

2016

Visualisation Studio for the analysis of massive datasets

Tucker, Roy Colin

<http://hdl.handle.net/10026.1/4870>

<http://dx.doi.org/10.24382/3308>

Plymouth University

All content in PEARL is protected by copyright law. Author manuscripts are made available in accordance with publisher policies. Please cite only the published version using the details provided on the item record or document. In the absence of an open licence (e.g. Creative Commons), permissions for further reuse of content should be sought from the publisher or author.

Copyright Statement

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the authors consent.

Visualisation Studio for the analysis of massive datasets

by

Roy Tucker

A thesis submitted to the University of Plymouth
in partial fulfilment for the degree of

Doctor of Philosophy

School of Computing and Mathematics
Faculty of Science and Technology

October 2015

Author's Declaration

At no time during the registration of the degree of Doctor of Philosophy has the author been registered for any other University award without the prior agreement of the Graduate Committee

Relevant scientific seminars and conferences were regularly attended at which work was presented and several papers were published.

1.1 Publications:

14th International Conference – Information Visualisation (IV 2010) Extended Abstract 'Using VISA to visualise multi-dimensional spike train data sets' – London

VISA Carmen Cross Correlation Service released – 20th July 2012

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications, 2 pp 352-358.

Tucker, R., Gunaratne, S., Barlow, N. & Stuart, L. (2014) 'A Scaling Cross Platform Tool for the Analysis of Neurophysiological Data'. International Journal of Computer Application, 3 (4). pp 41-56.

1.2 Conferences Attended:

14th International Conference – Information Visualisation (IV 2010)

CARMEN Consortium meeting 23rd – 24th May 2011

2012 International Conference on Parallel and Distributed Processing Techniques and Applications 18th July 2012, Las Vegas.

From Maps to Circuits: Models and Mechanisms for Generating Neural Connections (MAP 2014), 28/29 July 2014, Edinburgh UK

Signed.....

Dated: 22/03/2016

Thesis word count: 79,472

Acknowledgements

I would like to take this opportunity to express both thanks and gratitude to my supervisory team of Dr Liz Stuart and Dr Nigel Barlow for the endless patience and coffee supply that makes this thesis possible.

A special thanks to Professor Steve Furnell - Head of School for the School of Computing and Mathematics at Plymouth University for his support.

A special thanks to Dr Evelyne Sernagor of the Newcastle University Medical School who provided several data sets of 1500+ simultaneously recorded spike trains. These proved invaluable in testing the software.

Thanks is also due to the staff and research students of the School of Computing, Electronics and Mathematics (Faculty of Science and Engineering) there are too many amazing people to mention but my special thanks goes to Sam Gunaratne, Dean Thomas, Fahad Almansour, Torbjorn Dahl, Pushpa Subramaniam, Martin Beck, Shirley Atkinson, Mary Squire, Nicholas Outram, Julie Taylor, Sam Green, Carole Watson, Peter Mills, Antonio Rago and Kurt Langfeld.

As always the invaluable support of my family Mother, Father and Brother make my study a possibility. Finally my thanks and gratitude to Alan Griffiths, who is no longer with us, and without whom this work would never have been begun.

Abstract

Visualisation Studio for the analysis of massive datasets by Roy Tucker

This thesis describes the research underpinning and the development of a cross platform application for the analysis of simultaneously recorded multi-dimensional spike trains. These spike trains are believed to carry the neural code that encodes information in a biological brain. A number of statistical methods already exist to analyse the temporal relationships between the spike trains. Historically hundreds of spike trains have been simultaneously recorded, however as a result of technological advances recording capability has increased. The analysis of thousands of simultaneously recorded spike trains is now a requirement.

Effective analysis of large data sets requires software tools that fully exploit the capabilities of modern research computers and effectively manage and present large quantities of data. To be effective such software tools must; be targeted at the field under study, be engineered to exploit the full compute power of research computers and prevent information overload of the researcher despite presenting a large and complex data set.

The Visualisation Studio application produced in this thesis brings together the fields of neuroscience, software engineering and information visualisation to produce a software tool that meets these criteria. A visual programming language for neuroscience is produced that allows for extensive pre-processing of spike train data prior to visualisation. The computational challenges of analysing thousands of spike trains are addressed using parallel processing to fully exploit the modern researcher's computer hardware. In the case of the computationally intensive pairwise cross-correlation analysis the option to use a high performance compute cluster (HPC) is seamlessly provided. Finally the principles of information visualisation are applied to key visualisations in neuroscience so that the researcher can effectively manage and visually explore the resulting data sets. The final visualisations can typically represent data sets 10 times larger than previously while remaining highly interactive.

Contents

Copyright Statement	1
Author's Declaration.....	3
1.1 Publications:	3
1.2 Conferences Attended:	3
Acknowledgements.....	4
Abstract.....	5
Contents	6
List of Figures	11
Introduction.....	15
1 Overview.....	16
1.1 The Research Question	17
1.2 Outcomes and Contribution	17
1.3 A Tour of the Thesis.....	18
Visualisation.....	20
2. Overview.....	21
2.1 Historical Visualisation Applications	22
2.2 The Development of the Thermic Map, Flow Map and Information Graphics	24
2.3 The Visualisation Process.....	30
2.3.1 Data Gathering	31
2.3.2 Data Transformations	31
2.3.3 Graphics engine visual mappings	31
2.3.4 Visual and Cognitive Processing, Data Exploration and View Manipulation ...	36
2.4 Summary	39
Neuroscience.....	40
3 Overview.....	41
3.1 History of Neuroscience.....	41
3.2 The Neuron as the core component of the nervous system	41
3.2.1 Dendrites	43
3.2.2 Axon	43
3.2.3 Synapse.....	44
3.3 The action potential or 'spike' and the generation of spike trains.....	46

3.3.1	Overview of neuron encoding schemes	46
3.3.2	Rate encoding schemes	47
3.4	Connectivity between neurons in a biological neural network.....	48
3.4.1	Direct Coupling	48
3.4.2	Indirect Coupling.....	49
3.4.3	Common Input Coupling	49
3.5	Recording Neural Network Activity.....	49
3.5.1	The Multi Electrode Array (MEA's)	49
3.5.2	The Data Explosion in Neural Science.....	51
3.6	Analysing Neural Network Recordings	51
3.6.1	iRaster – The classic spike train raster plot.....	52
3.6.2	iGrid – The Cross Correlation grid.....	52
3.6.3	iAnimate – The Multi Electrode Array firing animation	52
3.6.4	Challenges of big data in neuroscience.....	52
3.7	Summary	54
	Parallelism	55
4	Overview.....	56
4.1	Historical development of parallel computation	56
4.1.1	Multiprogramming systems and the LEO III	56
4.1.2	Co-operative Multitasking.....	56
4.1.3	Pre-emptive Multitasking.....	57
4.2	Features of Multiprogramming / Multitasking processing.....	58
4.2.1	Categories of computer processes.....	58
4.2.2	Processes and threads of execution	58
4.2.3	The operating systems thread scheduler and the thread blocking process	59
4.2.4	Multiprogramming / multitasking is not parallel computation.....	60
4.3	Development of true parallel computation	60
4.4	Forms of computation	62
4.4.1	Single instruction, single data stream (SISD)	62
4.4.2	Single instruction, multiple data streams (SIMD).....	62
4.4.3	Multiple instructions, single data stream (MISD)	62
4.4.4	Multiple instructions, multiple data streams (MIMD)	63
4.5	Embarrassingly Parallel Problems	63
4.6	Parallel Programing – The failure to launch	64
4.6.1	Concurrency Frameworks – An abstraction layer for parallel coding.....	65
4.6.2	The spread of Concurrency Frameworks in modern development languages	65

4.6.3	General structure of a concurrency framework.....	67
4.7	The future for parallel / concurrent computation	69
5	Overview.....	71
5.1	The Information Processing Cycle	71
5.1.1	The Input Phase	72
5.1.2	The Processing Phase.....	72
5.1.3	The Output Phase.....	73
5.2	Dataflow Programming Languages & Visual Programming Languages.....	73
5.2.1	Dataflow implementation models	75
5.2.2	Dataflow execution models	76
5.2.3	Selection of dataflow model and execution method for VISA 3 implementation	77
5.3	The Visualisation of Inter-Spike Associations (VISA project).....	78
5.3.1	VISA Goals	78
5.3.2	VISA3 Development Goals	79
5.3.3	Neuroscience Visualisations	80
	Research and Software Development Methodology.....	84
6	Overview.....	85
6.1	Research Methodology	85
6.1.1	Theoretical Research.....	85
6.1.2	Empirical Research.....	86
6.1.3	Selected Research Methodology – Empirical Positivist	87
6.2	Project goals / aims.....	89
6.3	Software Development Methodology	90
6.4	The Spiral Development Methodology	91
6.4.1	Risk driven process model generator	91
6.4.2	Cyclic approach for incrementally growing a system	92
6.4.3	Key research outputs	93
	Designing iPipeline & the neuroscience visual programming language	94
7	Overview – iPipeline	95
7.1	Definition of terms.....	95
7.2	The ‘thin’ framework layer	97
7.2.1	Workflow Desktop.....	97
7.2.2	Process base classes	100
7.2.3	Data model support classes.....	103
7.2.4	Parallel execution engine	109

8	Creating a data model for neuroscience spike train recordings	111
8.1	Designing the data model	111
8.1.1	Managing big data in the data model	111
8.1.2	The theory of binary trees and self-balancing red-black binary trees.....	114
8.1.3	Supporting Shneiderman’s principle of “Overview first” and “Details on demand” 120	
8.1.4	Creating the dataflow ‘token’ using the developed data model.....	120
8.1.5	Supporting history and relationship tracking.....	121
	The i-Raster Visualisation	125
9	Overview of the i-Raster Visualisation.....	126
9.1	Drawbacks of Somerville’s i-Raster Visualisation with modern neuroscience datasets.....	128
9.1.1	Addressing the dataset density problem through the Visual Programing Language (VPL).....	129
9.1.2	Addressing the dataset density problem through additional i-Raster functionality.....	138
	The i-Grid Visualisation	143
10	Overview of the i-Grid Visualisation	144
10.1	Overview of Pair-wise Cross Correlation.....	144
10.1.1	Computational challenge of Pair-wise Cross Correlation.....	145
10.1.2	MPJ Express as an implementing environment for i-Grid’s cross-correlation algorithm.....	147
10.1.3	Implementation issues of i-Grid’s cross-correlation algorithm.....	148
10.1.4	Algorithm performance – Lab PC vs High Performance Computing (HPC) cluster 150	
10.2	Identifying interconnected neural clusters	153
10.3	Scaling the iGrid Visualisation.....	155
10.4	The Dendrogram Visualisation	157
10.4.1	The dendrograms visual encoding scheme.....	160
10.4.2	The dendrogram “forest” view	161
10.4.3	The dendrogram “un-clustered view”	168
10.4.4	The dataset’s “Overview” view	169
	The i-Animate Visualisation.....	171
11	Overview of the i-Animate Visualisation.....	172
11.1	Goals of the i-Animate Visualisation.....	173
11.2	Basic implementation details.....	173
11.3	The information overlay options in i-Animate.....	177

11.3.1	The MEA grid overlay.	177
11.3.2	The Heat map overlay selection.....	178
11.3.3	Implemented heat maps in the i-Animate visualisation.....	181
12	Conclusions.....	183
12.1	Contributions.....	183
12.1.1	The Visualisation Studio (i-Pipeline).....	183
12.1.2	The Neural Science Problem Domain Library (i-Pipeline)	183
12.1.3	The i-Raster Visualisation	184
12.1.4	The i-Grid Visualisation.....	184
12.1.5	The i-Animate Visualisation.....	185
12.2	Future Development	185
12.2.1	Exploitation of high performance GPU hardware	185
12.2.2	Distributed Computing and Cloud Computing	186
12.2.3	Application to other problem domains	186
12.3	Conclusion.....	186
	Appendix 1: Creating a new iPipeline toolbox process	188
1	Creating iPipeline Toolbox Processes.....	189
1.1	Data processing algorithm to segment data by a time window	189
1.2	Planning the algorithm	189
1.2.1	How the problem domain data model is modified by this algorithm	190
1.2.2	Permitted connections to / from the time segmenting algorithm	190
1.2.3	Process settings panel(s).....	190
1.2.4	Creating visual elements.....	191
1.2.5	The iPipeline Toolbox	191
1.3	Implementing a skeleton data analysis process	192
1.3.1	Time Segmenting algorithms settings panel GUI creation	195
1.3.2	Time segmenting algorithm's ISettingsPanel interface implementation	196
1.3.3	Responding to OK / Cancel button events on the settings panel.....	200
1.3.4	Integrating the settings panel with the parent process.....	201
1.3.5	Implementation of process core algorithm.....	202
1.3.6	Introducing the new process to iPipeline's problem domain	204
	Appendix 2: Glossary of Terms	206
	Appendix 3: Publications.....	209
	References	237

List of Figures

Figure 2-1: Recent history of visualisation (Friendly & Denis, 2001).....	22
Figure 2-2: Lascaux Cave art with modern constellations shown (Rappenglück, 1999).....	23
Figure 2-3: The Catal Hoyuk town map as it now appears (Slide 100, Museum at Konya, Turkey and Slide 100D Semitic Museum at Harvard University (Cambridge, MA))(Brock, 2001)	23
Figure 2-4: The Catal Hoyuk town map as it might have appeared when created (Brock, 2001)	23
Figure 2-5: The Turin Papyrus Map from ancient Egypt (Harrell & Brown, 1992a)	24
Figure 2-6: The “Designatio orbis Christiani” an early "Thermic Map" (Hondius & Purchas, 1607)	25
Figure 2-7 John Snows 1854 thermic map of the London Broad Street Cholera outbreak (Snow, 1855)	26
Figure 2-8: Florence Nightingale's Polar Area Diagrams or Rose Charts analysing causes of mortality among injured soldiers in the Crimea War of 1853 - 1856 (Nightingale, 1859).....	27
Figure 2-9: Minard’s Map of Napoleon’s losses in the Russian campaign of 1812, the maps title reads “Figurative map of successive losses in men of the French Army in the Russian campaign 1812-1813”. (Robinson, 1967).....	28
Figure 2-10: The original London Underground map incorporating both geographic and topological information (Creemers et al., 2014).....	29
Figure 2-11: Henry Becks topological map of the London Underground as it appeared when published in 1933 (Creemers <i>et al.</i> , 2014).....	30
Figure 2-12: The visualisation process (Ware, 2012)	31
Figure 2-13: Histogram representing student grades that applies Ware’s guidelines.....	35
Figure 2-14: Data table representing student grades the breaks several of Ware’s guidelines.	36
Figure 3-1: Major structures of a neuron (amended). (Jarosz, 2009).....	43
Figure 3-2: The structure and operation of a chemical synapse. (Wikipedia, 2014).....	45
Figure 3-3: A neural spike train showing spiking activity over a 2 sec time period.	46
Figure 3-4: Coupling option between connected neurons in a neural network. (Somerville, 2011)	48
Figure 3-5: An in vitro MEA array. (Potter, 2010).....	50
Figure 3-6: An in vivo MEA array. (Normann, 1993).....	50
Figure 3-7: MEA electrode recording from a neural network. Source (Buzsaki, 2004).....	50
Figure 4-1: Execution of the pre-emptive multitasking model	59
Figure 4-2: Amdahl's Law and realisable performance gains in parallel algorithms	61
Figure 5-1: The Information Processing Cycle	71
Figure 5-2: A dataflow system visualised as a directed graph. (Source: (Microsoft, 2012)).	75
Figure 5-3: Raster plot of 20 spike trains recorded for 2000ms	80
Figure 5-4: Raster plot re-ordered to identify when neurons started to spike (burst sort)	81
Figure 5-5: iGrid plot with neurons clustered using pair wise cross correlation. (Stuart, Walter & Borisyuk, 2003).....	82
Figure 5-6: Actual simulated neural network structure used in examples.	83
Figure 6-1: Research Strategies (Remenyi & Money, 2012).....	85
Figure 6-2: The "scientific method" or classic research process (ArchonMagnus, 2015).	86
Figure 6-3: Boehm's Spiral Development Model (Boehm, 2000).	91
Figure 7-1: Structure of the iPipeline workflow framework.....	95

Figure 7-2: iPipeline’s workflow desktop component	97
Figure 7-3: Workflow to merge five data files into a single data source.	98
Figure 7-4: Workflow to merge five datasets. Two new files are created, the first of raw data and the second after sorting the raw data.	99
Figure 7-5: Thin layer base process class implementation structure.	100
Figure 7-6: The visual representation (glyph) of a process	102
Figure 7-7: Core iPipeline interfaces and their relationships	104
Figure 7-8: Settings panel for burst sort algorithm	106
Figure 7-9: Visual encoding for incomplete parameter configuration	107
Figure 7-10: The parameter management sub-system structure	108
Figure 8-1: A binary search tree for a regularly firing (10Hz) neuron recorded for one second.	115
Figure 8-2: An unbalanced binary search tree	116
Figure 8-3: A Red-Black binary search tree for a regularly firing (10Hz) neuron recorded for one second	117
Figure 8-4: Left / Right rotation in a binary search tree. Source (Cormen <i>et al.</i> , 2009). NB: the symbols α, β and γ represent any arbitrary binary search tree	118
Figure 8-5: A binary tree structure both before and after a left rotation of the marked x , y connection. Source (Cormen <i>et al.</i> , 2009).	118
Figure 8-6: The iPipeline parameter and parameter manager system	122
Figure 8-7: Implementation of the 'Command' software design pattern	123
Figure 9-1: Raster chart of twenty spike trains recorded for two seconds.	126
Figure 9-2: Spike trains re-ordered in the Y-axis from minimum to maximum inter spike interval (ISI).	127
Figure 9-3: Spike train firing rate frequency overlaid onto raster chart.....	127
Figure 9-4: The dataset after filtering to show the 10 most active spike trains with a section zoomed to show detail.	128
Figure 9-5: Somerville's i-Raster chart for 700 spike trains recorded for 30 minutes	129
Figure 9-6: Simple i-Raster VPL program.....	130
Figure 9-7: Filter to exclude "noisy" data channels (those with firing rates).	131
Figure 9-8: Generation of a grouped spike train that summarises two spike trains.	132
Figure 9-9: Visual program to group 64 spike trains together	132
Figure 9-10: A researcher’s 1411 spike train dataset "grouped" with 64 spike trains per a group	133
Figure 9-11: Detailed view of summary spike train one	133
Figure 9-12: VPL program to “burst sort” spike trains prior to grouping them	134
Figure 9-13: A researcher’s 1411 spike train dataset burst sorted and grouped with 64 spike trains per a group.....	135
Figure 9-14: Detailed view of a burst sorted group of 64 spike trains	135
Figure 9-15: Time segment process added to VPL to create six five minute time segments	136
Figure 9-16: Division of the original raster chart using the time segmenting process.....	137
Figure 9-17: First five minute time segment of 1411 spike trains burst sorted and grouped with 64 spike trains per a group	137
Figure 9-18: Detailed view of a burst sorted group of 64 spike trains (first five minute time segment).....	138
Figure 9-19: Time filtered raster chart extracted from a 1411 spike train dataset	140
Figure 9-20: Components of the interactive time filtering tool	141

Figure 9-21: Accessing the electrode display from i-Raster's primary interface.....	141
Figure 9-22: A two second burst sorted recording of 500 spike trains with the electrode display shown.	142
Figure 9-23: Electrode display filtering a dataset by recording electrode.....	142
Figure 10-1: Generation of time bin totals in pair-wise cross correlation.....	145
Figure 10-2: Number of required cross correlation calculations for neuronal networks up to 2000 neurons in size.	146
Figure 10-3: Operation of i-Grids MPJ Express implementation of the cross correlation algorithm.....	148
Figure 10-4: Cross correlation results as a nested hash map structure.....	149
Figure 10-5: Cross-correlations per second vs neural network size in desktop and HPC environments.....	153
Figure 10-6: iGrid visualisation with cross correlation histogram for neuron spike trains 1 and 7. The red bar denotes the peak cross-correlation value used for the iGrid representation.	157
Figure 10-7: Resized iGrid visualisation & cross correlation histogram placing greater emphasis on the detailed histogram while retaining awareness of its place in the correlation grid. The red bar denotes the peak cross-correlation value used for the iGrid representation.	157
Figure 10-8: iGrid dendrogram overview visual encoding scheme.....	160
Figure 10-9: A "forest" composed of two trees.....	161
Figure 10-10: Spike train dendrogram collapsed and filtered to show a sub-cluster.....	162
Figure 10-11: Filtered iGrid for spike trains 4, 11 and 3, 19.....	163
Figure 10-12: Cross-correlation chart for spike trains 3 and 11.....	164
Figure 10-13: Predicted inter-neuron connections.....	165
Figure 10-14: Actual simulated neural network connections.....	165
Figure 10-15: Cross-correlogram for spike trains 11 and 19 showing the weak but statistically significant signal (i.e. in excess of the Brillinger threshold).....	167
Figure 10-16: iGrid visualisation for the second step of neural network mapping.....	168
Figure 10-17: The "Un-clustered view" with the spike trains filtered out from iGrid.....	169
Figure 10-18: Fully expanded overview showing the test data sets structure.	170
Figure 11-1: Primary i-Animate controls.....	173
Figure 11-2: i-Animate time and time filtering controls.....	174
Figure 11-3: Stimulation event triggers initial response from the neural network.....	175
Figure 11-4: A wave of neural activity has formed and is spreading through the network..	176
Figure 11-5: The wave's passage activates three neuron clusters which exhibit a period of intense spiking activity.....	176
Figure 11-6: By approximately 12 seconds after the stimulating event the nextwork has returned its unstimulated state.....	176
Figure 11-7: A wave of neural spiking activity moving through the neural networks most active region.	177
Figure 11-8: MEA Grid showing isolated clusters of active neurons and densely packed regions of active clusters.....	177
Figure 11-9: The classic rainbow colour map. (Moreland, 2009a).....	178
Figure 11-10: Divergent colour map for scientific visualisation.....	180
Figure 11-11: Moreland's colour space applied to geo-spatial data along with its RGB colour mapping for normalised scalar values. (Moreland, 2009b).....	180

Figure 1-1: Dividing a single spike train recording into ten data sets using a 200ms time window.....	189
Figure 1-2: Time segmenting process setting panel (mock up)	191
Figure 1-3: A deployed iPipeline framework with process library	204
Figure 1-4: Neural Analysis ProcessLibrary package	204
Figure 1-5: Batch script to generate the ProcLib.jar file from the compiled code.	205
Figure 1: Directed Graph of a VISA ³ Visual Program	217
Figure 1: An example of a typical spike train recording for three neurons over a period of 500ms.....	224
Figure 2: An example of (i) direct synaptic coupling and (ii) common input coupling	225
Figure 3: Example cross-correlogram for two connected neurons.....	225
Figure 4: An example of a Cross Correlogram calculation using a six-bin window	226
Figure 5: Example Brillinger normalised cross-correlogram with confidence interval.....	226
Figure 6: Example Correlation Grid showing only significant peaks (bin size 2ms, window size 100)	227
Figure 7: Example Correlation Grid, showing only significant peaks and clustered (bin size 2ms, window size 100).....	227
Figure 8: Neuron assembly for test data set.....	227
Figure 9: Number of required cross correlation calculations for neuronal networks up to 2000 neurons in size.....	228
Figure 10: Overview of the parallel cross correlation algorithm	231
Figure 11: Dendrogram plot of the first two spike train clusters in a 50 neuron network	232
Figure 12: iGrid showing significant cross-correlation peaks in a 50 neuron test dataset ..	232
Figure 13: Dendrogram for 50 neuron test dataset.....	233

Chapter 1

Introduction

1 Overview

The human brain consists of over 10^{12} neurons which form in excess of 1000 trillion synaptic connections between them. The neurons communicate by transmitting short (1ms) electrical signals (known as spikes) through the synaptic connections. From these three simple components; neuron, synaptic connection and electrical signal all the complex neuronal structures of the human brain arise. These structures grant us the ability to learn, memorise, reason, experience emotion and ultimately make us into conscious and self-aware entities. Despite the simplicity of the components scientific explanations for these key facets of the human brain are at best only partially complete. The fundamental question of “how does the brain work?” remains only partially answered.

The great barrier to progress in understanding the human brain is the extreme complexity of the organ. Certainly that complexity is required to give rise to such complex emergent properties as consciousness. Such complexity does, however, make it extremely difficult to understand how the brain functions. Historically scientific investigation has been limited by the ability to simultaneously record the activity of large numbers of neurons. These recordings are known as spike trains and have been limited to tens or hundreds of neurons. The introduction of silicon-based multi-electrodes comprising hundreds or more electrode contacts offers unprecedented possibilities for simultaneous recordings of spike trains from thousands of neurons (Einevoll et al., 2012). This increase in the number of recorded spike trains opens up the serious possibility of:

- mapping neuron connectivity in regions of the brain
- decoding the meaning of the signals transmitted between neurons
- expanding our knowledge of brain function and
- developing effective treatments for many brain function disorders

In order to realise the full benefits of this new technology science cannot however rely on the neuroscientist alone. It is not enough simply to record the activity of thousands of neurons and give that data to a neuroscientist. The neuroscientist must also be given tools with which to analyse and visualise the data. The creation of such tools must inevitably draw on scientific knowledge from fields outside of neuroscience. The computer stands at the heart of the modern world's ability to analyse data in large quantities. Inevitably the computer scientist must become involved in providing the tools to record and analyse large spike train data sets. Equally important is the knowledge of Visual Analytics as a tool to effectively study large data sets without overwhelming the researcher. The value of Visual Analytics to the analysis of neuroscience data has been shown by Somerville and Walter (Somerville, 2011; Walter, 2004). This research attempts to extend these techniques to the analysis of thousands of simultaneously recorded spike trains.

1.1 The Research Question

“How can Software Engineering and Visual Analytics be applied to aid the general analysis of scientific data and specifically current neural spike train data?”

To answer this question this research aims to:

- I. Develop a generic framework suitable for data analysis in any problem domain.
- II. To apply that framework to the specific problem domain of neuroscience visualisation for thousands of simultaneously recorded spike trains.
- III. To exploit parallel computation and programming as a solution for the problems of increased computational complexity arising from “big data”.
- IV. To apply data visualisation and visual analytics techniques to effectively analyse large (1000+) simultaneously recorded spike trains.

To accomplish these aims this research has the following objectives:

- I. To create a Visualisation Studio for the analysis of massive datasets (i-Pipeline).
- II. To build the Visualisation Studio in a cross platform manner.
- III. To exploit the delivery of increased computing power through multi-core architectures. This will require the application to support parallel computation.
- IV. To develop a library of data analysis processes suitable for the investigation of neuroscience data. This library should also exploit the delivery of increased computing power through multi-core architectures.
- V. To develop a set of interactive visualisations that support a researcher’s analysis of thousands of simultaneously recorded spike trains.
- VI. To identify neural coupling from spike train data using an algorithm that scales seamlessly from the researchers desktop to a university cluster computer environment.

1.2 Outcomes and Contribution

This work makes two significant contributions to the state-of-the-art. The first is the Visualisation studio itself (i-Pipeline). This uses the techniques of dataflow programming to produce naturally paralysable “programs” to analyse large data sets. The second is the enhancement of Somerville’s i-Raster and i-Grid visualisations to support large scale data sets (1000+ spike trains) and the introduction of the new i-Animate visualisation. In each case maintaining the interactive nature of the visualisations for 1000+ spike trains has placed extensive demands on the computer hardware. Modern computers deliver computational power through multi-core processor systems. Unfortunately exploiting this power requires programs to be rewritten in a manner that supports parallel computation. This work has adopted this approach from the beginning and reveals how even a researcher’s typical desktop computer can deliver “big data visualisation” performance with the correctly

developed software tools. The enhancements made to i-Raster and i-Grid can be summarised as:

- The data model underlying both visualisations has been re-developed to support much larger in memory datasets with rapid searching to maintain the performance of the interactive displays.
- Both have seen expanded “overviews” added to visually manage the larger datasets:
 - I-Raster has had a visual representation of the multi-electrode array that recorded the data set added to facilitate selection of spike trains to include on the raster chart.
 - I-Raster has also had the ability to collapse spike trains into groups added. This allows much more data to be represented by a single line of the chart. A drill down facility has been added to view the individual spike trains.
 - I-Grid has seen the cross-correlation data used with a clustering algorithm to create a dendrogram of potentially connected neurons. Filtering of items on the i-Grid visualisation is now achieved by expanding / collapsing sections of the dendrogram tree.
 - I-Grid is now rendered in an “infinite virtual space” to prevent loss of clarity with large datasets. A viewport allows the user to inspect the visualisation.
- Both visualisations have been re-coded to fully support and exploit the modern computers new paradigm of delivering compute power through multiple compute cores.

1.3 A Tour of the Thesis

After this introductory chapter a review of the history and current state of the art for the three key fields is provided in chapters 2-4 as follows:

- Chapter 2: Visualisation
- Chapter 3: Neuroscience
- Chapter 4: Parallelism

Chapter 5 explores the ideas of underlying “Visual Programming Languages” including dataflow programming. The Visualisation studio (i-Pipeline) falls into the category of a visual programming language and therefore these principles underlie its operation.

Chapter 6 examines the research and software development methodology adopted in this research project.

Chapter 7 examines the actual implementation of i-Pipeline from a technical point of view.

Chapter 8 examines the implementation of the neuroscience library, its analysis algorithms and their implementation.

Chapters 9-11 examine each of the neuroscience visualisations included in the neuroscience library as follows:

- Chapter 9: The i-Raster visualisation
- Chapter 10: The i-Grid visualisation
- Chapter 11: The i-Animate visualisation

Chapter 12 concludes the thesis with a review of the contribution and achievements made. This is followed by the appendix and references.

Chapter 2

Visualisation

“visualisation noun. A mental image that is similar to a visual perception”

Summary

In this chapter the term “Visualisation” is defined and the scope of the field examined with real world examples of its practical application.

2. Overview

Visualisation is a wide ranging field that can take many different forms. A simple review of its applications reveals how truly multi-disciplinary the field is. In brief the areas to which visualisation techniques have been applied would include:

- Information Visualisation
- Interactive Visualisation
- Scientific Visualisation
- Music Visualisation
- Geographic Visualization
- Flow Visualization

The common point across all these fields is that the user is constructing a 'mental model' of an aspect of the real world from a visual representation. In the modern information driven world this definition immediately conjures thoughts of complex 3D graphical displays created and managed by high power computers that present complex data that only skilled specialists understand. While this is certainly one aspect of visualisation in the modern world historically the field of visualisation was in effective use as a tool that helps its user make sense of the world long before the development of the modern computer, interactive graphical displays and global information sharing networks.

The construction of 'mental models' that reflect the real world is something that any creature (including humans) must undertake simply to survive in the natural world. The scientific name for humans is "Homo sapiens" literally meaning "wise man". This title is earned from our unequalled ability (in nature) to construct 'mental models' that reflect the reality of our world. The construction of mental models is however not a purely mental activity, indeed it has been observed that "It does not seem possible to account for the cognitive accomplishments of our species by reference to what is inside our heads alone. One must also consider the cognitive roles of the social and material world."(Hutchins, 1995). It therefore seems appropriate to assert that our mental models are formed both through interactions with others and the material world. Ultimately all such interactions are mediated by our five senses (sight, hearing, touch, smell and taste). Of these the sense that has evolved to provide the most data to our brains is our sense of sight (or vision). Indeed it has been observed that "Visual displays provide the highest bandwidth channel from the computer to the human. Indeed, we acquire more information through vision than through all of the other senses combined."(Ware, 2012). In the human brain there are some 20 billion neurons dedicated to the analysis of visual information and the identification of patterns within visual elements.

Throughout human history the combination of the human visual and cognitive systems has evolved to provide us with an impressive survival advantage based on the ability to identify / recognise specific patterns in visual data that is presented to us. These patterns then form the basis on which the cognitive system builds its mental model of the world.

The field of visualisation attempts to exploit this impressive, naturally evolved ability, to process and perform pattern recognition on visual data as a means of transmitting existing and discovering new knowledge. Fundamentally exploitation of this ability involves the

movement of the cognitive load from the cognitive system to the perceptual / visual system. A simple example of the effect can be seen by contrasting the time required to mentally solve an arithmetic problem with the time required to complete the same task using pen and paper. It has been shown that an individual will usually take five times longer to complete the task mentally than they do with pen and paper. (Card, Mackinlay & Schneiderman, 1999). Visualisation may then also be accurately described as a cognitive tool that aids and supports decision making. The key benefits derived from the visualisation process were summarised by Colin Ware as (Ware, 2012):

- Visualization provides an ability to comprehend huge amounts of data.
- Visualization allows the perception of emergent properties that were not anticipated.
- Visualization often enables problems with the data to become immediately apparent.
- Visualization facilitates understanding of both large-scale and small-scale features of the data.
- Visualization facilitates hypothesis formation.

While the modern computer is now the visualisation tool of choice (primarily because of its phenomenal ability to process huge amounts of data and generate a visualisation of it), the desire to derive the benefits listed above reaches back to before the dawn of recorded history. It seems then appropriate to first consider how visualisation has been applied in the historical context.

2.1 Historical Visualisation Applications

The history of visualisation is the history of the tools that humans use to amplify their cognition with the final goal of gaining knowledge. Figure 2-1 summarises the recent history, developments in and applications of visualisation.

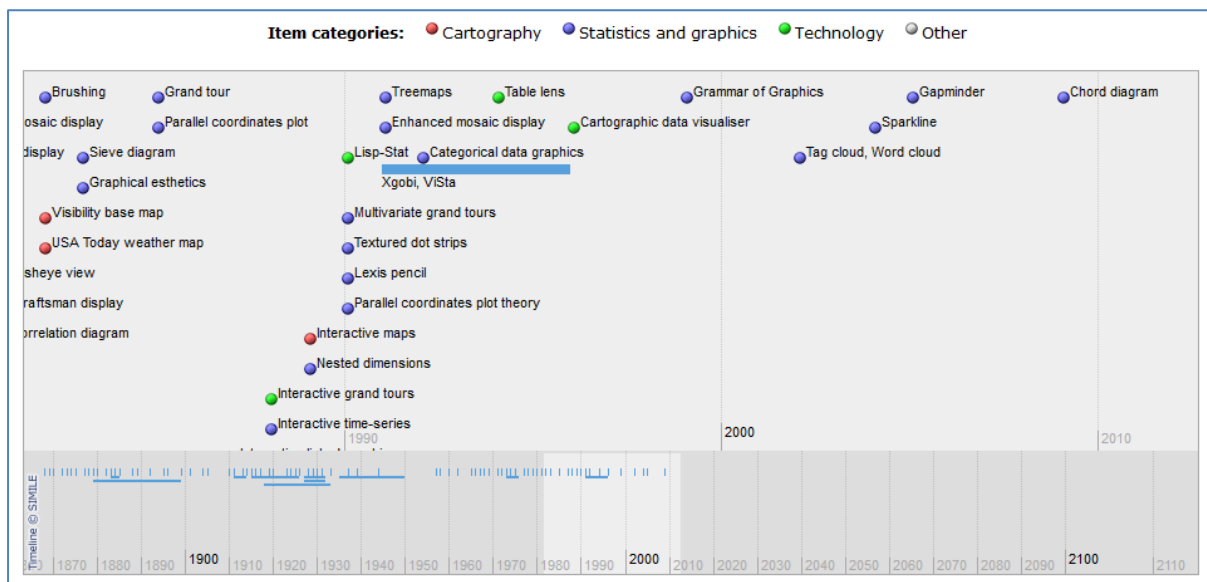


Figure 2-1: Recent history of visualisation (Friendly & Denis, 2001)

As can be seen from the visualisation in Figure 2-1 Friendly & Denis identify three primary drivers for advancement in the field of visualisation over time:

1. Cartography

2. Statistic and graphics
3. Technology

With such diverse drivers for advancement in the field and the separation of time it is impossible to say exactly at what point in time humans began visualising data. Do we include the oldest reliably dated cave art (37,300 to 40,800 year old art in the caves of El Castillo, Spain (Pike et al., 2012))? Are these graphics which are simple silhouettes of people's hands created by blowing paint onto the cave walls too primitive to be said to create a mental model causing us to prefer a more quantitative measure? Within these same caves we find a painting of a constellation Corona Borealis "Northern Crown" dated at 12,000 BC. This is however not the oldest map of the stars, in the Lascaux Caves representations of the night sky dated at 15,300 B.C. (Figure 2-2) are found (Rappenglück, 1999). Finally by circa 6,200 B.C. we find the first map of a man made settlement painted on the walls of a cave, the town of Catal Hoyuk in Turkey (Figure 2-3 & Figure 2-4).

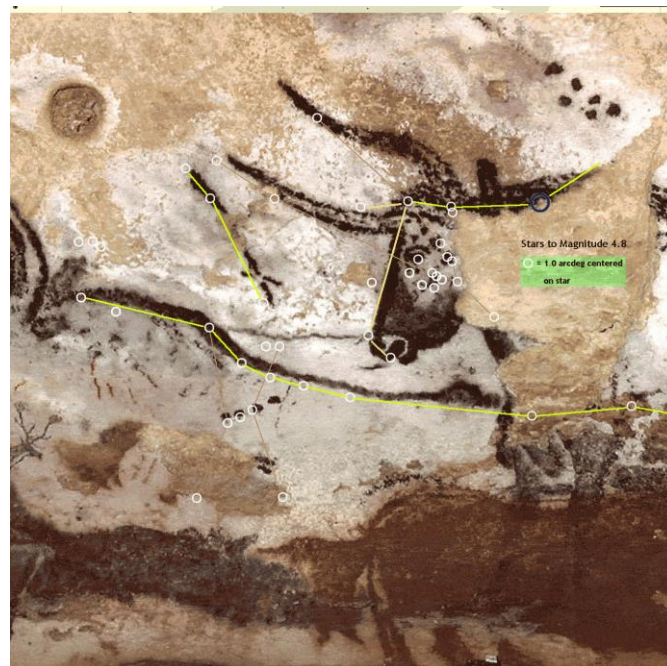


Figure 2-2: Lascaux Cave art with modern constellations shown (Rappenglück, 1999)



Figure 2-3: The Catal Hoyuk town map as it now appears (Slide 100, Museum at Konya, Turkey and Slide 100D Semitic Museum at Harvard University (Cambridge, MA))(Brock, 2001)



Figure 2-4: The Catal Hoyuk town map as it might have appeared when created (Brock, 2001)

By 1150 B.C. the Egyptians were producing maps that visualised multiple data items. Figure 2-5 shows the "Turin Papyrus Map" which is widely regarded as both the oldest

surviving topographical map and the oldest surviving geological map (Harrell & Brown, 1992a; Harrell & Brown, 1992b).



Figure 2-5: The Turin Papyrus Map from ancient Egypt (Harrell & Brown, 1992a)

In these examples can be seen all three drivers of progress in the field of visualisation at work; every example is some form of map expressing spacial relationships either on Earth or in the sky (Cartography). These relationships are represented using graphics to encode mathematical data (distances on the Earth or apparent angles in the sky – Statistics and graphics). Finally the progression of technology moves the maps from the impossible to transport cave painting, to the papyrus maps carried by Egyptian functionaries overseeing the quarrying of stone for the nation's monuments.

Given that mathematics, statistics and technology are key drivers in the advancement of visualisation as a field it is perhaps unsurprising that its true power only started to be exploited in the 17th and 18th centuries. A time that is often described as the dawning of the Age of Reason (Redwood, 1976). Over this time period many of the classical restrictions on scientific enquiry were relaxed leading to an explosion in scientific investigation. Ultimately this would lead to the industrial revolution with its profound impact on society around the world. The fields of mathematics, statistics and technology all benefited greatly from the new freedom of enquiry and, in turn, so did the fields that relied upon them - such as cartography. By the mid-17th century the production of “general maps” (that portrayed “base data” such as landforms, settlements and boundaries) had reach a point that allowed the first “thematic map” to be created.

2.2 The Development of the Thematic Map, Flow Map and Information Graphics

The thematic map is defined as “a map made to reflect a particular theme about a geographic area. Thematic maps can portray physical, social, political, cultural, economic, sociological, agricultural, or any other aspects of a city, state, region, nation, or continent.” (*Maps & Cartographic Information*, 2014). One of the oldest thematic maps (1607 A.D.) can be seen in Figure 2-6 below, in this case the map not only provides a geographic overview of the world but also uses colour and glyphs to show the distribution of Christian, Muslim and idolatrous regions of the world.



Glyphs Used	
Christianity	✝
Muslim	☪
Idoltrous	☪

Figure 2-6: The “Designatio orbis Christiani” an early "Thermic Map" (Hondius & Purchas, 1607)

The theme of the map is, of course, to show how much work still remained to convert the world to the “true faith” of Christianity (Hondius & Purchas, 1607). In such thermic maps the beginnings of modern Information Graphics can be seen. Over the next two centuries the use of thermic maps developed and began to be applied to many different fields. Ultimately this would lead to the creation of some of the most important maps in history. For example in 1854 John Snow, while investigating a cholera outbreak in London plotted 83 related deaths onto a map of the affected area and from this thermic map determined that the common factor in each case was the water pump from which the deceased had been drinking. After the pumping handle was removed from the Broad Street water well the outbreak immediately began to decline (Snow, 1855). Today the production of this thermic map (Figure 2-7) is regarded as the founding event of the modern science of epidemiology (the study of patterns, causes and effects of health and disease in defined populations).

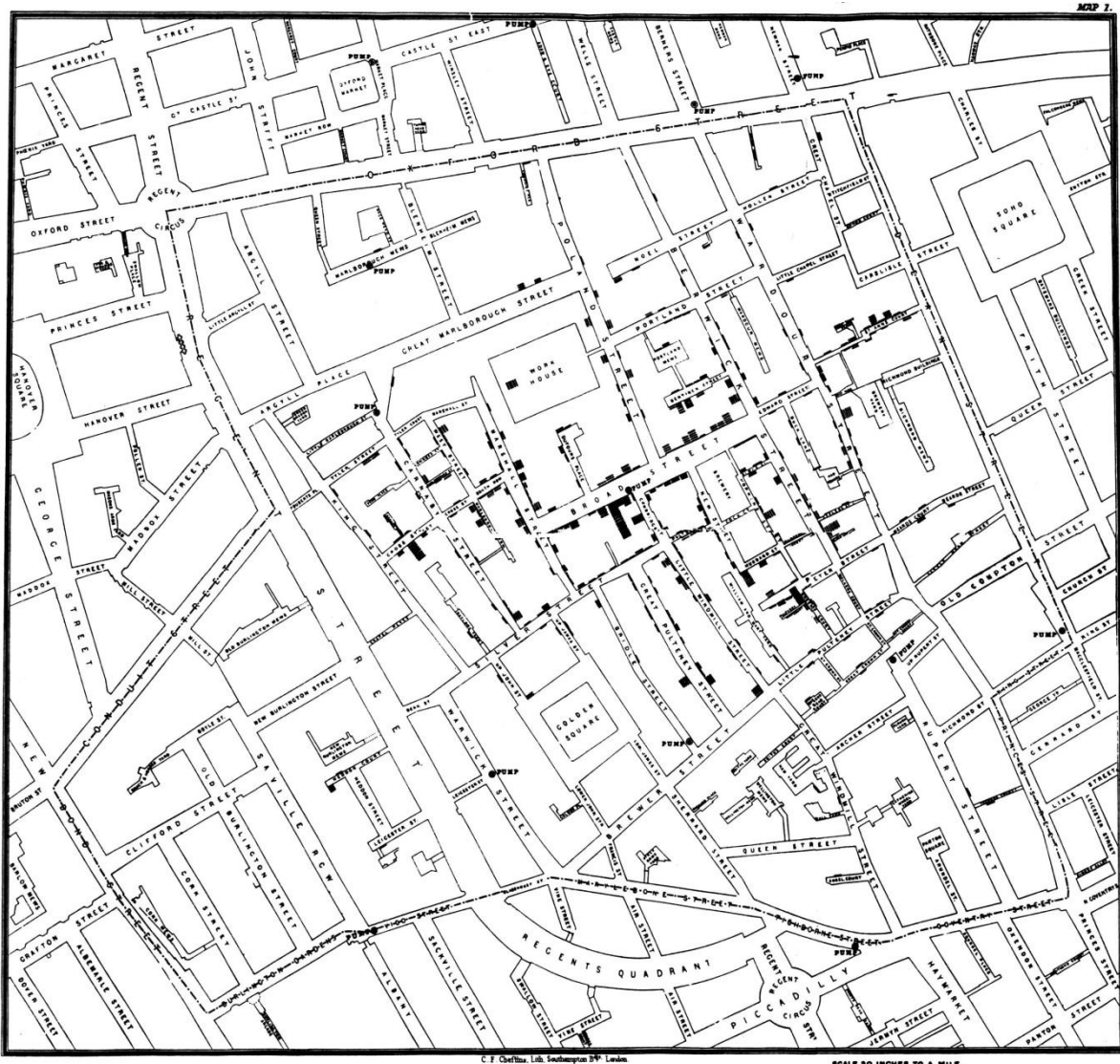


Figure 2-7 John Snow's 1854 thermic map of the London Broad Street Cholera outbreak (Snow, 1855)

John Snow's map served to challenge the majority view, that the primary means by which disease was spread was through the air. Looking back to the mid-19th century it is important to remember the theory of 'germs' was in its infancy and that Louis Pasteur's work confirming germ theory would not be completed for another decade.

At the same time John Snow was composing his thermic map another historical figure from the medical world was applying information visualisation techniques to their own work. Florence Nightingale is famous for her work in founding the field of modern nursing but she was also a highly capable mathematician and statistician who used information visualisation in the form of Polar Area Diagrams (or 'rose charts') to illustrate her findings to the members of the British government (Nightingale, 1858; Nightingale, 1859).

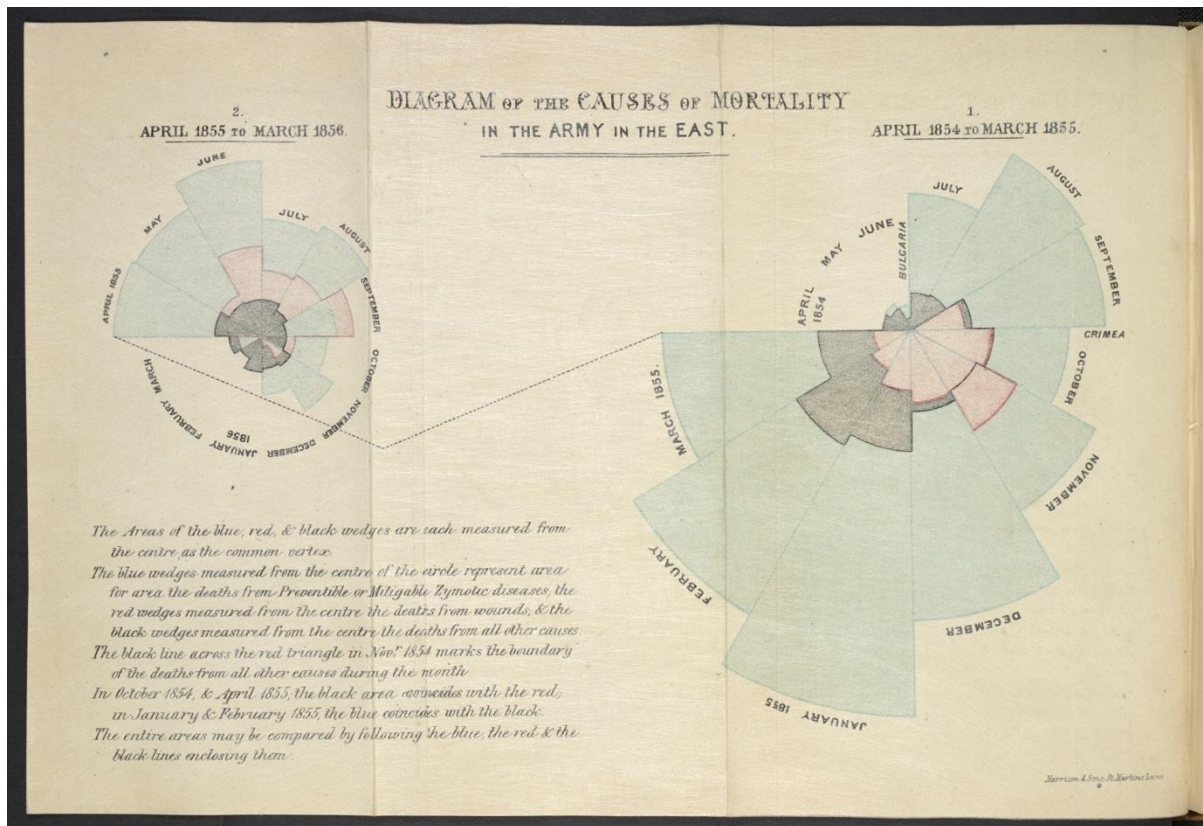


Figure 2-8: Florence Nightingale's Polar Area Diagrams or Rose Charts analysing causes of mortality among injured soldiers in the Crimea War of 1853 - 1856 (Nightingale, 1859)

Nightingale's demonstration of the impact of increasing sanitary conditions on mortality rates in British hospitals would serve as the foundation for the modern profession of nursing as well as the British government's overhaul of patient care at military hospitals.

