin, Harmondsworth
- [34] Maes, P. (1991), *A bottom-up mechanism for behavior selection in an artificial creature*, in Meyer, J. A. and Wilson, S. W. (Eds.) *From Animals to Animats*, MIT Press, Cambridge, MA.
- [35] Merkle, R. C (1996), *Nanotechnology and neurons*, a plenary lecture at the World Congress on Neural Networks WCNN'96, San Diego, CA
- [36] Minsky, M. and Papert, S. A (1969), *Perceptrons - An introduction to computational geometry*, MIT Press, Cambridge, MA
- [37] Mozer, M. C. (1993), *Neural net architectures for temporal sequence processing*, in Weigend, A. and Gershenfeld, N. (Eds) *Predicting the future and understanding the past*, Addison-Wesley Publishing, Redwood City CA.
- [38] Mountcastle, V. B. (1978), *An organization principle for cerebral function: the unit module and the distributed system* In *The mindful brain*, Schmitt, F. O., MIT Press, Cambridge, MA.
- [39] Mrsic-Flogel, J. (1992), *Aspects of planning with neural networks*, Doctoral Dissertation, Imperial College of Science, Technology and Medicine, The University of London.
- [40] Newell, A. and Simon, H. A. (1963), *GPS: A program that simulates human thought*, in Feigenbaum, E. A. and Feldman, J. (Eds.), *"Computer and thought"*, McGrawHill, New York.
- [41] Nilsson, Nils J. (1980), *Principles of artificial intelligence*, Tioga, Palo Alto, CA

- [42] Penfield, W and Jasper, H H (1954), *Epilepsy and the functional anatomy of the human brain*, Little, Brown, Boston, MA
- [43] Penfield W and Rasmussen, T. (1950), *The cerebral cortex in man*, Macmillan Publishing Co
- [44] Ramón a Cayal, S (1899), *Estudio sobre la corteza cerebral humana*, Rev Trim. Microscopia, 4.
- [45] Rosenblatt, F. (1961), *Principles of neurodynamics: Perceptrons and the theory of brain mechanisms*, Spartan Books, Washington, DC.
- [46] Rosenbloom, P. S., Laird, J. E., Newell, A. and McCarl, R. (1991), *A preliminary analysis of the Soar architecture as a basis for general intelligence*, Artificial Intelligence 47, pp 289-325, Elsevier Science Publishers B.V., Amsterdam, Netherlands
- [47] Rumelhart, D E., Hinton, G. E., Williams, R. J. (1986), *Learning internal representations by error propagation in Parallel distributed processing*, vol. 1, Rumelhart, D. E., McClelland J. L., Eds., MIT Press, Cambridge, MA
- [48] Russel, S and Norvig, P., (1996), *Artificial Intelligence: A Modern Approach*, Prentice Hall, Englewood Cliffs, NJ.
- [49] Shepard, G M (1990), *Synaptic organization of the brain*, Oxford University Press, New York, 3rd edition.
- [50] Shonwalder, J. (1996), *Distributed Network Management: Approaches and problems*, Institute Fur Betriebssysteme und Rechnerverbund, Technical Report, TU Braunschweig, Germany
- [51] Sun, R. (1994), *A neural network model of causality*, IEEE Transactions on Neural Networks, Vol 5, No 4
- [52] Sutton, R S (1988), *Learning to predict by the methods of temporal differences*, Machine Learning, vol. 3, pp. 9-44

- [53] (1995) *Stuttgart Neural Network Simulator 4.0, User Manual*, Institute for Parallel and Distributed High Performance Systems, University of Stuttgart, 1995
- [54] Szentagothai, J. (1969), *Architecture of the cerebral cortex* in Jasper, H. H. Ward, A. A. and Pope, A., *Basic mechanisms of the epilepsy*, Little, Brown, Boston
- [55] Szentagothai, J. (1979), *The module concept in cerebral cortex architecture*, Brain Research.
- [56] Tesauro, G. (1992), *Practical issues in temporal difference learning*, Machine Learning, vol. 8, pp. 257-277
- [57] (1996) *The Simple Times*, The Quarterly Newsletter of SNMP Technology, Vol. 4, Number 3, July.
- [58] Touretzky, D. and Hinton, G. (1988), *A distributed connectionists production system*, Cognitive Science, vol. 12, no. 3, pp. 423-466
- [59] Waibel, A., Hanazawa, T., Hinton, G., Shikano, K. and Lang, K. (1989), *Phoneme recognition using time-delay neural networks*, IEEE Transactions on Acoustic, Speech and Signal Processing, vol. 37, pp. 328-339.
- [60] Weld, D. S. (1994), *An introduction to least commitment planning*, AI Magazine, Winter 1994, pp. 27-60
- [61] Wernicke, C., (1874), *Der aphasische symptomengruppe*, M. Cohen et Weigert, Breslau.
- [62] White T., Bieszczad A. (1992), *The Expert Advisor: An expert system for real time network monitoring*, European Conference on Artificial Intelligence, Workshop on Advances in Real Time Expert System Technology
- [63] White, T., Bieszczad, A. (1992), *A customization environment for the expert advisor network management system*, Proceedings of Ninth Canadian Conference on Artificial Intelligence, pp. 82-89

- [64] Winston, P (1986), *Artificial Intelligence*, MIT Press, Cambridge, MA
- [65] Wooldridge, M and Jennings, N R , Eds , (1994), *Intelligent agents*, Springer-Verlag, Berlin
- [66] Wooldridge, M and Jennings, N R (1995), *Intelligent agents: Theory and practice*, submitted to *Knowledge engineering review*, October 1994, revised January 1995.
- [67] Tim Wilson (1994), *Current net mgm't inadequate for ATM*, Communications Week, CMP Publications, 16 November 1994
- [68] Zeki, S and Shipp, S (1972), *The functional logic of cortical connections*, Nature, Vol 335, pp 311-317

END

1 7 10 6 - 9 7

FIN