The increasing use of Information Visualisation to express statistics in the latter half of the 19th century was not limited to Britain and her empire. Indeed perhaps the most famous thematic map of this period was published by the French civil engineer Charles Minard. Technically this map was a 'flow' map, a sub category of thematic maps defined as maps that "can be used to show movement of almost anything, including tangible things such as people, products, natural resources, weather, etc, as well as intangible things such as know-how, talent, credit of goodwill." (Harris, 2000). Minard produced and published 51 thematic maps over the course of his life (Robinson, 1967) but he is famous for his "Carte figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813" map showing the devastation of Napoleon's Grande Armée during his invasion of Russia in 1812 (Figure 2-9).

If Minard's thermic / flow maps showed that the accurate portrayal of geography (arguably the core of cartography) was not paramount, at least for the purpose of Information Visualisation, then it was Henry Beck's London Underground map that took discarding strict geographic data to the extreme. Made in the early 20th century and still in use today this map discards geographic relationships completely in favour of the information it is trying to communicate – namely the interconnections between tube stations. Figure 2-10 shows the original London tube map that strictly presented geographic information and the interconnections between them while Figure 2-11 shows Henry Beck's initial version.

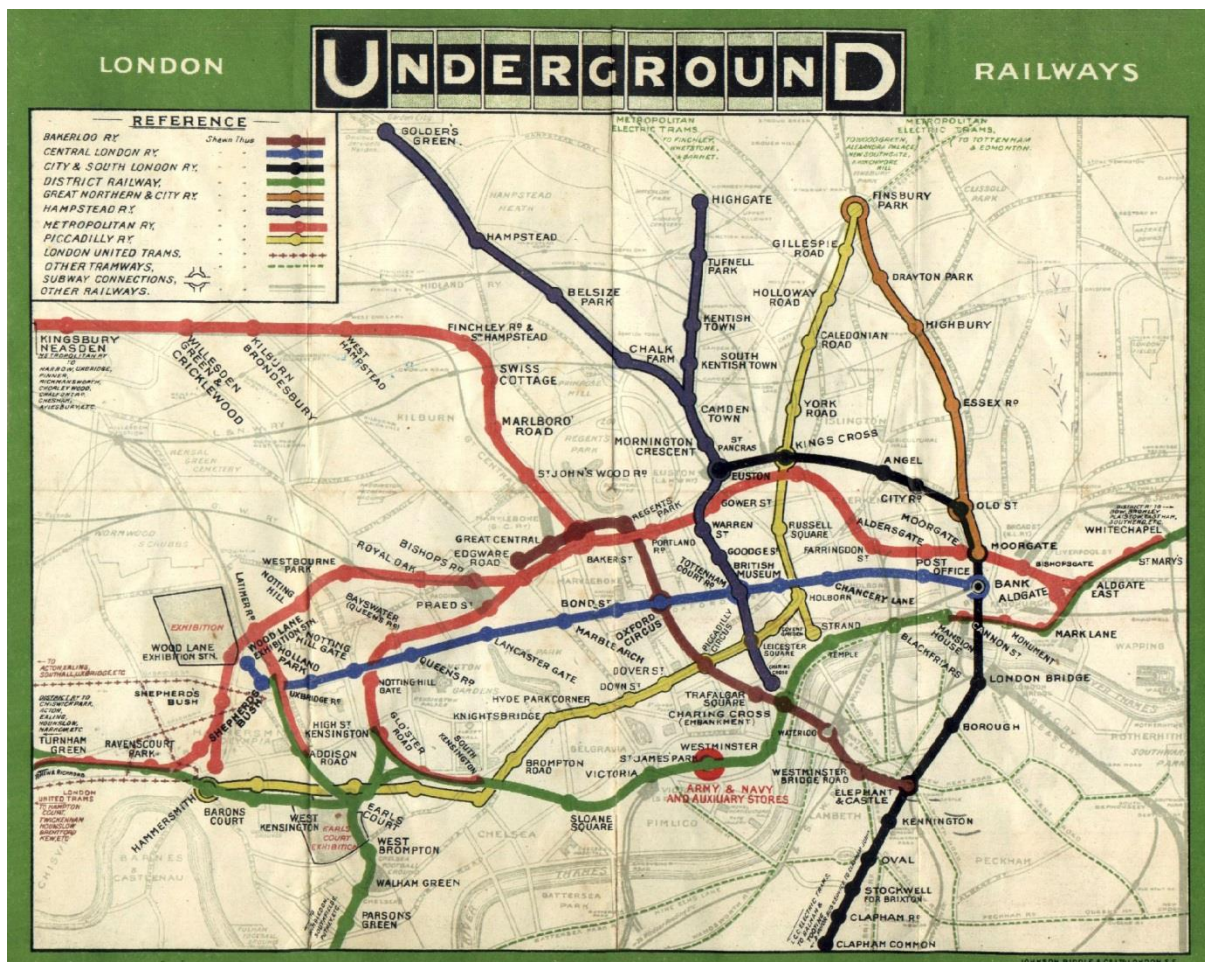


Figure 2-10: The original London Underground map incorporating both geographic and topological information (Creemers et al., 2014).



Figure 2-11: Henry Beck's topological map of the London Underground as it appeared when published in 1933 (Creemers *et al.*, 2014).

Beck's map places a far greater emphasis on the topological information about the London underground system than the original map which gives equal importance to the geographical information. Beck's map however was immediately more useful and relevant to the underground traveller. Users of the underground system lack the points of reference usually used to navigate by geographical features. In the original Figure 2-10 map its actual user was forced to discard half the information it presented as the geographical data was unusable for the task of navigating underground. The cost was an increased cognitive load on the user as they were forced to perform a "filtering" operation to extract the information that would permit them to navigate. It was Beck's recognition that the important data was the topology of the rail network, its stations and the connections between them, which made his map an immediate success. The cognitive load on his users was reduced as the data presented was already "filtered" into a usable form to complete the navigation task for which it was intended.

2.3 The Visualisation Process

Visualisation is the process of forming a mental picture of something that is invisible or abstract. It may itself be visualised as the set of processes and feedback loops shown in Figure 2-12.

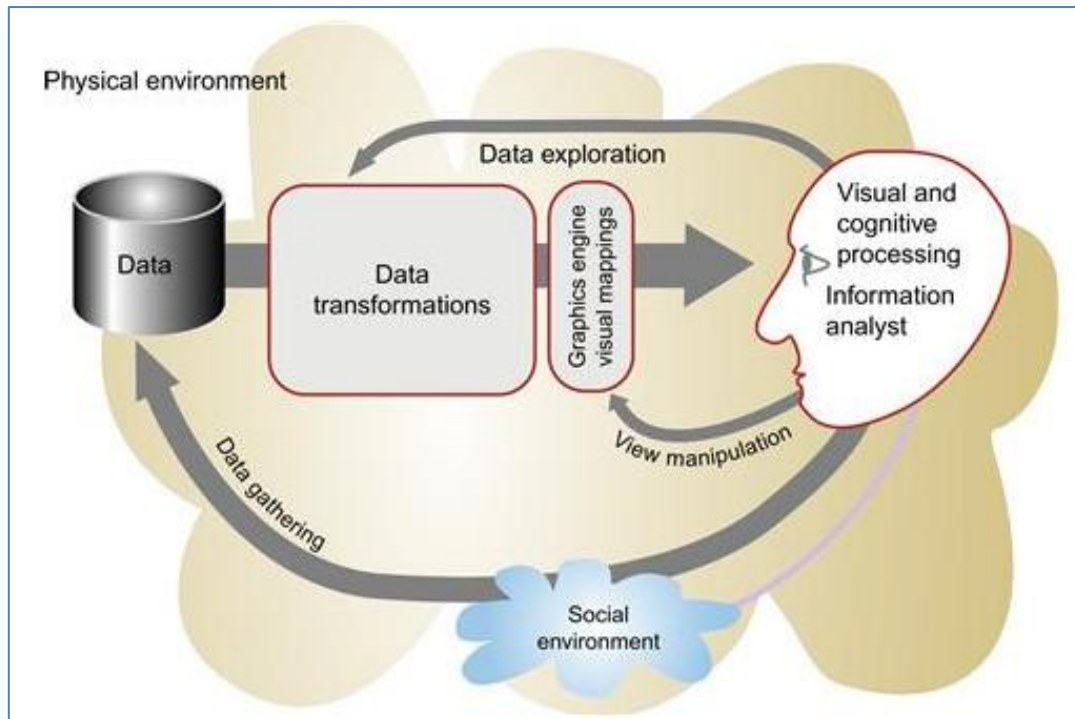


Figure 2-12: The visualisation process (Ware, 2012)

2.3.1 Data Gathering

The initial (and longest) loop is the process of gathering data, here the analyst may either be creating a new visualisation or researching new data to expand or enhance an existing one. The classic research process will be used with the analyst identify potentially relevant data to the topic under consideration, consolidating that data into a single data source and following up any interesting leads that can be readily identified. This stage is impacted by both the physical environment and the social environment. The physical environment is, of course, the ultimate source from which the data is derived and it may restrict the data available to the researcher and therefore the visualisation techniques that may be applied to extract meaning. Similarly the social environment may serve to influence both the ways in which data is collected and interpreted.

2.3.2 Data Transformations

Having prepared the data the analyst may decide that if certain transformations are performed on the data prior to visualisation it will be easier to extract meaning from the data. The exact transformations applied will depend ultimately on the field of study, the research question(s) being asked and the visualisation techniques that the analyst wishes to employ. Essentially this step can be seen as a data pre-processing stage that generates the final data to be presented in the visualisation. Its output provides the data that will undergo mapping to a visual representation and final rendering.

2.3.3 Graphics engine visual mappings

The graphics engine and visual mapping transform and present the pre-processed data to the analyst. The use of terms such as “graphics engine”, to the modern mind, immediately implies the use of a computer and this is usually the case in the modern world. It should be remembered however that this is not the only means of creating visualisations and many other techniques (such as model building, painting or sketching and sonification)

remain valid forms of visualisation beyond the 2D / 3D graphics abilities of the modern computer. The visual mapping phase selects appropriate symbols to represent data items; the study of symbols and how they convey meaning is known as semiotics. The field originated in the United States in 1868 with the work of logician and mathematician Charles Sanders Peirce (Brent, 1998; Peirce, 1868). Semiotics as a field is defined as “The study of signs and symbols and their use or interpretation” (*Oxford English Dictionary*, 2014) and is closely related to linguistics as the use of signs and symbols effectively forms a visual language. Attempts have been made to develop a general classification for all signs & symbols (Bertin, 1983) but no single system has gained wide acceptance with most seeming to be based on “arguments by example” rather than formal experiment (Ware, 2012). Despite this there is broad agreement that symbols can be divided into three areas:

- i. Semantic – The relation between signs and the things to which they refer (their meaning).
- ii. Syntactic - Relations among signs in formal structures (a set of symbols that may be constrained by rules that are specific to it).
- iii. Pragmatic - Relation between signs and the sign using agents (usually humans).

Reviewing the above areas it might be concluded that visualisation will be a tool only of use to those from the same or similar social / cultural background. Clearly all three elements seem to require some agreed social or cultural context for proper interpretation of the visualisation. To illustrate this point consider the semiotic breakdown shown in Table 2-1 where a complex UK road sign / symbol has been analysed by semiotic area.







Semiotic area	Sign / Symbol
Semantic – “Meaning”	 - Water / River / Sea  - Warning / Danger  - Wall / Barrier  - Car / Road user(s)
Syntactic – “Combination of symbols to give new meaning”	 - Quayside or river bank
Pragmatic – The using agents	 - Road user(s)

Table 2-1: An analysis of a UK road traffic sign by semiotic area.

It is certain that an understanding of the social / cultural background can greatly aid the effective interpretation of a visual representation. However there remains a difference between “greatly aid” and “is necessary / required”. In the first case we should proceed by establishing a set of conventions which are learned and adhered to whereas in the second it can be expected that different people from different social and cultural backgrounds will interpret an image in the same way. Fundamentally the question becomes whether or not symbols represents a **learned language** or a **universal language** that can express concepts across social and cultural divides. Research is divided on this point but the majority seem to favour the interpretation of symbols as a universal language. For example Deregowski has reported that both adults and children in a remote region of Zambia with little graphic art could still easily match photographs of toys to the actual toys (Deregowski, 1968). Similarly Hochberg and Brooks raised their own daughter to the age of two in a picture free house; they never read to her from picture books nor indicated that a picture was a representation of anything. Despite the lack of input telling her that pictures had any kind of meaning and any instruction on how to interpret pictures she could correctly identify objects in line drawings and black-white photographs demonstrating that interpretation of visual




images / symbols is not a learned skill (Hochberg & Brooks, 1962). Nevertheless counter arguments can be raised especially when non-pictorial images are used. In Table 2-3 we identify the red triangle as a warning / danger symbol. It is difficult to see how this designation can be anything but a learned social / cultural rule applied to the context of road signs. It bears little resemblance to any real world object. How then do we explain the difference between pictorial representations of a real world object as a universal language when other non-pictorial representations appear to be learned? Pearson et al., argue that “The most probable explanation is that, at some stage in visual processing, the pictorial outline of an object and the object itself excite similar neural processes”(Pearson, Hanna & Martinez, 1990). Ware argues that this view is plausible as “one of the most important products of early visual processing is the extraction of linear features in the visual array. These may be either the visual boundaries of objects or the lines in a line drawing” (Ware, 2012).

From the studies discussed above we may ultimately make a case that the interpretation of symbols used on visualisations ultimately depends on both the human visual system, the natural environment in which it has evolved and the social / cultural learning of the individual. This fact allows the division of symbols into the following categories:

Symbol Type	Description
Sensory	Refers to symbols / visualisations that express meaning by using the perceptual processing power of the brain without learning .
Arbitrary	Refers to symbols / visualisations that express meaning through learned social / cultural conventions (and therefore lack a perceptual basis).

Table 2-2: Colin Wares symbol classification scheme based on the symbols learned or un-learned meaning (Ware, 2012).

A graphical language can, therefore, be seen as a combination of symbols that fall into both the sensory and arbitrary categories. Whether or not any graphical language will represent a “universal language” or a “learned language” will depend on which category the majority of its symbols fall. There are probably very few graphical languages composed of entirely of only sensory or arbitrary symbols and therefore no completely universal language or completely learned language. In light of this we may re-examine our assessment that the language of road traffic signs is a learned one dependent on society and cultural training.

Sign / Symbol	Symbol Type	Comments
 - Water / River / Sea	Sensory	In our environment we encounter waves most frequently in liquids. The most common liquid we encounter is water. Our brains develop to pattern recognise waves and associate them with water.
 - Warning / Danger	Arbitrary	Both the use of a triangle and the adoption of the colour red are associated – in western culture – with warnings and danger.
 - Wall / Barrier	Sensory	Walls and barriers abound in the natural environment and our visual system adapts to recognise them. This symbol combines elements of a cliff face and a man-made wall.


Sign / Symbol	Symbol Type	Comments
 - Car / Road user(s)	Sensory	Our environment conditions our visual systems to recognise the pattern of a car / vehicle in much the same way as a dangerous predatory animal while associating this with a particular environment (the road).

Table 2-3: Quayside or river bank road traffic sign analysed using Ware's symbol classification scheme.

As can be seen in Table 2-3 when Ware's classification scheme is applied to the elements of the Quayside or river bank road traffic sign we can see that, of the signs four elements, three are categorised as sensory and one is arbitrary. From this we may categorise this symbol as a 'sensory' symbol that expresses meaning through the perceptual processing power of the brain without learning.

It would be inappropriate to infer that all road traffic signs attempt to fall into the sensory category and hence form a 'universal' graphical language from an examination of a single road sign. Fortunately the development of road traffic signs is a well-documented area and some research will quickly show that the designers employed seven guiding principles when designing traffic signs. These are (Shinar et al., 2003):

- i. Spatial Compatibility
- ii. Conceptual Compatibility
- iii. Physical Representation
- iv. Frequency
- v. Standardisation
- vi. Singular Functionality
- vii. Visibility

The most relevant of these are items (ii) and (iii). Conceptual Compatibility is defined as "a driver will know the meaning of a symbol without having to reflect and interpret its meaning". Clearly then the designers are targeting 'sensory' symbols where meaning arises from visual processing rather than the cognitive application of learned knowledge. Physical Representation is defined as "a driver will experience what is shown on the sign". This immediately requires the sign to reflect the physical world / reality that the user (driver) is or is about to encounter. Inevitably then the mental model being built in the drivers mind by the symbol should reflect the physical world. As we observed earlier the drivers most powerful perceptions of the physical world arise from the visual sense.

Given this it seems appropriate to classify the 'graphical language' of road signs as an attempt to create a universal graphical language. It also serves to inform us that if we wish our own visualisations to be universally understood by a wide audience we should endeavour to match data with a visual representation that falls into Ware's 'sensory' category.

Ware has examined the processes of the human visual system and has distilled from his studies eight guidelines for mapping data to a representation.

- i. Design graphic representations of data by taking into account human sensory capabilities in such a way that important data elements and data patterns can be quickly perceived.

- ii. Important data should be represented by graphical elements that are more visually distinct than those representing less important information.
- iii. Greater numerical qualities should be represented by more distinct graphical elements.
- iv. Graphical symbol systems should be standardised within and across applications.
- v. Where two or more tools can perform the same task, choose the one that allows for the most valuable work to be done per unit time.
- vi. Consider adopting novel design solutions only when the estimated payoff is substantially greater than the cost of learning to use them.
- vii. Unless the benefit of novelty outweighs the cost of inconsistency, adopt tools that are consistent with other commonly used tools.
- viii. Effort spent on developing tools should be in proportion to the profits they are expected to generate. (This means that small market custom solutions should be developed only for high value cognitive work).

The impact of Ware’s guidelines can be appreciated by contracting two visualisations of the same data. In Figure 2-13 the grades achieved by a group of university students has been visualised as a histogram that follows Ware’s guidelines. In Figure 2-14 the same grades are visualised as a data table that breaks several of Ware’s guidelines. In each figure whether guideline has been followed or broken is indicated (✓ or ✗).

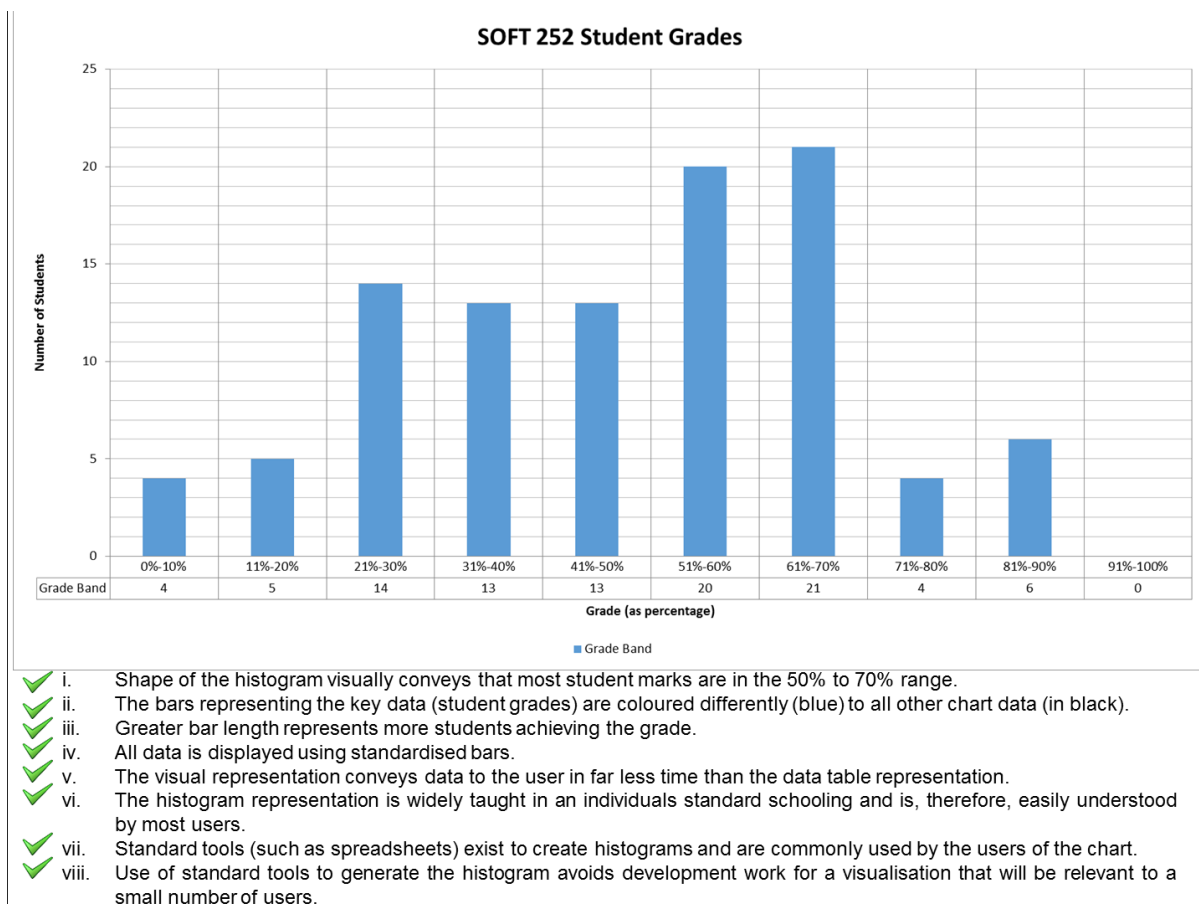


Figure 2-13: Histogram representing student grades that applies Ware’s guidelines.

- ✗i. Presented as a data table the user cannot immediately perceive that most grades are in the 50% to 70% bands. They must mental evaluate at least five values.
- ✗ii. The most important data is not visually distinct. (The user must mentally evaluate the five double digit grade band to find the best performing one.)
- ✗iii. The cognitive load for points i & ii is created because the greatest numerical quantity is not visually distinct.
- ✓iv. Standardised (numeric) symbols are used on the table.
- ✗v. Extracting key data from the table requires cognitive effort on the users part and therefore takes longer than the histogram visualisation.
- ✓vi. Almost all users will be familiar with tabular data.
- ✓vii. Standard tools (such as spreadsheets) exist to create tables and are commonly used by the users of the data.
- ✓viii. Use of standard tools to generate the table avoids development work for a visualisation that will be relevant to a small number of users.

Grade Band	Number of students
0%-10%	4
11%-20%	5
21%-30%	14
31%-40%	13
41%-50%	13
51%-60%	20
61%-70%	21
71%-80%	4
81%-90%	6
91%-100%	0

Figure 2-14: Data table representing student grades the breaks several of Ware's guidelines.

2.3.4 Visual and Cognitive Processing, Data Exploration and View Manipulation

The previous stages of the visualisation process have been concerned with the generation of a visualisation that effectively shifts the analyst's cognitive load onto the visual systems of the brain as they are more suited to perform pattern recognition tasks. In this final step of the visualisation process the Information Analyst must use the visualisation produced to attempt to answer the research question. When the visualisation is initially presented it is subject to visual and cognitive processing where the analyst attempts to identify interesting patterns within the data and attach meaning to them. A well designed visualisation will have shifted the majority of the pattern recognition tasks to the analyst's visual systems (which are optimised for this task). Determining potential causes for the identified patterns will be the responsibility of the analyst's cognitive system.

The process by which the analyst uses a visualisation to discover / reveal information about the research question was summed up by Ben Shneiderman when he developed the information seeking mantra "Overview first, zoom and filter, then details on demand" (Shneiderman, 1996). This mantra describes both how data should be presented and how the information analyst will interact with the visualisation. Assuming that it is an interactive visualisation where interaction is possible. In total Shneiderman identifies seven "Task-domain information actions" that visualisation users may perform. In summary these tasks are:

- | | | |
|------|-------------------|---|
| i. | Overview | Gain an overview of the entire data collection |
| ii. | Zoom | Zoom in on items of interest |
| iii. | Filter | Filter out uninteresting items |
| iv. | Details-on-Demand | Select an item or group and get details when needed |
| v. | Relate: | View the relationships between items |
| vi. | History | Keep a history of actions to support undo, replay and progressive refinement. |
| vii. | Extract | Allow extraction of sub-collections and of query parameters. |

The mantra also serves as a description of the final part of Ware's visualisation process (Figure 2-12) in which the analyst uses the visual and cognitive system to explore the data presented, identify patterns and features and then modify the presented visualisation to extract greater detail about interesting features.

2.3.4.1 Examining the Information Seeking Mantra

Each component of the Information Seeking Mantra serves to facilitate the analysts quest to answer their research question and is worthy of independent examination. An insightful analysis of the mantra is provided by Craft & Cairns who identified what can be learnt from the mantra (Craft & Cairns, 2005). Their findings are summarised by mantra section below:

2.3.4.2 Overview

The presentation of a complete overview of the dataset is of primary importance as it provides context for all the stages that follow. Many patterns and themes within a dataset can be seen only in the context of the entire dataset and therefore this forms the first step of the analysts 'visual thinking' in examining the patterns within a dataset. The primary goal is to identify features that are considered 'interesting' within the context of the research question being asked. The recognition of the 'interesting' patterns (mostly by the analysts visual cortex) and the selection of those relevant to the research question forms the basis for the next step of the visual information seeking process by selecting candidates to be retained or eliminated in the zoom & filter step.

2.3.4.3 Zoom and Filter

Both zooming and filtering serve the same purpose specifically "reducing the complexity of the data representation by removing extraneous information from view and allowing for further data organization" (Craft & Cairns, 2005). The difference between the two is subtle and is further complicated by the fact that zooming is itself usually divided into two possible actions – Zooming-in and Zooming-out. At the highest level the distinction between zooming and filtering can be stated as:

- i. Zooming – adjustment, by the user, of the size and position of data elements. Zooming may be regarded as "filtering by navigation and change of representational vantage point" (Craft & Cairns, 2005).
 - a. Zooming-in: removes extraneous information from the visual field. This in turn allows the cognitive centres to further organise the information into patterns to inform further interpretation and decision making.
 - b. Zooming-out: reveals hidden information – usually contextual information that is already known but cannot be recalled. Essentially the user is rediscovering

his location within the information space and integrating the detailed information revealed by the previous ‘zoom-in’ with their overall mental model.

- ii. Filtering – Reducing the complexity of the display by removing extraneous information without changing the data representation or the user’s view of it.

In either case both Shneiderman and Craft note that the visualisations responsiveness to the user’s interaction must be swift or its usefulness as an aid to cognition will be impaired.

2.3.4.4 Details on Demand

A typical information visualisation will contain many data points with the count ranging from tens to millions of points. Almost immediately it becomes impossible, given limited screen space, to display supplemental data on all these points. The mantra advocates a “details on demand” approach where supplemental data is provided on a data point by data point basis at the user’s request. The ‘request’ should be a simple action such as a mouse-over or selection of a data point that does not change the representational context in which the user is viewing the data.

2.3.4.5 Relate

An interactive visualisation should support its user in identifying and viewing the relationships that may exist between data points. Usually this is implemented as a change in viewpoint when the user makes a selection of a particular data item; the new viewpoint should present related items to the selected data point by degree of similarity. Of course what constitutes an appropriate measure of similarity will depend on the data being visualised and on any measure that might be calculated or made available to the visualisation engine during the data transformation step of the visualisation process.

2.3.4.6 History

Maintaining a record of the user’s actions as they explore the dataset and providing a facility to rapidly undo or redo actions is a key part of the user’s interaction with the visualisation. This accepts that the user is performing an ‘exploration’ of the dataset and that due to the exploratory nature it is possible for the user to need to restore the visualisation to a previous state. It may be that some user action that might have yielded a useful result does not, after all, achieve its goal. In this case a user will immediately wish to return to the previous state to continue the exploration of the dataset in a different direction. Equally the user may gain useful information from the action but still desire to return to the previous state as the contrast between the two states can itself provide further information. Finally of course the user may simply make an error and they should be able to rapidly recover from this.

2.3.4.7 Extract

An interactive visualisation frequently results in users performing a lengthy set of interactions to reveal the information needed to answer their research question. The information so revealed is often useful in many different tasks and it would be very labour intensive to re-create the interaction sequence every time the information was required for a task. Accordingly an interactive visualisation should provide the user with a means to extract and preserve the information exposed by their exploration of the dataset for use in other computer systems and projects.

2.4 Summary

Visualisation provides a tool that exploits the human sensory system – primarily the visual system – as a means to extract and reveal knowledge from a dataset while minimising the cognitive load on the researcher. The visualisation allows the researcher / observer to form a mental model of the revealed knowledge and how it fits into the area under study.

Chapter 3

Neuroscience

“neuroscience noun. Any or all of the sciences, such as neurochemistry and experimental psychology, which deal with the structure or function of the nervous system and brain”

Summary

In this chapter the field of neuroscience is defined, its history examined and the simultaneous recording of multi-dimensional spike train data is described.

3 Overview

Neuroscience is a multi-disciplinary field concerned with the scientific study of the nervous system. Its scope is broad ranging from the physical biology of neural networks, their cells and structure through development and functioning to their computational ability and the emergence of consciousness.

3.1 History of Neuroscience

The scientific study of the brain begins in Ancient Egypt and is perhaps surprisingly the subject of the oldest medical document now known. The Edwin Smith Surgical Papyrus written by the Egyptian physician Imhotep around 1700 BC contains the first recorded account of the brain's anatomy as well as documenting 48 cases of brain injury and recommended treatments (Feldman & Goodrich, 1999). Despite the scientific approach adopted in the papyrus the Egyptian's still believed that the seat of consciousness and reason was the heart and regarded the brain as a minor organ. This view was first challenged by Hippocrates who argued that the brain was the seat of reason. It was not until approximately 157AD that the Roman physician and philosopher Galen (a follower of Hippocrates) observed that gladiators in the arena lost their mental faculties when they suffered damage to the brain that the ancients began to accept the brain as the seat of consciousness and reason (Rocca, 2003).

Despite these vital insights study into the brain's functioning remained sparse and focused on documenting brain injury and its effects rather than its normal functioning. Detailed study of the brain and its operation waited on the development of technology and techniques that allowed researchers to examine its structure. The key technological breakthroughs that led to the first studies of brain structure were the development of the microscope and in 1873 the discovery of a 'staining technique' by Camillo Golgi that allowed the researcher to view the intricate structures of individual neurons. In 1887 this technique came to the attention of the Spanish neuroscientist Santiago Ramón y Cajal who further refined it and applied it to his studies of the central nervous system. Golgi and Cajal's work would establish the neuron as the 'functional unit' of the brain as well as amassing the scientific evidence for the 'neuron theory / doctrine' and mapping brain regions. Golgi and Cajal would share the 1906 Nobel Prize in Physiology or Medicine and their studies now underpin the modern field of neuroscience (Shepherd, 1991). Detailed study of the brain's structure and operation is therefore a relatively young research area which only developed in the late 19th and early 20th centuries. Despite this progress in the field has been considerable both because it is a young research area where much remains to be learned and because of the many practical applications in diverse fields such as biology, chemistry, computer science, mathematics, linguistics, engineering and medicine.

3.2 The Neuron as the core component of the nervous system

The brain constitutes a massively parallel computational system that, by a method not yet understood, gives rise to our consciousness as an emergent property of its operation. The brain exerts centralised control over the body with results similar to that of a Central Processing Unit (CPU) in a modern computer. While the results might be similar to a CPU the method by which control is achieved is radically different. Neuroscience has established the Neuronal Doctrine to explain how the biological neural network of the brain functions as an information processing system (Shepherd, 1991). The Neuronal Doctrine is an extension

of 'cell theory' based on Golgi and Cajal's work and has steadily evolved over the last few decades (and continues to evolve). The doctrine makes the following assertions (Finger, 2001):

- i. Neural units: The brain is composed of individual units – Neurons.
- ii. Neurons are cells: Neurons are biological cells with unique features (dendrites and axon's).
- iii. Specialisation: The neurons size, shape and structure vary according to its location or function.
- iv. Nucleus is the cells core: The centre of the cell is the nucleus; this contains the genetic material of the cell that is always replicated before cell division.
- v. Cell division: Nerve cells multiply through the process of cell division.
- vi. Axons are cell processes: Axons are outgrowths of nerve cells (whether myelinated or not).
- vii. Law of dynamic polarization: The axon can conduct in both directions BUT there is preferred direction of transmission from cell to cell. This preferred direction is created by a refractory period of 1-2ms in which the cell's Na⁺ channels that originally opened to depolarize the membrane remain open. During this period the cell cannot respond to any stimulus.
- viii. Synapse: A barrier to transmission exists at the site of contact between two neurons but it may permit transmission
- ix. Unity of transmission: The contact between any two cells may be excitatory or inhibitory but will always be of the same type.
- x. Dale's law: Each nerve terminal releases the same types of neural transmitter (Connors & Long, 2004; Dale, 1935).

To fully understand the neuronal doctrine it is necessary to examine what is known about the structure and operation of neurons in the brain. Figure 3-1 details the structure of a typical neuron:

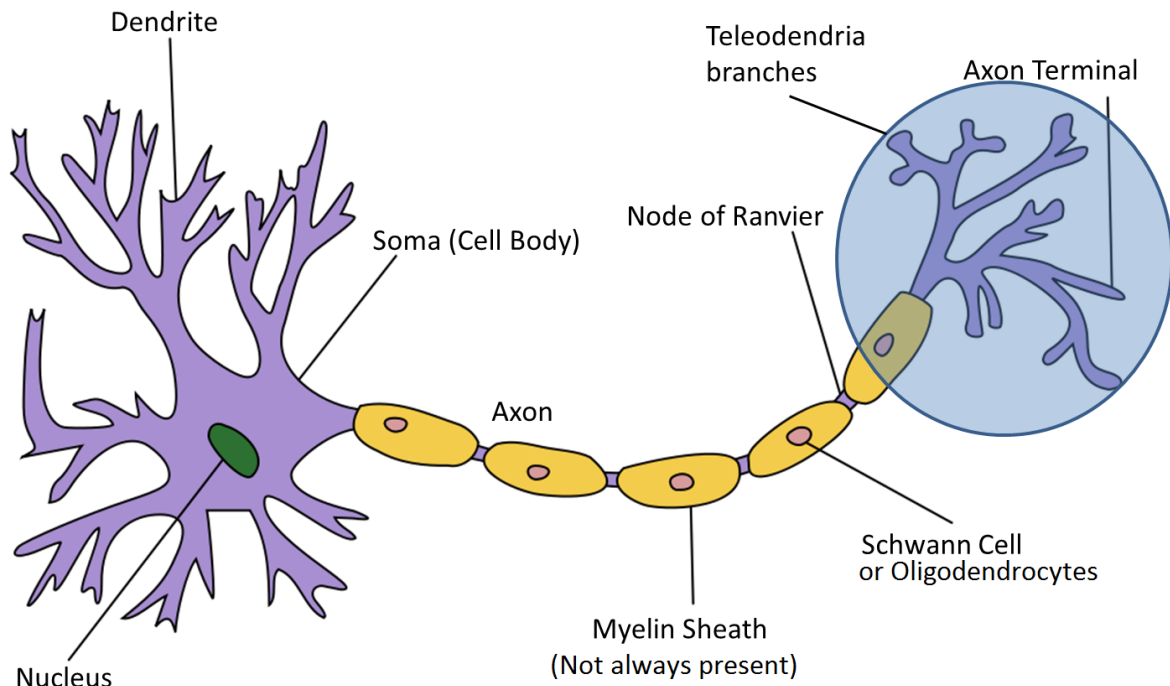


Figure 3-1: Major structures of a neuron (amended). (Jarosz, 2009)

As the neural doctrine asserts that the neuron is a biological cell the key features of any body cell are present in the form of a nucleus and soma (cell body). The specialised structures of the rest of the cell provide the biological system to create and receive the electrochemical signals used to encode and exchange data between the neurons in a neural network.

3.2.1 Dendrites

The dendrites are branched projections that receive electrochemical stimulation from connected neurons. Their primary task is to transfer the received signal(s) to the soma where if the combined signal from all connected dendrites is 'strong enough' the neuron will 'fire', that is generate its own electrical signal (action potential) to be propagated through the axon to other connected neurons within its neural network. It is believed that the dendrite is considerably more than a simple transmission system, the total surface area of the dendrite places a limit on the amount of information a neuron may gather, chemicals within the dendrite may serve to enhance or suppress the strength of the received electrical signal. Over time a neuron may even vary these factors; dendritic spines are protrusions on each branch that may grow to increase the surface area allowing additional connections to a neuron to be formed, and chemical changes within the cell may moderate the number and strength of electrical signals received (Roo et al., 2008). The dendrite / dendritic spine forms the second part of a synapse (see below) and this ability to vary the strength and number of connections over time gives rise to synaptic plasticity in which the strength of a connection varies over time depending on activity level. This synaptic plasticity is believed to form the biological basis by which the neural network achieves learning.

3.2.2 Axon

The neurons axon performs the task of delivering an electrochemical stimulation or action potential to the dendrites of connected neurons. Unlike the dendrite branches, which can be numerous, neurons have only a single axon however at the end of the axon it divides

into smaller branches called telodendria. Hence even though a neuron has a single axon it may connect too many dendrites and many other neurons within a neural network. The axon connects to the soma at the axon hillock. When a dendrite experiences an electrochemical stimulation at a synapse it transmits a signal to the neurons soma. The signals carried by dendrites are passive and they decrease with distance (much like signals in an electric cable). It is at the axon hillock that the signals received from all the neurons dendrites are summed over time. Should the combined sum of all signals from all dendrites exceed a threshold value that varies from neuron to neuron an action potential will be generated. For many years it was believed that creation of the neurons action potential occurred in the axon hillock but it has recently been shown that the initiation of action potentials usually begins in the adjacent (unmyelinated) segment of the axon proper (Clark, Goldberg & Rudy, 2009). The axon serves as the means to propagate the action potential to the next synapse. An axon may be myelinated or unmyelinated. An unmyelinated axon can be likened to a simple electrical wire with the action potential being transmitted through continuous conduction. In the case of myelinated axons the actual process of propagation is called 'saltatory conduction' meaning to 'hop or leap'. The process of saltatory conduction is achieved by the electrochemical interactions of the axons myelin sheath and the unsheathed segments called the nodes of Ranvier after their discoverer Louis-Antone Ranvier. The cytoplasm of the axon is electrically conductive while the myelin sheath inhibits charge leakage. Depolarization at one node of Ranvier elevates the voltage at the next node of Ranvier. This causes the action potential to be regenerated at the next node of Ranvier. The result is an electrical signal that appears to hop or leap from one node of Ranvier to the next without diminishing in strength as it travels. This results in a significantly faster transmission of the nerve impulse down the length of the axon (Huxley & Stämpfli, 1949; Tasaki, 1939). The speed of propagation will be dependent on the diameter of the axon.

3.2.3 Synapse

The branches of an axon's telodendria connect to the dendrites / dendritic spines of another neuron via a structure known as a synapse. The synapse transmits, usually via an electrochemical reaction, the action potential from the pre-synaptic neuron to the post-synaptic neurons dendrite. Figure 3-2 details the structure of a typical chemical synapse:

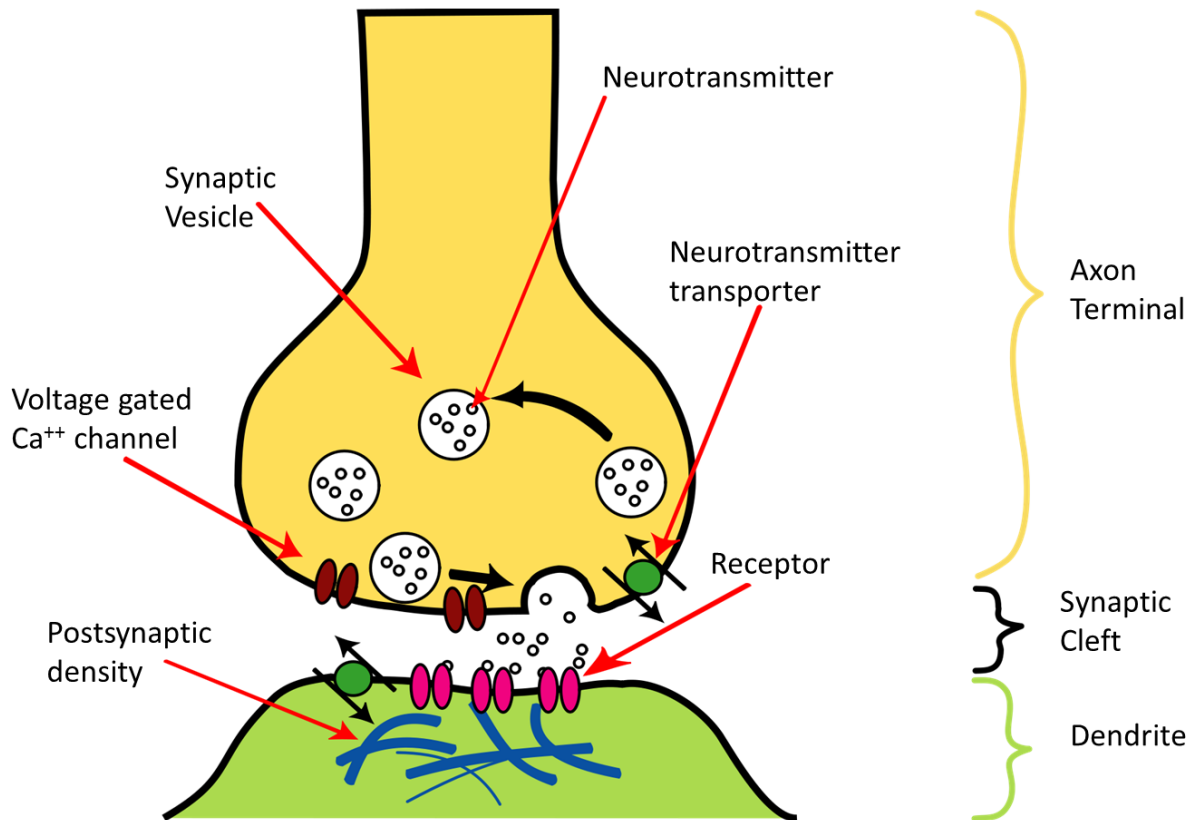


Figure 3-2: The structure and operation of a chemical synapse. (Wikipedia, 2014)

The primary features of a synapse are, of course, the axon of the transmitting neuron and the dendrite of the post-synaptic neuron. These two primary features are not, however, directly connected a small space remains between the membranes of the axon terminal and the dendrite. This space is known as the synaptic cleft and is on average 0.2 micron wide. Information is carried across the gap using a chemical called a neurotransmitter. The sequence of events to transmit the information from the axon terminal to the dendrite of the connecting neuron is summarised as ('synapse,' 2014):

- i. The arrival of an action potential at the axon terminal forces the movement of the synaptic vesicle(s) towards the axon terminal's membrane
- ii. A synaptic vesicle binds and fuses with the membrane of the axon terminal and releases a neurotransmitter into the synaptic cleft.
- iii. The neurotransmitter diffuses across the synaptic cleft and binds to receptor molecules on the postsynaptic membrane.
- iv. The binding action opens channel shaped protein molecules and electrically charged ions flow into or out of the neuron.
- v. The abrupt shift in electrical charge across the postsynaptic membrane changes the electrical polarisation of the membrane creating a postsynaptic potential (PSP).
- vi. If the inflow of positively charged ions is large enough then the PSP is excitatory and can lead to the generation of a new action potential in the post-synaptic neuron. Alternatively the response may be inhibitory, suppressing action potential generation.
- vii. After binding to a receptor the neurotransmitter is hydrolysed (broken down) by enzymes in the synaptic cleft.

- viii. The neurotransmitter is then re-absorbed by the presynaptic membrane of the axon terminal.

3.3 The action potential or ‘spike’ and the generation of spike trains

Having described the signalling mechanism between neurons from the perspective of its biological implementation it is now necessary to consider the signals exchanged by neurons. In one sense the signal itself is extremely simple; the neuron’s action potential is a short voltage pulse lasting only 1-2ms which is often termed a ‘spike’. A neuron that generates this electrical pulse is usually said to be ‘firing’. Plotting this spiking activity over time generates a ‘spike train’ such as the one seen in Figure 3-3. The similarity between this spike train and a binary sequence of 0’s and 1’s such as those encoding data in a modern computer is strongly evident.

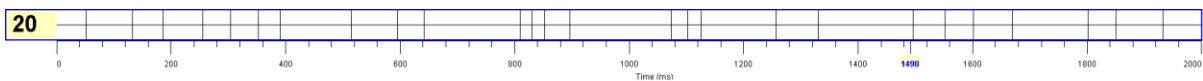


Figure 3-3: A neural spike train showing spiking activity over a 2 sec time period.

Binary data in a modern computer encodes data and information and it is believed that the firing rate and pattern of the neurons serves the same function in the ‘biological computer’ that is the brain (Adrian & Zotterman, 1926). Initially it was believed that the neuron firing rate was the primary carrier of data and information, particularly after Adrian & Zotterman demonstrated that stimulation of the skin and body hair produced corresponding increases in neuron firing rates (Adrian & Zotterman, 1926; Zotterman, 1939). Research has, however, proposed many different methods by which a spike train can carry information. These are summarised in Figure 3-3:

Coding Scheme			
Rate Coding	Temporal Coding	Population Coding	Sparse Coding
Spike-count rate	Phase-of-firing code	Correlation coding	Linear Generative Model
Time-dependent firing rate		Independent spike coding	
		Position coding	

Table 3-1: Coding schemes for information in neural networks.

A detailed discussion of neural coding schemes is beyond the scope of this project however an understanding of the broad categories is necessary for a full appreciation of the visualisations that will be described later. Accordingly a summary of the principles underlying each category of information encoding scheme is now provided. All of these data encoding schemes have been found to be in use by the brain.

3.3.1 Overview of neuron encoding schemes

Individual spiking events, when recorded intracellularly, are always all-or-nothing events with identical characteristics. When recorded extracellularly individual spiking events can appear to show variations in duration, amplitude and shape but this is simply a recording artefact rather than a real variation. These variations are introduced by such factors as distance between neuron and recording electrode or poor electrical conductivity. Individual spiking events are of such short duration, usually 1ms, that they are always treated as an all-

or-nothing point event in time (Dayan, 2005; Dayan & Abbott, 2005; Gerstner & Kistler, 2002). Analysis of spike trains and neural firing patterns is a highly mathematical field applying statistical methods, probability theory and stochastic point processes. This work has shown that some of these schemes are definitely in use, such as the time-dependent firing rate used by motor neurons to determine the strength at which a muscle is flexed, but this does not preclude other methods also being used to transmit different types of information (Gerstner et al., 1997).

3.3.2 Rate encoding schemes

Rate encoding schemes can be seen as the classic encoding scheme for information in the neural network as laid out by Adrian and Zotterman in 1926. Such schemes encode information in the 'firing rate' of a neuron's spike train. This scheme is definitely in use by motor neurons to control muscle flexing with higher firing rates triggering a larger response from the muscle.

3.3.2.1 Temporal encoding schemes

In temporal encoding schemes information is encoded through the precise timing of spikes on the millisecond level. The difference between temporal and rate encoding is best illustrated by example. Drawing an analogy with the modern computer it is possible to encode the decimal numbers 455 and 819 using their binary representations of 0111000111 and 1100110011 respectively. Assume that each binary '1' digit is represented by a neuron spike event and each binary digit ('1' or '0') represents a 1ms time period. Given this arrangement it is possible to compute a 'firing rate' for a neuron encoding these values. In both cases the firing rate is 6 spikes / 10ms. Hence representing these values purely in terms of firing rate is not possible as both have the same firing rate. The information is conveyed by the exact timing of individual spikes within the 10ms period (Theunissen & Miller, 1995). It has been shown by Theunissen and Miller that organisms perform sound localisation tasks within milliseconds but that sufficient information could not be gathered and transmitted purely using the firing rate models at this timescale. Hence modern neural science believes that temporal encoding must at least supplement rate encoding. Studies of this type of encoding usually focus on measurements of, and variations in, the inter-spike interval(s) of a spike train.

3.3.2.2 Population encoding schemes

Population encoding schemes argue that information in the neural network is not encoded exclusively by the activity of a single neuron. Rather the data input / stimulation of the neural network is represented by the joint activities of many neurons. This method of neural encoding is employed throughout the neural cortex and many examples could be given. As an example consider the medial temporal (MT) lobe of mammalian brains. The MT lobe is primarily responsible for long term memory of both facts and events (Smith & Kosslyn, 2006). The MT lobe is closely associated with processing both auditory and complex visual stimuli. It has been experimentally demonstrated that the MT lobe, while analysing motion in a visual scene, extracts the direction of motion using a collection of neurons that each encode a preferred direction (Georgopoulos, Schwartz & Ketiner, 1986). Motion in a neurons preferred direction results in an elevated firing rate giving rise to a mathematical vector. The final direction of motion is encoded as the vector sum of all MT neurons analysing the scene (Maunsell & Van Essen, 1983).

3.3.2.3 Sparse encoding schemes

A sparse encoding scheme attempts, given a large set of possible inputs, to find a small number of representative patterns which when combined reproduce the original input. A relatively simple example might be the encoding for an English sentence. There are a very large number of possible sentences but only a small set of symbols needed to represent them these being the letters, numbers, punctuation and spaces. In the brain this type of encoding is characterised by the activation of only a small subset of all available neurons. It has been shown experimentally that this form of encoding is utilised in the primary visual cortex of the mammalian brain (Olshausen & Field, 1996).

3.4 Connectivity between neurons in a biological neural network

From the overview of data encoding schemes it can be seen that neurons work together to encode data. One of the defining properties of the biological neural network is its ability to reconfigure itself through synaptic plasticity in response to its environment. It can, therefore, be concluded that the interconnections that can be formed between individual neurons is key to implementing an effective data encoding scheme. Any individual connection between two neurons may be excitatory or inhibitory. That is the action potential or spiking event delivered by the connected neuron may either increase or decrease the likelihood that its partner will generate its own action potential, or spiking event. With this in mind there are three basic connections that can be formed between two neurons as detailed in Figure 3-4. Each of these will now be examined.

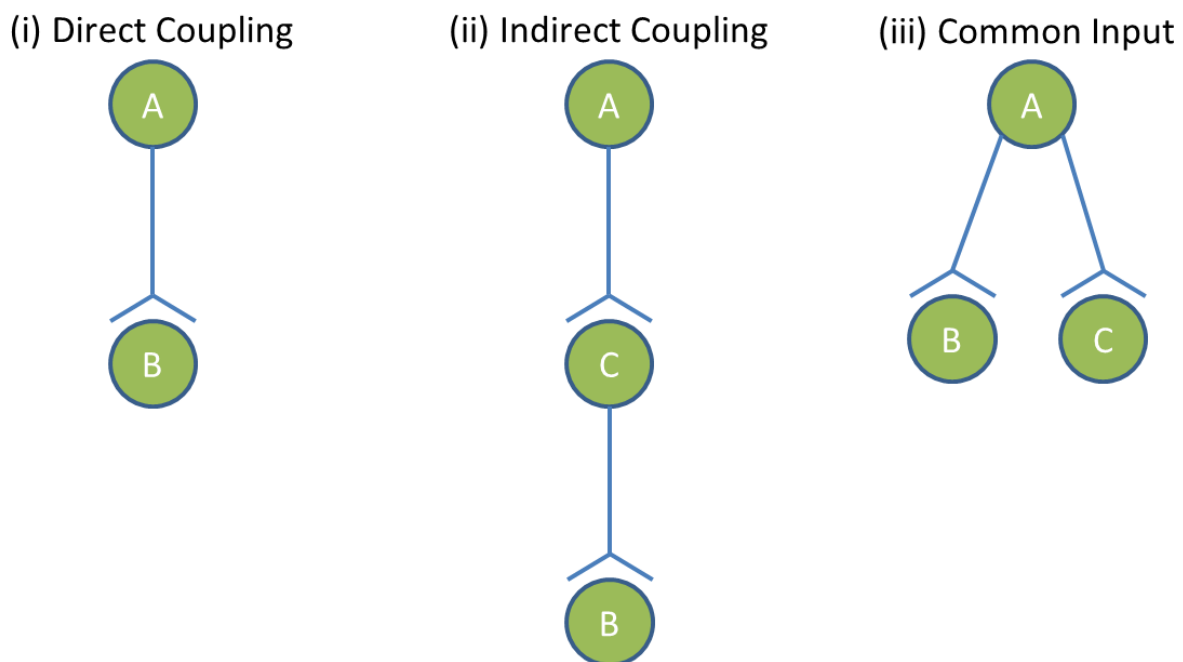


Figure 3-4: Coupling option between connected neurons in a neural network. (Somerville, 2011)

3.4.1 Direct Coupling

Figure 3-4 (i) shows the simplest connection between neurons. Termed direct connection the axon of neuron A links, via a synapse, directly to neuron B. Spike events generated by neuron A will be delivered directly to neuron B and may serve either to increase the likelihood of neuron B firing or suppress its firing activity. In this case an analysis of the spike trains for neuron A and B will show a strong correlation between their

firing patterns with neuron B's activity (or lack of activity) closely following Neuron A's spiking activity.

3.4.2 Indirect Coupling

Figure 3-4 (ii) Neuron A and B are still connected but not directly. An intermediary neuron (C) mediates signals passing between the pair. In this instance two connections are involved in connecting neuron A and B each of which may be independently excitatory or inhibitory. All three spike trains will show correlation with each other in their spiking pattern however the strength of the correlation will vary with A->C and C->B exhibiting a stronger correlation than A->B.

3.4.3 Common Input Coupling

Figure 3-4 (iii) shows the case of common input coupling. Here neuron A's axon terminals have established a synaptic connection to two neurons (B and C). Each connection is, of course, an instance of direct coupling and therefore both neuron's B and C will exhibit a correlation between their spiking events and neuron A's spike train. However this relationship implies that a (weaker) correlation will also exist when examining the correlation between B->C as both share a common source of spike events from neuron A.

Neural networks themselves are composed of many instances of the couplings shown in Figure 3-4 with individual neurons exhibiting complex correlations between their spiking patterns with a wide degree of variability in the strength of these correlations. The next section examines both how spike train recordings are made and the strength of the correlation between spiking events is measured.

3.5 Recording Neural Network Activity

Before any determination of neural connectivity or neural data encoding scheme can be attempted it is first necessary to record the activity of a neural network. This section examines the physical challenges associated with the recording of neural network activity.

The typical human brain cerebral cortex contains some 200 billion individual neurons that have established 125 trillion synapses (Micheva KD et al., 2010). For all practical purposes it is impossible to record the activity of the entire cerebral cortex given the limits of current technology. Despite this limitation modern technology does allow for the examination of the activity in the neural network on a smaller scale. If useful information is to be derived it is important to record the activity of as many neurons as possible at the same time. This has given rise to neural network recordings known as "Multiple simultaneously recorded neural spike trains".

3.5.1 The Multi Electrode Array (MEA's)

The spiking activity of a neural network can be detected using electrodes inserted into the tissue of the neural network. Electrodes are usually arranged into an "array" that records activity in a particular area of the network (see Figure 3-5 and Figure 3-6). The electrodes when placed close to a neuron's soma or axon transduce the electrical charge of an action potential (spike event) recording the firing of the neuron. Multi electrode arrays generally come in two classes.

- i. The *in vitro* (Latin: within glass) array and
- ii. The *in vivo* (Latin: within the living) array.

The *in vitro* array (Figure 3-5) is used to study neural network samples that have been removed from their usual biological setting such as brain slices. The primary benefit of *in vitro* work is that the system under study is greatly simplified being detached from the immense complexity of a living organism. This allows for detailed study of a small number of components (Vignais & Vignais, 2010). The primary drawback to such studies is that it is difficult to extrapolate its results back to the intact organism (Rothman, 2002).

The *in vivo* array (Figure 3-6) is implanted into a living organism and permits study of the activity within the neural network in its natural setting.

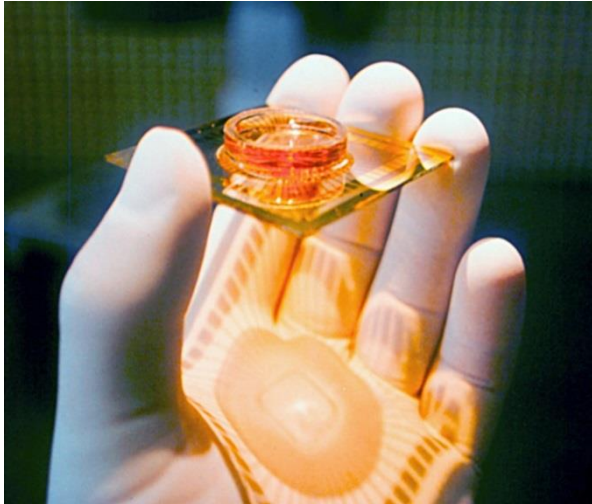


Figure 3-5: An *in vitro* MEA array. (Potter, 2010)

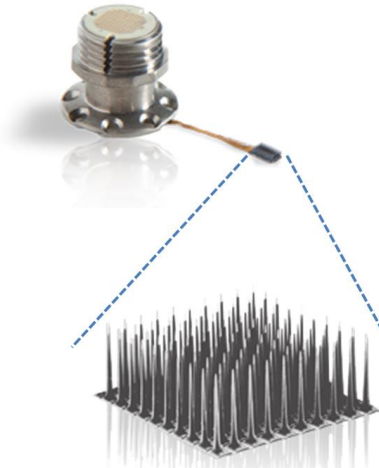


Figure 3-6: An *in vivo* MEA array. (Normann, 1993)

The simultaneous recording of multiple neurons is not, however, as simple as inserting an array into a tissue sample or live animal. Each electrode has the potential, on average, to record spiking events from over a 1000 neurons situated within 140 μm of the electrode (Buzsaki, 2004). A recording of the electrical activity from the electrode will, therefore, represent the sum of all spiking events from an unknown number of neurons within $\sim 140 \mu\text{m}$ of the electrode. To be useful the recorded signal must be analysed, individual neurons identified and the recorded spiking events assigned to neurons. This task is complicated by the fact that all spiking events appear in the recording to have the same characteristics. The process is termed spike sorting with each spiking event be identified and then assigned to a neuron that produced it. The use of an array of electrodes or sometimes the division of the electrode tip into multiple tips (a tetrode) allows this to occur through a process of triangulation with different electrodes (or tips) receiving the same signal at different strengths. The varying signal strength provides the basis for measuring the distance to the pre-synaptic neuron while the physical location of the receiving electrode in the array will provide the basis for triangulation. Figure 3-7 illustrates the principle. Despite the application of these techniques there remains no universally

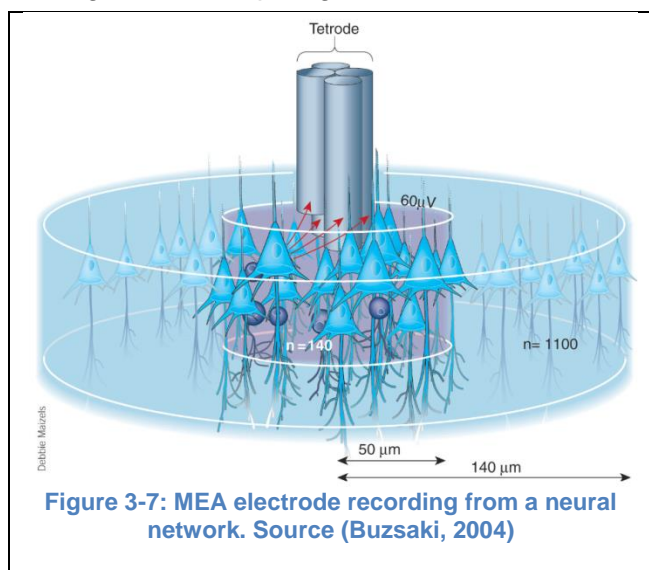


Figure 3-7: MEA electrode recording from a neural network. Source (Buzsaki, 2004)

agreed algorithm for spike sorting with different algorithms' pre-synaptic different results from the same input data (Brown, Kass & Mitra, 2004). Development of an effective spike sorting techniques remains a challenge for neuroscience that places considerable constraints on recording multiple simultaneously generated spike trains. The practical impact of this is that the typical number of reliably recorded spike trains per an electrode of the recording array is markedly below the theoretical limit of ~1000 neurons. Nevertheless work continues in this field and the last few years have seen a marked growth in the number of simultaneously recordable spike trains. It is a reasonable expectation that this trend will continue in the near future as the number and sensitivity of electrodes in recording arrays increases and superior spike sorting algorithms are developed.

3.5.2 The Data Explosion in Neural Science

Given the historical limitations on spike sorting algorithms' discussed above the number of simultaneously recorded neural spike trains has been small, typically in the hundreds of neurons range. However as technology has advanced more modern recording equipment is now able to identify thousands of neurons in a typical recording session (Taketani & Baudry, 2006). While even this is significantly below the theoretical maximum it still represents a flood of data that requires significant processing power to analyse. While the computer provides the power to record this mass of data the development of software to analyse it has not kept pace with the ability to record the data. This is a trend that has been seen in many areas of science due to the fast pace of technological change in the information technology field (Ward & Barker, 2013). In addition to this Brown et al, believe that "Multiple spike trains are multivariate point processes, yet research in statistics and signal processing on multivariate point process models has not been nearly as extensive as research on models of multivariate continuous-valued processes" (Brown, Kass & Mitra, 2004). Brown also observes that such analysis techniques as are available tend to restrict themselves to analysing neuron pairs rather than considering the wider connection network. As a further analysis failing Brown identifies that neural plasticity "makes non-stationarity in neural data a rule rather than an exception". Despite this there is a lack of "explicit adaptive estimation algorithms to track these dynamics for multivariate point processes". This lack of proven analysis methods serves to hold back progress despite the wealth of data now being recorded.

3.6 Analysing Neural Network Recordings

The analysis of multiple simultaneously recorded spike trains has as its goal the identification of the recorded networks functional connectivity between neurons and the mapping of the network. This information will then allow researchers to formulate and refine computational models describing the operation of the recorded network.

This project presents three visualisations aimed at allowing the researcher to explore a set of simultaneously recorded spike trains with the goal of identifying connectivity between neurons. These are:

1. The spike train raster chart (iRaster),
2. The pairwise cross correlation grid (iGrid) and
3. The MEA firing animation (iAnimate)

Each of these will be discussed in the relevant implementation chapter but in summary they are used as follows:

3.6.1 iRaster – The classic spike train raster plot.

The iRaster visualisation presents the raw data being analysed as a collection of spike trains plotted as a time series of discrete spiking events. Exploring the dataset in this view is primarily a re-ordering and filtering task. The interactive plot will provide a set of built in analysis algorithms to order / reorder spike trains (primarily sorting on inter-spike intervals and bursts of spiking activity). The VISA analysis pipeline will provide a means to introduce the researchers own custom developed ordering algorithms. Finally the raster plot will be able to group and filter individual spike trains. It provides a means to inspect visually the raw data and to identify visually the recurring patterns that indicate potential connectivity.

3.6.2 iGrid – The Cross Correlation grid

Based on the work of Stuart, Walter and Borisyuk the cross-correlation grid is a visualisation from which it is possible to determine the neural networks connectivity (Stuart, Walter & Borisyuk, 2005). This is a computationally intensive visualisation which while very effective does not scale well to large data sets. The computational load grows exponentially as the number of recorded neuron spike trains increases. The visual grid representation also becomes quickly un-usable as neuron counts rise. This project will use parallel computation to increase access to compute power and provide a pre-processing algorithm to handle the computational load. This algorithm will scale effectively from a researchers laptop to a high performance compute cluster (HPC) without code modifications. To prevent cognitive overload on the part of the user a clustering algorithm will identify connected neurons. The clusters will be used to create a dendrogram allowing user navigation of the data set by neuron cluster.

3.6.3 iAnimate – The Multi Electrode Array firing animation

The physical spacial relationship between firing neurons can also be used to visually identify clusters of neurons. By plotting firing of neurons over time on a 2D plane representing the recording multi electrode array clusters can be visually identified. This allows further analysis, such as the cross-correlation grid to target these neurons. Where position data is available in addition to the spike train data this animation can be used to identify potentially connected neurons.

3.6.4 Challenges of big data in neuroscience

As with many fields the information technology age has had considerable impact on the field of neuroscience. The 125 trillion synapses or connections between neurons in the brain, remains a problem that not even today's computers can completely model. Indeed at the moment it is not possible to simultaneously record the activity of the 200 million+ neurons. Nevertheless it is possible to record a subset of this and ask meaningful questions based on these recordings. Extracting answers from the mass of data generated by recording even a small sub-set of neuron activity requires researchers to confront the 'big data' problem. To be complete it is necessary to define what is meant by the term 'big data' and exactly what the problems are in its analysis and presentation.

Big data is a term which has been very poorly defined, usually by salesmen determined to push their product as a solution to extracting useful information. Usually this is a data mining product that attempts to identify sales opportunities from purchasing data or internet browsing histories. Ward and Barker however provide a more academically satisfying definition of the term. After reviewing its history and use they decided to define big data as:

“Big data is a term describing the storage and analysis of large and or complex data sets using a series of techniques including, but not limited to: NoSQL, MapReduce and machine learning” (Ward & Barker, 2013).

From this definition some key points regarding big data may be extracted:

1. The term big data may be applied to data sets which are:
 - a. Physically large in terms of data points or
 - b. large in terms of complexity, i.e. high data dimensionality even if small in terms of physical size or
 - c. Both of the above where the data set is physically large and complex.
2. Effective analysis of the data is a non-trivial task requiring considerable computing power. This may include the latest in machine learning and artificial intelligence algorithms.

The recording and analysis of multi-dimensional spike train data for neuroscience clearly falls into 1(c) and 2 above. Such data usually exhibits certain attributes that present challenges to the data analyst. These are usually summarised as (Laney, 2001):

- i. Volume – High volume data refers to the number of individual data points being collect. Neuroscience already provides far more potentially collectable data points than can be recorded. Advances in technology over time will serve to make a steadily growing number of data points recordable.
- ii. Velocity – High velocity data refers to the recording rate at which data points are created. As with volume the rate at which MEA’s record the spike train signal, the sampling rate, is growing with time. The simple storage or high velocity data can present a considerable challenge.
- iii. Variety – High variety data refers to the range of different sources that might generate spike trains. MEA data is only one method for observing neural network activity. Many other forms of recording exist such as Voltage Sensitive Dyes, Functional magnetic resonance imaging (fMRI) and Positron emission tomography (PET). Successful analysis will involve combining data from a variety of sources

This definition pre-dates the development of the concept of big data. Some organisations argue to add variability and complexity to the above (SAS, 2014). Given Ward and Barker more precise definition of big data this would seem sensible.

- iv. Variability – High variability refers to periodic peaks or bursts of activity which can place burdens both on recording and storing the data. Neural networks in particular generate bursts of high velocity data when subject to stimulation.
- v. Complexity – Neuroscience data is unavoidable complex with many different data encoding schemes and the need to represent all the experiences of a living organism. In addition neural networks analyse and dictate responses to stimulation as well as giving rise to consciousness in living creatures. That the data will be complex is an unavoidable conclusion even if science does not yet fully understand how all these processes are achieved.

Bringing to bear the computing resources need to store and analyse neuroscience data will present a considerable challenge in and of itself.

3.7 Summary

Neuroscience is a complex science combining elements of classical biology, chemistry and electronics to explore the operation of neural networks. As with many fields the application of technology to the collection and analysis of neural networks has revealed a wealth of new data. However converting the raw data into usable information challenges even the modern computer with a truly 'Big Data' problem. Additionally this is a relatively young science (100-150 years old), Golgi and Cajal were its founders and they shared the 1906 Nobel Prize. There remains much yet much to be learned about the operation of biological neural networks, ranging from data encoding schemes to the operation of neuron clusters' as data processing centres. Progress will require software tools and new mathematical algorithms that address the problems of 'Big Data' and the analysis of point processes. It will also require a far wider sharing of data recordings, analysis code and expertise (Gibson et al., 2008). Finally, to be usable, the information extracted must be presented in a way that avoids cognitive overload to the user.

Chapter 4

Parallelism

“parallel adj. Of or relating to the simultaneous performance of multiple operations: parallel processing”

Summary

In this chapter the term “parallel processing” is defined and its use as a means of delivering increased computing performance is examined.

4 Overview

The term parallelism has arisen in computing to describe the simultaneous performance of operation of multiple operations. It has emerged as the primary means of delivering increase computing power in the 21st century but its effective exploitation requires software developers to re-think software design. This has led to the proliferation of software written to historic design standards that fails to fully exploit the available power of the modern computer. It is argued that this waste of computing power must be avoided if the challenges of applying the computer to significant “Big Data” problems are to be met.

4.1 Historical development of parallel computation

Historically parallel computation was developed not to address the processing of large quantities of data but the considerable difference in operating speed between a computer's CPU and its attached peripherals. In the early age of computing ‘time on the CPU’ was an expensive commodity. Expensive hardware meant most businesses utilised only a single computer with departments submitting jobs to a central administrator. In this situation the primary measure of efficiency was the work done per CPU cycle. However a large number of CPU cycles were effectively wasted when interacting with attached peripherals. Printers, hard disks and data transmission over a network were all tasks that required CPU's to pause program execution and await completion of the operation. Some element of these time consuming operations was unavoidable, such as loading the program and persisting its results to disk. This markedly extended the time required to execute a program and lowered the efficiency of the computer.

4.1.1 Multiprogramming systems and the LEO III

The first attempt to address the CPU / peripheral time imbalance came in 1961 in the form of the LEO III (Lyons Electronic Office). This computer was the first multiprogramming system (Aris et al., 1997) which enabled a batch of programs to be loaded into the CPU simultaneously. This allowed multiple programs to effectively queue for access to CPU time. In this system the first program would execute until it reached an instruction requiring the use of a peripheral device. Rather than pausing CPU operations while the peripheral completed its task the ‘context’ of the executing program would be saved and work on the next queued program would commence. Once the peripheral completed operation(s) the ‘context’ of the currently executing program would be saved and the original restored. Execution could then proceed as if no interruption had occurred but no CPU cycles were wasted as another task had been making progress during peripheral operation. The primary limitation of this system became the availability of additional tasks to maintain CPU usage. In the modern computing age this would be termed a problem of granularity. Queuing entire programs for access to CPU cycles will not maximise CPU usage as the executing units are too large.

The issues of granularity and the size of the executing unit were not initially recognised while the batch processing of jobs remained common. However this changed as the use of computers moved from the batch processing environment to interactive use.

4.1.2 Co-operative Multitasking

The rapid expansion in the use of information technology over the 1970's and 1980's led to the computer becoming far more ubiquitous both in the business and home

environments. Production costs for computer hardware fell as printed circuit boards were replaced with the single chip microprocessor. In 1975 the first commercial successful microcomputer the Altair 8800 shipped with its creators expecting to ship a few hundred units, actual demand topped a thousand systems in the first month (Ceruzzi, 2003). This spread of cheap computing power spelled the end for the large corporate IT computing department. No longer would users submit jobs and await results from the single IT department controlled computer. The microprocessor placed computing power onto the desks of staff and researchers. It also exposed the flaws of the multiprogramming's queuing system. Users who would previously wait hours for a program to be executed and the results returned expected a far more interactive experience from the new microprocessor. For the first time effective human-computer interaction (HCI) became important with the need for interactive displays that showed levels of progress. Shneiderman codified the important "Golden Rules" of HCI as (Shneiderman & Plaisant, 1998):

- i. Strive for consistency.
- ii. Enable frequent users to use shortcuts.
- iii. Offer informative feedback.
- iv. Design dialog to yield closure.
- v. Offer simple error handling.
- vi. Permit easy reversal of actions.
- vii. Support internal locus of control.
- viii. Reduce short-term memory load.

Several of these rules do not operate well in the multiprogramming environment, particularly (iii) and (vi - viii). In these cases the 'all or nothing' approach of batch processing and the need to wait until a task is complete before updating the user limits compliance with these rules. For example a long running peripheral operation should provide feedback on progress but if the CPU is executing a different program the original cannot report progress.

The solution, adopted in early operating systems such as Windows (prior to Win 95) and the Mac (prior to Mac OS X), was co-operative multitasking (Apple, 2001; Microsoft, 1995). In co-operative multitasking programs are expected to regularly 'offer' time on the CPU to other programs that need to perform ongoing tasks. So long as software developers respected this requirement the impression of an interactive system could be maintained for the user. The obvious drawback was that a program could still 'hog' CPU time by refusing to offer time to other programs. As a result any number of well written applications could have their operation disrupted by a single poorly written application. Despite this in the early era of microcomputers co-operative multitasking provided a much improved alternative to multiprogramming systems.

4.1.3 Pre-emptive Multitasking

At the same time as co-operative multitasking was being adopted as a solution to the drawbacks of multiprogramming systems a rival strategy termed pre-emptive multitasking was emerging. In 1969 this approach was selected for the UNIX operating system and is now standard in UNIX and its derived operating systems (Aikat et al., 1995). In the pre-emptive multitasking paradigm the available CPU cycles are divided into 'time slices' with each executing process being allocated a number of time slices. This paradigm makes an implicit guarantee that each executing process will receive CPU time on a regular basis regardless of other activities. This approach provided pre-emptive multitasking systems with

a key advantage over co-operative multitasking – control of CPU time remained with the operating system. It was therefore impossible for a poorly coded or badly implemented application to ‘hog’ access to the CPU at the expense of other programs. Ultimately this advantage would persuade both Microsoft and Apple to abandon their original implementation of co-operative multitasking. By the mid 1990’s, Microsoft Windows had adopted the pre-emptive multitasking system incorporating it into both Windows NT and Windows 95. Apple Inc. followed suit with the MAC OS 9.x, released in October 1999.

4.2 Features of Multiprogramming / Multitasking processing

4.2.1 Categories of computer processes

In the broadest sense a process executing on a computer can be categorised into two categories that describe the limiting factor that controls its operation.

- i. An I/O Bound process. This category of process is the original reason for the development of multitasking / multiprogramming systems. Performance is limited by the peripheral devices and the speed at which data can be requested and delivered (Corporation, 2008)
- ii. A CPU Bound process. In these processes the total computation time to complete the process is limited by the operating speed of the CPU. Typically a computationally intensive process will have its performance bound by the number of CPU cycles available to it.

The primary reason for the emergence of the pre-emptive multitasking model is that it can efficiently allocate CPU time between both process types via the concept of “thread blocking”. Unlike the co-operative multitasking system which caters effectively only for I/O bound processes and rely’s on software developers to ensure that CPU cycles are offered for re-allocation to other (primarily CPU bound) processes. Each executing process contains at least one “thread of execution” representing the instruction currently being executed by that process. Time on the CPU is determined by allocating time slices to the various threads. Each time slice represents a number of CPU cycles that may be used to execute instructions. The pre-emptive model interrupts a threads execution at the end of its time slice and performs a context switch to allow execution of another thread. The sequence for the pre-emptive multitasking model can be seen in Figure 4-1. The sequence for a co-operating multitasking model is similar but the concept of time slices triggering context switches is removed. Instead a context switch occurs only when a process ‘offers’ to allow another thread to run.

4.2.2 Processes and threads of execution

In Figure 4-1 the term “process” could be considered synonymous with “thread” and in the earliest systems this was true. Originally each process could be considered to be a thread executed on a classic Von Neumann computer (Neumann, 1945). This of course means that each process is completely isolated from another with its own memory space and variables. The ‘context switch’ involved saving the executing processes memory space and loading the previously saved memory space for the next executing process. However it was rapidly realised that granularity would again become an issue. Individual processes could increase granularity by having several threads of execution. For example the word processor on which this document is written is currently:

- i. Accepting text input from a keyboard,
- ii. spell and grammar checking the document and
- iii. background saving the document

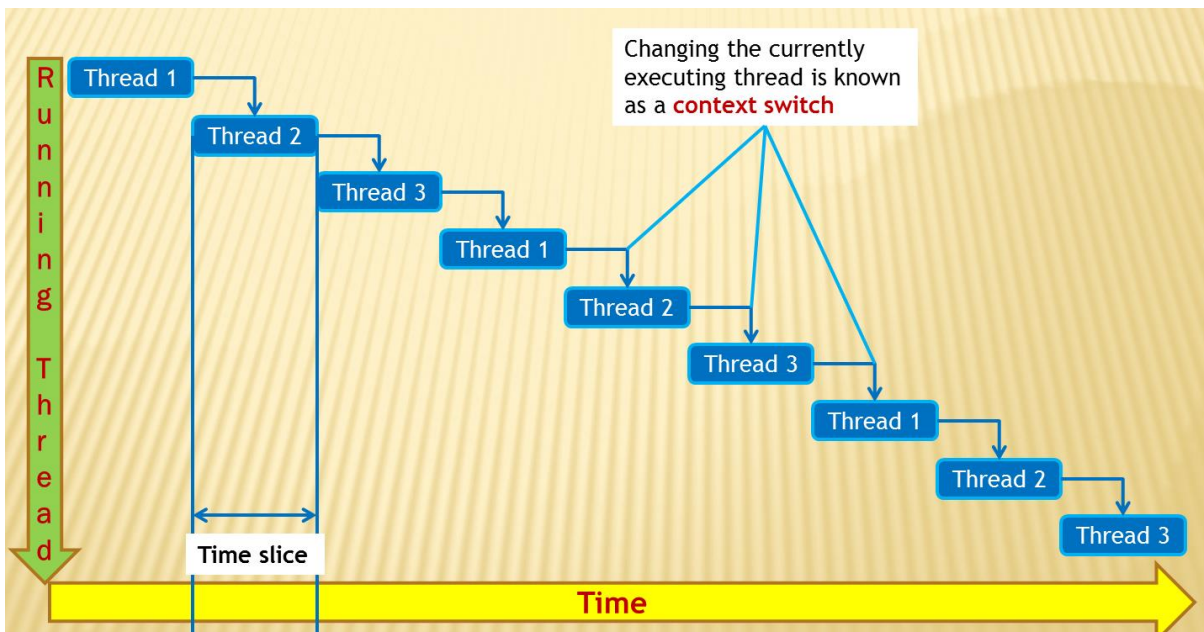


Figure 4-1: Execution of the pre-emptive multitasking model

Of these three items (i) and (iii) are I/O bound processes while item (ii) is a CPU bound process. Items (i) and (iii) will, therefore, frequently have to wait for extended time periods while item (ii) could effectively utilise processor cycles almost continually. To perform a context switch to a completely different process every time (i) or (iii) needed to pause would deprive (ii) of usable CPU time. By dividing a single 'process' into 'multiple threads of execution' that shared the same memory space but received their own CPU time slice allocation the CPU bound tasks could proceed without interruption from I/O bound tasks. This leads to a more efficient allocation of time slices to processes and threads through the 'thread blocking' process.

4.2.3 The operating systems thread scheduler and the thread blocking process

Modern operating systems now use the thread as the smallest unit of programmed instructions that can be independently managed (Lewis & Berg, 1995). At the core of a multitasking operating system is the thread scheduler. This component is part of the operating system and manages the allocation of CPU cycles to the threads of execution. When the pre-emptive multitasking model is used the thread scheduler enforces the allocation of time slices to threads. When a time slice expires threads are forced off the CPU and replaced by the next queued thread. In the co-operative model a thread cannot be forced to yield the CPU but may offer to do so allowing the next queued thread to execute. This round robin system continues allowing each loaded thread to execute in sequence. An I/O bound operation will lead to frequent cases where a thread must pause while it awaits data to be delivered. In these cases the thread scheduler 'unloads' the thread from the queued list of threads awaiting a time slice on the CPU. At this point the thread is said to be 'blocked' and unable to proceed as it is no-longer queued to receive CPU time slices. The thread is held in this suspended state until a signal is received indicating the required data is available. The thread is returned to a queued state once the peripheral unit signals that the

I/O task has been completed. Of course when a *process* is divided into '*multiple threads of execution*' it is possible for a different thread within the same process to proceed even though a different thread within the process is 'blocked'. The result is that the thread & blocking mechanism provide the required granularity to maximise the effective use of CPU cycles between CPU bound and I/O bound operations.

4.2.4 Multiprogramming / multitasking is not parallel computation

From the users point of view an effective multitasking system appear to function as if true parallel computation was occurring. However this is merely an illusion delivered through the high number of context switches between the executing threads and processes. Prior to the development of multi-core CPU systems in 2001 a computer did not perform true parallel computation. This is defined as "the simultaneous performance of multiple operations" and despite the rapid context switching at any given instant only a single operation was being performed. To meet the requirements of true parallel processing the system requires at least two CPU's so that simultaneous performance of multiple operations can occur.

4.3 Development of true parallel computation

In 2001 the International Business Machines Corporation (IBM) released to the market the first non-embedded dual core CPU system, the POWER-4 (Tendler et al., 2002). This was the world's first non-embedded dual-core processor and the forerunner to the modern multi-core CPU systems in use today. The initial driver for the development of multi-core systems was not simply to exploit parallel computation. Rather it came from the industry desire to maintain the phenomenon known as Moore's Law. Formulated in 1965 by Intel co-founder Gordon E. Moore his law states "the number of transistors on integrated circuits doubles approximately every two years" (Moore, 1965), meaning that the computers computational power doubles at the same rate. By the 1990's manufacturers were pushing the physical limits for miniaturisation of components and this was threatening to prevent further growth in the CPU's computational power. Indeed it is something of a misnomer to call Moore's Law a 'Law' it is more an 'observation' of trends in the industry (Dubash, 2005). Parallel computation became seen as a way to maintain the growth in computing power. Instead of packing ever more power into a smaller CPU the new paradigm would be to use multiple CPU's and operations to deliver more compute power in the same time period.

The paradigm shift by hardware manufacturers towards the delivery of increased compute power through multiple cores has given rise to a new 'Law' that rivals Moore's. Known as Amdahl's law it argues that the potential speedup in program / process execution depends on the amount of code parallelisation that can occur in an application (Hill & Marty, 2008). In the case of parallel computation using multiple cores Amdahl's law can be stated mathematically as:

$$S(N) = \frac{1}{(1 - P) + \frac{P}{N}}$$

Where:

- i. $S(N)$ = the maximum speed up from using N processors
- ii. P = the portion of a program that can be made parallel
- iii. N = the number of CPU compute cores working on the problem

In Figure 4-2 Amdahl's law has been graphed to demonstrate the key features required to realise significant performance increases.

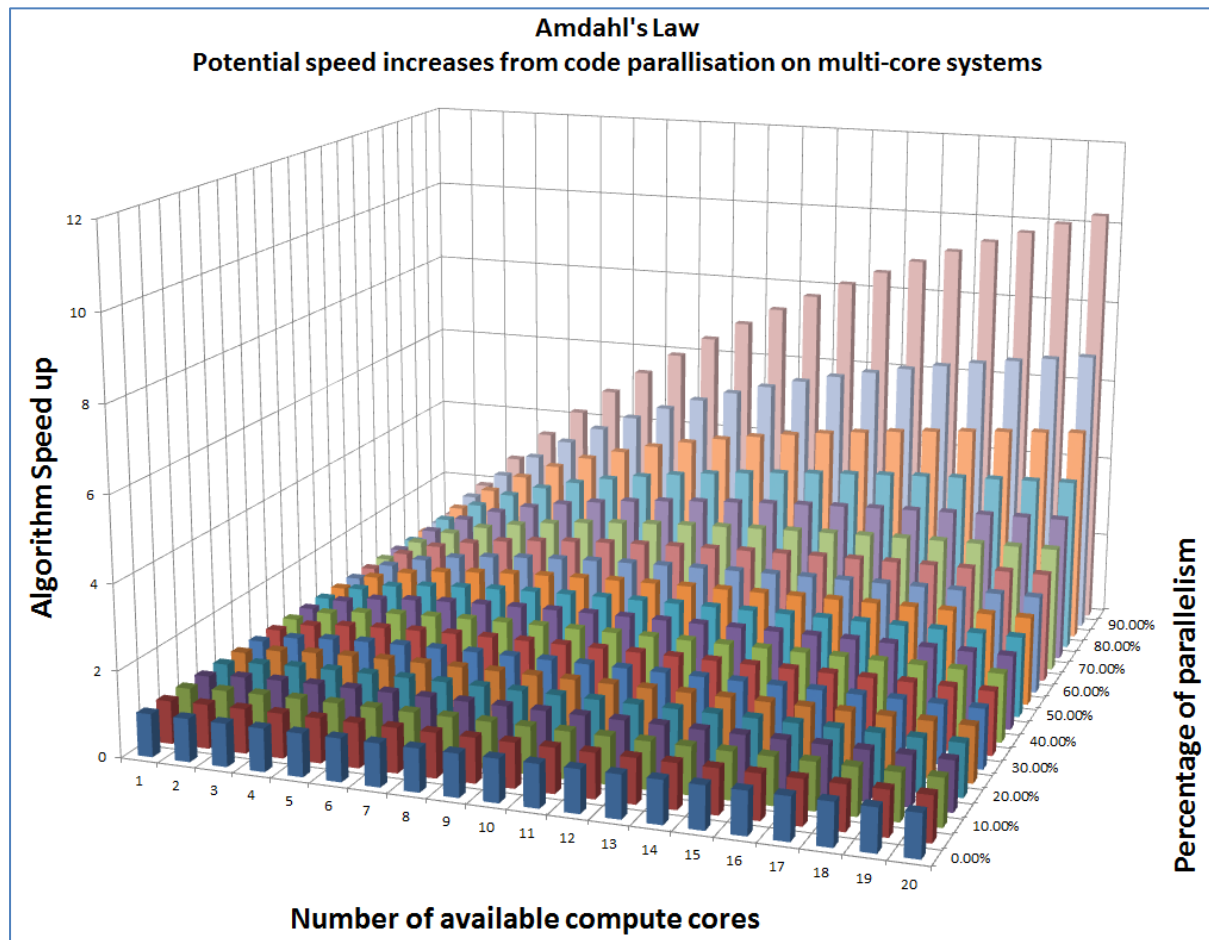


Figure 4-2: Amdahl's Law and realisable performance gains in parallel algorithms

As can be seen from Figure 4-2 the critical factors for determining performance increase are the available number of compute cores and the degree of parallelism that can be introduced into the executing algorithm. It is also clear that a sharp law of diminishing returns is in effect for both factors. By far the most important of these factors is the percentage of parallelism that can be introduced into an algorithm. This is because the average desktop / laptop system incorporates only a relatively low number of compute cores. The typical commercial computer system in 2014 features 4-6 compute cores on average. The high end research machine deployed in universities might expect 8 compute cores to be available. To secure access to larger numbers of cores requires execution of the algorithm on a cluster computer system or in a distributed cloud based environment. From an examination of the data underlying Figure 4-2 it can be seen that moving a highly parallel algorithm (say 95%) from a 4 core to a 6 core system yields a 38% decrease in execution time. The same algorithm moved to an 8 core system would realise a 70% decrease in execution time. This represents a significant performance boost and makes it possible to conclude that the relatively low number of compute cores in an average computer system avoids the full impact of the law of diminishing returns. The degree / percentage of parallelism in an algorithm is a quite different story, historically this percentage has been 0% for the majority of algorithms. This has arisen because there has been no point or benefit to

writing parallel code for the majority of algorithms that ran on a multitasking system. As multitasking was not true parallelism no benefit could arise from writing code to execute in parallel. It would always be serially executed regardless. Amdahl's law asserts that in a multicore environment this is no longer true. In this case an algorithm must be divided into a parallel component and a sequential component. Increasing the availability of compute cores speeds execution of the algorithms parallel component but has no impact on the sequential component (Hill & Marty, 2008). It has been proven that any algorithm will contain a sequential and un-parallisable element (Kirk & Hwu, 2012). Most problems also include a parallisable element allowing them to exploit a multicore computer to improve performance. The exact performance boost that can be realised will depend on the exact form of parallel computation that the algorithm can exploit.

4.4 Forms of computation

Research into parallel computation has identified four forms or category's into which algorithms can be divided (Flynn, 1972). In his taxonomy Flynn identified these as:

1. Single instruction, single data stream (SISD)
2. Single instruction, multiple data streams (SIMD)
3. Multiple instructions, single data stream (MISD) and
4. Multiple instructions, multiple data streams (MIMD)

With the exception of the first category (SISD) all of the others can be subjected to some level of parallelism. Each will be examined in turn and illustrated with an example. Note that the sequential component of the algorithm, storing a value into memory in each example given, has been omitted for clarity.

4.4.1 Single instruction, single data stream (SISD)

This category defines the algorithms executed on a classical single core computer system. In this case at any given instant a single instruction is executing against a single item of data. For example scaling the vector by $\begin{pmatrix} 3 \\ 2 \end{pmatrix}$ by the value 5 might be implemented as a single instruction (multiply by 5) and a single data stream delivering the values 3 and 2. A single compute core sequentially applying the instruction to each value of the data stream will yield the solution vector of $\begin{pmatrix} 15 \\ 10 \end{pmatrix}$. As with the classical single core computer system this type of computation cannot be considered parallel computation.

4.4.2 Single instruction, multiple data streams (SIMD)

Algorithms falling into this category define a single instruction but need to execute it multiple times. Each compute core takes on responsibility for execution of the instruction against a data stream. The vector scaling operation used above may be modified to demonstrate this case. The instruction remains unchanged (multiply by 5) but each compute core receives a stream of data that contains a single data item (one core receives the value 3 the second the value 2). The same solution vector is reached but the data items are processed 'in parallel' on their own compute core.

4.4.3 Multiple instructions, single data stream (MISD)

Algorithms falling into this category define multiple instructions but provide a single stream of data against which all instructions must be executed. Again we may restate the

vector scaling problem used above as a MISD algorithm. In this version two instructions will be defined:

1. Multiply by 3 and
2. Multiply by 2.

The single data stream holds the scalar value of 5. Each compute core will execute an individual (but different) instruction against the delivered data stream value but the solution vector will still be correctly derived.

4.4.4 Multiple instructions, multiple data streams (MIMD)

Completing the categories of processing algorithms the multiple instructions, multiple data streams could be considered the most parallelisable form of algorithm. The vector scaling example used previously does not fall into this category. However a simple matrix multiplication problem will serve to illustrate this category of algorithm. Here two matrices are multiplied together to give a solution ($A * B = AB$).

$$\begin{matrix} \begin{bmatrix} 3 & 2 \\ 1 & 0 \end{bmatrix} & \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} & = & \begin{bmatrix} 14 & 19 \\ 2 & 3 \end{bmatrix} \\ A & B & = & AB \end{matrix}$$

Matrix 'A' forms the multiple instructions set with each row in the matrix generating an instruction set. Each value in a row will generate its own instruction to form the instruction set. The columns of matrix 'B' will each form a data stream which is delivered to each instruction set. Each instruction set will contain at a minimum two instructions (multiply by a value and add the result to a running total). Equally at a minimum there will be two streams of data delivered to the compute core (a column of matrix 'B').

In addition to the various *forms* of computation there is at least one *type* of problem that is suited to solution by parallel computation.

4.5 Embarrassingly Parallel Problems

Cleve Moler a co-founder of MathWorks and developer of the MATLAB software first coined the term 'embarrassingly parallel' to describe problems "for which little or no effort is required to separate the problem into a number of parallel tasks. This is often the case where there exists no dependency (or communication) between those parallel tasks"(Foster, 1995; Moler, 1986). If Amdahl's law is applied to such a problem it will have a P value close to 1 (remembering that it can never be 1). Figure 4-2 shows that such a problem is a perfect candidate for parallel computation. High increases in performance being available with even a small number of compute cores. The potential increase in execution performance is effectively limited by the number of available compute cores. These problems are, therefore, prime candidates for solution by cluster or cloud computing systems.

Embarrassingly parallel problems abound both in science and nature making the multicore computing system ideal for their solution. Examples would include:

1. Event simulation in particle physics.
2. Ensemble calculations of numerical weather prediction.
3. Genetic algorithms and many other evolutionary computing techniques.
4. Brute-force searches in cryptography (such as Bitcoin mining).
5. Serving static files on a web server to multiple users simultaneously.

6. Computer simulations comparing many independent scenarios, such as climate models.
7. Pairwise cross correlation of neural spike trains.

As the name of these problems implies failure to exploit the paradigm shift to the delivery of increased computing power through multicore hardware and parallel programming could be seen as 'embarrassing' when solving these problems. Almost all modern hardware on which computer programs are developed is a multicore system delivering compute power through multiple cores. If multicore hardware has become ubiquitous then it is to the shame of the modern software development community that the parallel programming skills needed to fully utilise it have not.

4.6 Parallel Programming – The failure to launch

Ever since programming emerged as a distinct discipline developers have been trying to abstract themselves away from the hardware on which the code is executed. Modern programming paradigms such as Object Oriented Programming (OOP) focus on describing the logic of an algorithm without any reference to the executing hardware. Popular programming languages such as Java and the .NET Framework family (including C#, C++, Visual Basic) now use 'virtual machines' such as Java's JRE and Microsoft's Common Language Runtime (CLR). This abstracts even the concept of the code's execution environment. Abstraction is usually implemented as a layered architecture with each layer building on the previous to provide a different representation of the same process (Floridi & Sanders, 2004). An instance where abstraction breaks down and the physical hardware becomes relevant to the higher abstraction layers is not well handled by programmers / developers. An example of this from the history of parallel computation would be the co-operative vs pre-emptive multitasking models of thread scheduling. As described in the co-operative multitasking model developers were required to write code that 'offered' time on the CPU to other executing threads. Traditionally management of access to CPU cycles falls into the operating systems layer and not the creation of application code. This is the primary reason for the co-operative multitasking model being dropped in favour of pre-emptive multitasking model in the modern operating system. Pre-emptive multitasking manages CPU access entirely within the operating system and so respects the traditional layered model.

Parallel computation represents another programming challenge that violates the traditional layered approach that abstracts the software developer from the hardware. Indeed Amdahl's law expresses beautifully the close relationship between an algorithm's performance, the executing hardware and the degree of parallelism built into the algorithm (see Figure 4-2). Nevertheless traditional software development emphasises ignoring hardware and views the algorithm's logic as sequentially executing functions. The training of software engineers at colleges and universities continues to be focused primarily on traditional programming models. Parallel computation, if taught at all, appears in only 1-2 final year undergraduate modules that often have little hands on experience (Higginbotham & Morelli, 1991). This occurs even though almost every computer in use is at least a dual core system. Most historians mark the end of the single-core era as May 2004 when the Intel Corporation dropped development of its last single core systems (Flynn, 2004). A decade later software engineers are still only making slow progress to accept that the paradigm of adding compute power through multiple cores requires a new coding paradigm. Despite the slow progress the development community is fully aware that a parallel coding paradigm is

required. In 2009 David Stewart CEO of CriticalBlue and chairperson for the Multicore Programming Practices (MPP) working group comments on this situation stating that “There's capability in (multicore) platforms which is not being utilized or not being optimized by the software development community” (Myslewski, 2009).

Given the clear need for parallel code production, or concurrent programming, it must be asked why software engineers and the development community have made such slow progress? Research has been underway in the field of parallel code production. The most critical barrier to progress was identified by Apple Inc. which has observed that “the dominant model for concurrent programming - threads and locks - is too difficult to be worth the effort for most applications” (Apple, 2009). This conclusion would suggest that the current parallel programming models are not fit for purpose. Indeed it has been observed that this style of programming has “a well-deserved reputation for introducing bugs that are difficult to find and fix” (Myslewski, 2009). What is needed then is either:

1. A new way of doing parallel / concurrent programming or...
2. An easier and more productive way to utilise the current model of threads and locks.

At the moment no-one has advanced an approach for concurrent programming that fully respects the traditional abstraction layers between developer and hardware. Therefore most research has focused on the second option of simplifying how the current model is used.

In summary, while the hardware to support parallel / concurrent programming has spread into almost every environment and computing device, the software development community has been slow to design and develop applications that exploit the full potential of this hardware. In turn this has restricted the opportunities to exploit this power in both business and scientific communities. Software development languages are now belatedly adopting concurrency frameworks for parallel computation. However the teaching and training of software engineers to identify and exploit opportunities for parallelisation of code remains spotty at best.

4.6.1 Concurrency Frameworks – An abstraction layer for parallel coding

Research into simplifying the current thread and locks model has taken the form of the development of “Concurrency Frameworks” that are added to the traditional development languages. These frameworks offer a new abstraction layer between developer and hardware. In principle these frameworks accept that traditional threads and locks are too close to the executing hardware. At the application coding level developers want to work with abstract concepts. In the case of concurrent / parallel coding the most obvious abstraction is the *parallel task*. This is the concept that the code to complete a task should be grouped into a logical whole, assigned to a compute core and that at some indeterminate point in the future it will produce a result. This clearly leaves the developer with more hardware dependent considerations than normal, like a need to consider the number of available compute cores. It does, however, abstract away responsibility for managing threads via the software design pattern of a thread pool.

4.6.2 The spread of Concurrency Frameworks in modern development languages

The creation and incorporation of concurrency frameworks into modern coding languages has been a slow process. It may still be considered a second best solution to a

completely new coding paradigm and as Goetz the primary developer of the Java concurrency framework notes “writing correct programs is hard; writing correct concurrent programs is harder” (Goetz et al., 2006). Nevertheless in the absence of any better paradigm the major software development languages have seen the need to develop concurrency frameworks.

Language	Concurrency Framework	Release Date	Source
Python	Stackless Python	01/2000	(Tismer, 2000)
Java	JSR 166: Concurrency Utilities	09/2004	(Lea, 2004)
C++	Intel® Threading Building Blocks	08/2006	(Reinders, 2007)
.Net Framework (v4)	Task Parallel Library (available in all .Net languages)	04/2010	(Microsoft, 2010)

Table 4-1: Creation of concurrency frameworks in application development languages. (Tucker, Barlow & Stuart, 2012)

Table 4-1 details the creation and deployment dates for concurrent programming frameworks in major application development languages. As can be seen deployment has been slow with Microsoft being the latest adopter to introduce a concurrency framework for its .NET programming languages. Early adopters of concurrency frameworks have already enjoyed considerable commercial success. Stackless Python for example is credited with the commercial success of EVE Online with over 500,000 paying subscribers (Loktofeit, 2013). EVE Online was developed by CCP Games based in Reykjavik, Iceland and the CEO Hilmar Veigar Petursson has been cited as crediting Stackless Python's concurrency features as the prime reason for the language selection as the EVE Online implementation language. In endorsing the language he said:

“When embarking on the creation of EVE Online, a single shared persistent world we realized two things: We could not given constraints of time and commercial reality do this in a compiled language and we needed innovative concurrency control for such a large scale shared state simulation across tens of thousands of CPUs (EVE Clients included). After many experiments with various combination of existing scripting languages and NT fibers we arrived at Stackless Python. Stackless Python offered us the power of Python coupled with a vastly superior concurrency control mechanism over anything we had seen before, first as continuations and later with an innovative channel based API. CCP's commercial success today is built on the single decision of selecting Stackless Python as our foundation” (Petursson, 2011).

Despite the difficulties of the thread and locks model for concurrent programming the developers of concurrency frameworks have defended it. Oracle's Java language architect Brian Goetz has observed “threads are the easiest way to tap the computing power of multiprocessor systems” and “as processor counts increase, exploiting concurrency effectively will only become more important” (Goetz *et al.*, 2006). It is such statements as these and the successful commercialisation of concurrency frameworks that the drive to deploy and use concurrent frameworks arises. This drive finally seems to be overcoming the development communities' general resistance to adopting concurrent / parallel programming due to the difficulties involved.

The advantages of a concurrency framework may be summarised as (Oracle, 2004):

1. Reusability and effort reduction: The effort required to create concurrent code is considerable but the core elements are used and reused throughout development. Making these available within a framework increases code re-use and reduces effort.
2. Superior performance: The primary goal of all concurrent code is to balance the degree of parallelism and the number of compute cores to minimize execution time.
3. Higher reliability: Parallel coding has long been considered a 'bug magnet'. By working with tested and proven framework code a programs reliability should be increased.
4. Maintainability and Scalability: Reusable framework components are easier to maintain and can be scaled as technology advances.
5. Increased Developer Productivity: Abstraction of the developer from the hardware is (partially) restored allowing them to focus on *what to do* and not *how the computer does it*.

4.6.3 General structure of a concurrency framework

The problems associated with concurrent programming have been understood for some time and the production of concurrency frameworks does now have at least some common elements. This is true regardless of programming language since the underlying problems remain the same. This project will draw extensively on improved execution time performance to generate interactive visual displays for large datasets. This will be mainly achieved through concurrent code exploiting the Java concurrency framework. It seems worthwhile at this point to examine the elements that commonly appear in a concurrency framework with emphasis on the Java implementation.

The formal specification for the Java Concurrency Framework is given in Java Specification Request - 000166 Concurrency Utilities (Lea, 2004). It is divided into the following six core component:

1. Task Scheduling Framework
2. Concurrent Collections
3. Atomic Variables
4. Synchronizers
5. Locks
6. Nanosecond-granularity timing

Each of these framework components will be considered in turn and their role in Java's concurrent programming model explained from the framework documentation (Oracle, 2004).

4.6.3.1 Task Scheduling Framework

This component is charged with the execution of the concurrent tasks submitted by the code for parallel computation. This manages invocation, scheduling, execution and control over parallel code tasks within a set of configurable policies. At its heart the Executor framework manages these tasks, interacts with the sequential application code and executes tasks either on a single background thread or through an application of the thread pool software design pattern.

4.6.3.2 Concurrent Collections

All modern development languages provide a set of common data structures, called collections, such as lists, queues, stacks, and maps. Each data structure provides a means to manage a collection of data or data objects. The primary barrier preventing the use of such structures in the concurrent environment is the pre-emptive multitasking thread scheduler itself. With its aggressive enforcement of time slice allocation to application threads the scheduler can corrupt data by forcing a thread off the CPU before a memory update can be completed. Classic data structures are not generally 'thread safe' in that they cannot allow multiple threads to access the data without the risk of corruption.

This component provides a set of 'thread safe' data structures that are immune to data corruption when accessed by multiple threads. This reduces developer workload, as most thread safety issues are eliminated, and improves reliability as the opportunity for many common concurrency bugs to enter the code base is eliminated. In addition these classes are optimised to offer high performance making them more suitable for implementing high performance concurrent applications.

4.6.3.3 Atomic Variables

As with data structures and collections the classic variables such as Integer and Boolean are vulnerable to data corruption by a pre-emptive thread scheduler. In this component they are re-implemented as 'atomic' variables where the thread scheduler may not corrupt their values by interrupting update operations.

4.6.3.4 Synchronizers

The need to synchronise the operation of multiple threads when writing concurrent code cannot be completely avoided, despite it being a relatively low level operation that is usually abstracted away from the application developer. This component provides pre-written and tested synchronisation / locking systems. In addition to the classic semaphores and mutex locks the Java concurrency framework provides barriers, latches and exchangers. This provides a set of reusable components that avoid the need to manually code most thread locking strategies.

4.6.3.5 Locks

Synchronisation between thread operations is normally performed through the locking and thread blocking mechanism described in 4.2.3. The concurrency framework includes many pre-written locking strategies, through its synchronizers. However the complexity of concurrent programming ensures that situations can arise where developers must write their own. This component provides developer access to both the classic memory locks and more advanced timed, multiple condition and non-lexically scoped locks.

4.6.3.6 Nanosecond-granularity timing

The timing component provides access to a nanosecond granularity time source (assuming the executing platform support nanosecond timing). Many portions of the concurrency framework depend on accurate timing. The parallel code developer may also require such timing accuracy if (or when) forced to interact with the system at a low level.

4.6.3.7 Recent changes to the Java Concurrency Framework

In the course of this projects execution the Java concurrency framework has undergone a major update. While the project uses Java's JDK 1.6 and does not utilise these

changes no review of the development of parallel / concurrent programming could be complete without covering the most recent changes. Concurrent programming research is an expanding field which in the case of Java 7 has led to the following major updates to the concurrency framework (Konda, 2011; Oracle, 2014):

1. The Executor framework in the Task Scheduling framework has been extended to support Fork & Join operations such as those common to UNIX systems.
2. The pre-written synchronizers have added the Phaser that merges functionality from a CyclicBarrier and CountdownLatch as well as allowing an application to be divided into a set of 'phases' during execution.
3. The generation of pseudo-random numbers without cross thread communication has been added. This will reduce thread contention and further optimise performance.

4.7 The future for parallel / concurrent computation

The road ahead for the effective development of parallel computation and concurrent programming remains unclear. Parallel computation became necessary as hardware shifted from delivering compute power via a single core CPU to the multicore CPU. Advances in computer hardware have not slowed. Manufacturers seem committed to multicore and have started to open up the massively parallel compute cores of graphics cards for general purpose computing operations (GPGPU's). The increasingly interconnected world has also seen cloud computing emerge as a practical means to deliver compute power that was historically restricted to the high performance compute cluster in a practical way to the masses (Apache, 2014; NVIDIA, 2014a; NVIDIA, 2014b). Many big data problems such as that of neuroscience may benefit from leveraging this additional compute power. Additionally many computers already include the hardware for GPGPU programming and frameworks such as CUDA and OpenCL are emerging to program them (Khronos, 2014; NVIDIA, 2014c).

While still lagging behind hardware development, it may be that these software frameworks and the distributed computing technologies of cloud computing represent the future. A future that will deliver greater compute power in a massively parallel but distributed way.

Chapter 5

Visual Programming Languages

“Visual Programming Language (VPL) is an application development environment designed on a graphical dataflow-based programming model.”

Summary

This chapter examines Visual Programming Languages, how they work and the rationale behind creating one for the third iteration of the VISA project.

5 Overview

One of the primary contributions of this project is the development of a Visual Programming Language (hereafter a VPL) as a means to create a concurrent data processing environment that can be customised to perform data analysis for any type of data. This chapter examines VPL's and their suitability as a means of creating a user customisable data processing pipeline. Additionally it introduces the Visualisation of Inter-Spike Associations project and examines why development of a VPL for the project was undertaken for the third iteration of its software.

5.1 The Information Processing Cycle

The operation of any computer system can be described at the highest level by the information processing cycle. This cycle is considered to have four parts which are:

- i. Input
- ii. Processor / Processing
- iii. Storage
- iv. Output

In describing the system the terms 'data' and 'information' must be carefully defined. Data represents the raw unorganised facts that need to be processed. Information is processed data which is organised, structured or presented in a given context so as to make it *useful*. Computers, and all other information processing systems, transform raw data into useful information. The information processing cycle describes at a high level how this is performed.

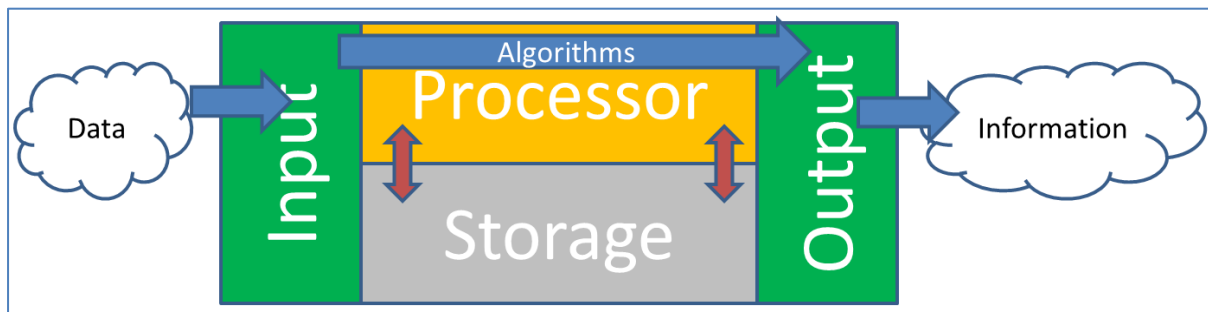


Figure 5-1: The Information Processing Cycle

Figure 5-1 provides a representation of the information processing cycle in which raw unorganised data is input into a processing system. Algorithms are then applied to this data to organise and structure it. The selection of algorithms depends, of course, on what the user deems will be 'useful' for their purpose. One or more algorithms may be used giving rise to the need to store data being processed. The final processed output depends on the algorithms used and their order of application. No matter how it was created the output must be presented to the user in some fashion, classically on a computer monitor though this is not a requirement.

Despite having four distinct components the (iii) component storage serves to support the processing operation. There are many forms of data storage but they simply act

to persist and retrieve data on demand. No actual processing occurs to stored data and as such the information cycle has three 'active' phases.

5.1.1 The Input Phase

This phase is primarily focused on providing the raw data to the processing system. As such it serves as the interface between the data processing system and the outside world. The data deluge of the modern information age has led to the creation of a wide array of data storage and delivery systems that deliver data in an equally numerous number of formats. Examples would include keyboards, microphones, hard disk drives, mice and track pads. Before the processing component of the cycle can begin data must first be read and placed into a storage location from which it can be delivered to the processing elements on demand. In the classical computer system this will be some form of memory ranging from CPU registers to random access memory (RAM) or a hard disk drive. The key requirements to complete the input phase are:

- i. The ability to access the raw data – Appropriate permissions, physical access to the data storage mechanism or some sort of remote access such as modern cloud and distributed processing systems can provide.
- ii. An understanding of the data's semantics. Data represents discrete facts and therefore includes an element of meaning. For example the integer number 8 is a piece of data but is incomplete without meaning. It can represent a wide range of data such as 8 planets, 8 miles per hour or 8 degrees Celsius. Hence to be used by information processing systems data is not simply values but also their meaning.

The input process then receives data and stores both discrete values and their meaning in a form suitable for further processing. In the modern computer a 'data model' is often created in memory that represents the raw data.

5.1.2 The Processing Phase

In this phase a sequence of algorithms is applied to the raw data aimed at extracting 'useful features' from the data. What exactly constitutes a useful feature depends entirely on the information that the user desires to extract and the use to which it will be put. In any significantly sized data set there will be a wide range of possible information depending on the meaning of the raw data and the overall goal of the processing. Nevertheless the processing phase will always have the following features:

- i. There will be one or more goal(s).
- ii. There will be one or more algorithm(s), even if the only algorithm is to produce an in memory representation of the raw data. The algorithm transforms the data into information and has been called the defining activity of computers (Illingworth, 1997).
- iii. The processing result will be delivered to some kind of output phase.

The creation of goal(s) for the processing will define what constitutes 'useful' information and will usually involve answering one or more research question(s). Complex goals may break down into a number of sub goals that must be achieved before the primary goal can be completed. The overall goal will constrain the data that must be made available and the algorithms that will be applied.

When the goals have been established algorithm selection and sequencing can occur. Here the user identifies the individual operations that must be performed to transform

the raw data into the required information. The algorithms could be viewed as tools in a toolbox with the user selecting those that reveal the information needed to meet the processing goals. As with the classical engineers toolbox the 'tools', aka algorithms must be selected and applied in the correct order to achieve the processing goals. For complex tasks or multiple goals it may be useful to change the order in which the algorithms are applied either to provide different information about the same data or to provide the same information presented in a different way. The ability to derive new information by viewing the same information in different ways is one of the strengths of the human visual processing system (see chapter 2).

Regardless of the algorithm(s) applied and in what sequence the end result must finally be presented to the information user.

5.1.3 The Output Phase

To be useful the results of processing must be presented to a user who applies the information to the solution of a problem. As with the input phase there is a wide range of possible outputs. Broadly the output can fall into two categories:

- i. Static – Typical examples would be printouts and fixed information displays
- ii. Interactive – In this case the user has the option to further interact with the output essentially performing further processing themselves.

The classic examples of static output is printed output be it numeric, textual, graphical or some combination of these three. Other examples would be a static image rendered to a display screen or saved to a storage medium. The increasing power of computers and their impact on the field of data visualisation (see chapter 2) has allowed far greater use of interactive output systems. Such systems are particularly useful were it is uncertain exactly what algorithms or processing might yield useful information. In the interactive system the user may manipulate the output to reveal additional information beyond that identified in the initial processing. These exploratory data operations if they consistently reveal useful information may then be consistently applied to future analysis as a new algorithm. Some output systems, such as information displays and sonification, may blur the line between static and interactive output. Such systems may present static information views or if the user can manipulate them become truly interactive.

In summary then the Information Processing Cycle can be seen as describing a 'flow' of data (the input) through a sequence of transformations (processes) to produce useful information (the output).

5.2 Dataflow Programming Languages & Visual Programming Languages

The classical computer has at its core a "von Neumann" processor (Neumann, 1945) however this architecture is not the only solution to constructing a data processor. In the early 1970's research into parallel processing showed that the von Neumann architecture suffered from two bottlenecks (Ackerman, 1982; Backus, 1977):

- i. The global program counter.
- ii. The global updatable memory.

The global program counter is simply a register that stores the address of the next executable code instruction. This hinders parallel execution as that requires the execution of multiple instructions whereas the classic program counter can only point to one instruction. Global updatable memory refers to the processors access, and ability to modify any value in memory. In parallel code this leads to conflicts as different instructions may attempt to access the same memory location. Different instructions executing at the same time may update or read incorrect values from global memory.

Researchers proposed a new processor architecture that was named a dataflow system to address these deficiencies. In a dataflow system there is no concept of an object held in global memory, such as a variable. Instead the dataflow model deals only with values and processing is achieved by manipulating values to produce new values. Each programming 'statement' has a precise mathematical meaning given by a mathematical function. No global counter is employed instead the mathematical functions execute as their input values become available. The function's output is routed to another function where it provides an input value. This function will in its turn wait until all values that are routed to it as inputs have been computed and then immediately execute. A program constructed in this manner can be represented as a 'data dependency graph' and their execution is said to be data driven.

The advantage of the dataflow system is that it is naturally paralysable with each function being independently executed as its input values become available. As such programs also form a data dependency graph they may be visualised as sequence of connected nodes in a directed graph. This leads to an interesting possibility where such a system may be 'visually programmed'. The programmer adds functions to a directed graph and routes the output of each function to the input of another. The resulting graph represents a naturally paralysable program where each node executes as previous nodes make data available. This approach allows the construction of visual programming languages (VPL's). Figure 5-2 illustrates the concept using the Microsoft Visual Programming Language.

The coupling of dataflow with a graphical representation as a directed graph was first proposed by Sutherland in his 1966 PhD thesis (Sutherland, 1966). This resulted in the creation of many dataflow languages of which LabVIEW is perhaps the most famous (*Graphical Programming*, 2013). While the naturally paralysable nature of dataflow programs provides a significant advantage given that modern hardware is now delivering increased compute power through parallelism a number of problems remain for dataflow system. These can be summarised as:

- i. The handling of complex data structures as values is inefficient, but no complete solution to this has been proposed. In this project a high degree of Object Orientated Programming (OOP) and abstraction will be used to address the need for complex data structures.
- ii. Dataflow computation tends to involve long 'pipelines' of interconnected processes and this can degrade performance where the application being programmed is not sufficiently paralysable.
- iii. The programmer does not have explicit control over memory and a highly branched pipeline will lead to extensive data duplication as different branches seek to independently process the same data. Shared data structures are possible but will

require development of thread safe data structures and careful memory management / garbage collection mechanisms.

- iv. The complexities introduced by (ii) and (iii) will result in a management overhead that consumes both memory and time. Therefore although the pipelines parallelism is exposed to the hardware its full advantages may not be realised.

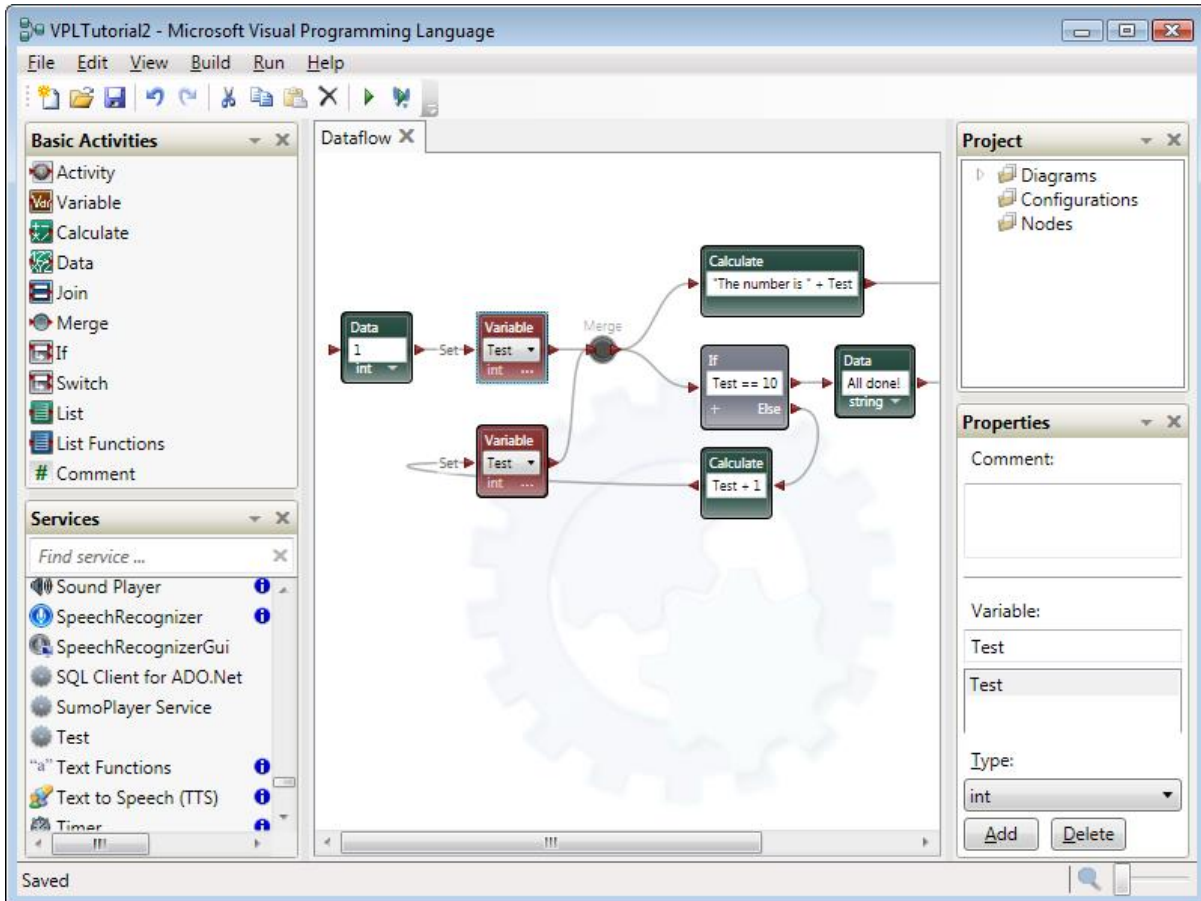


Figure 5-2: A dataflow system visualised as a directed graph. (Source: (Microsoft, 2012)).

Visual Programming Languages (VPL's) have enjoyed success in the academic field and in the research labs of the world, however mainstream business programming continues to favour text based compiled or interpreted languages. The construction of data processing pipelines in which the user can introduce new algorithms by manipulating graphical elements rather than re-coding an application offers great flexibility.

5.2.1 Dataflow implementation models

There are essentially two implementation models for dataflow / workflow programming. The token based dataflow model and the structure based dataflow model (Johnston, Hanna & Millar, 2004). In either of these models execution of the dataflow program is controlled using either a data-availability-driven approach or a demand-driven approach. This section will examine these options and the rationale behind selecting a structure based data-availability-driven approach for implementation.

5.2.1.1 The token based dataflow model

In token based dataflow the connections between nodes of the dataflow directed graph carry 'tokens' that encapsulate the data being processed and convey it between

processing nodes. The connections are termed 'edges' or 'arcs'. These arcs may either be input arcs that deliver data for processing or output arcs that transmit processing results to the next node in the directed graph. From the implementation point of view these arcs are unbounded first-in first-out (FIFO) queues (Khan, 1974) that deliver tokens to processing nodes. A processing node becomes fireable, or executable, when at least one token is available on an input arc. Nodes may have multiple input arcs that form a 'firing set' all of which must carry a token before the node becomes fireable (Davis & Keller, 1982). To begin the computation special input nodes are required that generate the first set of tokens. Once a node is fireable it removes a data token from its input arc(s) and performs its operation. At the conclusion of processing the node places a new token on one or more of its output arcs. This in turn makes further nodes fireable and advancing the program to the next processing step.

5.2.1.2 The structure based dataflow model

The structure based dataflow model emerged from research in the early 1980's and uses the same arc and node directed graph structure as the token model (Davis & Keller, 1982; Keller & Yen, 1981). The key difference is that while each node in the token model can generate multiple tokens, i.e. it is a stream processor, in the structure model each node creates only one data object or 'structure' on each of its output arcs. This structure is not removed from a FIFO queue by later nodes but remains on the arc. The primary advantage of the structure approach is:

"It is possible for these structures to hold infinite arrays of values, permitting open-ended execution, and creating the same effect as the token model, but with the advantage that the structure model permits random access of the data structures and history sensitivity" (Johnston, Hanna & Millar, 2004).

The structure model is attractive given its ability to define a structure for almost any data model while offering random data access and a record of processing history. Nevertheless it has not seen wide adoption in dataflow programming because of its primary disadvantage. The disadvantage to the structure based approach is that it is inherently less efficient at storing data than token based systems. This arises because to preserve the ability to examine the history of the program all generated data must be preserved. Data structures can grow very large as the program progresses if a complete processing history is preserved.

5.2.2 Dataflow execution models

Regardless of the selected dataflow implementation model it will be necessary to control the execution of the dataflow program. This involves identifying when nodes of the directed graph have become 'fireable' and scheduling them for execution. Either of the two dataflow implementation models creates 'tokens' that are placed on the 'arcs' of the dataflow diagram. Note that the data structures from the structure implementation can be considered a form of 'token'. Research has shown that rather than treating these tokens as passive conveyors of data they are the best means to control program execution (Kosinski, 1978). There are two approaches to controlling the execution of a dataflow program.

5.2.2.1 Program execution using the data-availability-driven approach

The data-availability-driven approach could be considered the 'pure' dataflow approach. Execution of program nodes in the directed graph begins when a node becomes

'fireable'. The status of a node as fireable depends on the availability of data in the form of a token (or structure) on a nodes input arc(s). Only when all input arcs have a data token available is a node fireable and becomes eligible for execution at the next opportunity.

At the end of execution each node will place a token (or structure) on its output arc(s). These in turn will cause further nodes in the directed graph to become fireable. This process repeats until a terminating condition is reached. In the case of the VISA 3 project the normal terminating condition is that a visualisation node is reached. The processed data is delivered to the visualisation application encapsulated by the node.

5.2.2.2 Program execution using the demand-driven approach

The demand-driven approach reverses the data-availability approach. Instead of a node passively waiting to become 'fireable' it monitors its output arc(s) waiting to receive a demand for data. When a demand is received the node places demands for data onto its input arc(s). This in turn serves to trigger demands for data from nodes higher up the directed graph. Ultimately a node is reached that either has or can acquire / load the demanded data. Once a node receives its demanded data (a token / structure on an input arc) it processes the data and places its own token / structure onto the output arc. Johnston et al describe the demand-driven approach as a four step process (Johnston, Hanna & Millar, 2004):

1. A node's environment requests data.
2. The node activates and requests data from the environment
3. The environment delivers the requested data
4. The node places a token / structure onto its output arc(s).

5.2.3 Selection of dataflow model and execution method for VISA 3 implementation

In the final implementation of the VPL's directed dataflow graph this project has selected a structure based dataflow model with a data-availability-driven approach to the VPL's execution.

The choice of a data-availability-driven approach to the VPL's execution is not particularly contentious. This approach mirrors closely the information processing cycle of input -> processing -> output around which the VPL is built. The majority of the VPL design follows a top down approach and it made sense to continue this with the execution method. Switching to a demand-driven approach would have required users to select which visualisations should be activated ahead of execution. This seems an unnecessary step as the data availability approach can automate the process of determining a complete workflow. This avoids both work and cognitive load on the user. In summary then the data availability approach was selected for this project because:

- The data-availability approach mirrors the input -> processing -> output cycle of the VPL's processing nodes.
- The data-availability approach is "top-down" which replicates how the typical user would read a flow chart and build processing pipelines in the VPL.
- The step of defining which visualisations will "demand" data can be eliminated.
- Cognitive load on the user is reduced because:
 - Most users will be familiar the flow chart analogy

- Users will naturally build their visual processing pipelines by defining inputs, the processing to be performed and the output visualisations to be produced.

More contentious is the choice of a structure based dataflow model. The majority of dataflow programming languages use the token based dataflow model with the structure approach not proving popular. The primary reason for selecting the structure based dataflow model was the broad specification given for this VPL. Concrete information was available for the neuroscience data that would be used with the VPL. However the specification required the VPL to adapt to potentially process any type of data. It would have been very difficult to design a 'token' based system without imposing some limits on the type or amount of data it could carry. The structure based approach is inherently more suitable being more generalised. Its use also offered the problem domain specific process developer the opportunity to define the structure for themselves. This provides the ability to tailor the structure to the problem domain data. The structure approach also naturally allows for random access to the data structure and the tracking of execution history. This is a significant number of advantages that could be useful when analysing a wide range of data. The drawback that had to be accepted was the potential that data structures may grow quite large. Problem domain developers should look to ensure they store data in an efficient manner and track processing history only where it is needed. In summary then the selection of the structure based dataflow model was made for the following reasons:

- The definition of the "structure" that carried data through the dataflow model could be done by the problem domain developer.
- The problem domain developer can design and tailor the structure to support the data and processing they wish to perform.
- The structure approach has the potential to carry an unlimited amount of data (within the constraints of the computer systems memory)
- The structure approach affords random access to data as needed by the processing nodes.
- The structure approach allows several input arcs to be activated simultaneously. This supports execution of processing nodes in parallel.

5.3 The Visualisation of Inter-Spike Associations (VISA project)

The Visualisation of Inter-Spike Associations project lead by Dr Liz Stuart at Plymouth University will provide the problem domain for the initial iPipeline process library. To fully understand the potential benefits of employing a VPL to perform data processing it is necessary to review the project goals:

5.3.1 VISA Goals

The VISA project is developing a software toolkit suitable for the analysis of multi-dimensional spike train data recordings. The toolkit will provide software capable of deriving the functional connectivity of the recorded neural network. This project will be the third iteration of the software development cycle for the project. The overall project goals can be summarised as:

1. To produce an open source toolkit that allows researchers to visualise and explore multi-dimensional spike train recordings.

2. Assist researchers in mapping the functional connectivity of the recorded neural network.
3. Provide the above in an operating system agnostic toolkit.

The previous development cycles produced the Neurigma toolkit which in principle met the goals above. However the third development cycle addressed a number of deficiencies in the toolkit.

The problems identified for the Neurigma toolkit were:

1. Data processing to identify patterns in the recorded spike train data was highly structured. Neuroscience however has not identified a single set of data analysis algorithms. Researchers often wish to develop new and novel algorithms for spike identification, sorting and visualisation. Neurigma while an excellent tool was not flexible enough to allow user introduced algorithms or to change the sequence of analysis.
2. Neurigma failed to exploit the full computational power of the computers using it. Written as a classic desktop application and, despite the considerable computational workload of spike train analysis, it only employed a single compute core for calculations. The modern research machine has at a minimum two compute cores and far more likely four or more cores. This practically limited the toolkit to small datasets where the computational workload could be handled by a single core in a reasonable time.
3. Over time researchers ability to simultaneously record multiple spike trains has grown considerably. When originally conceived the toolkit was expected to deal with at most a few hundred spike trains. Modern recording equipment can now effectively record thousands of spike trains necessitating a software tool capable of analysing them.
4. While the visualisations iRaster and iGrid in the original toolkit have proven effective at deriving the recorded neural networks structure they do not scale well to large networks. As networks size grows the researcher begins to suffer from 'data deluge'. It becomes impossible to identify patterns in the data or make meaningful data filtering decisions due to the amount of data and the resulting cognitive load on the researcher. Finally with large datasets the limited on screen display space can no longer effectively present the dataset for researcher exploration.

5.3.2 VISA3 Development Goals

For the third iteration of the VISA toolkit the drawbacks of the Neurigma software were targeted with the aim of creating a toolkit that allowed users to:

1. Develop their own algorithms in a mainstream programming language and introduce them to the software. This would be achieved by introducing a VPL that allowed researchers to create modules and connect them into a data processing pipeline.
2. Made full use of the available compute power of the typical researcher's lab computer system. Exploiting parallelism wherever possible to speed compute intensive operations such as the generation of cross-correlograms for a spike train dataset.
3. Operated on significantly larger datasets both in terms of:
 - a. Number of simultaneously recorded spike trains.

- b. Increased recording length in terms of total time.
4. Addressed the difficulties of scaling the existing visualisations to cope with both the greater number of spike trains and the longer duration recordings.
5. Finally the data processing pipeline would be adaptable beyond neuroscience so that, in principle, any data could be analysed. Possibilities would include, but are not limited to, financial, economic and other science fields.

5.3.3 Neuroscience Visualisations

The Neurigma toolkit provided two visualisations for the analysis of neural spike trains and discovery of the connection architecture in the network. iRaster is an interactive raster chart intended to provide a time series plot of spiking events. By sorting and re-ordering the individual spike trains it is possible to identify visually recurring patterns in the data. iGrid utilised the formal method of pairwise cross correlation to identify connectivity within the neural network. These visualisations form the ultimate goal for the data processing pipeline and the end point for the data pipelines. The final implementation of each will be examined in its own chapter but to understand the design choices made and challenges faced an overview of each is presented here.

5.3.3.1 iRaster interactive spike train raster plot

This is the most basic visualisation where spike events are displayed as point events along a timeline. Mathematical algorithms, such as inter-spike interval and burst sorting may be applied to identify recurring patterns in the dataset. Figure 5-4 gives an example of a burst sort where twenty spike trains have been plotted. In Figure 5-3 the raw data is presented with no attempt to analyse it. In Figure 5-4 the same raster plot has been re-ordered using a burst sort algorithm. This re-orders the spike trains based on when they first started to show spiking activity. The contrast between the two presentations of the same data is striking. A visual inspection of the raw unprocessed data reveals little structure within the data set. No attempt has been made to present the raw data in a way that exploits the viewer's visual processing abilities. The data seems, therefore, unrelated with each spike train being disconnected from all others.

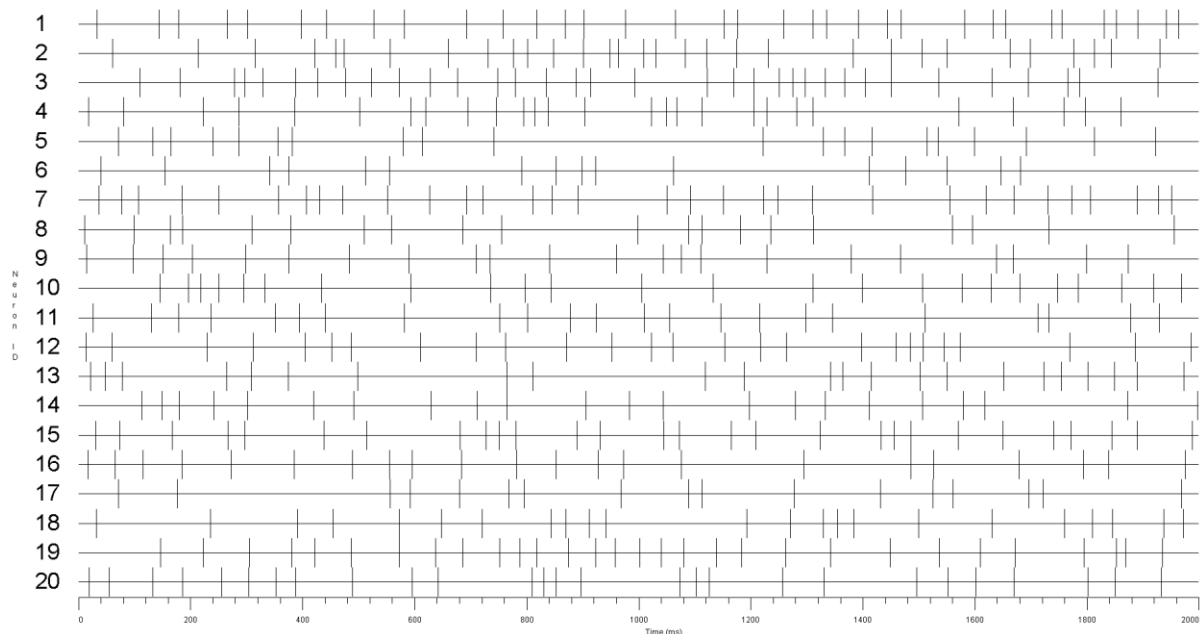


Figure 5-3: Raster plot of 20 spike trains recorded for 2000ms

The un-ordered raw data of Figure 5-3 has few distinguishing features. When visually inspected it seems to show no recurring patterns. There are clearly points where synchronous or near synchronous spike events occur but the patterns do not seem to repeat and could easily be simply chance artefacts from ordering the spike trains in a random order. Hence examination of the raw data does not seem to convey any useful information.

Applying a burst sort algorithm to re-order the spike trains starts to reveal the hidden structure within the data. This algorithm uses a sliding window of 300ms and re-orders the spike trains based on the earliest spike event in the window. The window must contain at least three spike events before the first spike time is used to re-order the spike train.

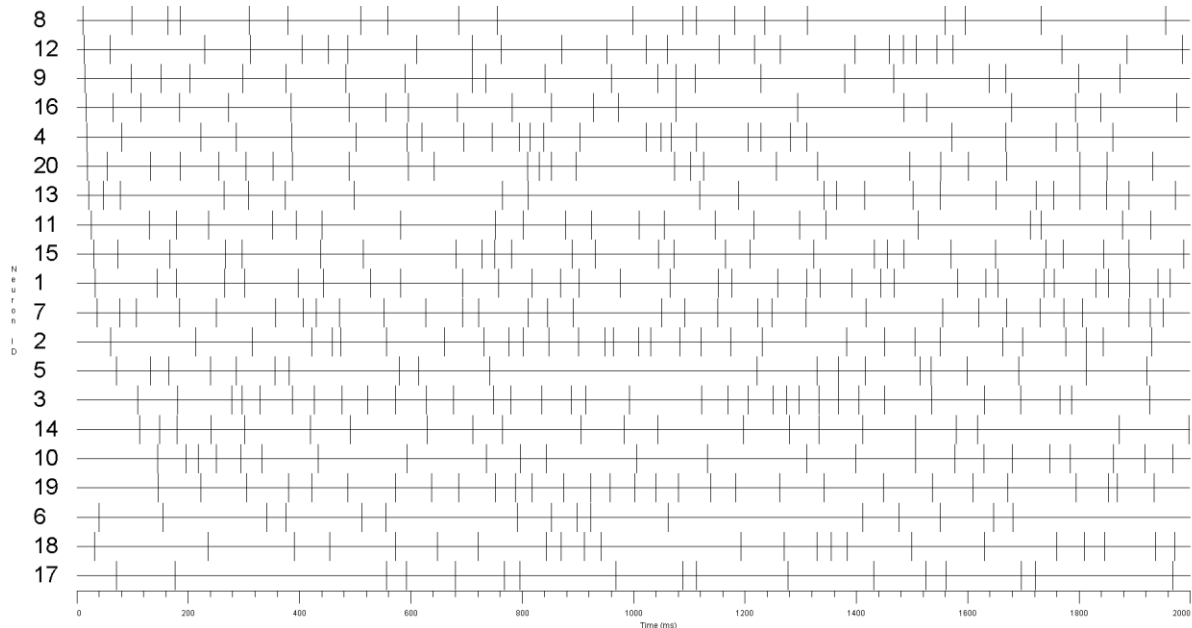


Figure 5-4: Raster plot re-ordered to identify when neurons started to spike (burst sort)

After the burst sort is applied more meaningful patterns start to appear in the data. Of the 20 recorded neurons it would appear that 11 of them started to spike very close together. This indicates that at a large cluster of connected neurons exist. Figure 5-4 shows these as neurons 8, 12, 9, 16, 4, 20, 13, 11, 15, 1 and 7. This is by no means a definitive proof of connection but provides us with a potential starting point. Of the remaining neurons 6, 18 and 17 are likely to be either disconnected or very loosely connected. The others may / may not be connected to the large cluster or form clusters of their own.

The burst sorting of the raw data has permitted rapid identification of *potentially* interconnected neurons by visual inspection. These potential neuron groupings will form the basis for the assessment of connectivity using the iGrid visualisation.

5.3.3.2 iGrid interactive visualisation of pairwise cross-correlation spike train data

The iGrid visualisation primary use is to confirm the connectivity between neurons. The technique of pairwise cross-correlation is used to assign a metric to the 'strength' of the relationship between two neurons. Each neuron pairing is plotted onto a grid with grey scaling being used to encode the strength of the relationship between the two neurons. Black denotes a perfect correlation, where the neurons fire together, and white representing no significant correlation. A sliding window and binning of spike events combined with the

statistical technique of Brillinger normalisation is used to assess correlation strength. The techniques will be examined when the implementation of the iGrid visualisation is reviewed. Figure 5-5 shows the result of plotting the 20 neuron dataset used to demonstrate iRaster. The strength of the correlation in the neurons firing pattern has been used to cluster the neurons into connected groups.

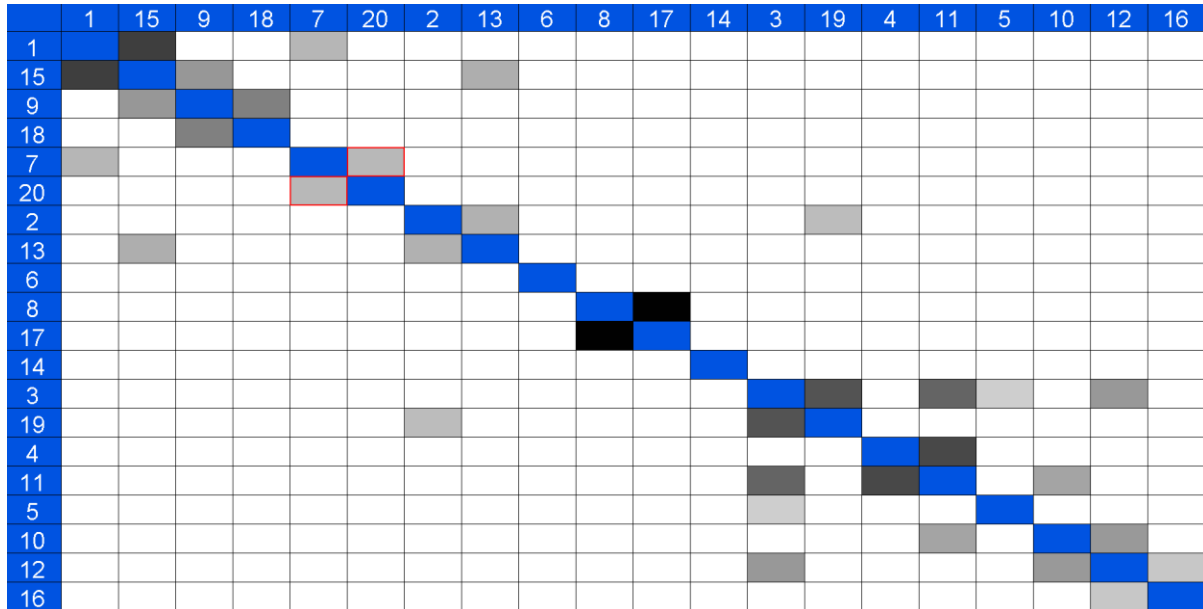


Figure 5-5: iGrid plot with neurons clustered using pair wise cross correlation. (Stuart, Walter & Borisyuk, 2003)

The iGrid plot allows more detailed investigation of the neural networks connectivity. If a cluster is observed on the grid which also appears on the raster plot it is possible to infer a connection between the neurons. Equally a strong correlation between spike trains, i.e. completely black iGrid cell, implies connectivity. A visual inspection of Figure 5-5 identifies several clusters; neurons 1, 15, 9 and 18 form a cluster while 7 and 20 seem to form their own group with a connection to neuron 1. Neurons 2 and 13 group together and have a connection to neuron 15. Neuron 6 and 14 do not seem to correlate with any group. Neurons 8 and 17 form an isolated cluster with no connectivity to other clusters in the network. Neurons 3 and 19 form a cluster with connections to neurons 5, 11 and 12. Similarly neurons 4 and 11 form a cluster with connections to neurons 3 and 10. This leaves neuron 16 which is involved in a connection with neuron 12.

The actual architecture of the neural network used in these visualisation examples is shown in Figure 5-6. As can be seen the structure of the network has been predicted almost exactly by the iGrid visualisation. Many of the potential groupings seen in iRaster have been proven and others resolved into more clearly defined clusters and connections.

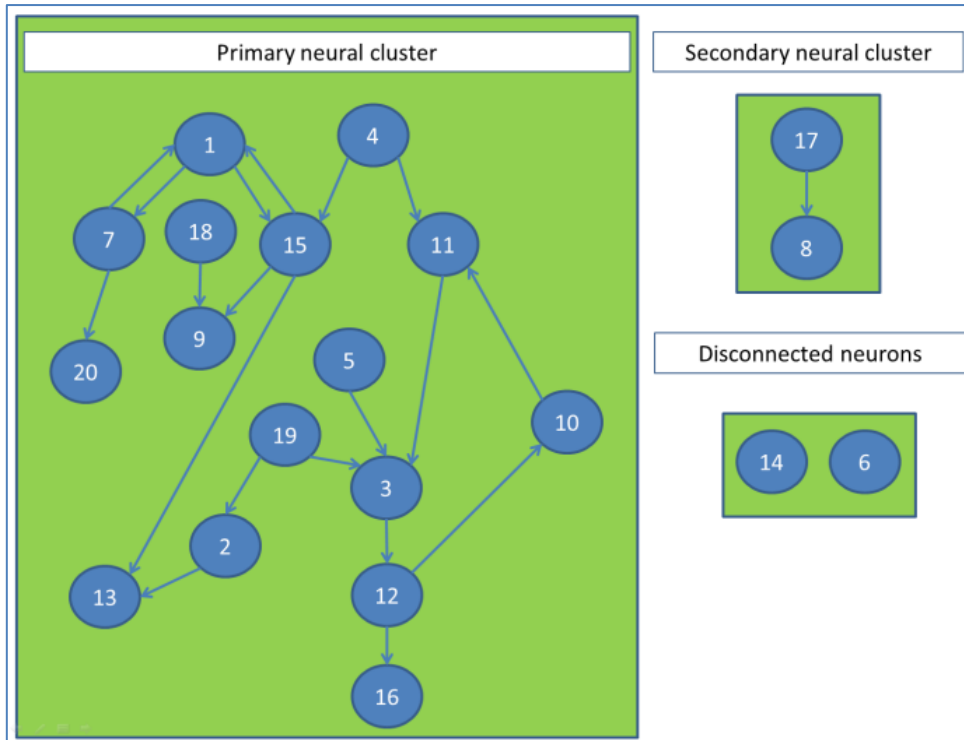


Figure 5-6: Actual simulated neural network structure used in examples.

Chapter 6

Research and Software Development Methodology

“methodology noun. The process used to collect information and data for the purpose of making business decisions.”

Summary

In this chapter the research and software development methodology is explained covering the data captured and the software development process employed.

6 Overview

This chapter examines both the research and software development methodologies applied to create i-Pipeline, the neural science analysis algorithms and data visualisations.

6.1 Research Methodology

The successful completion of a research project often rests on the selection of a suitable methodology on the part of the researcher. Research methodology is broadly categorised into two strategies; Theoretical research and Empirical research. The empirical strategy is usually further sub divided into two approaches; Positivist (or quantitative) research approach or Interpretivist (or qualitative approach) (Remenyi & Money, 2012).

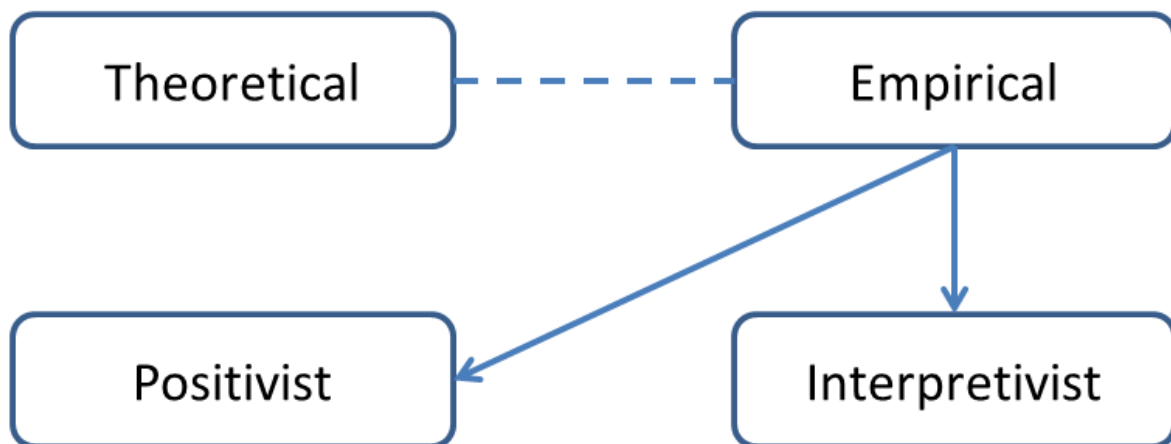


Figure 6-1: Research Strategies (Remenyi & Money, 2012)

Each of these approaches will be briefly examined and the reason for selecting an empirical positivist research approach explained.

6.1.1 Theoretical Research

Theoretical research “draws on ideas and concepts which represents the cumulative body of previous research and through a process of reflection and discourse develops, extends or in some other way qualifies the previous work” (Remenyi & Money, 2012). This type of research is seen primarily as a cerebral activity that can be considered a modern equivalent of “Rationalism”. Rationalists regard reason as the chief source and test of knowledge. A rationalist asserts that (Blanshard, 2016):

- reality itself has an inherently logical structure
- a class of truths exists that the intellect can grasp directly
- certain rational principles exist in logic, mathematics, ethics, and metaphysics that are so fundamentally true that denying them causes one to fall into contradiction

There is generally no “right” or “wrong” way to conduct theoretical research and it can be done almost anywhere. Albert Einstein was one of the 20th century’s greatest theoretical scientists who when asked to show someone his laboratory took out his fountain pen and said “There it is!” (Remenyi & Money, 2012).

6.1.2 Empirical Research

Empirical research “draws on observation of primary evidence in order to understand a phenomenon being studied. This evidence can be quantitative or qualitative” (Remenyi & Money, 2012). The Encyclopaedia Britannica gives the definition of empiricism as “the view that all concepts originate in experience, that all concepts are about or applicable to things that can be experienced, or that all rationally acceptable beliefs or propositions are justifiable or knowable only through experience” (Duignan, 2016). This research approach might be considered to be the “classic” research process (sometimes called the “Scientific method”) illustrated in Figure 6-2 below:

The Scientific Method as an Ongoing Process

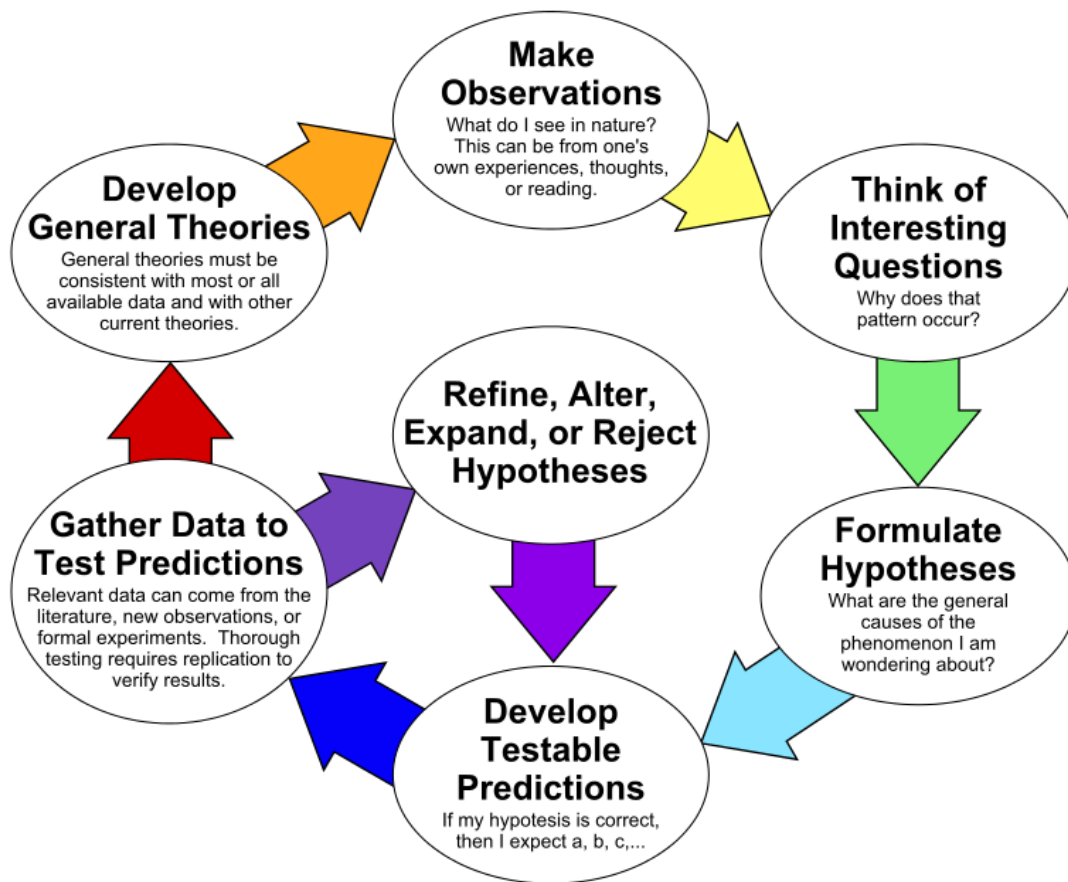


Figure 6-2: The "scientific method" or classic research process (ArchonMagnus, 2015).

Whilst an understanding of theoretical concepts and issues is a prerequisite of empirical research the foundation of the research must be the data the researcher gathers. Hence the empirical researcher might say “It is a capital mistake to theorise before one has data”. Empirical research is, therefore, grounded in the observations (aka experience) of the researcher. Those observations form the data for the research.

As can be seen in Figure 6-1 the conduct of empirical research is usually further subdivided into positivist (quantitative) or interpretivist (qualitative) research.

6.1.2.1 *Positivist / Quantitative Research*

The essence of positivist (quantitative) research is that it relies on numbers (Remenyi & Money, 2012). Hence information derived from sensory experience, interpreted through reason and logic, forms the exclusive source of all authoritative knowledge. Verified data received from the senses is known as empirical evidence (Macionis & Gerber, 2010). Research done in the positivist style will therefore rely on numbers and on the researcher's skill in mathematics or statistical analysis. Historically, this has been the approach adopted in the natural sciences.

6.1.2.2 *Interpretivist / Qualitative Research*

Interpretivist or qualitative research aims to understand and interpret interactions whereas quantitative research tests hypotheses (Johnson & Christensen, 2010). Qualitative research methods examine the “why” and “how” of the subject under study in addition to the “what”, “where”, “when” and “who”. Qualitative methods produce information on the particular case under study. Where more general conclusions are drawn (propositions) quantitative methods can be used to seek empirical support for such research hypotheses. This approach is often employed in the social sciences to gather an in-depth understanding of human behaviour and the reasons for that behaviour.

6.1.3 *Selected Research Methodology – Empirical Positivist*

This project selected an empirical positivist approach to the development of i-Pipeline, the neuroscience analysis library and the i-Raster, i-Grid and i-Animate visualisations. To understand why this research methodology was adopted it was necessary to examine the hypothesis being tested and the research question derived to test it.

6.1.3.1 *Hypothesis*

The original application of visual analytics to the problem of mapping a neural network's functional connectivity produced the original i-Raster and i-Grid visualisations (Somerville et al., 2011; Stuart, Walter & Borisjuk, 2005). The usefulness of these visualisations was being limited because of the following factors:

- In the field of neuroscience:
 - Technological advancement has seen the number of simultaneously recordable spike trains move from the hundreds into the thousands. The i-Raster and i-Grid visualisations as implemented cannot scale to datasets of this size.
- In the field of visual analytics:
 - The i-Raster / i-Grid visualisations can be scaled to manage large neuroscience datasets if the emerging “big data” visualisation techniques are incorporated into them. These include:
 - The addition of tools that summarise data while offering the ability to “drill down” to more detailed data.
 - The addition of tools to filter data within the visualisations.
 - Creating interactive visualisations of big data on the average research computer require software that exploits the full compute power of the computer hardware.
- In the field of software engineering
 - There has been a paradigm shift in how the modern computer delivers increased compute power. Historically, this has been achieved by

replacing the computers CPU with a faster version every 18 months. Hardware vendors are now delivering increased compute power through the provision of multiple CPU's.

- Access to the full compute power of modern computers requires software to be written in a way that permits the parallel execution of code on multiple CPU's.
- Software engineering issues and the need to access multiple compute cores (CPU's) was not considered in the original implementation of i-Raster and i-Grid. This prevents the original versions from scaling to use "big data" visual analytics techniques while remaining interactive.

6.1.3.2 The Research Question

To test the hypothesis above, the following research question was formulated:

"How can Software Engineering and Visual Analytics be applied to aid the general analysis of scientific data and specifically current neural spike train data?"

This question challenges the researcher to improve upon the work of Somerville and Stuart by introducing more recent developments in visual analytics to their original visualisations. Previous work was limited to the fields of neuroscience and visual analytics, but the recent developments in visual analytics rely heavily on the field of software engineering. Hence, if interactive "big data" visualisation is to be achieved software must be re-engineered in light of the paradigm shift in the delivery of compute power.

The joining of the software engineering and visual analytics fields is, of course, not limited to producing visualisations for neuroscience. Hence the research question also required that the techniques developed in the project should be applicable to "the general analysis of scientific data" even if they are, in principle, demonstrated using neuroscience data.

6.1.3.3 Selection of research methodology

Having reviewed the underlying hypothesis and the projects research question the research methodology of empirical positivist was selected. Empirical research is grounded in observation (experience) and the key field of visualisation / visual analytics draws most of its effectiveness from how a user "experiences" an interactive visualisation. The positivist approach argues that "information derived from sensory experience, interpreted through reason and logic, forms the exclusive source of all authoritative knowledge. Verified data received from the senses is known as empirical evidence" (Macionis & Gerber, 2010). This is a reasonable definition for visualisation where knowledge is created through the combination of the researcher's senses with reason and logic. The verified data for the project is of course the simultaneously recorded spike trains. This data was either recorded from live animal experiments or generated from simulated neural networks. The live animal simultaneously recorded spike train data was provided by researchers at the Newcastle University Institute of Neuroscience (Sernagor, 2016). Simulated simultaneously recorded spike train data was generated using Plymouth Universities Neural Network Simulator and Neural Network Creator (Borisjuk, 2002; Borisjuk, 2008).

The success or failure of the project in addressing the research question was determined empirically by measuring:

- i. The number of spike trains visualised
- ii. The number of data points processed and the rate of processing.
- iii. The number of compute cores utilised.
- iv. The time required to complete computationally expensive operations (such as cross correlation).
- v. The ability of the software to operate in multiple environments (laptop, desktop computer and high performance compute cluster (HPC)).

6.2 Project goals / aims

The first stage was the identification of the project's key goals from the research question. These can be summarised as:

- a. Develop a data processing system with the following properties:
 - i. The system must accept any data for processing regardless of its storage format.
 - ii. Provides a means for a researcher to define data processing algorithms and incorporate them into the data processing system.
 - iii. Provide a means for the researcher to define a data “pre-processing” pipeline using the algorithms they have created.
 - iv. Provide a means to rapidly add, re-order execution of and remove data processing algorithms from the pipeline.
 - v. To deliver the pre-processed data to an independent “visualisation program”. This program should run independently of the data processing system.
- b. Develop a set of data pre-processing algorithms appropriate to the analysis of neural spike train data. These algorithms were used to demonstrate the operation of the data processing system as an analysis tool. Broadly the algorithms were categorised as:
 - i. Input algorithms that accept a selection of commonly used neural spike train data formats.
 - ii. Analysis algorithms that process the spike train data with the goal of deriving the functional connectivity of the neural network that generated the spike train data.
 - iii. Output / Visualisation algorithms that deliver the processed data to the independent visualisation programs.
- c. Develop a set of independent visualisation programs that facilitate the visual exploration of the dataset. Three visualisations were selected for development and expanded over previous implementations to fully exploit the computing power of the modern research computer.
 - i. iRaster – based on the work of Somerville (Somerville *et al.*, 2011) this visualisation provides the researcher with a detailed view of neural spike train data at the level of individual spike events.
 - ii. iGrid – also based on the work of Stuart (Stuart, Walter & Borisyuk, 2005) this visualisation permits the researcher to infer functional connectivity between neurons from the spike train recording.
 - iii. iAnimate – a new visualisation of the multi-electrode array (MEA) that recorded the spike train data. Animation of the recorded spike trains

spiking activity over time, allowing the researcher to visual identify patterns within and across the neural network.

In addition, the following supporting goals were set for the project software:

1. The software should be cross platform and deployable to any research computer running a modern operating system.
2. Multi-threaded programming should be used throughout to exploit fully the compute power of modern computer systems.
3. The most computationally intensive algorithm (cross-correlation analysis) should have the option to run in a cluster computer environment or on the researcher's machine (with an extended run time) without the need to modify the code.
4. Visualisation techniques should be applied throughout the project to enhance the user's experience. Specific attention was paid to ensuring responsiveness from the software despite the large data processing demands.

6.3 Software Development Methodology

The uncertain and unpredictable nature of the challenges that this project generated required an iterative software development approach. In addition, the project carried significant risks that would have to be addressed over the product development. Specifically:

1. Large processing demands would be placed on hardware without knowing exactly what hardware would be available at runtime.
2. Responsiveness of the application could be threatened by the size and quantity of the data to be held in memory.
3. The availability of a cluster computing facility could not be guaranteed.

These constraints required the selection of a software development methodology that supports both iterative development and allows the effective management of risks. Several possible development methodologies were considered and ultimately the Spiral Model (Boehm, 1986) was selected. This model of development emphasises the production of prototypes and the management of risk as part of the iterative development cycle. Figure 6-3 illustrates the structure of a spiral development process:

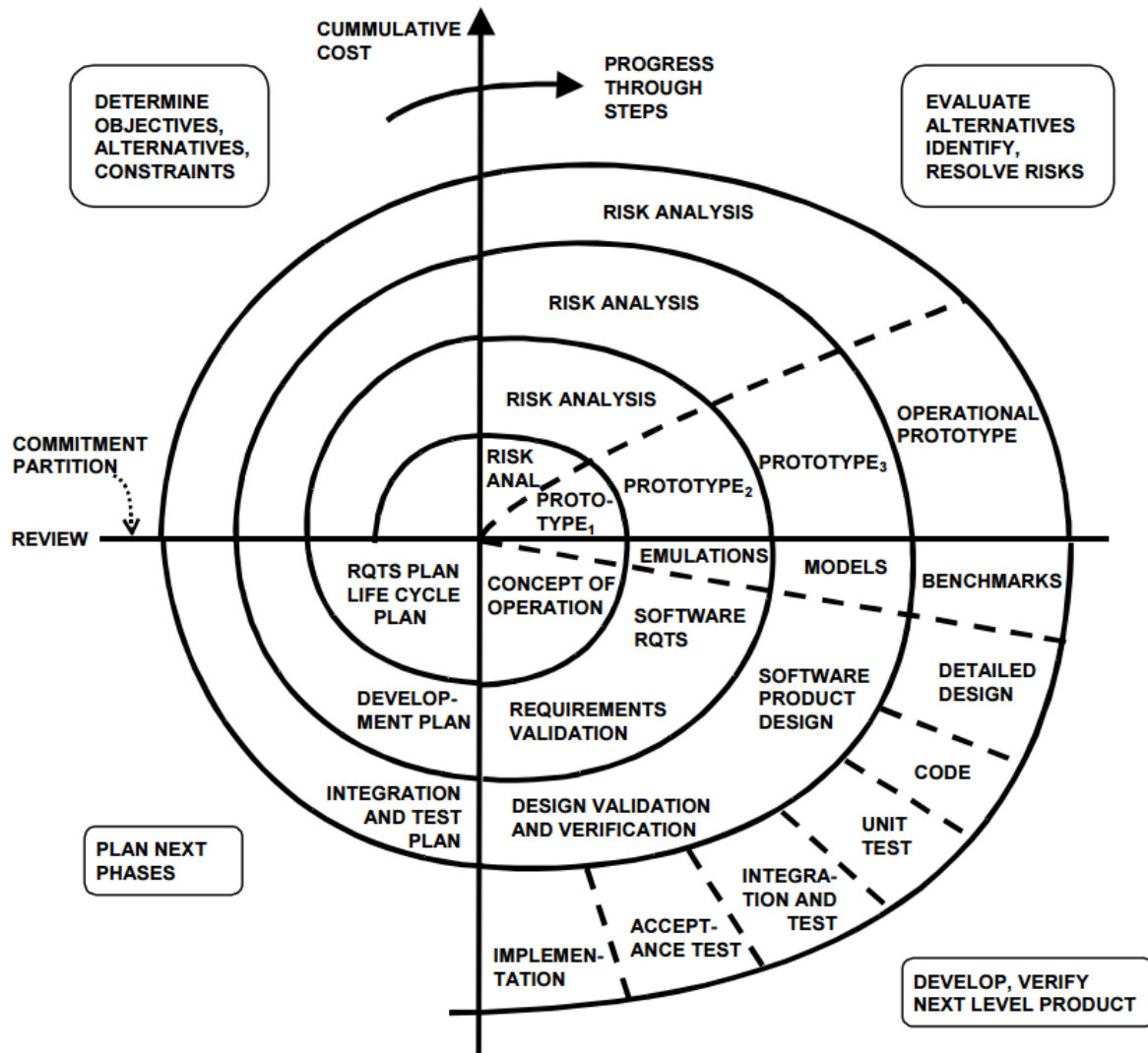


Figure 6-3: Boehm's Spiral Development Model (Boehm, 2000).

6.4 The Spiral Development Methodology

Boehm defines the primary characteristics of the spiral development method as follows (Boehm, 2000):

- a **risk-driven process model generator**
- a **cyclic** approach for incrementally growing a system
- a set of **anchor point milestones** for ensuring stakeholder commitment

6.4.1 Risk driven process model generator

Boehm defines **risk** as “situations or possible events that can cause a project to fail to meet its goals”. This project had several major risk factors arising from its multi-disciplinary nature. Examples would include:

- Several of the desired project goals were in mutual conflict with each other, such as implementing visually responsive applications (Visual Analytics) vs cross-platform compatibility (Software Engineering).
- As initially conceived the project would have utilised the CARMEN Virtual Laboratory (VL) for neurophysiology (Gibson et al., 2008; Weeks, 2010) to

provide the required cluster computing hardware but this was not under the researcher’s control.

- It was expected that the implemented features would evolve over time as each development cycle would provide greater clarity on what was physically possible for the hardware to accomplish.

It was therefore considered vital that a software development methodology should be employed that allowed risk to be evaluated and addressed at each stage of the development. The spiral model directly links risk to the generation of its “process model” and is, therefore, particularly well suited to the development of projects with a large number of (initially) unknown risk factors. A **process model** should answer two main questions:

1. What should be done next?
2. For how long should it continue?

In the spiral model, the answers to these questions are driven by risk considerations. Ultimately, the answers to these questions will define the work and the amount of time dedicated to the next **cycle** of development.

6.4.2 Cyclic approach for incrementally growing a system

In software development it is common to time box or time limit each development cycle. It is also common to require that each cycle has a distinct identifiable output. Software development methodologies that employ this approach are generally described as “incremental” methodologies. The spiral model’s development cycles produce incremental product prototypes. Each cycle builds on the prototype from the previous cycle with the completed software product “evolving” with each cycle (see Figure 6-3).

In the case of this project the development of functional prototypes proved critical to identify areas where further refinement was needed. The initial research plan called for the following prototypes:

Prototype	Description	Chapter
Visual programming language (core i-Pipeline / VPL) implementation.	Develops the VPL directed graph, base classes for different types of graph nodes and the i-Pipeline virtual desktop	See: Chapter 5 for Visual programming Languages (VPL’s). See: Chapter 7 for i-Pipelines design.
Development of neuroscience data model	Develops an efficient data storage system with support for rapid filtering and sorting operations	See: Chapter 8 for the design and development of the neuroscience data model.
Development of initial neuroscience processing algorithms	Development of the burst sort and inter spike interval sorting algorithms for i-Raster	See: Chapters 9, 10 and 11 for processing algorithms and their use.
Integration of data model and neuroscience algorithms with i-Pipeline	Algorithms are “wrapped” into graph nodes and the i-Pipeline Toolbox implemented	See: Chapters 7, 8 appendix 1
Development of the parallel execution engine	A data-availability-driven approach to the VPL’s execution is implemented on top of a structure based dataflow model	See: Chapter 5 (Section 5.2.2) – Dataflow execution models

Prototype	Description	Chapter
Development of the i-Raster visualisation	Implementation of a revised i-Raster visualisation incorporating the new overview and filtering features	See: Chapter 9
Development of a pairwise cross correlation and clustering algorithm to run over the CARMEN compute cluster	Developed and deployed a CARMEN service for pairwise cross correlation and clustering.	See: Chapter 10 (Section 10.1.1) – Computational challenge of Pairwise Cross Correlation
Development of i-Grid visualisation	Implementation of the i-Grid visualisation with a new dendrogram overview using the clustering algorithm.	See: Chapter 10
Development of i-Animate	Implementation of i-Animate to visualise recording MEA array with heat map overlays.	See: Chapter 11

Table 6-1: Research prototypes, work undertaken to produce them and associated thesis chapter(s).

The primary risk with the prototype plan as set out in Table 6-1 was the reliance on an external third party (the CARMEN Virtual Laboratory (Gibson *et al.*, 2008)) to provide the compute cluster for the most computationally intensive data analysis. Ultimately this risk was realised with the provided hardware and access restrictions preventing the CARMEN compute cluster from delivering the needed computing power. The fall back plan of using the Plymouth University high performance computing cluster was adopted and the pairwise cross correlation and clustering algorithms were implemented using this hardware.

6.4.3 Key research outputs

In this project a framework for data processing and analysis has been developed modelled on the VPL concept. This framework will allow a developer to provide a library of pre-coded algorithms which are represented as graphical nodes. Each node is deployable to a desktop environment and can be linked with other nodes to form a visual data processing pipeline. The pipeline will follow the dataflow execution model with each algorithm executing concurrently. The final output is delivered to a visualisation module that acts as the terminus for the pipeline. The visualisation module is an independently executing program which applies the principles of information visualisation and allows an interactive exploration of the processed data. The visualisation may also present options for further processing.

The created VPL, named iPipeline is a general framework into which domain experts may place a library of analysis algorithms. By exchanging libraries the VPL is customisable for data analysis in any problem domain. For the purpose of development a library for the analysis of neuroscience data has been developed. The neuroscience problem addressed was the discovery of a neural network's architecture from the analysis of simultaneously recorded neural spike train data. This problem was selected as it is a computationally challenging "big data" problem that can be solved efficiently through combining the disciplines of neuroscience, software engineering and information visualisation.

Chapter 7

Designing iPipeline & the neuroscience visual programming language

“pipeline noun. In computing, a pipeline is a set of data processing elements connected in series, where the output of one element is the input of the next one.”

Summary

This chapter reviews the design of the iPipeline data processing environment including both the technical design and the visualisation choices made.

7 Overview – iPipeline

The iPipeline workflow environment provides the framework for the visual data processing pipeline at the heart of the software implementation. It is designed as two ‘layers’:

1. A ‘thin’ framework layer which handles the creation of the workflow’s directed graph and its execution.
2. A ‘thick’ problem domain layer that provides:
 - a. The problem domain specific data model and
 - b. The problem domain specific algorithms that manipulates and visualises the data model.

Describing and studying a problem using different “layers of abstraction” is a common technique in computer science (Floridi & Sanders, 2004).

Figure 7-1 provides an overview of iPipeline’s layered structure and its individual components. This is followed by the definition of several key terms that will be used throughout this chapter.

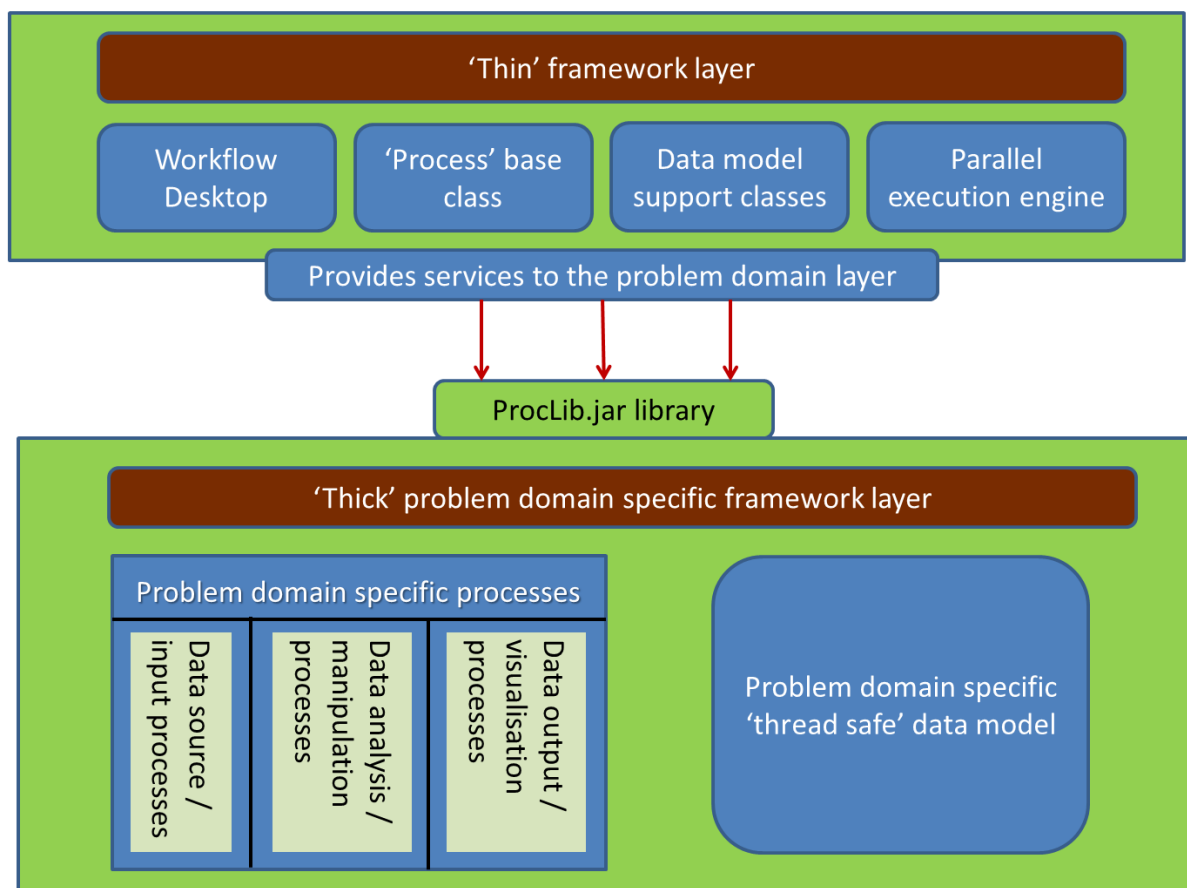


Figure 7-1: Structure of the iPipeline workflow framework

7.1 Definition of terms

This section provides the exact definition for a number of technical terms that will be used throughout this chapter.

Term	Definition
Problem domain developer	The external third party developer who provides the implementation of the ‘thick’ problem domain specific layer.
Workflow	A directed graph of connected ‘process nodes’ that define a sequence of processing that usually ends with the creation of a visualisation.
Workflow Desktop	The virtual space in which the user creates the workflow using process nodes drawn from the toolbox.
Toolbox	A floating toolbox visible on the workflow desktop that contains the problem domain specific processes loaded from the ‘thick’ framework layer.
Process or process node	A process / process node encapsulates an algorithm allowing its insertion into a workflow. The information processing cycle can be used to classify a process as a: <ol style="list-style-type: none"> 1. A data source / input process or 2. A data analysis / manipulation process or 3. A data output / visualisation process ‘Base processes’ representing each of the above steps have been created and the problem domain developer should extend these classes to implement a process algorithm
Process algorithm	The algorithm encapsulated inside a process . The algorithm will implement some part of the information processing cycle and can be categorised as either: <ol style="list-style-type: none"> 1. An input algorithm 2. A data processing algorithm 3. An output algorithm
Process glyph	The graphic used to represent a process on the workflow desktop.
Process settings panel	A graphical user interface component that permits user configuration of the problem domain algorithm encapsulated in a process.
Data model support classes	The development of a ‘thick’ problem domain layer is supported by providing access to a collection of pre-written classes.
Thin framework layer	Collectively refers to all pre-developed iPipeline components. This layer provides: <ol style="list-style-type: none"> 1. The ability to visually create a directed graph. 2. The base definition for a ‘process’ in the directed graph. 3. The data model base process classes. 4. The parallel execution engine that executes the user defined workflow to produce a visualisation.
Thick problem domain specific framework layer (or the thick framework layer)	Created by the problem domain developer this layer has two primary tasks: <ol style="list-style-type: none"> 1. provide a set of process nodes specific to the field being studied 2. provide a ‘thread safe’ data model to represent the data being processed
ProCLib.jar	The name for the Java jar library developed by the problem domain developer. It contains all classes needed to implement the “Thick problem domain specific framework layer”. Changing to another problem domain simply requires swapping the ProCLib.jar file.

Table 7-1: Definition of terms used in this chapter

7.2 The ‘thin’ framework layer

This layer is the core of the iPipeline implementation. It provides all of the functionality needed to make visual programming using a pipeline of processes a reality. The four key components of this layer can be seen in Figure 7-1 and each component will be reviewed in turn.

7.2.1 Workflow Desktop

The user interacts with the application via the workflow desktop component. It is here that the user creates their workflows using the domain specific processes provided by the problem domain developer. Figure 7-2 illustrates the desktop with its key features identified.

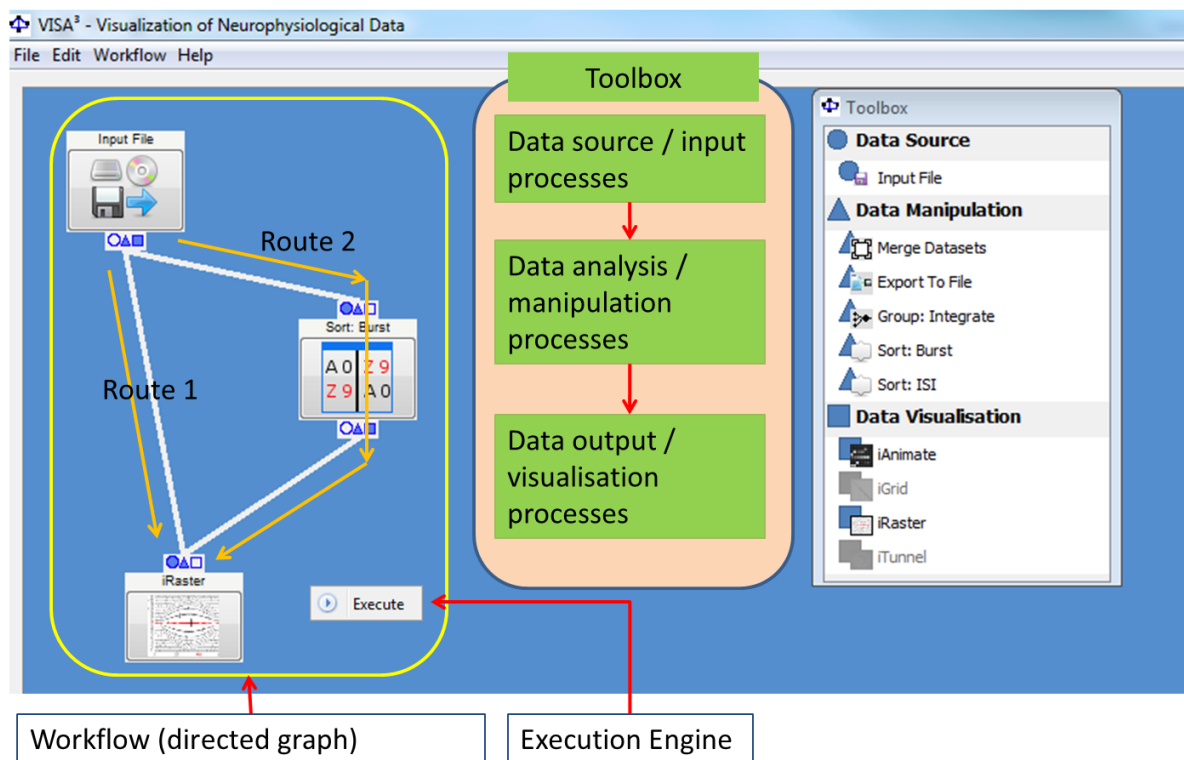


Figure 7-2: iPipeline’s workflow desktop component

The desktop can expand infinitely in any direction to accommodate any sized workflow or any number of workflows. The single most important part of the workflow desktop is the toolbox. It is through the toolbox that the user is provided with access to the process nodes from which they will build a workflow. The contents of the toolbox will change depending on the specific problem domain being studied. It is through the toolbox that iPipeline’s ‘thin’ framework layer connects to the ‘thick’ framework layer. The content of the toolbox is defined in the ‘thick’ framework layer. The toolbox is divided into three sections mirroring the three phases of the information processing cycle (see Chapter 5). The toolbox divisions, their mapping to the information processing cycle and a description of each division is given in Table 7-2.

Toolbox division	Information processing cycle phase	Description
Data Source	Input Phase	Processes encapsulating algorithms that provide data for processing.
Data Manipulation	Processing Phase	Processes encapsulating algorithms that manipulate the data in the pipeline.
Data Visualisation	Output Phase	Processes encapsulating algorithms that visualise the data in the pipeline.

Table 7-2: Divisions of the toolbox and their relationship to the information processing cycle

To construct a workflow the user places nodes from the toolbox onto the desktop surface and connects them together. At a minimum a workflow must be composed of one data source process and one data visualisation process. Such a workflow will simply create a visualisation that displays the data loaded into the pipeline. More commonly one (or more) data manipulation processes will be inserted between the data source and data visualisation processes. The definition of a **valid workflow** can therefore be given as:

1. A workflow that contains a **route** through the workflow's directed graph which starts at a data source process and ends at a data visualisation process and
2. Optionally the route may contain any number of data manipulation processes.
3. All processes must have correctly configured settings panels.

A valid workflow must have at least one (1) route through its directed graph that meets the conditions above but may have more. Figure 7-2 has two (2) such routes. Figure 7-2 shows both the minimal case (route 1) where a data source and data visualisation process is directly connected and the optional case where a data manipulation process has been inserted (route 2). Each route can be executed to produce a set of data that is

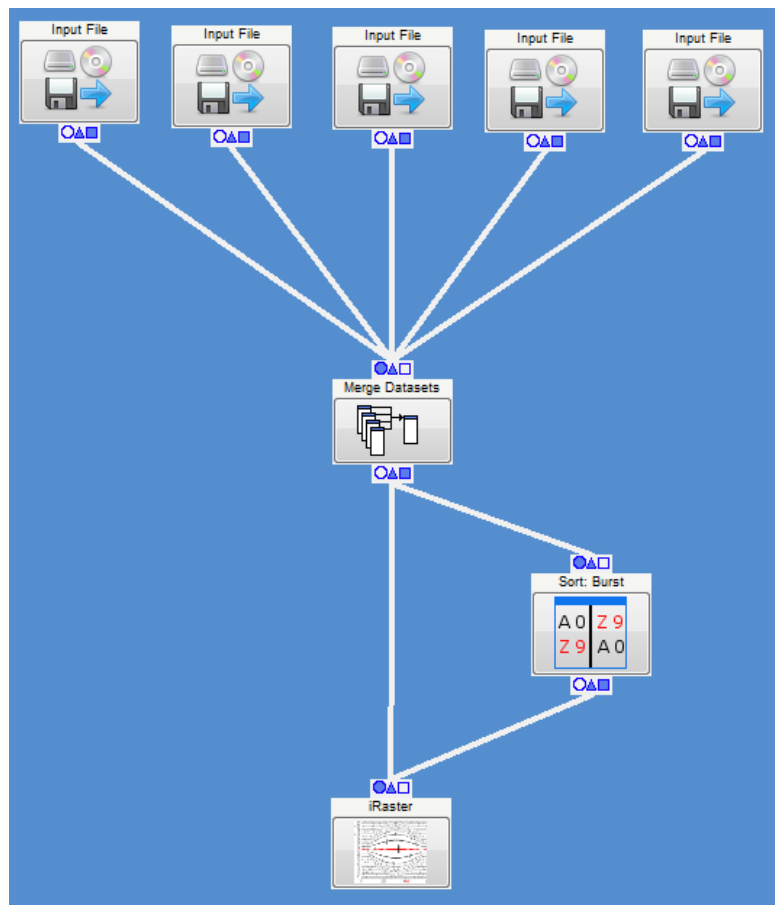


Figure 7-3: Workflow to merge five data files into a single data source.

delivered to the data visualisation process at the end of the route. Since Figure 7-2 has two routes through its directed graph it will deliver two sets of data to the iRaster visualisation at the end of the workflow. These will be:

1. The raw data read from a data file by the input file process at the start of the workflow (route 1 in Figure 7-2).
2. A sorted set of data created by the optional data manipulation process (route 2 in Figure 7-2).

The power of visual programming becomes apparent when the workflow becomes more complex. Figure 7-2 dealt with a simple case where a single file of data is loaded into the pipeline. Figure 7-3 extends the workflow to load five (5) files of data. This situation might arise where a researcher wishes to combine simultaneous recordings from five different sensors into a single set of data. The problem domain developer has provided a data manipulation process to merge multiple datasets into a single dataset. Modifying the workflow only takes moments. A classical computer program would require changes to the source code and re-compilation to achieve the same result. This can realise considerable time savings as operations can be inserted into (or removed from) the workflow without having to re-code or recompile the application. The processes encapsulate computer code that is “written once; re-used anywhere”.

Figure 7-4 further extends the Figure 7-3 example by introducing a new process – the export to file process. The workflow in Figure 7-3 loads merges and sorts five sets of data. This processing takes time and it might be useful to create and permanently store a single file with this processing already completed. Figure 7-4 shows such a workflow that produces two files. The first

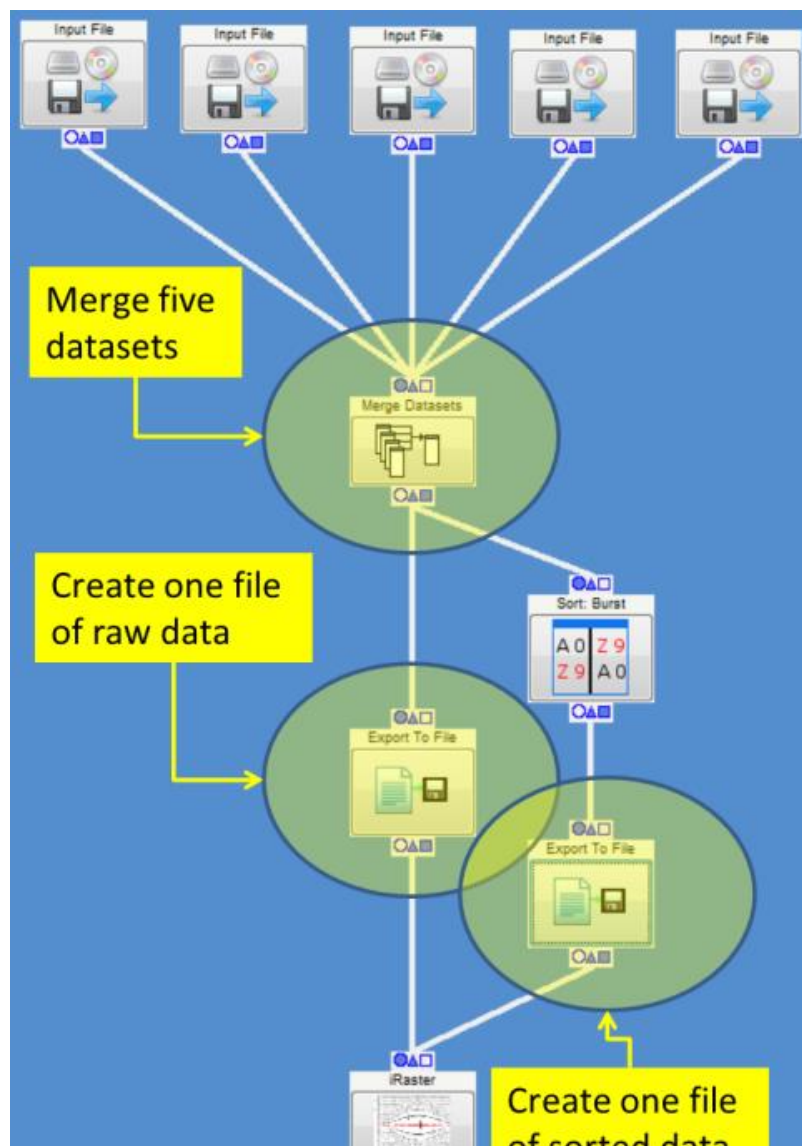


Figure 7-4: Workflow to merge five datasets. Two new files are created, the first of raw data and the second after sorting the raw data.

file stores the result of combining five sets of raw data. The second file stores the same data after it has been sorted. The benefits to the researcher can be summarised as:

1. The single file of raw data is easier to use and distribute than five files of raw data.
2. The result of merging the raw data files and sorting the result is preserved allowing these steps to be eliminated in future.
3. The workflow is naturally paralysable. Writing the merged raw data file to disk can occur *at the same time* as sorting the raw data as they appear on different routes through the workflow.

The workflow desktop provides a flexible visualisation of the “program” that the user is creating. At the same time it provides the ability to rapidly reconfigure that program and preserve the results.

7.2.2 Process base classes

The ‘thin’ framework layer must interact with the “domain specific visual programming language” built by the problem domain developer (The ‘thick’ layer). However the ‘thin’ framework has no knowledge either of the problem domain or the specific algorithms that have been implemented by the problem domain developer. Nevertheless this gap must be bridged if any “visual program” is to be created and executed. The process base classes (and the IProcess interface) provide the

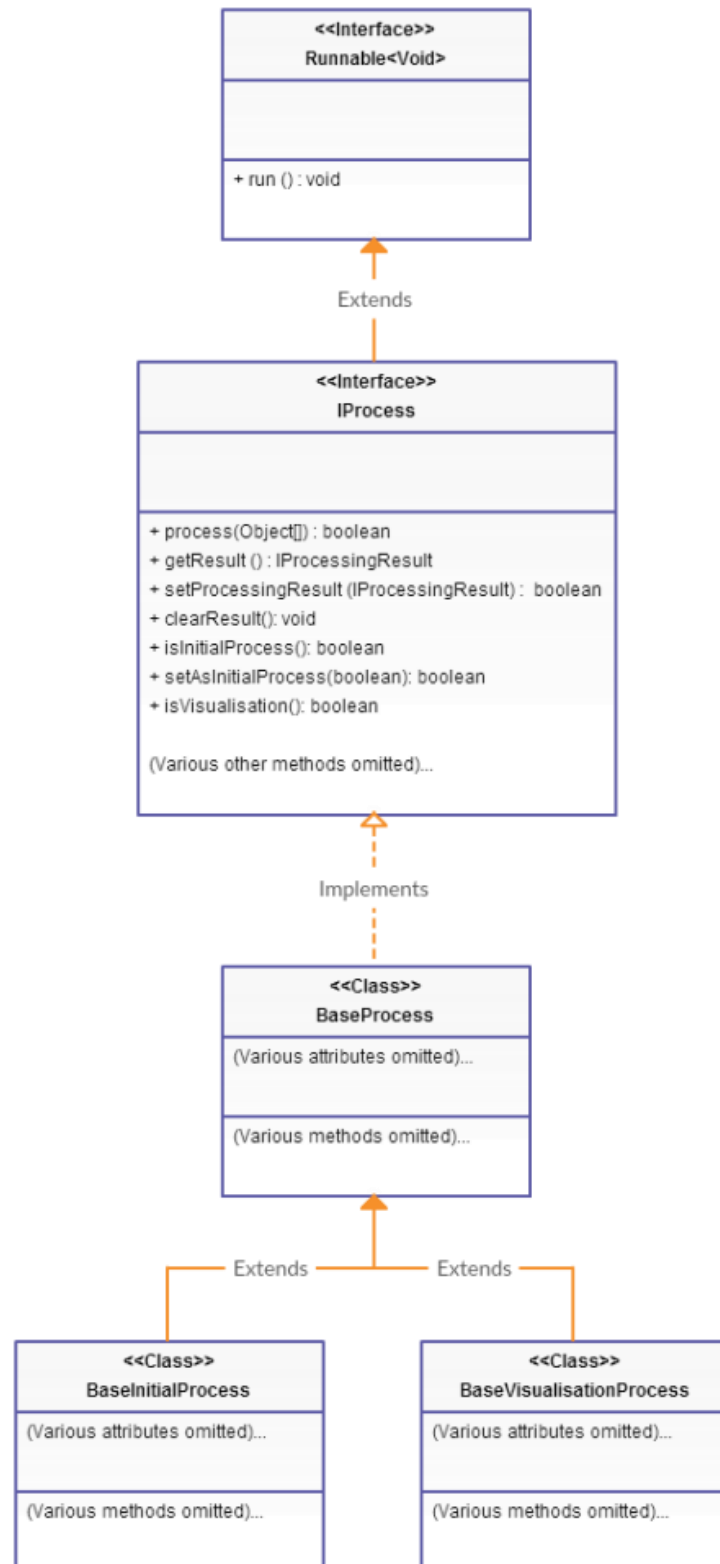


Figure 7-5: Thin layer base process class implementation structure.

Nevertheless this gap must be bridged if any “visual program” is to be created and executed. The process base classes (and the IProcess interface) provide the

means to bridge the gap (see Figure 7-5). The bridge is built on the idea that whatever workflow may have been created by the user it must take the form of a directed graph. As such a guarantee exists that the workflow takes the form of a collection of 'nodes' linked together with parent / child relationships. The 'thin' framework layer is therefore aware only of 'nodes' and 'links':

- Nodes represent '**something that is done to data**'.
- Links define the **order** in which the nodes '**do something to the data**' (a parent node must finish 'doing its thing to the data' before a child node starts to 'do its thing').

The above statements both contain an abstract "something". That 'something' is an algorithm that manipulates data. The "BaseProcess" class can therefore be considered an abstraction that represents an algorithm created by the problem domain developer. Hence any 'BaseProcess' may be a 'node' in the workflows directed graph. To make any algorithm interoperate with iPipeline it must first be "wrapped" in a "BaseProcess" class. Wrapper classes are a form of software design pattern formally identified by the 'gang of four' in their seminal book on software design patterns (Gamma et al., 1994). This class exposes a common interface that iPipeline uses to interact with the encapsulated algorithm. Within the iPipeline code base this interface is called 'IProcess' the UML diagram in Figure 7-5 shows the IProcess interface and its implementing classes.

The IProcess interface is itself derived from the Java languages Runnable interface. This means that each process is itself 'runnable' as an independent entity – essentially a mini-program. The runnable interface has however been expanded to add the functionality needed by a 'node' in a directed graph. The most basic function is to encapsulate the problem domain developer's algorithm (whatever it might be). The process(Object[]) : boolean method provides this facility. The problem domain developer will extend the base class (which implements IProcess) and overrides this method to implement their algorithm. The method is invoked by the parallel execution engine during execution of the workflow.

Every node in the workflow's directed graph must accept the result from previous parent nodes and produce a result of its own. The IProcess interface provides three methods to manage the production of results:

- getResult() : IProcessingResult
- setProcessingResult (IProcessingResult) : boolean
- clearResult() : void

These methods are used by the parallel execution engine to control the flow of processing results through the workflow. The interface IProcessingResult is the abstraction representing the structure produced by iPipelines structure based dataflow model (Davis & Keller, 1982; Dennis & Robinet, 1974; Keller & Yen, 1981)(see Chapter 5). This abstraction will be discussed further in the next section.

The final three methods of the IProcess interface reflect the information processing cycle. As discussed in chapter 5 this cycle is composed of three primary steps:

1. Input
2. Processing and

3. Output.

A 'base' class is required for each of these steps. These three base classes will form the link between the 'thin' framework layer that manages the workflow and the 'thick' framework layer which defines the algorithms to be executed. The BaseProcess class seen in Figure 7-5 provides an implementation of the IProcess interface sufficient to represent a 'general processing node' (step 2 of the information processing cycle). Step 1 and step 3 are essentially more specialised versions of the general processing node. Specialised sub-classes are provided to implement these steps. Step 1 is represented by the BaseInitialProcess class while step 3 is represented by the BaseVisualisationProcess class.

Taken together the three 'base' classes provide the problem domain developer with a means to wrap their algorithms into a 'thin' framework class that:

1. Can be represented as a node in a workflow's directed graph and
2. Be executed by the thin frameworks parallel execution engine.

In addition to the software implementation of a process each process also needs to be represented visually on the workflow desktop. The representation is known as the process glyph. The user will connect these process glyphs together to create the workflow. Figure 7-6 shows the visual elements of a process glyph.

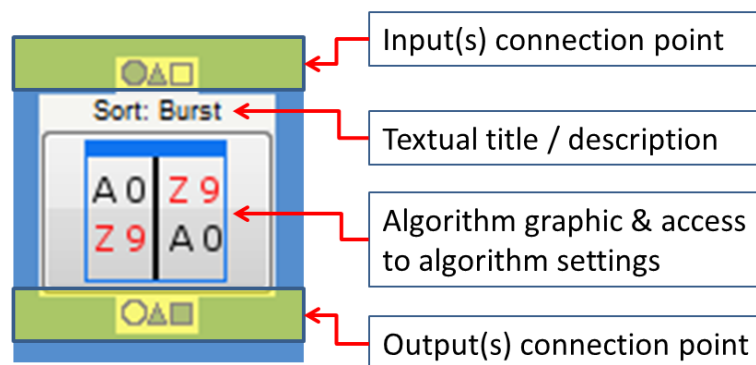


Figure 7-6: The visual representation (glyph) of a process

The process glyph has two primary tasks to perform. Firstly to identify the algorithm encapsulated by the process and secondly to define which other processes are allowed to connect to this process. The majority of the process glyph is given over to describing the algorithm encapsulated by the process. The BaseProcess class and its child classes allow both a textual description and an image to be associated with the encapsulated algorithm. At the heart of the glyph a button prominently displays the image with the textual description above it. The button provides access to the process settings panel created by the problem domain developer.

The remainder of the process glyph is an input and an output connection points. As described in chapter five iPipeline uses a structure based dataflow model. In this model each process receives input and generates output by receiving and transmitting 'tokens' that encapsulate data. The input / output connection points define which processes will deliver dataflow tokens for processing and where the token created by the process will be sent. The BaseProcess class has both an input and an output connection point. Hence the class both accepts dataflow tokens as input and produces them as output. Its child classes

BaseInitialProcess and BaseVisualisationProcess (see Figure 7-5) do not have both connection points as they represent the start or end of a workflow. The BaseInitialProcess has no input connection point because it represents the start of the workflow and therefore there can be no earlier process that provides input dataflow tokens. Equally the BaseVisualisationProcess has no output connection point since it represents the end of a dataflow.

In addition to the controlling processes connectivity through their presence or absence each connection point carries a series of three glyphs that define the types of process that can connect to that point. These glyphs are inspired by the UK road traffic sign system and reflect the information processing cycle and are presented in Table 7-3 below.







Process Category	Visual encoding used	UK Road Traffic sign	Visual road traffic encoding
Input Process		Order / Command	
Data Manipulation Process		Warning / Danger	
Output Visualisation Process		Information	

Table 7-3: Visual encoding of process type and its similarity to UK road signs

As Table 7-3 shows the stages of the information processing cycle are connected to a geometric shape. Input processes (stage one) are related to a circle / disk glyph. Circular signs on the road system encode commands and usually instruct a driver to start doing something. Input processes are associated with this symbol as they represent the start of a workflow. Very often such processes will draw data from a hard disk. The second stage of the information processing cycle (data manipulation / processing) is represented by a triangle. The road sign analogue is a warning / danger sign. The intent is to prompt the user to ensure that they are selecting the correct data manipulation process + algorithm to generate the data visualisation they need. The final stage (output) is represented by a rectangle intended to represent a computer monitor or sheet of paper. The road sign analogue is a traffic sign that presents information to the driver. This reflects that the production of information is the end goal of the information processing cycle and that the visualisation processes are the end of an iPipeline workflow. To indicate the types of process that may connect to a connection point the glyph is either filled with a solid colour (connection allowed) or unfilled (connection dis-allowed).

7.2.3 Data model support classes

The 'thin' framework layer is primarily concerned with the implementation and execution of iPipeline's workflow. However it does need to provide a means to build the 'thick' problem domain specific framework layer "on top of itself". The data model support classes are intended to provide a group of software entities that support the development of the 'thick' problem domain layer. The 'base process classes' discussed in the previous section could fit this definition but are of such key importance they warranted a separate discussion. Figure 7-7 expands upon Figure 7-5 to show the data model support classes / interfaces.

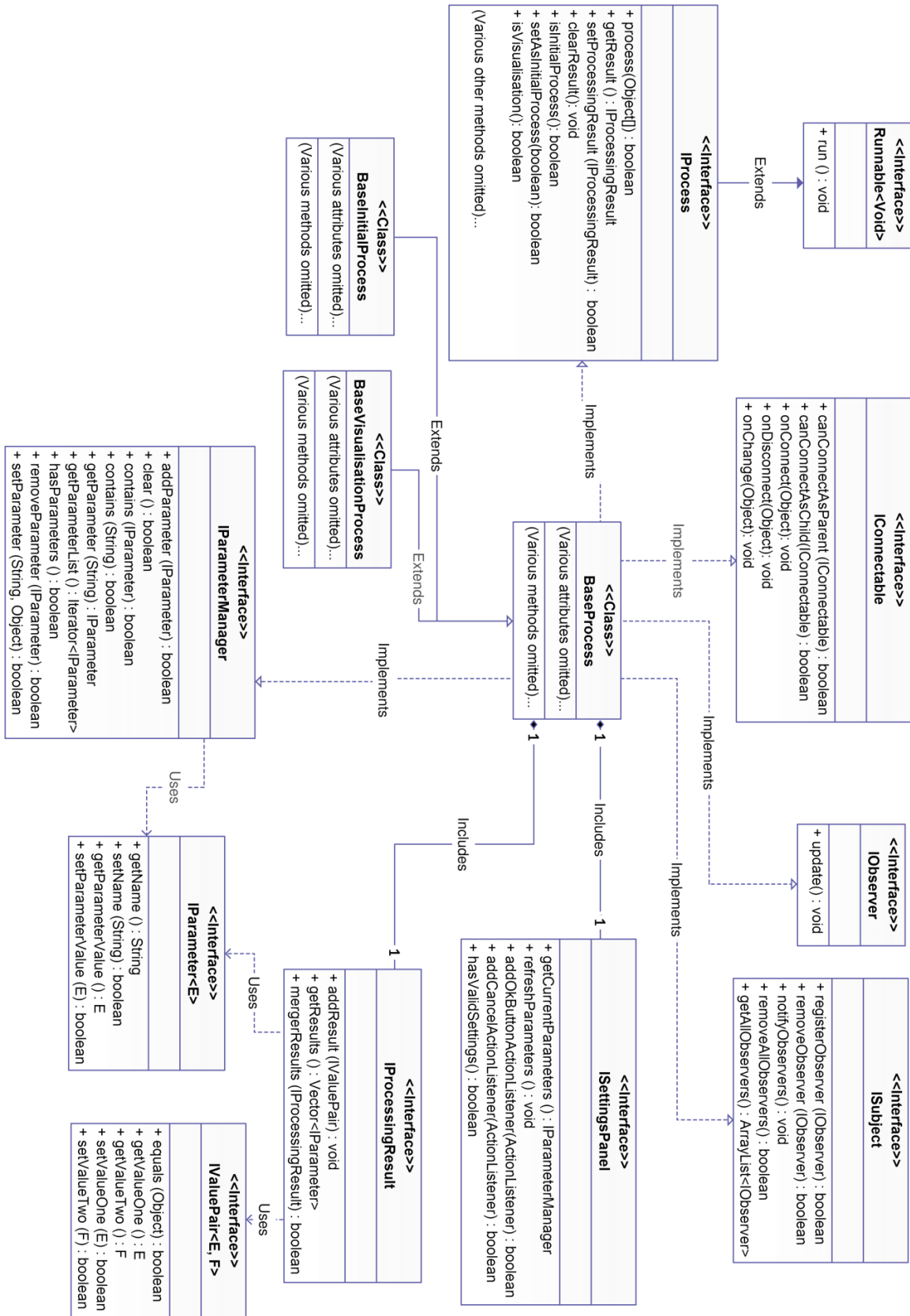


Figure 7-7: Core iPipeline interfaces and their relationships

The data model support classes divide into four “sub-systems” to support the operation of the thin framework layer. These sub-systems may also be used by developers

implementing the “thick” problem domain layer. Table 7-4 summarises the provided sub-systems, their purpose and the classes / interfaces that provide their implementation.

Sub-System Name	Classes / Interfaces	Description
Dataflow token	IProcessingResult	The interface to be implemented by the class that will serve as the dataflow token (see Chapter 5)
Settings Panel	ISettingsPanel	Interface to be implemented by a graphical component that allows customisation of problem domain algorithms
Parameter Management	IValuePair<E,F> IParameter<E> IParameterManager	Provides a generic system for transferring any data value without knowing its data type
Observer Pattern	IObserver ISubject IConnectable	Interfaces used to implement the observer software design pattern

Table 7-4: Thin framework layers supporting sub-systems

7.2.3.1 The dataflow token sub-system

As chapter 5 explained iPipeline employs a structure based dataflow model that generates a single output token from each workflow process. The token needs to ‘flow’ through the workflows directed graph. The ‘thin’ framework layer is responsible for ensuring that this happens. At the same time the ‘thin’ layer knows nothing about the token’s implementation. The token is highly specific the problem domain and is therefore implemented in the ‘thick’ problem domain layer. To solve this problem a software interface has been created in the ‘thin’ framework layer. This interface (IProcessingResult) must be implemented in the ‘thick’ problem domain layer. This interface ensures that the ‘thin’ framework layer can transfer the dataflow token between processes of the workflows directed graph.

7.2.3.2 The settings panel sub-system

While individual processes within a workflow have a consistent visual representation it is possible – even likely – that some of the problem domain algorithms encapsulated within them will require user defined inputs. In neuroscience for example many algorithms require the researcher to define a time window (the cross-correlation algorithm would be one case). The user defined inputs for a workflow process are dependent on the algorithm encapsulated by the process. The ‘thin’ framework layer can, therefore, have no knowledge of these (they are defined in the ‘thick’ problem domain layer). Despite this the ‘thin’ framework must still store and manage such user input since it is part of defining a workflow. This problem has been solved through the creation of the ISettingsPanel interface and the parameter manager sub-system.

The ISettingsPanel interface provides the ‘thin’ framework layer with the ability to attach a Java JPanel to any processes glyph. Accessed via the process glyph button (see Figure 7-6) this panel allows algorithm specific values to be supplied. The problem domain developer is free to develop any panel they need for their algorithm simply by implementing this interface. However the ‘thin’ framework layer needs to make the user inputted values

available to an algorithm in a consistent manner. This has to be achieved without the thin layer placing any restrictions on the user input and without knowledge of the number or type of values to be passed to the algorithm. The solution adopted models values as a name / value pair where the name is always a string and the value is a Java generic datatype that can become any type of value. This structure is known as a “Parameter” and is highly suited to inclusion in a list data structure. The Parameter Manager sub-system discussed in the next section manages an array list structure of these parameters. The ISettingsPanel provides interface methods to access “parameters” created by the JPanel that implements it. The BaseProcess uses these methods to populate a “parameter manager” object which the problem domain developer can access from the algorithm they are encapsulating into a process. This provides the problem domain developer the freedom to create and store as many “parameters” as they wish when creating the settings panel for a process. At the same time access is provided to these parameters via a software interface. This ensures that the developer’s parameters (whatever they might be) are delivered to the algorithm encapsulated in the process being created.

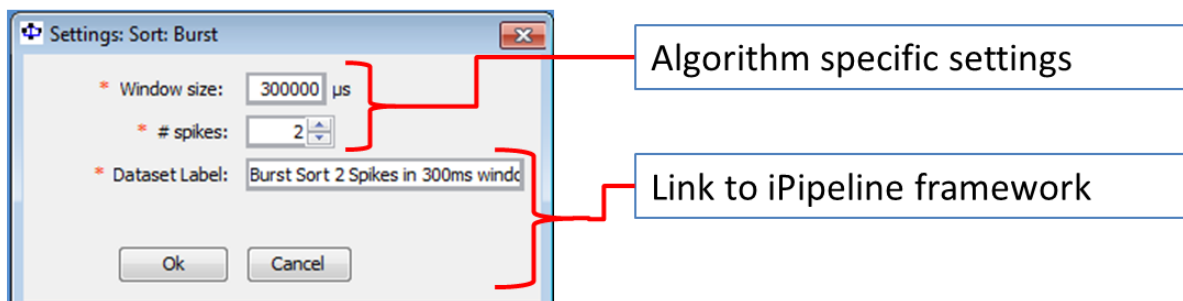


Figure 7-8: Settings panel for burst sort algorithm

Figure 7-8 provides an example of a settings panel for sorting neural spike train data based on when the spike train shows the first ‘burst’ of spiking activity. The panel is completely configurable by the problem domain process creator in terms of algorithm specific parameters. However it does require the presence of the OK and Cancel buttons as these store, or remove, the parameters from the processes ParameterManager software object. In addition the problem domain developer should name the dataset with a descriptive name which will be used to identify the dataset in any visualisation. The final design of a settings panel will contain two elements. The OK / Cancel and dataset name will always be present and forms the link between iPipelines ‘thin’ framework layer and the ‘thick’ problem domain layers. The remaining controls, values and the parameters generated from them are specific to the encapsulated algorithm and form part of the ‘thick’ problem domain framework layer.

A final issue faced when linking the ‘thin’ framework layer to the algorithm specific settings panel is the validation of the parameters. The ISettingsPanel provides a method to validate the parameters. This of course must be coded by the problem domain developer but the true / false Boolean result indicates the validity of the algorithm’s parameters to the ‘thin’ framework layer. A configuration that fails to pass this validation method cannot be part of a valid route through the workflow’s directed graph. The ‘thin’ framework layer will also indicate this failure to the user visually on the workflow desktop. Figure 7-9 shows an input process which should load data from a file but which has not yet been pointed at the required data file.

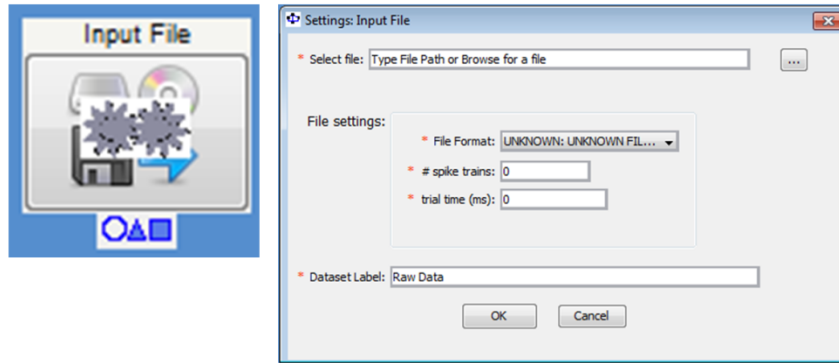
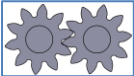


Figure 7-9: Visual encoding for incomplete parameter configuration

For incorrectly configured settings panels the ‘thin’ framework overlays an animated icon on top of the process glyph. Visually this icon is a set of rotating gears  that indicate the need for further configuration of the process settings panel.

7.2.3.3 The parameter management sub-system

The primary problem faced by the ‘thin’ framework layer is to find a way to use, transmit and manipulate data without knowing in advance what that data will be. The parameter management sub-system is the ‘thin’ framework layers solution to this problem. Figure 7-10 shows the system’s structure via a UML class diagram.

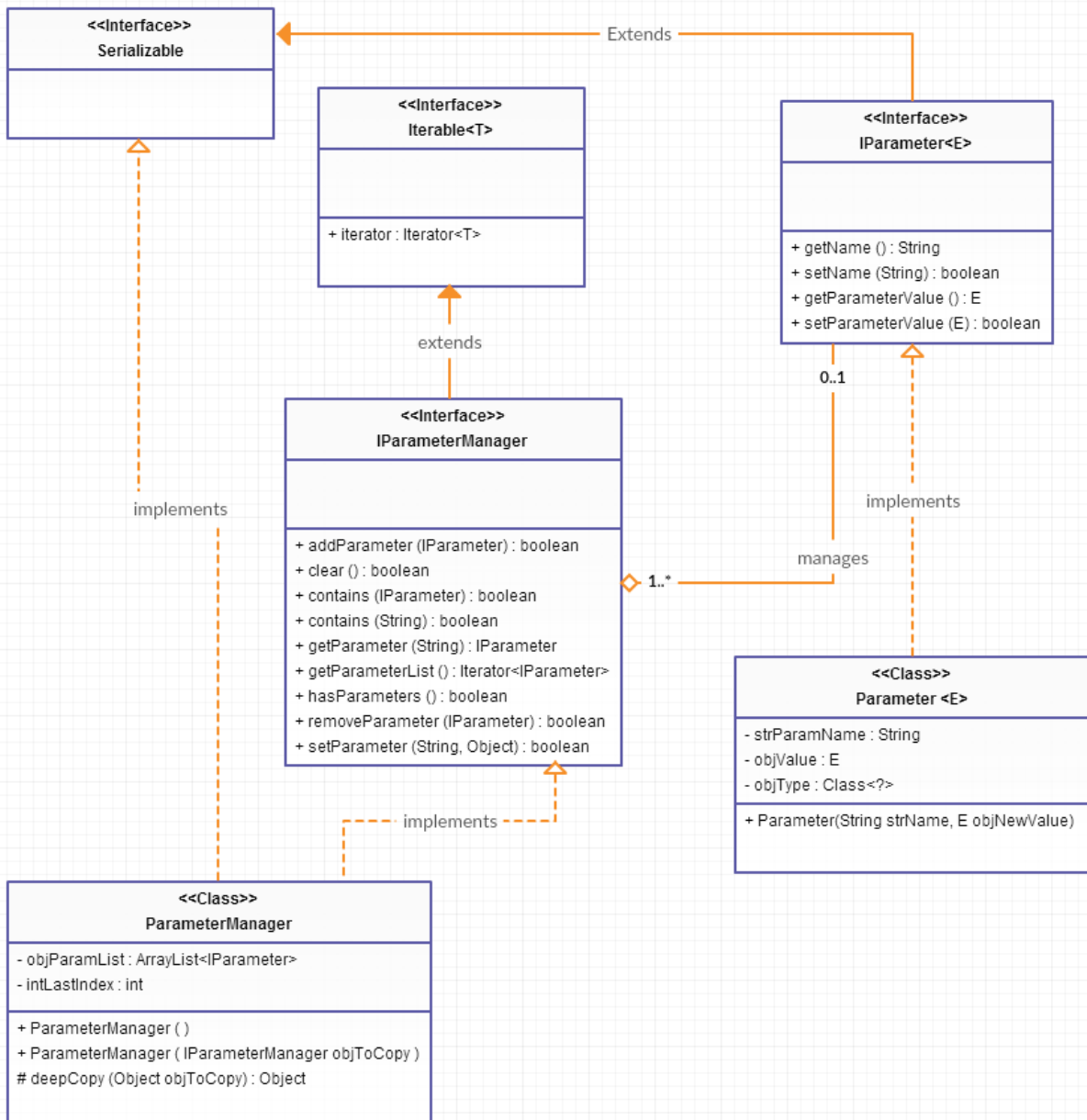


Figure 7-10: The parameter management sub-system structure

The core functionality of the system is described by two software interfaces. The IParameter interface represents the abstract concept of a name & value pair while the IParameterManager interface represents a collection of IParameter objects. For each of these two interfaces a concrete implementation is provided. These implementations are then used throughout the ‘thin’ framework layer to manage data that cannot be predefined. Examples of such data would include; the problem domain data model, the definition of the dataflow token and delivering algorithm specific data in a generic way. This sub-system therefore see’s use both within the ‘thin’ framework layer to manage data and also in the ‘thick’ problem domain layer.

The core of the system is a name-value pair where a string holds a name associated with a ‘value’. In this case the ‘value’ is any object or datatype. Java generics are used to allow a parameter to accept any ‘value’. The provided concrete implementation class examines the data type assigned on creation of a new Parameter and while it will allow the

value to change it will now allow a change in the *values type* after construction. The name component of the name-value pair is always a string. The interface `IPParameter<E>` defines these rules while the `Parameter<E>` concrete class provides the implementation (see Figure 7-10).

The `IPParameterManager` interface and its concrete implementation `ParameterManager` provide a means to store and pass parameter objects between software entities. In the ‘thin’ layer knowledge of the meaning of the parameters is not required. Its primary application within the ‘thin’ layer is to exchange values between a process and an associated algorithm specific settings panel. Every process implementation encapsulates a `ParameterManager` object which is exposed to the attached settings panel. Any settings values can therefore be stored as name-value pairs and accessed by the processes encapsulated algorithm.

The parameter / parameter manager system is also the primary tool for building the dataflow token that will be passed through the workflows directed graph. Its generic data allows structures to be built where the ‘value’ component is itself either another parameter or a list of parameters. The ‘thin’ layer represents the dataflow token via the `IProcessingResult` interface and this interface gives access to the dataflow token’s data through parameters (see Figure 7-7). In this way the ‘thin’ framework layer can manage the dataflow token without knowledge of its contents or structure

7.2.4 Parallel execution engine

The final component of the ‘thin’ framework layer the parallel execution engine is charged with actually executing the algorithms encapsulated in the workflow processes. Execution begins with the identification of all **valid workflows** currently defined on the workflow desktop. For each valid workflow execution begins with an input process that creates a dataflow token and places it on its ‘output arc(s)’. Whenever a child process detects that a dataflow token is available on every input arc then execution of that child process. Execution terminates when a visualisation process passes the final dataflow token to a visualisation. Each workflow process is a Java `Runnable` (see Figure 7-5) and can therefore be executed on an independent thread. The Java concurrency framework (Goetz et al., 2006; Lea, 2004) is leveraged to create a pool of worker threads that execute each process once all dataflow tokens from parent processes have been created. This approach exploits the natural parallelism of a directed graph with processes executing as soon as data becomes available to them. It is also the source of the single restriction placed on the problem domain developer. Specifically that the data model they create must be thread safe. The data model will implement (or be wrapped by a class that implements) the `IProcessingResult` interface. Since the dataflow token will be propagated through the directed graph and each process is expected to be running on a different thread the underlying implementation must be thread safe.

Chapter 8

Neuroscience Problem Domain Layer

“Problem Domain Analysis is the process of creating a model describing the problem to be solved”.

Summary

This chapter examines the implementation of the iPipeline problem domain layer for analysis of neural spike trains. It serves to provide the implementation on which the neuroscience visualisations are built and as a model for implementing this layer both in neuroscience and for other fields.

8 Creating a data model for neuroscience spike train recordings

Before iPipeline can be used to analyse and deliver data to interactive visualisations an implementation of the problem domain layer must be created. At a minimum this implementation must provide:

1. A thread safe data model representing the multi-dimensional spike train recordings to be analysed.
2. A set of analysis processes (algorithms) that can be combined to create a dataflow directed graph.
3. A set of interactive visualisations that the researcher may use to further explore the data.

This chapter considers points (1) and (2) while the interactive visualisations provided will be described in their own chapters.

8.1 Designing the data model

The creation of the spike train data model required a number of factors to be taken into account. These can be summarised as:

1. Neural spike train recordings constitute ‘big data’
2. The data model should be thread safe
3. The use of memory should be minimised

8.1.1 Managing big data in the data model

The phrase “big data” has become something of a buzz word in information technology and a clear definition can be difficult (Snijders, Matzat & Reips, 2012). The two most commonly agreed upon qualities of big data is its physical size and complexity. Generally to be big data one or both of these must “Big Data usually includes data sets with sizes beyond the ability of commonly used software tools to capture, curate, manage, and process the data within a tolerable elapsed time”. The various types and definitions of big data were reviewed in chapter 3 which examined why spike train recordings form a “big data” set. This section considers how the “big data” can be managed in a practical implementation of a spike train data model.

In designing the data model it is important to consider:

1. Data Storage
2. Data Searching
3. Interactive Data Visualisation

These areas are identified from considering the use to which the data model will be put. Simultaneously recorded spike trains generate huge amounts of data at even modest neuron firing rates. For example a 1000 neuron data set recorded for 30 minutes and exhibiting a 10Hz firing frequency rate will generate 18 million data points ($1000 * 30 * 60 * 10$). Given that a primary goal of the VISA project is to allow analysis of more data using a typical researcher desktop or laptop system an efficient storage model must be developed. This already large number was further complicated by the storage model in use in earlier versions of the VISA software.

8.1.1.1 *Flaws in the original VISA data model*

Previous versions of VISA used a Boolean array to represent whether a neuron had fired. Typically a true / false entry would be made for every millisecond describing whether a spike event had been detected. This would require some 1,800,000,000 individual data points for 1000 neurons recorded for 30 minutes. The implementing language is Java which normally uses 1 byte to represent a Boolean value. Simply holding this data structure in memory will require 1.67638 Gigabytes of storage. Such a structure, while suitable for 100's of neurons rapidly grows as the number of recordings and their length grows. As such it cannot be said to scale to the number of neurons that modern hardware can record or is expected to record in the near future. The original Boolean array data structure suffers from two failings that lead to considerable inefficiency:

1. It does not encode in a single entry all relevant information about a spiking event
2. It stores irrelevant data that produced considerable memory inefficiencies.

8.1.1.2 *Encoding and storing relevant data*

A spike event looked at in isolation, is an all or nothing event that has only one relevant piece of data associated with it – the time at which it occurs (Dayan & Abbott, 2005; Gerstner & Kistler, 2002). A neural spike train is a time ordered sequence of spikes recorded from the same neuron. The original VISA data model encoded the time a spike occurs into a sequence (array) of Boolean values. This means that the time a spike occurs is represented by its position in the array. The time between spike events is expressed as the number of array elements between two true Boolean values. To extend the analogy a single neuron firing with a rate of 10Hz would be encoded as an array of 1000 elements where ten have a true value and 990 are false. Assuming a constant firing rate for the neuron every 100th element of the Boolean array is a true value. In terms of memory this data structure is extremely inefficient as some 99% of the stored values are the same, a Boolean false value. A solution to this problem is already known, the described data structure could be implemented far more efficiently using a sparse array.

A sparse array eliminates the repetitive data and stores only the points of interest. In this case the 'interesting data' is the times at which spike events occur. Traditionally implemented as a linked list this approach discards the Boolean in favour of recording the time of the spike event. Each array entry becomes a single time value and a pointer to the location in memory of the next spike event in the spike train. To be meaningful a time value requires two components, a numeric value and the units in which the numeric value is stated. In the final data model this implementation is followed with a double value denoting spike time and an integer enumeration for the timescale. In addition a third integer value representing the number of spikes occurring at this time was added for use when groups of spike trains are created. In the Java language the memory requirements for this data structure becomes 16 bytes being one 8 byte double value and two 4 byte integer values. Using this data structure the example of one neuron recorded for 1000ms and firing at a rate of 10Hz requires 160 bytes to store. Java fails to define the size in memory of a Boolean but most implementations of the Java Virtual Machine (JVM) use one byte per Boolean value. Assuming this the memory saving is $1000 / 160$ or 6.25 times smaller when a sparse array is used. Returning to the original example of a 1000 neuron data set recorded for 30 minutes and exhibiting a 10Hz firing frequency rate the storage requirement becomes 0.26822 Gigabytes or 274.6582 Megabytes. The adoption of the sparse array model minimises memory demands and adapts itself to the specific number of neurons and firing rate.

Theoretically the data for a large number of relatively inactive neurons could be stored in less memory than a small number of highly active neurons. The important point is that the model adapts its memory use to the data being stored unlike the original VISA model.

8.1.1.3 Searching the data model & interactive displays

The VISA project ultimately aims to deliver the processed data to interactive visualisations that a researcher may use to explore the dataset. In designing the data model it is important to remember the use to which it will be put. From chapter two Shneiderman identifies seven “Task-domain information actions” that visualisation users may perform. In summary these tasks are (Shneiderman, 1996):

- | | | |
|-------|-------------------|---|
| viii. | Overview | Gain an overview of the entire data collection |
| ix. | Zoom | Zoom in on items of interest |
| x. | Filter | Filter out uninteresting items |
| xi. | Details-on-Demand | Select an item or group and get details when needed |
| xii. | Relate: | View the relationships between items |
| xiii. | History | Keep a history of actions to support undo, replay and progressive refinement. |
| xiv. | Extract | Allow extraction of sub-collections and of query parameters. |

It is appropriate then to design a data model that supports these operations as much as possible. The iPipeline framework leaves the definition of the data model so completely open specifically to allow its design to be appropriate for the data held. So the developer must ask what are the most common operations performed on this data and how can they be supported? What constraints exist for the data model? These two questions can both be answered by looking at how the data will be used. Of Shneiderman’s seven task domain actions only (vi) seems completely independent of the data model’s implementation. For the others a time efficient means to sort and extract sub-sets of the data in the model is required. Why time efficient? Time efficiency is necessitated since this data model is used to generate interactive displays. In such a display the visualisation developer aims to “promote an experience of being in direct contact with the data” (Ware, 2012). (Rutkowski, 1982) calls this the Principle of Transparency which when successfully applied allows the user “to apply intellect directly to the task; the tool itself seems to disappear”. One of the key requirements to produce this effect is the tool’s responsiveness. Long delays in responding to user interactions violate the principle of transparency. As a general rule of thumb Shneiderman recommends that the visual response to an interaction should be provided within 1/10th of a second of the interaction (Shneiderman & Plaisant, 1998). The user’s interaction will inevitably require the display to perform some transformation of the data. When faced with ‘big data’ ensuring the system’s responsiveness can be challenging. Indeed the very definition of big data used in section 8.1.1 required a data set that was difficult to curate and manage in a tolerable time period. Since the problem domain involves big data the developer must consider not only how data points are described and stored (curation) but also how they can be *managed in a tolerable time period*.

The question that must be asked by the data model developer is: How can the model manage a large number of data points (millions) and still be responsive? The key requirements to answer this can be drawn from Shneiderman’s task domain actions:

1. The ability to create sub-sets of data (ii, iii, iv, v and vii)

2. The ability to rapidly search and organise large amounts of data (i, ii, iii, v and vii)

Meeting the first requirement will greatly reduce the amount of data that must undergo transformation when the user interacts with the data. Meeting the second will allow transformations to occur at high speed. The most fundamental structure in the neuroscience data is the spike event however there is a higher level structure. The definition given for a spike train in section 8.1.1.2 was:

A neural spike train is a time ordered sequence of spikes

The previous VISA data model translated this to mean an array of values but the question can be asked, is there a more efficient data structure than an array? In this case the efficiency being sought is the ability to rapidly search, sort and create sub-sets of the data. Much research has historically focused on rapid searching and sorting of data sets and has identified numerous data structures that perform these tasks (sets, stacks, queues, hash maps, arrays and lists to name a few). The simple array is the least efficient structure in terms of searching, sorting and extracting sub-sets. The next implementation step for the problem domain layers data model was to select a more suitable structure to manage the spike-train data. After reviewing the various options available the Java TreeSet which provides an implementation of a red-black binary search tree was selected.

Ultimately the Java TreeSet class was selected as the storage mechanism for spike train data points. This class provides an implementation of a red-black binary search tree. The next section overviews how binary trees and specifically red-black binary trees work.

8.1.2 The theory of binary trees and self-balancing red-black binary trees

8.1.2.1 Definition

A binary search tree has the following properties (Hibbard, 1962):

1. There is one and only one node, called the root, such that for any node p there exists one and only one path which begins with the root and ends with p .
2. For each node p the number of links beginning with p is either two or zero. If the number is two, then p is said to be a proper node. If the number is zero then p is said to be a blank node
3. The set of links is partitioned into two sets L and R . Each link belonging to L is called a left link. Each link belonging to R is called a right link.
4. For each proper node p , there is exactly one left link beginning with p and exactly one right link beginning with p .

Each node of the binary search tree is associated with a value, usually termed its key, which controls the ordering of the node within the tree so that:

1. The left sub-tree of a node contains only nodes with keys less than the node's key.
2. The right sub-tree of a node contains only nodes with keys greater than the node's key.
3. The left and right sub-tree also form a binary search tree.
4. There must be no duplicate nodes.
5. A unique path exists from the root node to every other node.

Performing operations, like search, insert and delete on a binary search tree requires comparisons between the nodes keys. These comparisons are made by a *comparator* which defines the ordering of any two keys. In VISA3 the implemented neuroscience data models fundamental unit of data – a neuron spike event – implements the Java Comparable interface to provide a natural ordering based on the time a spike occurs. This reflects the definition of a neural spike train as a time ordered sequence of spike events. Figure 8-1 illustrates a binary search tree for a neuron with a regular firing rate of 10Hz. Time values are recorded in milliseconds with a spike event every 100 milliseconds.

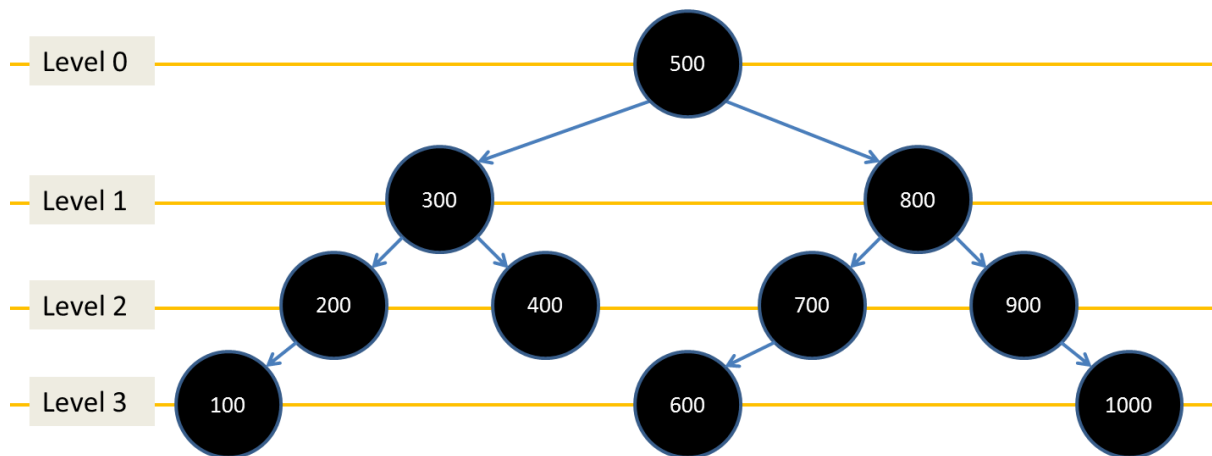


Figure 8-1: A binary search tree for a regularly firing (10Hz) neuron recorded for one second.

From Figure 8-1 the key features of a binary search tree can be derived. The root node is the spike event at 500ms. Leaf nodes have no child nodes associated with them and are nodes 100, 400, 600 and 1000. Proper nodes have exactly one left and one right child node being nodes 300, 500 and 800. The structure is in sorted order with earlier spike events to the left and later events to the right. The tree may be broken at any node and the resulting structure is itself a binary search tree. For example if the 500 to 800 link is broken then 800 becomes the root node for a new binary search tree. It is this property that makes the extraction of sub sets highly efficient. Duplication of data is not possible, as two nodes with the same value cannot exist in the same binary search tree. In addition it is normal to speak of binary search trees having a size and a depth. As with the traditional array structure the size is the number of elements, 10 in Figure 8-1. The depth is given by the number of levels, occupied by nodes of the tree after the root. Figure 8-1 is a depth three (3) binary search tree. The depth of the tree is critical to the performance of the search, insert and delete operations which perform faster with lower depth value trees. This introduces the concept of the ‘balanced binary search tree’.

8.1.2.2 *Balanced binary search trees*

Figure 8-1 serves as an example of a ‘balanced binary search tree’. A tree is said to be balanced when its nodes are organised to comply with the rules of a binary search tree *and the depth of the tree is minimised*. This property is *not* implicit in a binary search tree; it is possible to construct a tree with the same data which is not balanced. In Figure 8-2 this has been done for the spike train used in Figure 8-1. Figure 8-2 shows the worst case scenario where the individual spike events have been added to the tree in a time sorted order. The result is still a binary search tree as all the rules outlined in section 8.1.2.1 still

hold true but the depth of the tree has grown to nine (9). Operations on this unbalanced binary search tree are considerably less efficient. For example the Figure 8-1 search tree could locate an item with no more than four comparisons between the nodes key and the requested key (worst case). The Figure 8-2 search tree will, in the worst case, require ten comparisons when searching for the spike event at 1000 milliseconds.

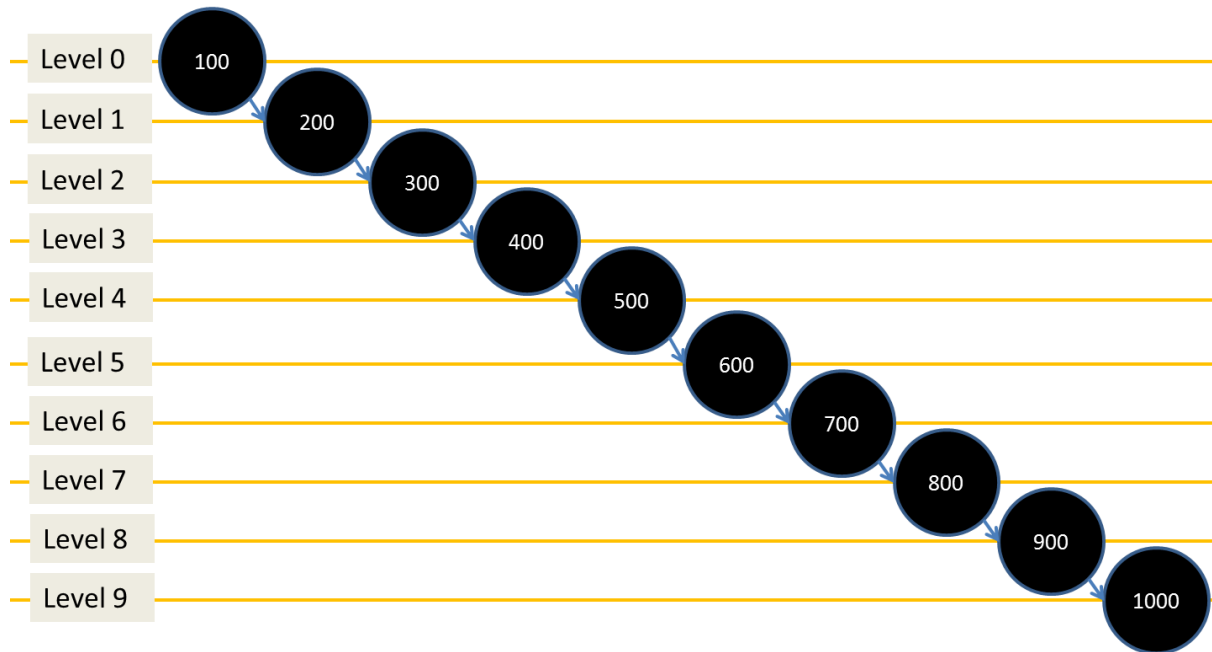


Figure 8-2: An unbalanced binary search tree

It would be very unusual for modern recording equipment not to store spike train recordings as a sorted time sequential list of spike events (i.e. a spike train). As Figure 8-2 illustrates building a binary search tree by adding sorted data to the structure inefficient. Indeed the resulting data structure is functionally equivalent to a linked list and would lose all of the benefits associated with binary search trees. Nevertheless the input data for the data model will almost certainly be stored in exactly this sorted form. What is required then is a means to accept sorted / ordered data but generate a balanced binary tree structure. Once again the storage method for the neural data needs to adapt to the data received. Simply put a binary search tree is required that minimises its depth regardless of the order in which data is added or removed. This is the definition of a self-balancing binary search tree (Knuth, 1998). Fortunately in the case of a binary search tree Rudolf Bayer proposed a solution to this problem in the form of the Red-Black tree in 1972.

8.1.2.3 The Red-Black self-balancing binary search tree

To create a Red-Black binary search tree the requirements of section 8.1.2.1 are expanded to impose the following additional constraints (Cormen et al., 2009):

1. Each node of the tree is assigned a colour – either Red or Black.
2. The tree's root node is always black.
3. All leaves are the same colour as the root node (Black).
4. Every red node MUST have two black child nodes.
5. Every path from a given node to any of its descendent leaves contains the same number of black nodes.

The impact of these additional constraints is best illustrated by example. Therefore Figure 8-1 has been re-stated as a red-black binary tree in Figure 8-3.

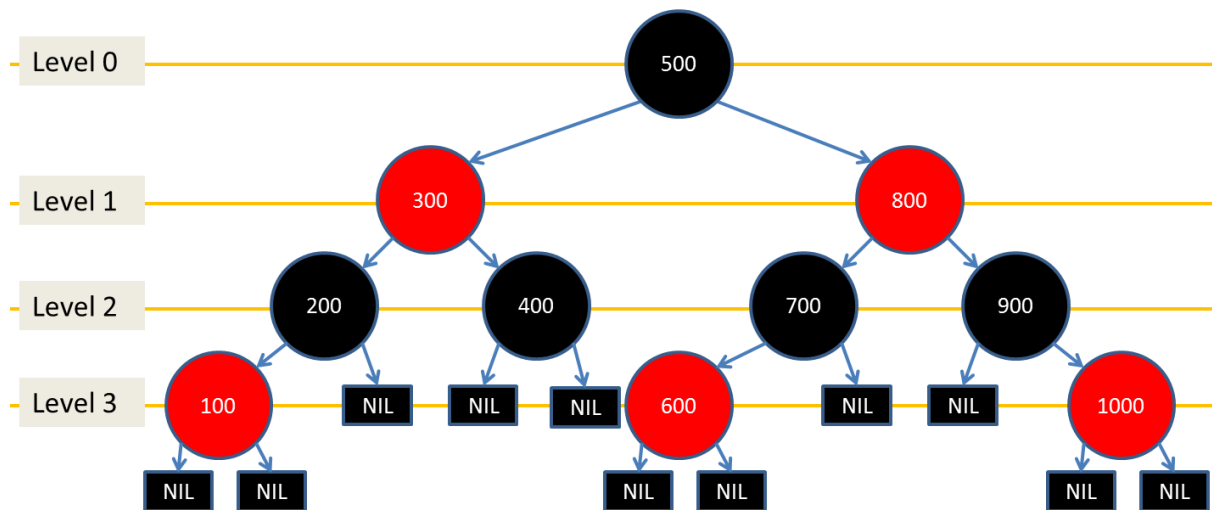


Figure 8-3: A Red-Black binary search tree for a regularly firing (10Hz) neuron recorded for one second

The additional constraints introduced to create the Red-Black tree enforce a new property. The length of the path from the root to the furthest leaf is not more than twice the length of the path to the leaf nearest to the root. So long as this property is maintained the resulting tree is at all times roughly height balanced. This property can be stated mathematically as; A Red-Black tree with n internal nodes has a height no greater than $2 \log(n + 1)$.

Any modification of the tree has the potential to violate one or more of the additional constraints that create the red-black tree. Therefore the insert and delete operations may require the tree to re-balance. The creation / extraction of sub sets could only violate additional constraint 2 in which case the correction is trivial as the sub-tree nodes are simply re-coloured. Hence it is true to say that any sub-tree of a red-black tree is itself expressible as a red-black tree. Structural modifications are more complicated but the constraints of a red-black tree can always be restored using one or more **rotations**. A rotation is defined as: “a local operation in a search tree that preserves the binary-search-tree property”. There are two types the left or the right rotation. In each case the rotation “pivots” around a link in the tree changing the connections between nodes and possibly re-colouring them. Figure 8-4 illustrates both the left and right rotations.

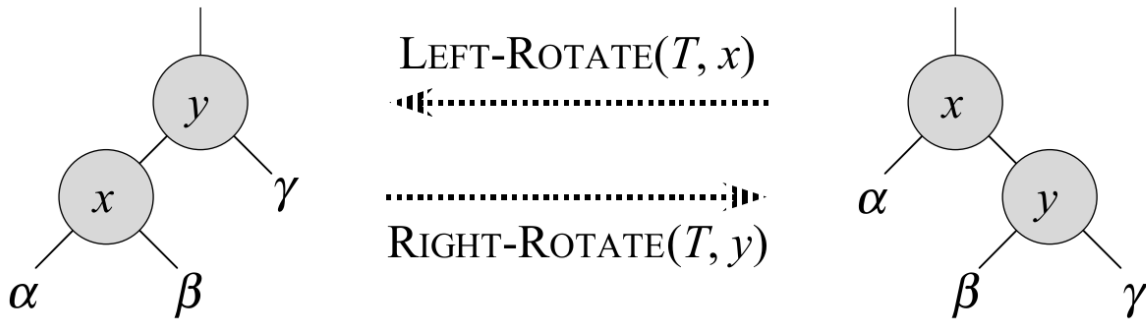


Figure 8-4: Left / Right rotation in a binary search tree. Source (Cormen et al., 2009).
 NB: the symbols α, β and γ represent any arbitrary binary search tree

Performing a rotation may lead to a violation of constraint 2, 4 or 5; in this case either nodes must be re-coloured or a further rotation one level higher in the tree must be made. In the worst case scenario this may cascade up the tree to the root node. If a rotation of the root node changes its colour to red then the nodes of the tree must be re-coloured to restore constraint 2. Figure 8-5 shows the impact of a left rotate operation on a binary tree structure.

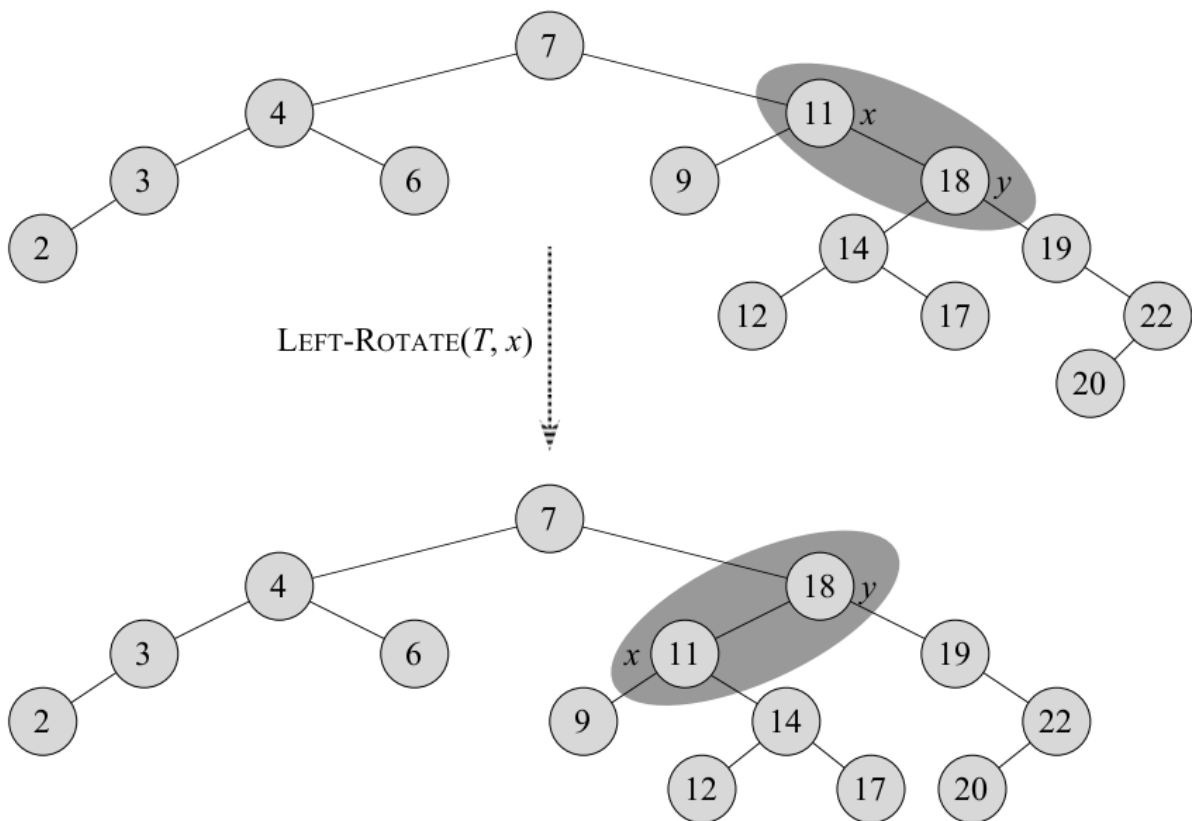


Figure 8-5: A binary tree structure both before and after a left rotation of the marked x, y connection. Source (Cormen et al., 2009).

Fortunately the Java SDK provides an implementation of Bayer’s Red-Black binary tree in the form of Java TreeSet class. Therefore the data model implemented for the problem domain of spike train analysis makes extensive use of this class. All individual spike events are stored in the data model in a TreeSet which is the foundation of the models spike train implementation. ISpikeTrain and its concrete implementation function as a Java

collection of ISpike objects representing a single recording with the TreeSet as the storage mechanism for collection objects. The result is a data structure that can manage a large data quantity while making a guarantee of performance when filtering, zooming and creating sub sets of the data.

In summary then the key qualities of a red-black binary tree that make it suitable for spike train data management can be summarised as (Bayer, 1972):

1. The tree provides an efficient search operation that guarantees search times of $O(\log_2 n)$ where n is the number of elements in the tree.
2. The tree guarantees that the insert, delete and re-ordering operations for data also occur in $O(\log_2 n)$ time.
3. The tree is a self-balancing binary search tree, meaning that even in the face of an arbitrary number of insertions or deletions it will maintain performance.
4. In order traversal of the tree is possible using an asymptotically optimal algorithm requiring $O(n)$ time.
5. A self-balancing binary search tree is one of the most efficient data structures for *incremental sorting*, adding items to a list over time while *keeping the list sorted at all times*.

Using a data structure that is essentially self-sorting simply mirrors the key property of a spike train recording, that it is a time ordered sequence of spikes. The self-balancing binary tree enforces this definition by using the ‘natural ordering’ of the data it contains. In this case the natural order is the distinguishing feature of a neural spike – the time at which it occurs. Furthermore Shneiderman’s “Task-domain information actions” are also easily supported by the data structure. The creation of overviews, for example, often involves combining detailed information to present a simplified view. With neural spike train data this would involve creating a single spike train that represents several individual spike trains. This merging would result in many insertion operations. The self-balancing binary trees maintain performance regardless of the number of insert or delete operations performed (unlike traditional binary tree structures). Equally the Zoom, Filter and Details on Demand operations will require frequent searches, insertions and deletions. All these occur in $O(\log_2 n)$ time in a red-black binary tree. Returning to the example of 1000 neurons recorded for 30 minutes with an average 10Hz firing rate the performance impact of the red-black binary tree can be assessed. Such a recording generates 18 million data points. The simplest request that can be made of the data set is; did a spike event occur at time x milliseconds? In the original VISA data structure this would involve searching 1000 arrays and determining if the x element was true. This same test can be made in a red-black binary tree in no more than $(\log_2 18,000,000) = 24.10149357$ or 25 operations. This reduction in workload leads to significant time savings when querying the data model. Finally there is Shneiderman’s task domain action – Extract (point vii) which requires the extraction of a sub collection of the data set. The tree structure of the red-black binary tree also supports this operation. Any red-black binary tree may be ‘broken’ at any node of the tree to create a new binary tree. The node at which the break occurs becomes a new root node of a new binary tree which is already a sorted, in-order representation of a sub-set of the original binary tree. Hence the creation of a data subset is a rapid operation requiring almost no compute time.

This section has demonstrated the importance of storing and managing large data sets using efficient data structures. In addition Rutkowski’s “principle of transparency” and

Shneiderman’s “task-domain information actions” have been used to identify a suitable data model structure that supports the visualisation and management of ‘big data’.

8.1.3 Supporting Shneiderman’s principle of “Overview first” and “Details on demand”

The lowest levels of the data model have been focused on providing efficient management of the ‘big data’ problem. Conversely the higher levels provide a structure that supports effective visualisation of the large data set. Key to this is Shneiderman’s principle of providing the user with a high level ‘overview’ of the data sufficient to identify key features. The user then requests more detailed information on the features that interest them. The primary aim is to avoid a ‘data deluge’ or ‘cognitive overload’ of the user’s data processing ability. The general approach is to provide a summary of the data with which the user may interact to reveal more detailed information. To support the generation of data summaries the data model provides a means to manage spike trains both as individual recordings and on the level of groups. The INeuron interface and its concrete implementation provide the wrapper around individual spike train recordings. This interface is then extended to create the INeuronGroup interface which manages arbitrary collections and groupings of neuron spike train recordings. This sets up an ‘is-a’ relationship within the data model that allows any INeuron object to be seamlessly replaced with an INeuronGroup object. Equally the opposite is true so that an INeuronGroup can be replaced with a single INeuron spike train recording. The concrete implementation for the INeuronGroup interface holds references to each INeuron object placed into the group. From these it constructs a single Red-Black binary tree that contains all spiking events from all group members. The complication with this approach is that multiple spike trains may contain synchronous spike events and binary trees cannot contain duplicate values. This is resolved by introducing an attribute to the ISpike object to track frequency. For single spike trains this value is always 1 but as groups are created synchronous spike events are represented by a single entry in the binary tree with frequency > 1. This structure permits visualisations to group and summarise data and then treat those summaries as a single spike train entity. Behaviours are provided to break an INeuronGroup back to its constituent parts. This allows visualisations to present groups of spike trains either as a single entity or multiple spike trains.

8.1.4 Creating the dataflow ‘token’ using the developed data model

So far the structures in the data model have been concerned with individual data points or individual recordings. However the data model is also required to act as a dataflow token that can be placed on the arcs of the iPipeline dataflow directed graph. The iPipeline framework exposes an interface to represent the dataflow token and a default implementation that allows any data model to be added to it using a generic name / value pair. The data model must then be reducible to the value component of a name value pair. This is not a significant issue for a data model since the developer simply defines an appropriate data model and then provides a wrapper to contain it. In the case of the spike train data recordings it is logical that this wrapper should represent the entire network of recorded neurons. The INeuronAssembly and its concrete implementation provide both a container for the data recordings and a means to attach meta-data such as size, duration of recording and timescale used (milliseconds, seconds etc.).

With the data model defined two utility methods were created that allowed any number of INeuronAssembly objects to be stored / extracted from iPipelines IProcessingResult object. This defines the general structure that each processing operation

in the neuroscience data processing library would follow. In pseudo code each processing operation follows the following high level structure:

1. Receive an arbitrary number of IProcessingResult objects; one on each input arc of the dataflow directed graph
2. Extract all INeuronAssembly objects and associated parameters from each IProcessingResult and compile them into a list. This requires a deep copy to be made of the data model as the same IProcessingResult object will potentially be delivered over multiple arcs of the dataflow graph.
3. For each listed INeuronAssembly apply the data processing logic to transform the data model to its new state.
4. When all identified INeuronAssembly objects have been transformed create a new IProcessingResult object that contains the transformed data model(s).

Once created the iPipeline framework will place the post processing IProcessingResult object on all outgoing arcs for the process. This will trigger processes further down the iPipeline directed graph as processing results complete.

8.1.5 Supporting history and relationship tracking

Shneiderman's task domain actions also identified a need to track relationships and the processing history through which the data has passed. These domain actions are primarily focused on the users interactions with the visualisation displays. However there is clearly value in maintaining, in the data model, a record of the transformations, or processing that has been applied to the data. Additionally as new processes generate information and meta-data there is a need to provide a mechanism where the data model becomes extensible. The iPipeline framework exposes two data structures to manage history and relationship tracking:

1. The IParameterManager interface and its concrete ParameterManager implementation provide the means to track processing history and append new data structures to the model making it extensible.
2. The ICommandTracker interface and its concrete CommandTracker implementation are a pre-made implementation of the 'Command' software design pattern. This provides a ready-made data structure that allows interactive visualisations to "encapsulate a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations" (Gamma et al., 1994).

8.1.5.1 Parameter Manager – creating an extensible data model

One of the primary goals of the iPipeline framework is that users should be able to create and introduce their own data processing algorithms. It is inevitable such algorithms will need to provide their own data structures. The iPipeline framework therefore provides a general means to package new data structures into an existing model and manage them as a part of the data model. Indeed the IProcessingResult views the developed data model as a 'Parameter' that is special only in that it provides the dataflow 'token' with the data it is carrying. In this system a parameter is a name value pair where the name is any descriptive string and the value is any object or collection of nested objects. The 'collection of nested objects' can constitute a data model in its own right allowing the existing model to be expanded to meet changing requirements. One of the drawbacks of the structure dataflow

model is the demands placed on system memory. If this becomes a concern it is possible to output data to long term storage devices and store references to files into the parameter system.

The parameter / parameter manager system also provides the means to store individual process parameters. This forms part of the developer created settings panels with the system providing the storage mechanism for an arbitrary number of settings values per a process. Figure 8-6 provides the high level UML diagram for the system. As can be seen the ParameterManager provides an iterable collection of IParameter objects that supports serialisation. The concrete Parameter class uses Java generics to provide a type safe name / object data structure. The Parameter constructor accepts any object <E> for its value but on creation enforces that any future updates to the parameter must supply an object of type <E>. Hence parameter type is initialised at construction and becomes immutable thereafter, despite the use of a generic parameter.

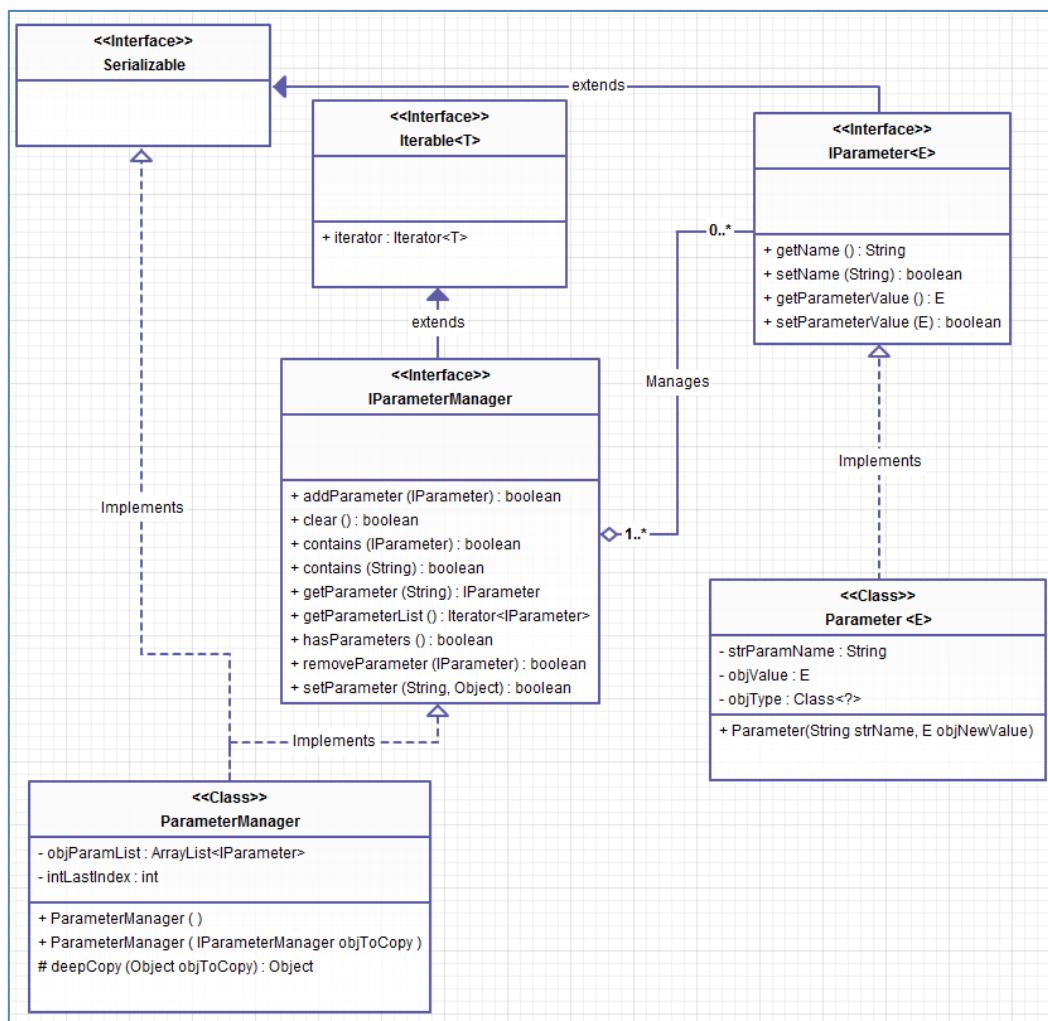


Figure 8-6: The iPipeline parameter and parameter manager system

8.1.5.2 Command Tracker – A software pattern to support interactive visualisation & tracking of user interactions

As Shneiderman described it is an important property of any interactive visualisation that the user should be able to rapidly see the result of an interaction. It is of equal importance that they should be able to rapidly return the visualisation to a previous state.

The reasons for doing so range from an interaction that did not produce the desired result to the user seeking to remind themselves of the broader context of the detailed data they have been examining.

To facilitate this task domain action iPipeline provides an implementation of the 'Command Pattern', a software design pattern that supports this behaviour (Gamma *et al.*, 1994). The iPipeline framework itself makes extensive use of this implementation to allow users to freely redefine the structure and connections in the dataflow graph. The implementation is deliberately provided as a separate package that can also be added to the problem domain visualisations. Figure 8-7 shows the UML for the implementation:

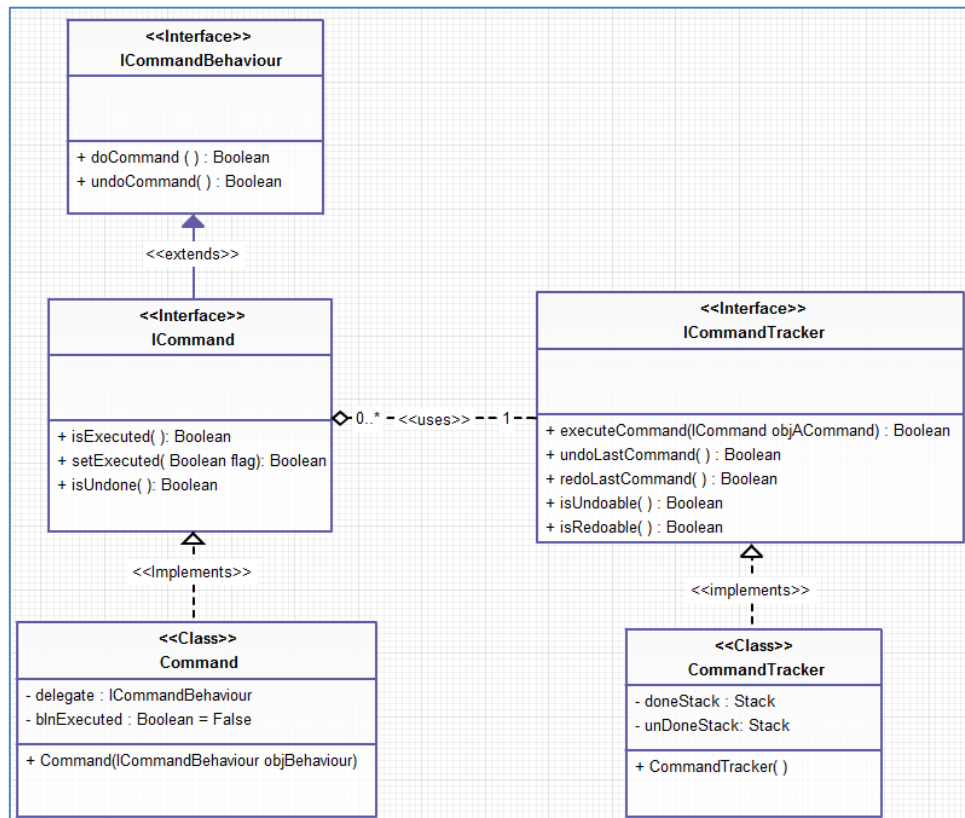


Figure 8-7: Implementation of the 'Command' software design pattern

The core of the system is the `ICommandTracker` interface and its concrete implementation `CommandTracker`. To create a new command the developer must implement the `ICommandBehaviour` interface and provide code both to execute and undo the command. A wrapper class 'Command' adds the functionality needed to track execution to the developer coded command.

To effectively utilise this system it is recommended that the developer creates a single controller class for a visualisation that maintains a `CommandTracker` object. The most flexible option is to use composition where the controller class implements `ICommandTracker` but delegates its functions to a `CommandTracker`. All interactions with the visualisations should be coded as commands. Specifically the command should be encapsulated into a class that implements `ICommandBehaviour`.

Collectively the design of the neuroscience data model and the supporting iPipeline framework address all of Shneiderman's 'Task-Domain actions' to produce effective visualisations for exploring large neuroscience spike train recordings.

Chapter 9

The i-Raster Visualisation

“Raster noun. A pattern of closely spaced rows of dots that form an image (as on the cathode-ray tube of a television or computer display)” (The Merriam-Webster dictionary, 2004)

Summary

This chapter describes the iRaster visualisation, its implementation and the visualisation techniques added to it, to manage large datasets.

9 Overview of the i-Raster Visualisation

A raster chart is one of the most common visualisations. In principle a raster chart can be generated by taking any two variables, assigning them to the X and Y axis of a chart and plotting the variables data. They have been used in many fields from the production of navigational charts (National Oceanic and Atmospheric Administration, 2015) to neuroscience (Somerville et al., 2011). Within neuroscience, they provide the researcher with the most elementary view of their raw data; the neural spike train. Figure 9-1 shows a raster chart of twenty simulated spike trains recorded for 2 seconds. The x-axis shows time in milliseconds and the y-axis denotes the id number of the electrode that recorded the spike train.

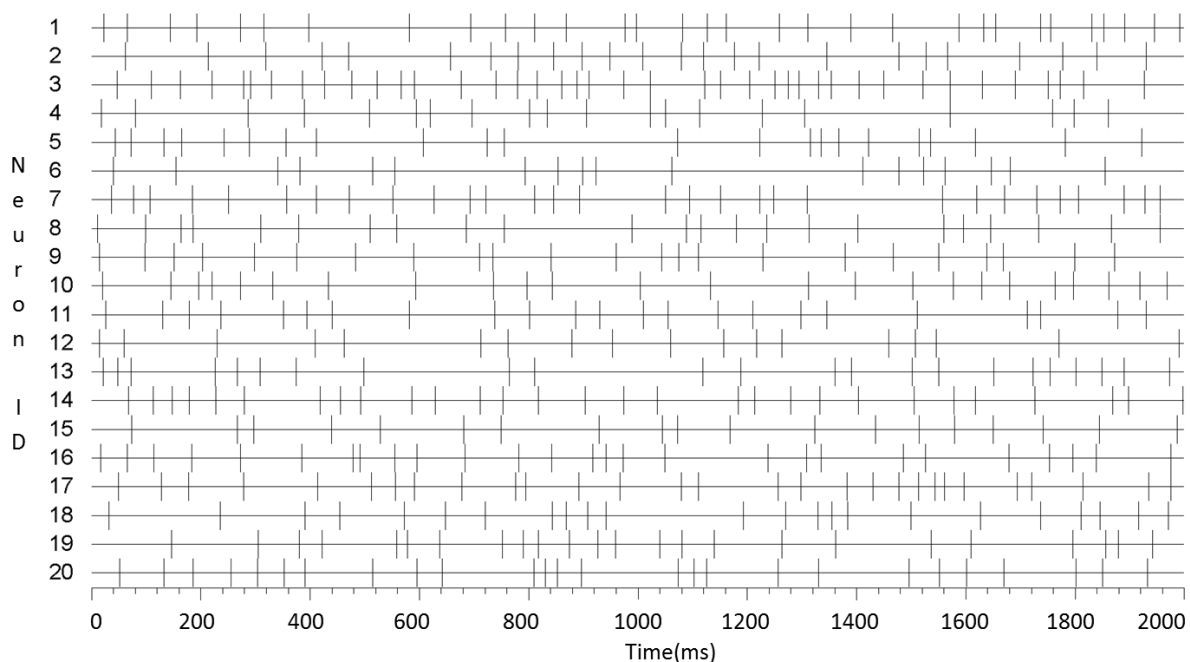


Figure 9-1: Raster chart of twenty spike trains recorded for two seconds.

Somerville demonstrated that visual analytics could be effectively applied to a raster chart in three ways (Somerville, 2011):

- i. Reordering spike trains along the y-axis (Figure 9-2)
- ii. The accurate representation of spiking rate (Figure 9-3)
- iii. Reduction of complexity using filters and zooming (Figure 9-4)

Through the application of these three methods visual analytics makes it possible to identify patterns within the dataset through visual inspection of the raster chart. Figure 9-2 to Figure 9-4 show examples of these techniques applied to the dataset seen in Figure 9-1

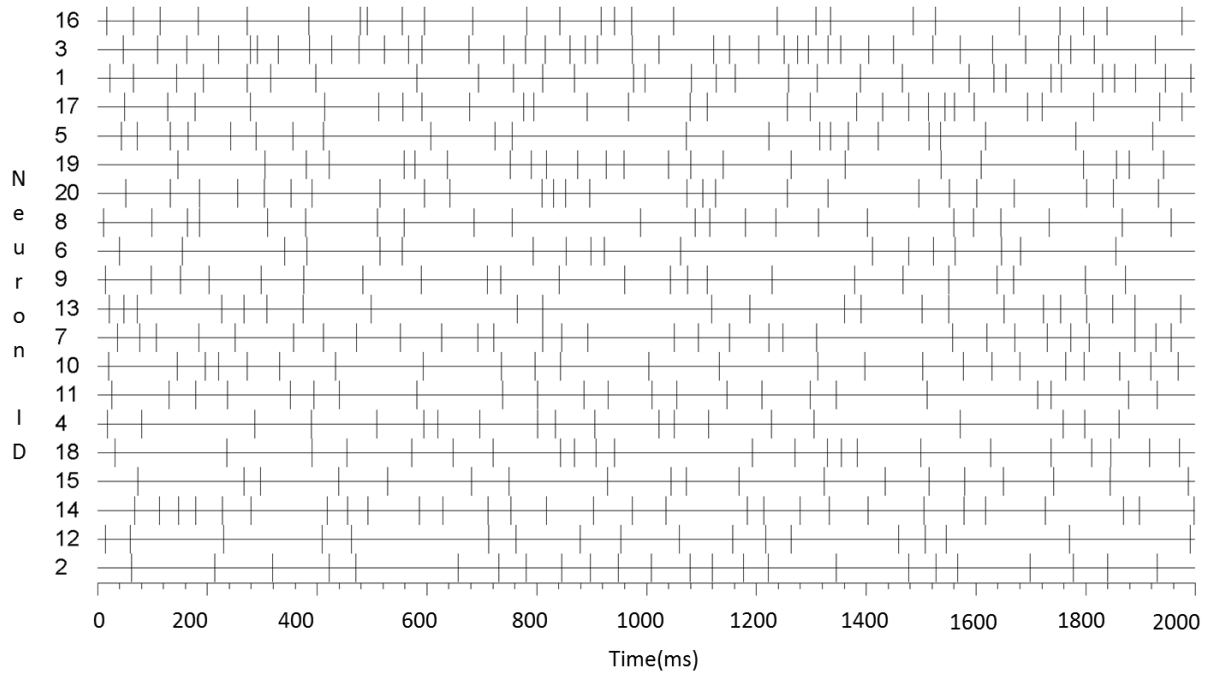


Figure 9-2: Spike trains re-ordered in the Y-axis from minimum to maximum inter spike interval (ISI).

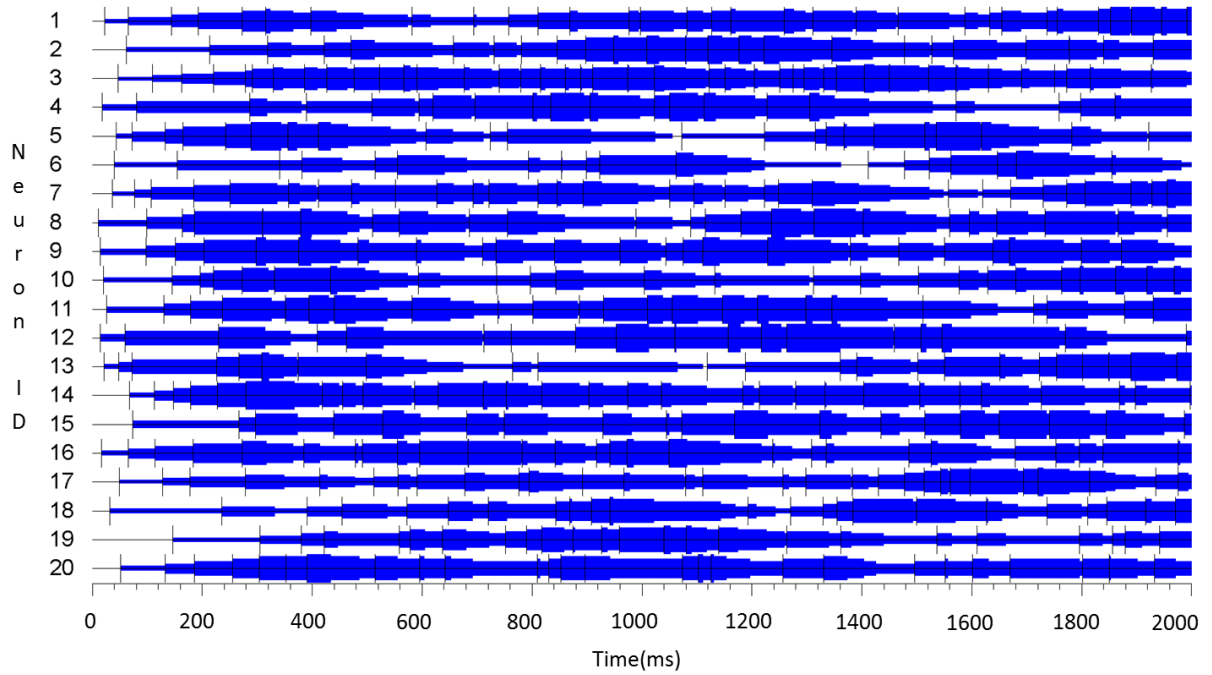


Figure 9-3: Spike train firing rate frequency overlaid onto raster chart

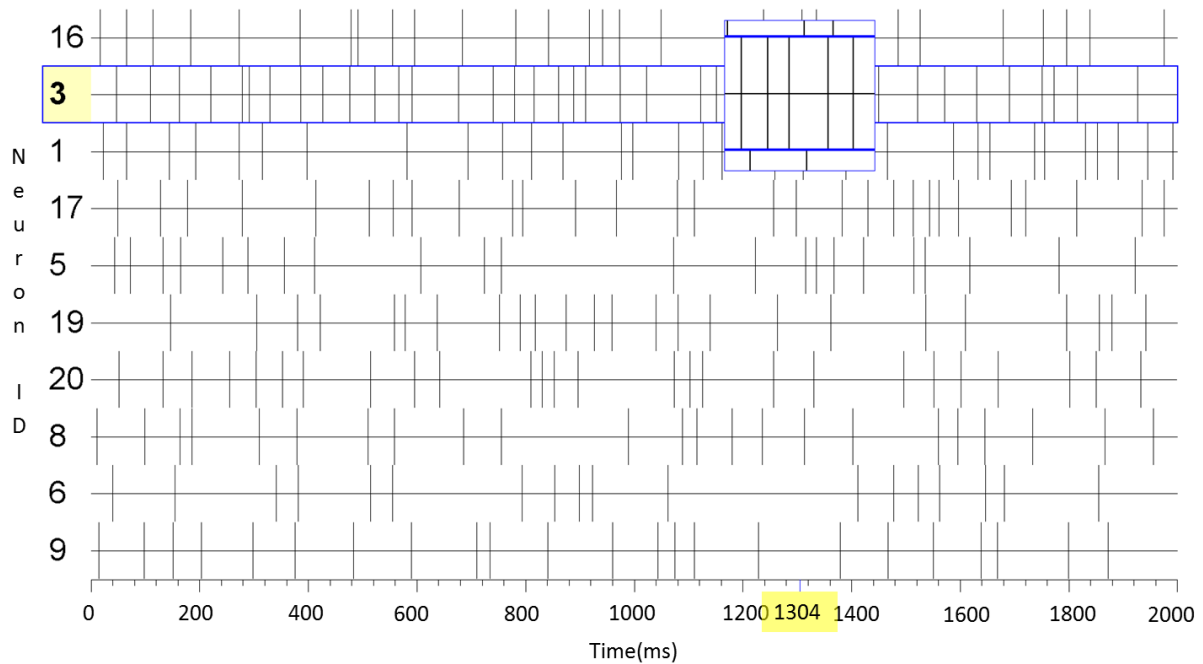


Figure 9-4: The dataset after filtering to show the 10 most active spike trains with a section zoomed to show detail.

Somerville's interactive raster chart (known as i-Raster) is a valuable tool for the examination of spike train datasets. However since its development the technology used by neuroscience to record spike train datasets has advanced considerably. The result is that dataset size has increased significantly in terms of:

- 1) The number of spike trains that can be recorded (the Y-axis of an i-Raster chart).
- 2) The time period of the recording (the X-axis of an i-Raster chart).

9.1 Drawbacks of Somerville's i-Raster Visualisation with modern neuroscience datasets

Figure 9-5 presents an i-Raster chart generated from a collaborating researcher's dataset. The recording is of 700 spike trains over a 30 minute period (this is approximately half of the 1411 recorded by the researcher). Performing any meaningful *visual analysis* of the dataset in Figure 9-5 has clearly been rendered impossible by the increased density of the dataset. Of course this is caused by the fact that the display area for the i-Raster chart has not been increased. This research has focused on how Somerville's i-Raster visualisation can be improved to once again make visual analysis possible with modern spike train recordings. This has been achieved by two primary methods:

- 1) Introduction of i-Pipeline's visual programming language (VPL) provides the opportunity to pre-process the dataset before it is visualised and
- 2) The provision of further functionality within i-Raster to manage both the increased number of recorded spike trains and the increased recording period.

In each case the guiding principle applied to create a solution has been the Visual Information Seeking Mantra developed by Shneiderman:

“Overview first, zoom and filter, then details-on-demand.” (Shneiderman, 1996)

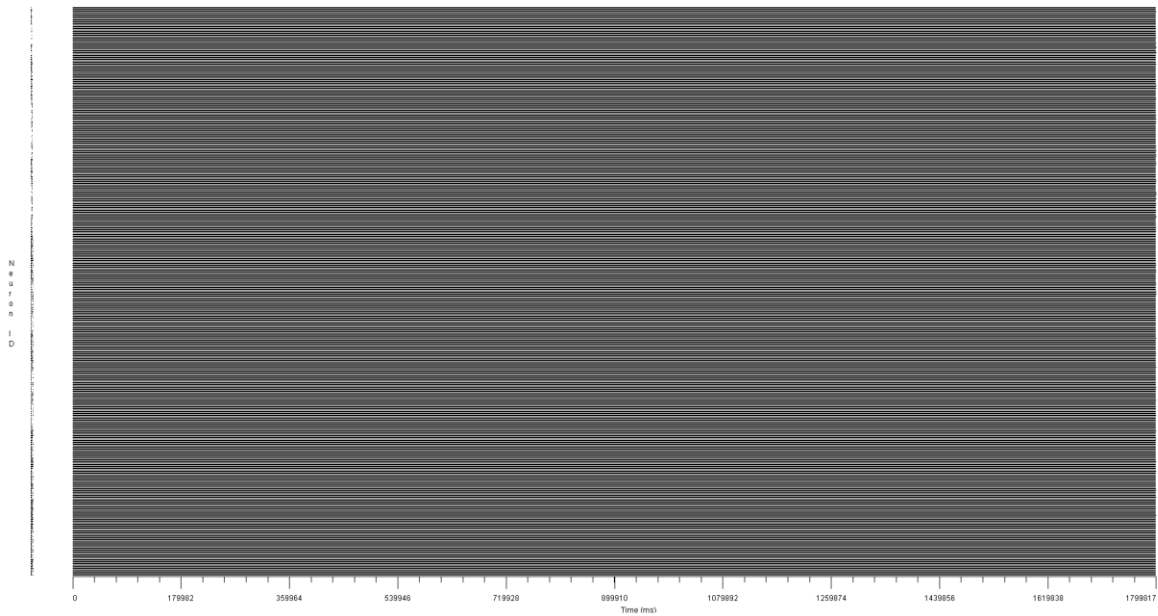


Figure 9-5: Somerville's i-Raster chart for 700 spike trains recorded for 30 minutes

9.1.1 Addressing the dataset density problem through the Visual Programming Language (VPL)

The raster charts seen in Figure 9-1 to Figure 9-5 can be represented by the simplistic VPL program seen in Figure 9-6.

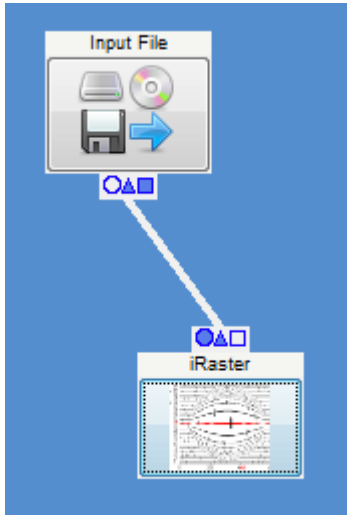


Figure 9-6: Simple i-Raster VPL program

In this program a file containing a dataset of multiple simultaneously recorded spike trains is loaded into memory and delivered directly to the i-Raster visualisation. The dataset is rendered as a raster chart by the visualisation. As demonstrated in Figure 9-5 this approach fails with modern spike train recordings.

Shneiderman advises that in cases where the on screen density of data has become overwhelming to a user an overview should be introduced. The overview should either filter or present a summary of the data. The user then has the option to access more detailed data on demand. The primary benefits of incorporating an overview element were summarised by Craft and Cairns as:

- It paints a "picture" of the whole data entity that the information visualization represents.
- Patterns and themes in the data can often be seen only from a vantage point that comprises the whole view.
- Major components and their relationships to one another are made evident.
- The overall "shape" of the data itself can provide assistance in understanding the information.
- Significant features can be discerned and selected for further examination.
- Revealing these features at the outset can aid the user in filtering the extraneous information (Craft & Cairns, 2005).

The density of the i-Raster chart can be reduced both by filtering the data and by summarising it.

9.1.1.1 Filtering the i-Raster charts data

Simultaneously recording the spiking activity of a thousand plus neurons is by its nature an imperfect process. Researchers devote considerable effort to ensuring that good electrical contact is made between the recording electrodes and the tissue sample. Despite this it is inevitable that some recording channels will suffer from "noise". This noise usually appears in the form of a very high inter spike interval (ISI) where the recorded neuron appears to be firing nearly constantly. What constitutes an excessively noisy recording channel will depend on a number of factors and can only be accessed by the neuroscientist based on the details of the sample and the recording method used. Nevertheless it is a relatively simple matter to introduce a third step (or process) into the VPL program in Figure 9-6 to filter out these "noisy" recording channels. Figure 9-7 shows the amended program configured to filter out spike trains with an average firing rate in excess of 1.5 spike events per microsecond.

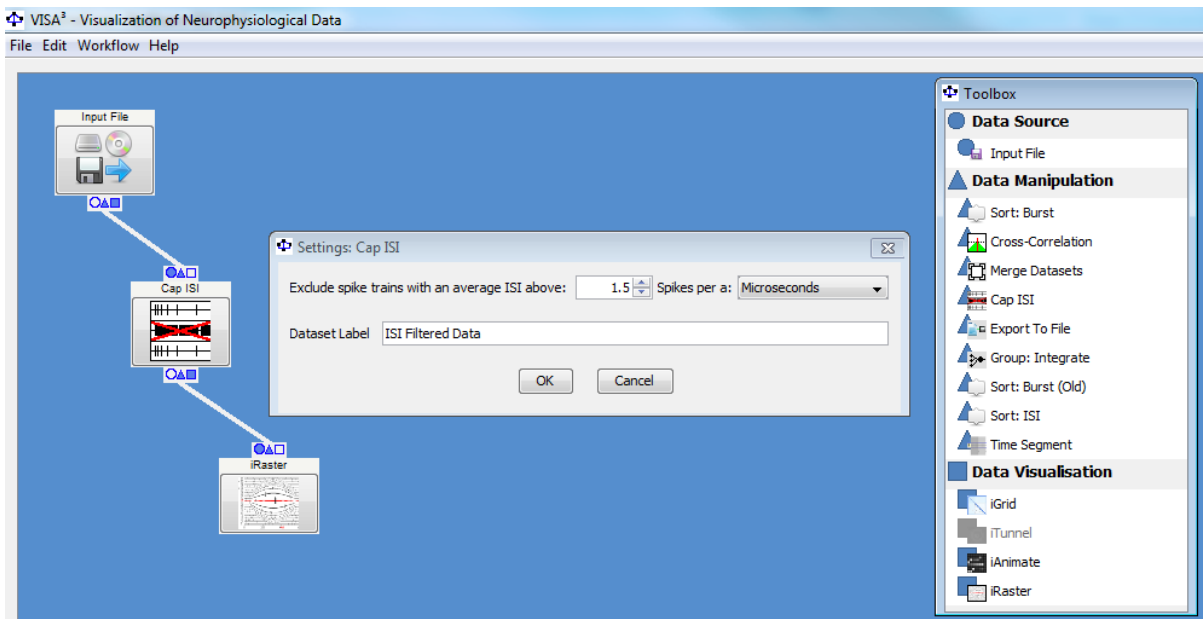


Figure 9-7: Filter to exclude "noisy" data channels (those with firing rates).

9.1.1.2 Summarising the i-Raster charts data

It is unlikely that filtering out “noisy” channel from the recording will in and of itself solve the density problem. For the VPL to be useful in solving the density problem another step (referred to as a **process**) must be introduced which generates an overview. This overview can be presented to the user instead of the overwhelming raw data. Additionally incorporating this process into the VPL would allow the summarised data to be used by other visualisations beyond i-Raster. The question that must then be asked is *what* data should be summarised to generate the overview? The available choices in the case of a neural science raster chart are:

- 1) To summarise data by the recording electrode (the y-axis data type) or
- 2) To filter by time (the x-axis data type) and present only a portion of the data.

A solution for each of the above cases has been implemented into the VPL and is described in the next section.

9.1.1.3 Case 1: Summary by recording electrode

When generating an overview using the VPL the researcher needs to ask themselves what “picture” they wish to paint of the dataset. The researcher’s visual exploration of the dataset will begin with this picture and they will explore the features revealed by it. The picture presented in Figure 9-5 has two primary draw backs:

- 1) Only half the dataset is displayed (if all 1411 are displayed Figure 9-5 is a black rectangle).
- 2) No “features” can be seen in the dataset and therefore the researcher has no point from which to begin a visual exploration of the dataset

Given that Figure 9-1 proved to be usable when presenting 20 spike trains it might be argued that the y-axis dataset is usable if limited to 20 spike trains. A “process” can therefore be proposed where multiple spike trains are combined (grouped) and rendered as a single

spike train. The grouping operation is illustrated in Figure 9-8 where the first two spike trains of Figure 9-1 have been grouped to produce a summary spike train.

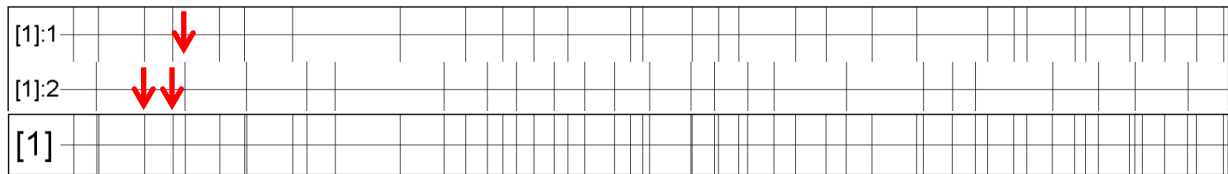


Figure 9-8: Generation of a grouped spike train that summarises two spike trains.

Applying this approach to the 1411 spike train dataset seen in Figure 9-5 would require 70 spike trains to be grouped together into a summary spike train ($1411 / 70 = 70.55$ per group). However it is inappropriate to group data simply on the basis of what is needed for a workable display. Grouping spike trains is dangerous as neighbouring cells can be doing very different things. Ultimately only the neuroscientist using their knowledge of the recording hardware and the piece of neural network that was recorded can make a sensible choice for grouping. The i-Raster chart therefore allows spike trains to be moved between groups by the researcher. After consultation with the neuroscience group that recorded the projects test dataset it was agreed that groups of 64 spike trains would be more appropriate. The recording MEA array was laid out as a 64 x 64 grid and each summary spike train would therefore reflect one “row” of the recording array

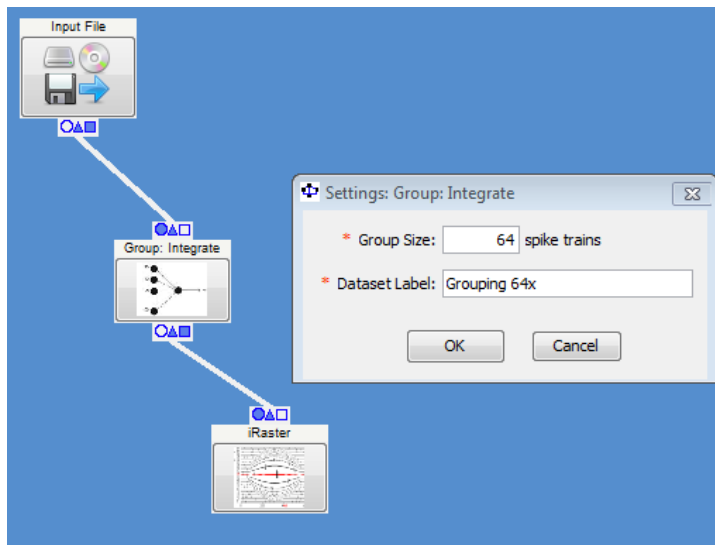


Figure 9-9: Visual program to group 64 spike trains together

Figure 9-9 shows the VPL program to perform this grouping operation. A “Group: Integrate” process has been introduced into the VPL pipeline. The process is configured to divide the dataset into groups of 64 spike trains. The processed dataset is then delivered to the i-Raster visualisation for display. Visualisation is performed as shown in Figure 9-8. The resulting raster chart can be seen in Figure 9-10. The new visualisation is considerably more useful than

the original Figure 9-5 version. The researcher is no longer confronted with essentially a black featureless rectangle. The visualisation now includes all of the data (1411 spike trains vs 700) and “features” are starting to become apparent. Distinct bands of activity have become visible which may serve as starting points for further investigation. The “shape” of the dataset (to borrow Craft & Cairns term) is also becoming apparent. The upper half of the dataset shows a highly active collection of spike trains while the lower half is less active.

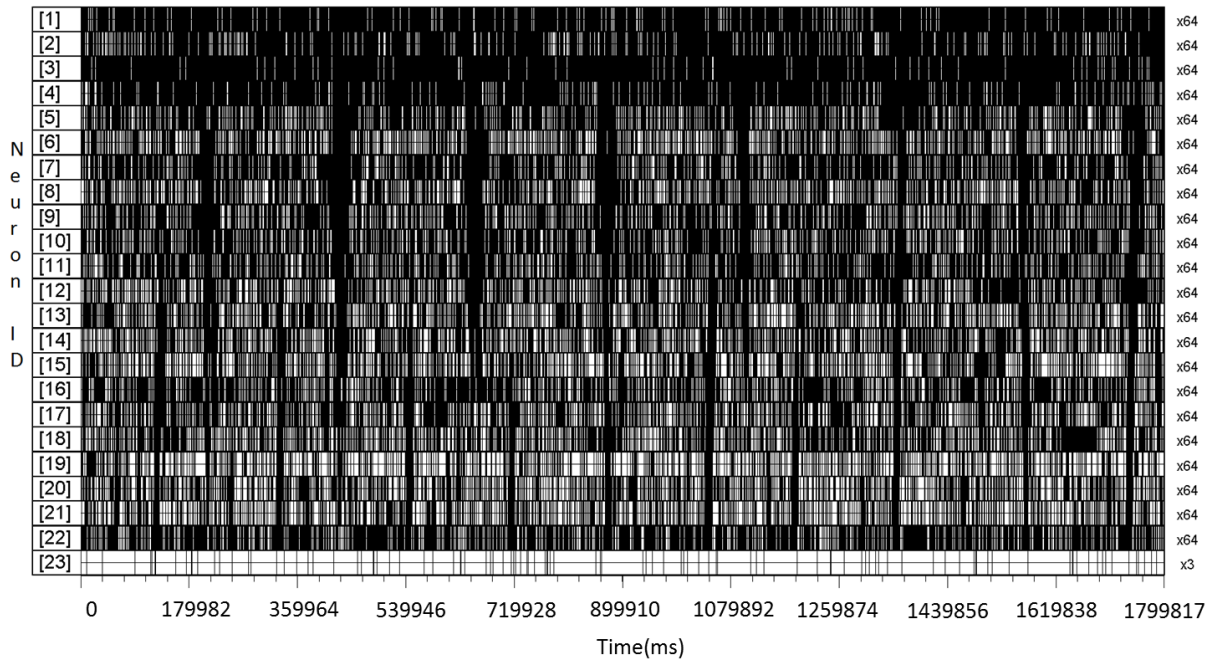


Figure 9-10: A researcher’s 1411 spike train dataset "grouped" with 64 spike trains per a group

The Visual Information Seeking Mantra also advises that it is important for the visualisations user to obtain detailed information on demand. Hence it is important that i-Raster provides some means to display the individual spike trains that compose a summary spike train. This must be done in a way that does not render the raster chart unusable. Figure 9-11 shows the solution that has been adopted.

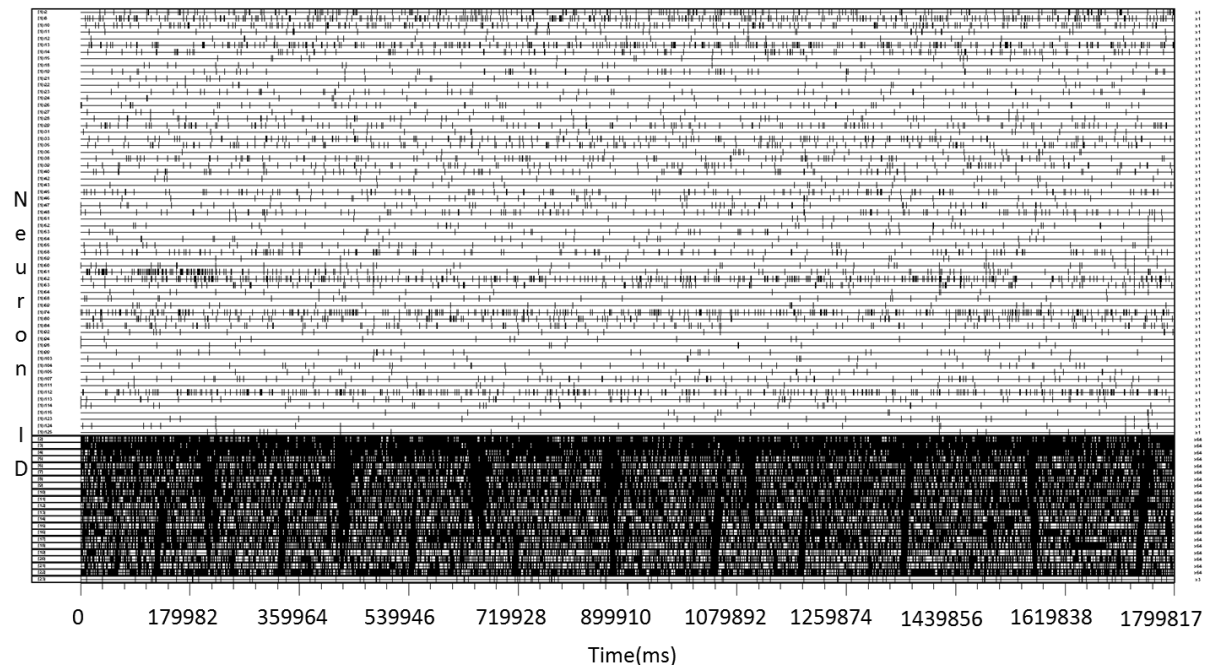


Figure 9-11: Detailed view of summary spike train one

In Figure 9-11 the user has requested a “detailed” view of summary spike train one. The summary spike train has been removed from the raster chart. It is, however, not practical to plot the 64 spike trains in the space freed by removing summary spike train one.

The result would have been as usable as the Figure 9-5 raster chart. Instead the space allocated to plotting the summary spike trains along the y-axis is considerably reduced to free additional space for the “detailed” view. It is however important to maintain a representation of the remaining 20 summary spike trains. This allows the user to maintain an overall context for the detail they are viewing (for example the spiking activity bands are still visible). The majority of the raster chart’s space is now available for plotting, in detail, the 64 spike trains of summary spike train one. Figure 9-11 clearly shows that it is possible to study firing patterns in the detailed data. The Figure 9-11 revised raster chart has remained useful to the researcher despite displaying twice as much data as the original and unusable chart (Figure 9-5). The continuing usefulness of the Figure 9-11 raster chart can be attributed to the application of two key components of the visual information seeking mantra:

- 1) Overview first – provided by summary spike trains representation of multiple detailed spike trains
- 2) Zooming – The space allocated to representing a set of spike trains increases as more detailed data is requested by the user.

The visual programming languages group integrate process (as seen in Figure 9-9) has provided a means to modify the dataset to produce a more effective visualisation. This approach can be developed further. Somerville argued that the re-ordering of spike trains in the y-axis prior to visualisation could also be used to reveal features to the researcher. In principle then it should be possible to not only group but also re-order the spike train dataset. Figure 9-12 shows the VPL with just such a sorting process introduced into it.

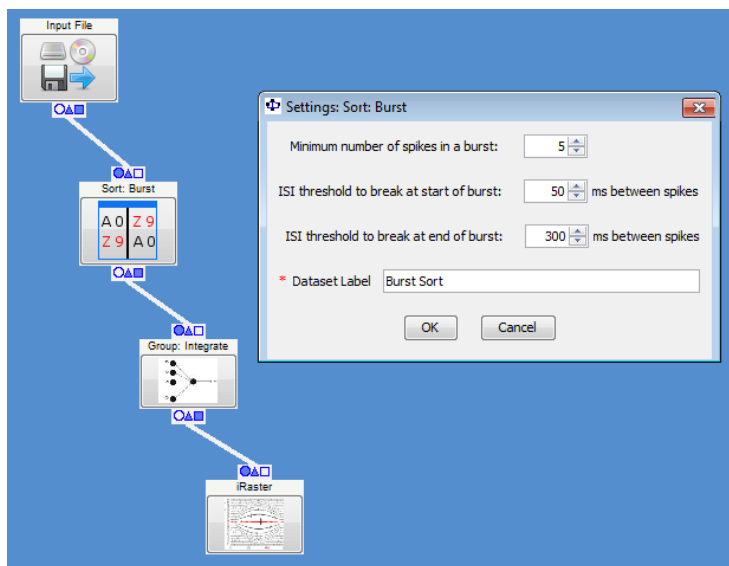


Figure 9-12: VPL program to “burst sort” spike trains prior to grouping them

After the spike trains are re-ordered they are once again grouped into collections of 64 spike trains as before (Figure 9-9). Finally the processed dataset is again visualised using i-Raster. The results can be seen in Figure 9-13 (summary view) and Figure 9-14 (detailed view). The summary view does not appear to show any major visual change from Figure 9-10 however the detailed view has revealed a new feature of the dataset. In Figure 9-14 summary spike train one has been expanded to show its 64 constituent spike trains in detail. The result is a clear feature running down the left hand side of the raster chart. Such a visual feature, where many spike trains show activity immediately after each other, is not proof of neural connectivity between the neurons that generated the

The “burst sort” process allows the researcher to re-order the spike trains based on when they first show a “burst” of spiking activity. The burst sort seen in Figure 9-12 has been configured to define the start of a burst of activity as five spike events with an average inter spike interval of 50ms. The start time is the time of the first spike in the five spike sequence. The burst is considered to end when the average inter spike interval rises above 300ms.

spike trains. It does however providing a starting point to the researcher by identifying possible target spike trains for more rigorous analysis. Somerville identified a total of seven sorting algorithms commonly used to re-order spike trains. These seven are available in the new i-Raster implementation. In addition these seven sorting methods have also been made available as VPL processes to pre-process datasets prior to visualisation.

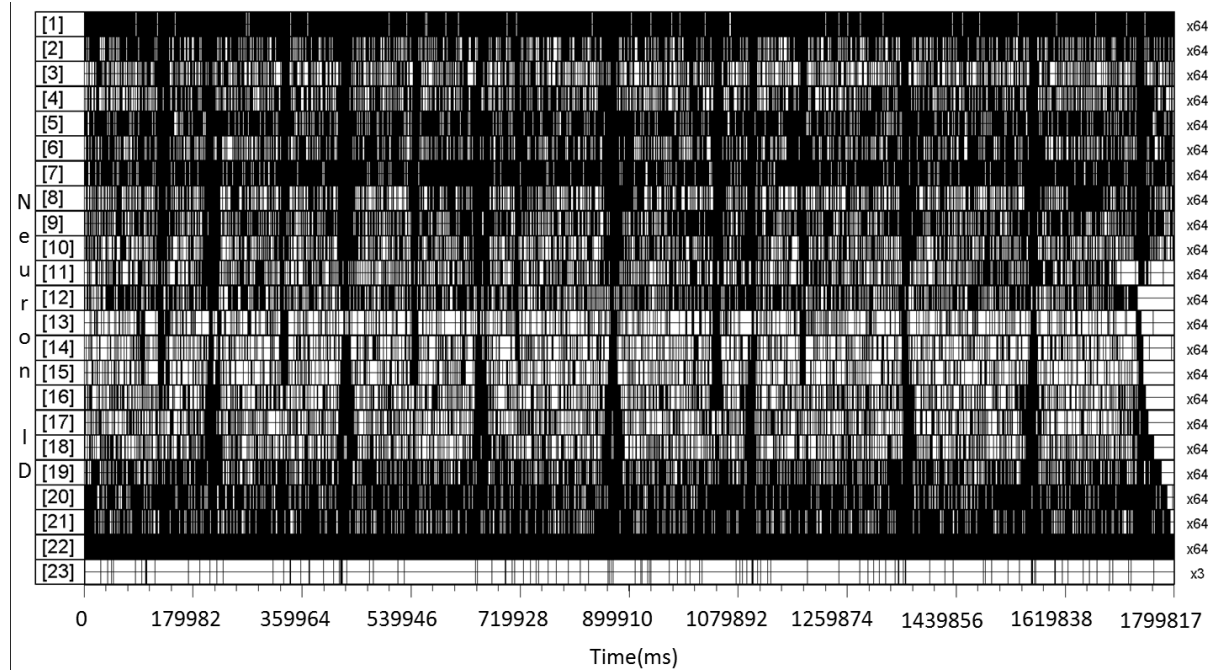


Figure 9-13: A researcher's 1411 spike train dataset burst sorted and grouped with 64 spike trains per a group

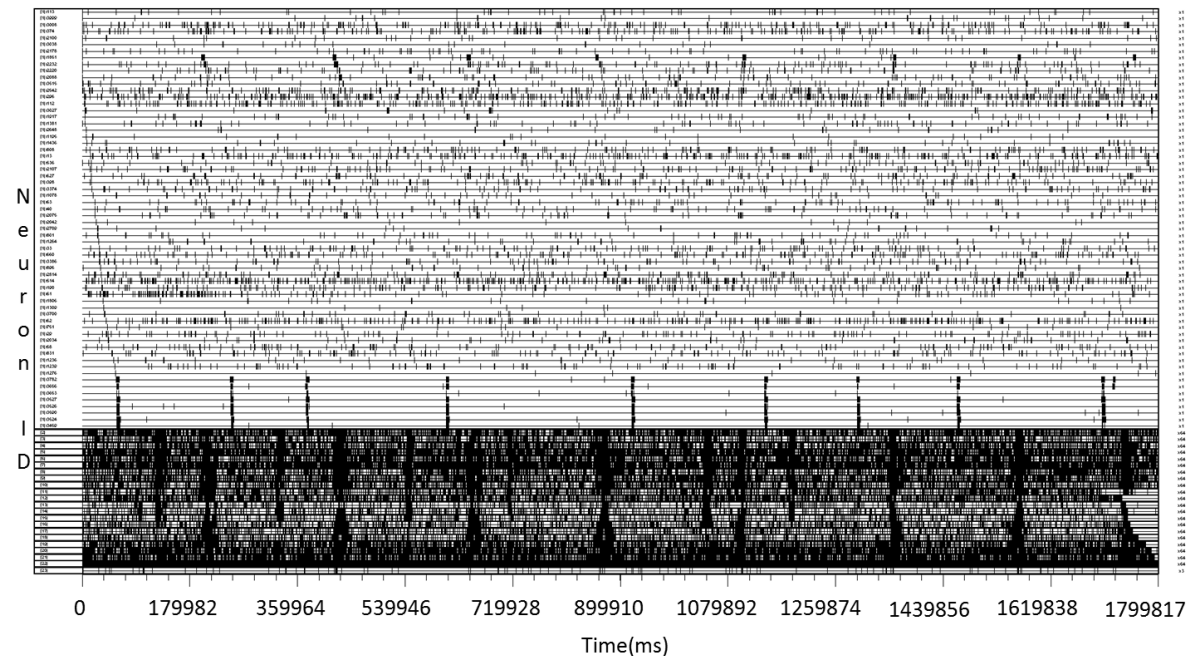


Figure 9-14: Detailed view of a burst sorted group of 64 spike trains

9.1.1.4 Case 2: Filtering data by time

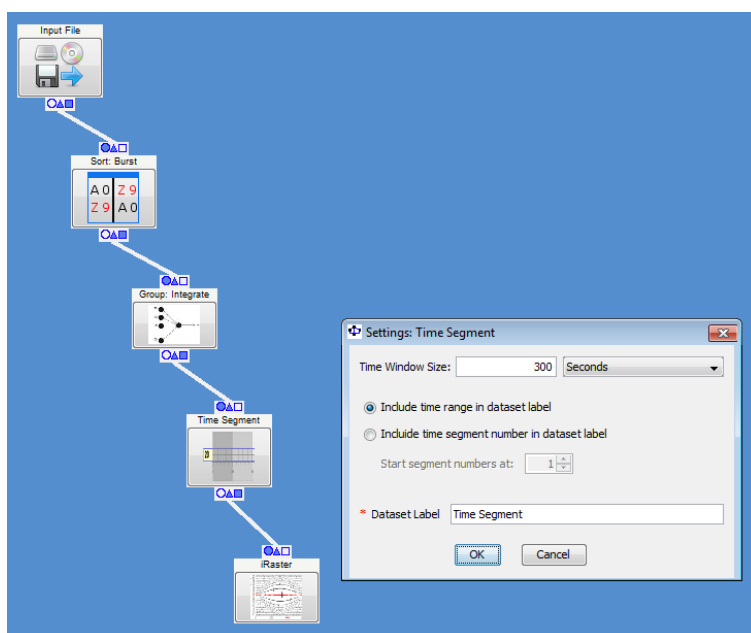
The grouping and sorting operations seen so far focus on the y-axis of the raster chart. However it would be inappropriate to ignore the other dimension of the dataset – time. Plotted along the x-axis of the raster chart it shows the same density problem experienced on the y-axis. The length of the recorded dataset has increased but the total screen space available to plot the data is unchanged. The result is clear from comparing Figure 9-13 and Figure 9-14. Summary spike train 1 in Figure 9-13 is not useful being essentially a black featureless rectangle. However when summary spike train 1 is viewed in detail (Figure 9-14) hidden structure becomes apparent. The question must therefore be asked; can the data plotted on the x-axis be visualised more effectively?

To answer this question it is necessary to consider the nature of the dataset. A neural spike train is a sequence of times at which a neuron “spiked”. A spike is an electrical signal transmitted through the neurons axon which is detected and recorded by an electrode. Generation of the signal is triggered by electrical signals received through the neurons dendrites (a stimulus). Information is encoded into a sequence of spikes. Researchers typically record two types of data in their experiments:

1. Spontaneous activity where the neural network is not stimulated and the recording simply shows the “resting state” activity in the neural network.
2. Interleaved stimulation where a different stimulus (usually light) is applied at different times. The recording shows the networks reaction to the stimulus.

Researchers make use of this by noting the time at which they applied a “stimulus” to the neural network. The activity recorded immediately after the stimulus represents the neural networks encoding of information about the stimulus. Hence rather than considering the whole recording there is value in viewing it as a series of time segments between the applications of stimuli.

The addition of a VPL that pre-processes the dataset prior to visualisation provides an opportunity to manage a dataset as a series of time segments. Figure 9-15 shows the VPL program that divides the 30 minute dataset used so far into six time segments each five minutes long.



Page 126

Figure 9-15: Time segment process added to VPL to create six five minute time segments

Figure 9-16 shows what will happen to the dataset seen in Figure 9-13 immediately prior to visualisation by i-Raster. The 300 seconds (or 5 minute) time window will divide the original raster chart into six new raster charts. After this division the x-axis will no longer represent a 30 minute time period but instead a 5 minute time period. The Visual Information Seeking Mantra would categorise this change of

scale as a “zoom” operation to reveal more detail.

Figure 9-17 and Figure 9-18 present the first 300 second time segment. In Figure 9-17 only summary spike trains are shown in the same manner as Figure 9-13. In Figure 9-18 summary spike train one has again been expanded to present a detailed view of the 64 spike trains that were grouped together to create summary spike train one (equivalent to Figure 9-14 for the whole dataset).

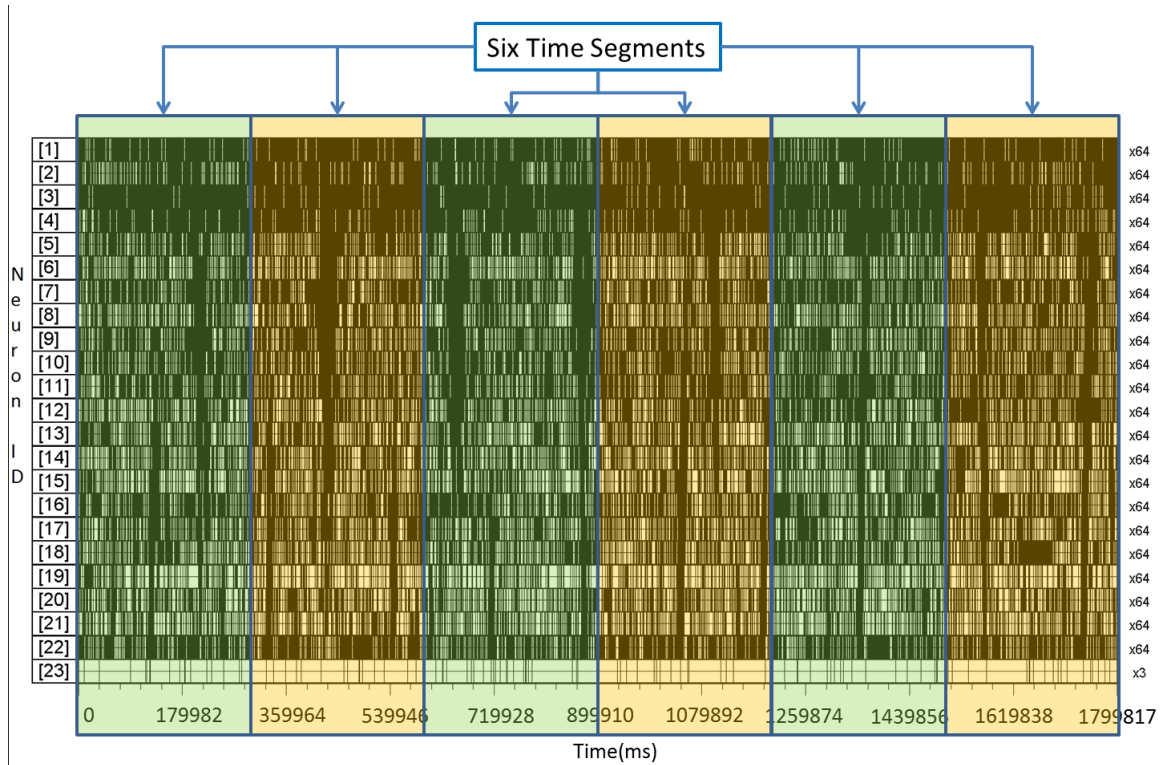


Figure 9-16: Division of the original raster chart using the time segmenting process

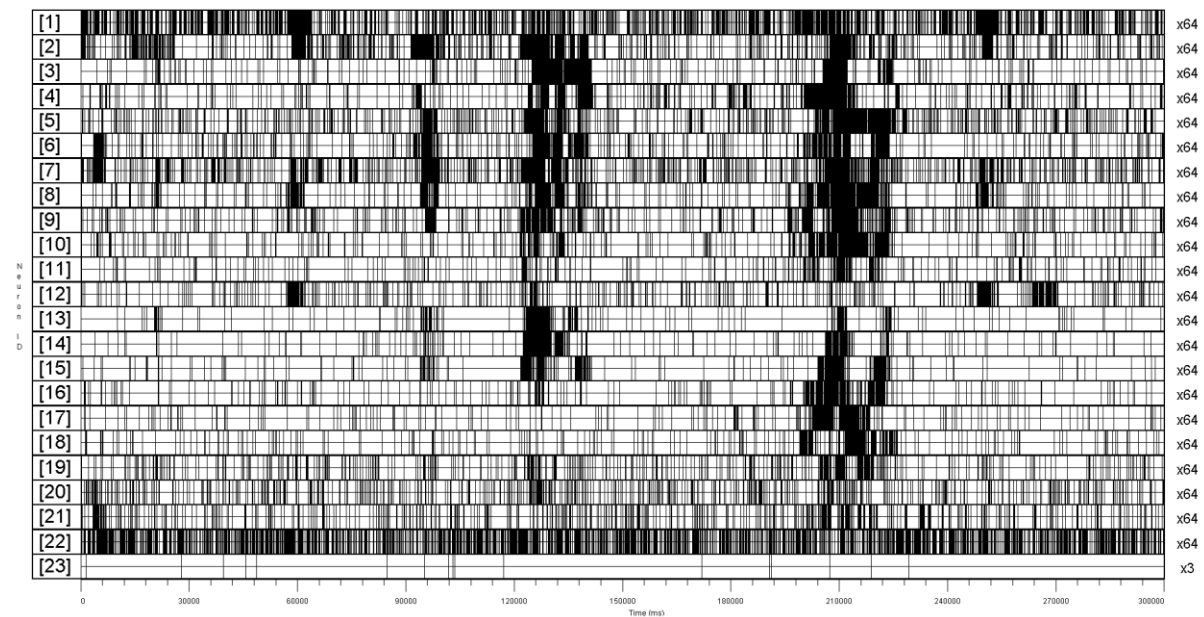


Figure 9-17: First five minute time segment of 1411 spike trains burst sorted and grouped with 64 spike trains per a group

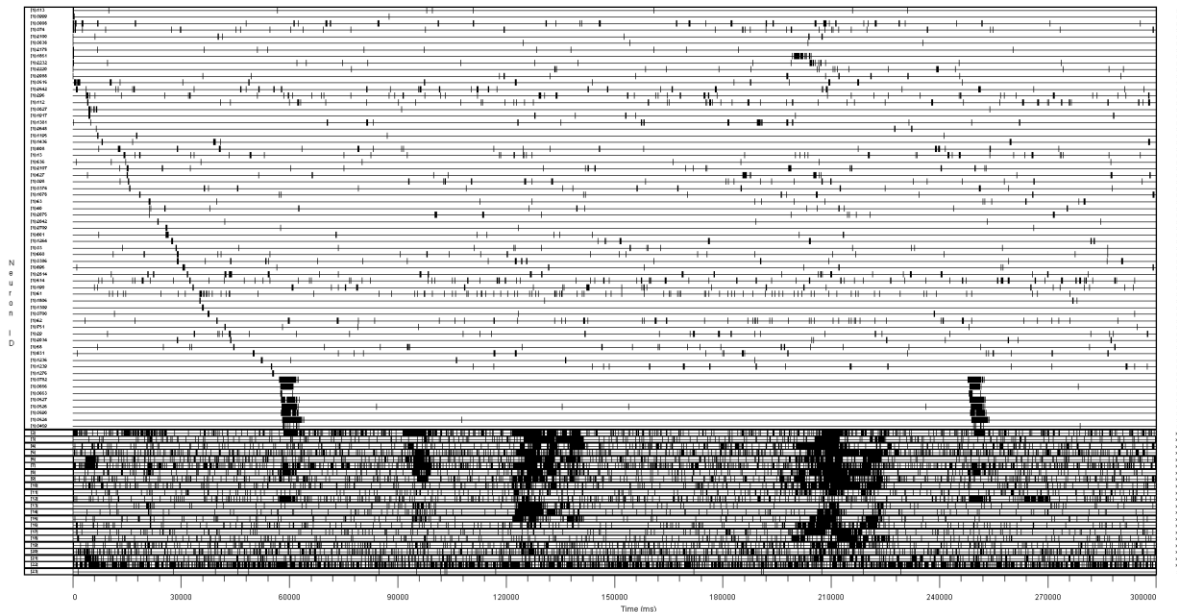


Figure 9-18: Detailed view of a burst sorted group of 64 spike trains (first five minute time segment)

Contrasting Figure 9-17 which presents the first five minute time segment on its own with the representation of that time segment shown on Figure 9-16 the benefit of time segmenting is apparent. Clearly the same features of the dataset are present. In Figure 9-17 however these features are much more visually apparent. Furthermore summary spike train 1 which in Figure 9-16 is a featureless black rectangle now starts to exhibit structure. Periods of intense spiking activity followed by periods of lower spiking frequency are now apparent.

The detail view has also revealed more information about the burst sorted spike trains. In Figure 9-14 it appeared that each spike train in summary group four began a burst of spiking activity immediately after its predecessor. Examining this feature in Figure 9-18 it is clear that this is not the case. Some spike trains do indeed begin spiking immediately after their predecessor. However in other cases there are distinct gaps of longer time periods between the first and second spike train starting to fire. Further analysis will be required to determine if these apparent groupings indicate functional connectivity in the neural network. They do however provide the researcher with a place to start.

9.1.2 Addressing the dataset density problem through additional i-Raster functionality

In addition to applying the new VPL to manage modern neural science data sets it is also possible to introduce additional functionality to the i-Raster visualisation. As with the VPL the aim of this functionality is to efficiently sort, zoom and filter sections of the dataset. This additional functionality should not be viewed in isolation as it both extends Somerville's i-Raster functionality and complements the processing performed by the VPL.

Two key pieces of functionality have been added to achieve the goal of managing increased neural dataset size:

1. A time filter that allows the idea of time segmenting a raster chart (as seen in the VPL) to become an interactive user experience and

2. A graphical representation of the multi-electrode array (MEA) used to record the spike trains. Individual electrodes or groups of electrodes can be selected and the raster chart interactively reconfigures itself to show only the spike trains recorded by those electrodes.

9.1.2.1 *The interactive time filter for i-Raster*

The usefulness of visualisation as a tool is enhanced by the modern computers ability to allow the user to interact with the visualisation. This fact is at the core of the Visual Information Seeking Mantra in the form of its advice to use filtering and zooming to explore datasets. Somerville's i-Raster implementation was primarily concerned with sorting and filtering data in the y-axis of the raster chart (by electrode id number). Using the VPL this has been extended by sorting and grouping spike trains. Within i-Raster individual spike trains may now be filtered into or out of groups and moved between groups. However the x-axis (or time dimension) offered no such filtering or zooming abilities. This deficiency has been addressed through the introduction of an interactive time filter.

In principle the operation of the interactive time filter is relatively simple. The user is presented with a scale bar representing the entire x-axis of the raster chart. A start and end slider define the beginning and end of a **time range**. Only spike events that fall between the start and end of this time range will be shown on the raster chart. Figure 9-19 shows the concept in action. The 1411 burst sorted and grouped spike trains seen in Figure 9-13 have been time filtered to show only the first 60 seconds of spiking activity.

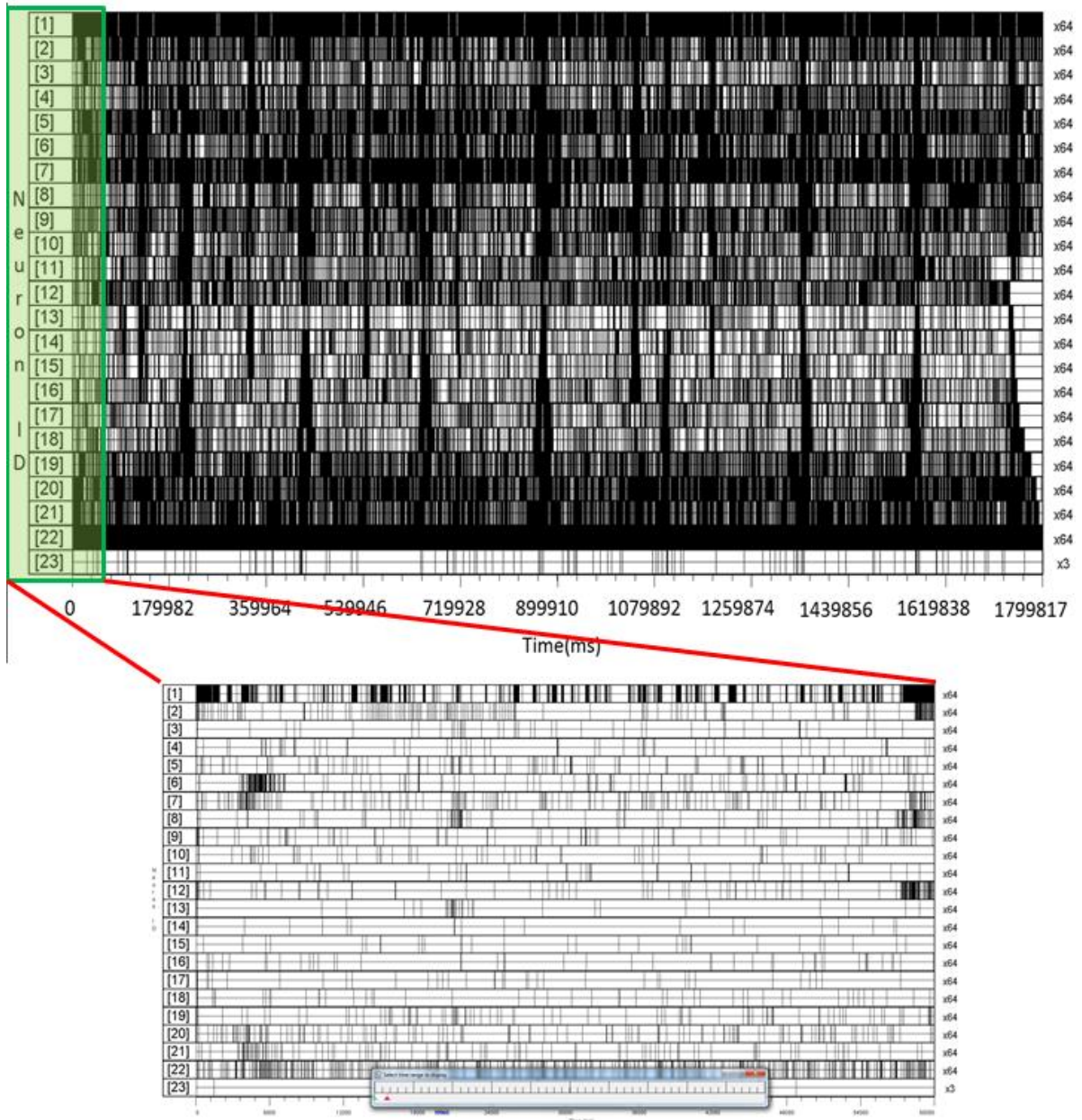


Figure 9-19: Time filtered raster chart extracted from a 1411 spike train dataset

As can be seen in Figure 9-19 the time filtering achieves on the x-axis of the raster chart what grouping does for the y-axis. By filtering out the spike train outside of the time range the scale of the raster chart is changed. This has a zooming effect allowing for a more detailed view of a smaller section of the dataset. Figure 9-20 shows the key components of the time filtering tool.

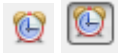
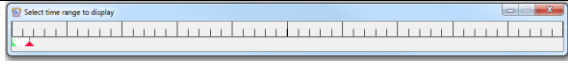


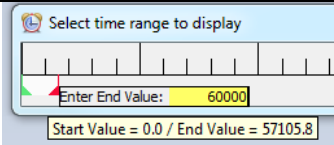
Component	Function
	Menu bar push button toggles the time range selection tool on / off.
	Time range selection tool represents the x-axis of the raster chart.
	Time range selections start slider . Colour coded green in a traffic light style.
	Time range selections end slider . Colour coded red in a traffic light style.
	Time point entry box permits entry of an exact time point value for either time range slider. Double click start or end slider to directly type a time point value for that slider.

Figure 9-20: Components of the interactive time filtering tool

Taken together all the components in Figure 9-20 allow the user to view any portion of the recorded dataset at any desired scale. The raster chart updates in real time as the user modifies the start and end range selection sliders. This allows the user to zoom seamlessly between the two views seen in Figure 9-19. The user is able to maintain an awareness of the overall dataset while concentrating their attention on particular time periods.

9.1.2.2 The multi-electrode array display (Electrode Display)

Accessed through i-Raster's main interface the electrode display provides a visualisation of the multi-electrode array (MEA) that recorded the currently viewed spike train data.

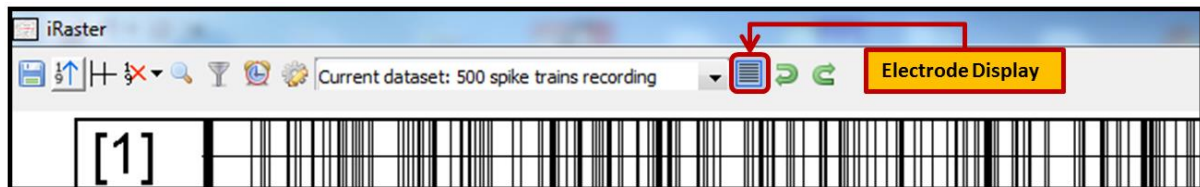


Figure 9-21: Accessing the electrode display from i-Raster's primary interface.

The increase in the number of recorded spike trains in a modern neuroscience dataset has, in part, been achieved by using ever increasing numbers of electrodes on the recording MEA devices. Neurons that are physically close to each other in a biological sample are more likely to form connections than those which are widely separated. Proximity should not be taken as a guarantee of connectivity (some neuron axon's can be over a meter long). Proximity between neurons generating the recorded spike trains does, however, provide a starting point for exploring the dataset. The electrode display provides a means to rapidly select and filter spike trains on a raster chart to show only those recorded from specific regions of the MEA.

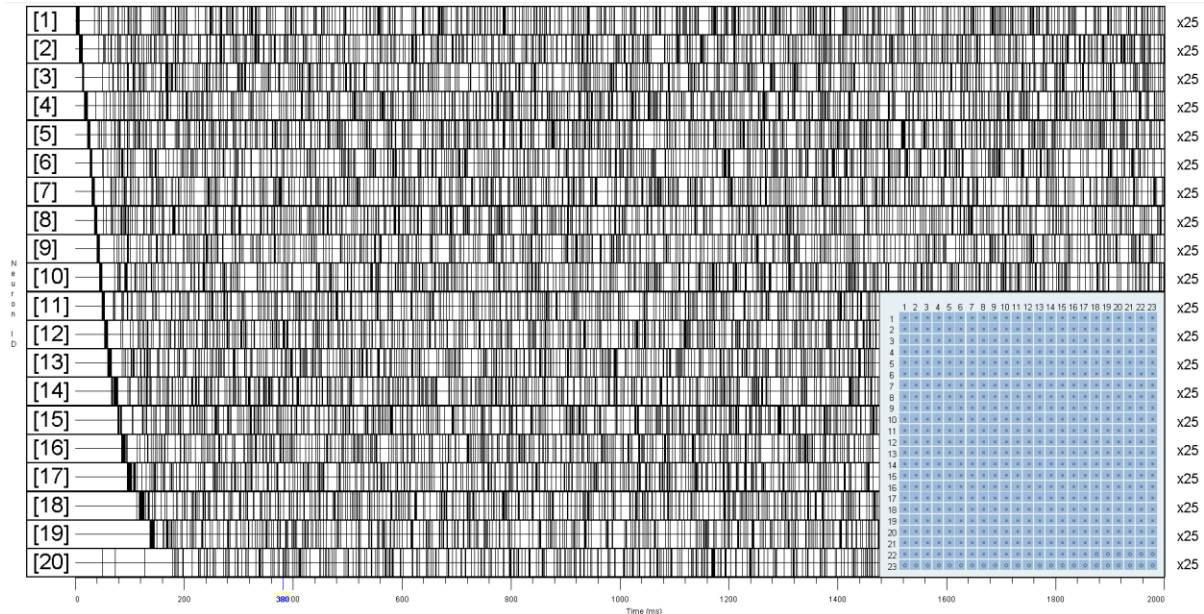


Figure 9-22: A two second burst sorted recording of 500 spike trains with the electrode display shown.

Figure 9-22 shows a two second burst sorted recording of 500 spike trains. The electrode display has been activated and since the whole dataset is being viewed all electrodes are selected. The VPL program that loaded this dataset is identical to the one seen in Figure 9-12 earlier. To filter the dataset to show spike trains from a single region the user simply drags a selection rectangle around the area. Multiple regions of the MEA can be selected by holding down the right control key while selecting. When individual spike trains are being displayed (rather than summary groups) placing the mouse over an electrode will highlight the corresponding spike train. Figure 9-23 shows the same dataset filtered to show data from three regions. One of the summary spike trains has been expanded and the electrode that recorded spike train 101 identified.

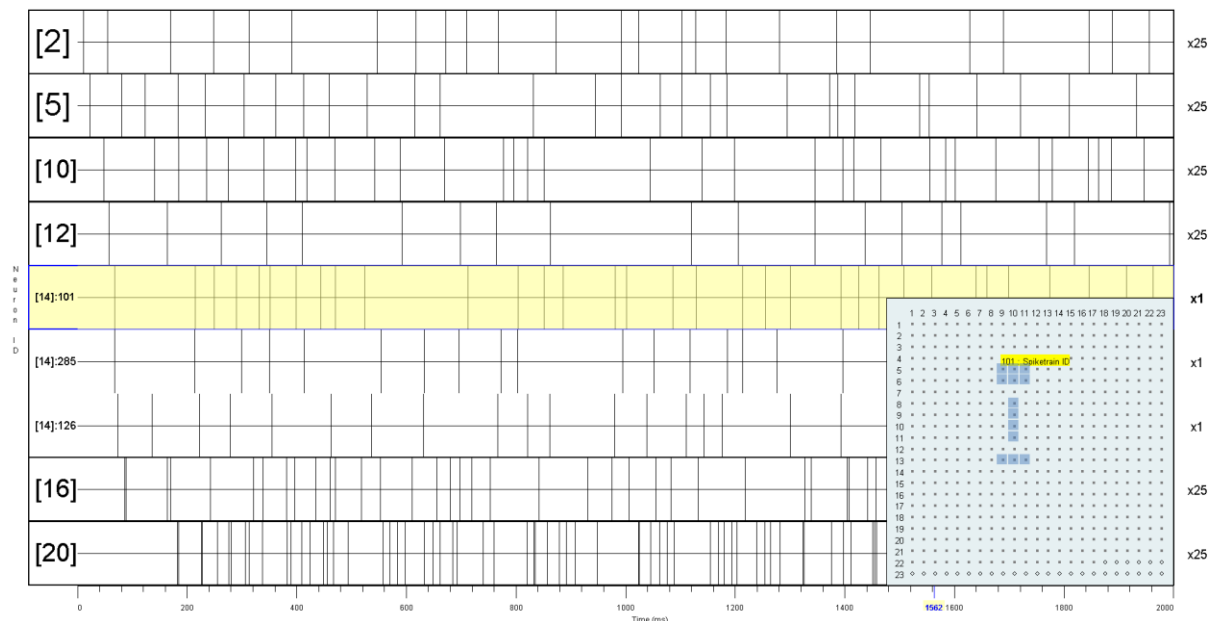


Figure 9-23: Electrode display filtering a dataset by recording electrode.

Chapter 10

The i-Grid Visualisation

“Grid noun. A network of lines that cross each other to form a series of squares or rectangles”

Summary

This chapter describes the iGrid visualisation its implementation and the visualisation techniques add to it to manage the large datasets now being produced by recording hardware.

10 Overview of the i-Grid Visualisation

The i-Raster visualisation demonstrated how potentially connected neurons could be identified from a visual inspection of spike train recordings. This chapter considers the i-Grid visualisation which provides a means to make this assessment for the entire data set. In addition challenges exist in scaling the grid to cope with the increased data set size generated by the modern multi-electrode recording array (MEA). In a similar manner to i-Raster the provided implementation of i-Grid includes a number of new features aimed at managing the growing data size. Again Shneiderman's Visual Information Seeking Mantra – "Overview first, zoom and filter, then details-on-demand" is applied so users can extract useful information by visual inspection.

Before the new i-Grid implementation can be examined in detail it is first necessary to understand the mathematical theory that underpins the grids operation. The generation of the grid rests on the creation of spike train pair-wise cross correlations as a metric to determine functional connectivity between neurons. The metric is then applied to order and group spike trains into clusters. The clusters of inter-connected spike trains reflect the underlying functional connectivity between neurons in the recorded neural network.

10.1 Overview of Pair-wise Cross Correlation

Cross correlation is a commonly used mathematical technique to identify recurring patterns in recorded signal data. In i-Grid's case this signal is the recorded neuron spike train with the researcher looking for repeating patterns. Cross correlation is a 'binning process' that generates a histogram describing the degree of similarity between spike trains. The procedure to generate a pairwise cross correlation is as follows:

- i. Select a pair of simultaneously recorded spike trains. To compile a meaningful i-Grid visualisation all recorded spike trains must be paired with all other spike trains in the data set.
- ii. Determine a time window and bin size. The exact determination of these factors must be made by the researcher based on the data set being examined. Key factors in making these decisions are:
 - a. The length of the spike train recording. Cross-correlation is a statistical technique that requires a reasonable data sample in order to be effective.
 - b. The spike train firing rates should be similar between the compared spike trains. The i-Raster visualisation provides a firing rate display for recorded spike trains to visually verify this.
- iii. One of the selected spike trains is designated as the reference spike train and the second as the target spike train.
- iv. For each spiking event on the reference spike train the target spike train is examined. The centre of the time window is placed on the target spike train at the same moment in time as the reference spike occurred. The bin size divides the time window into a number of "bins". The total number of spike events on the target spike train is summed for each bin and then added to a running total for each time bin. Figure 10-1 illustrates the procedure.
- v. The aggregate totals are then "normalised" so that the average bin count value is 1. This normalisation, known as "Brillinger normalisation" allows the program

to make meaningful statistical judgements about the relationship between the target / reference spike trains (Brillinger, 1979). The normalisation allows the visualisation to establish a threshold for a significant degree of correlation between the reference and target spike trains. Spike trains showing a statistically significant degree of correlation will be clustered together in the final visualisation. The threshold is usually set at about two standard deviations from the mean.

- vi. The usual visual representation of a pair-wise cross correlation is as a histogram showing the normalised bin counts for the time window bins (called a Cross-Correlogram). All Cross-Correlogram peaks in excess of the Brillinger threshold denotes a significant cross correlation with the strongest peak providing a measure of correlation between the reference and target spike trains.

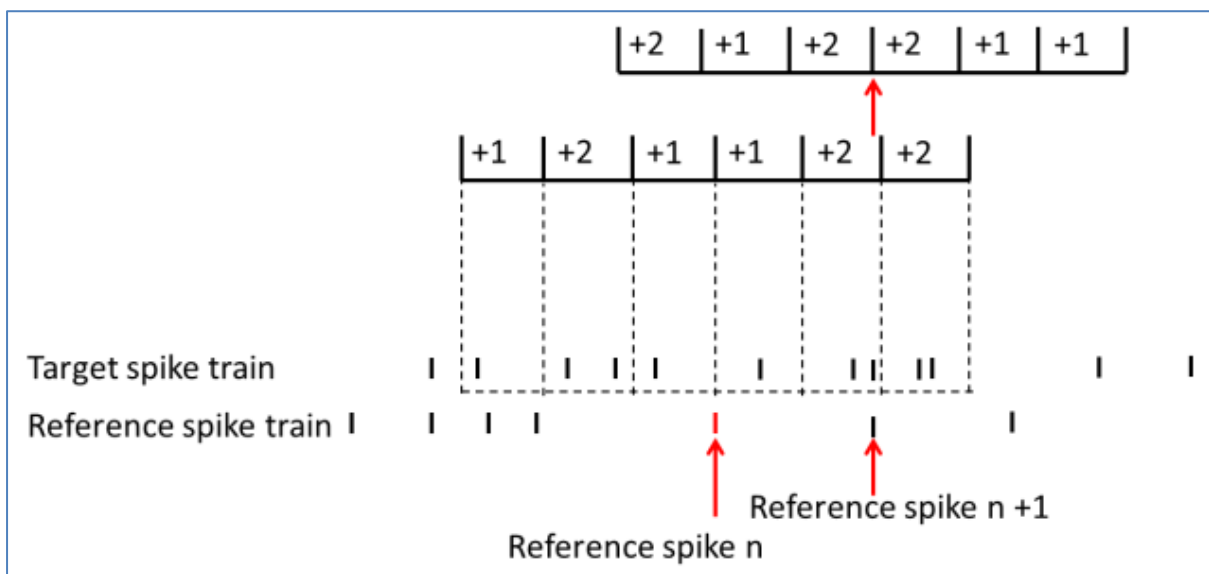


Figure 10-1: Generation of time bin totals in pair-wise cross correlation

10.1.1 Computational challenge of Pair-wise Cross Correlation

The generation of any individual cross correlation between two spike trains may not appear computationally challenging. However the scientific communities' ability to record spike trains is rapidly growing both in terms of the recording length and the number of simultaneously recordable spike trains. This translates into an increasing number of data points (growth in recording length) and an increase in the total number of cross correlation pairs. Within the project both of these challenges need to be addressed in order to develop a scalable solution that cope effectively with current recording technology and near future technologies.

Chapter 8 described how the challenge of increased recording length has been met by employing sparse arrays and red / black binary trees to manage the growth in data points. However if the project is to achieve the goal of a scalable solution it must also address the growth in the number of recorded spike trains. The pair-wise cross correlation process rapidly becomes computationally intensive as the number of pairs in the data set grows. The number of cross correlations that must be completed to identify all statistically significant relationships within a data set is given by:

$$\frac{n^2 + n}{2}$$

Where n = the total number of neuron spike trains recorded in the data set.

Graphing this function illustrates the rapid growth in computational load as data set sizes grow. Figure 10-2 illustrates the number of pair-wise cross correlations that must be computed for data set sizes between 0 and 2000 recorded neurons:

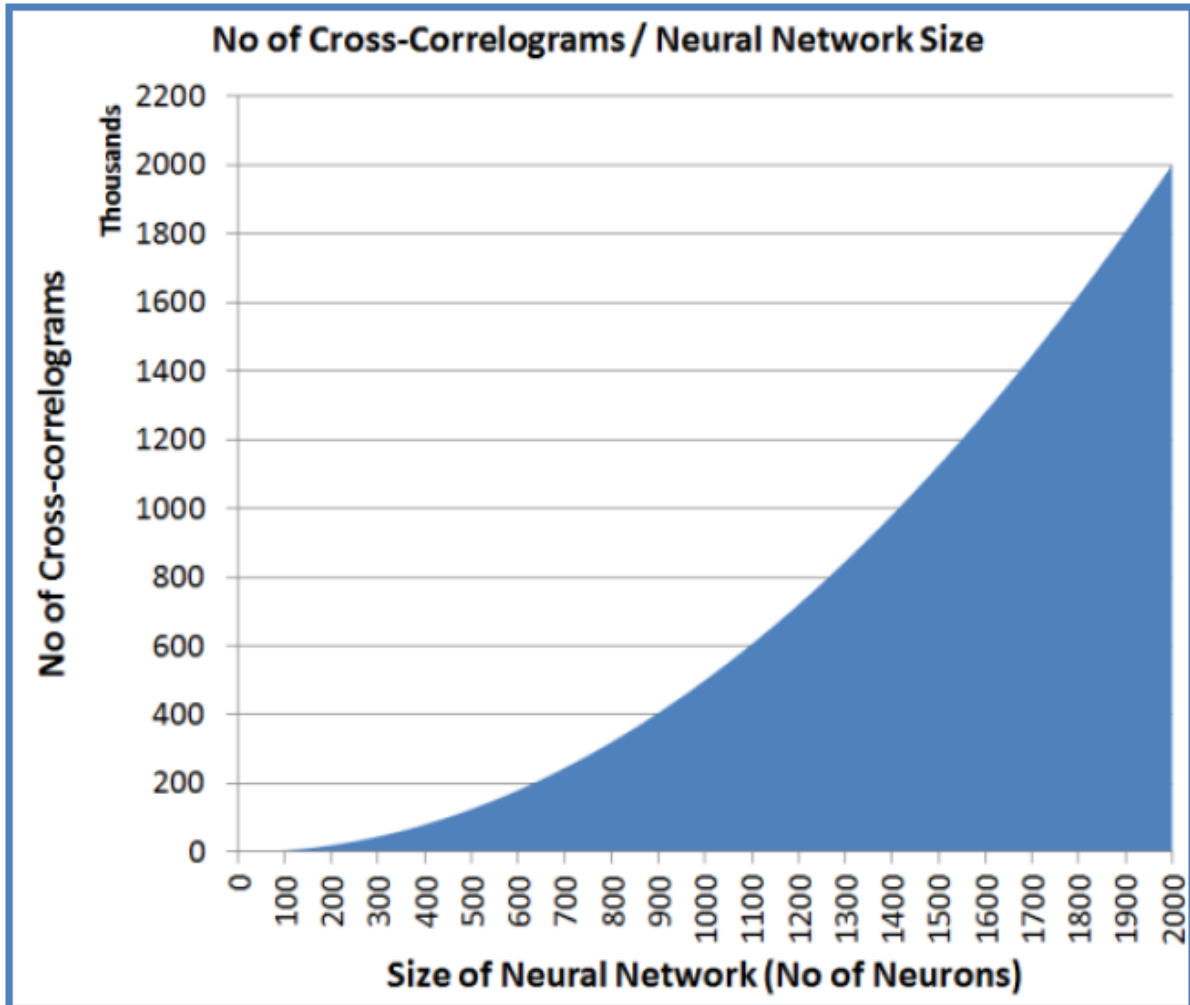


Figure 10-2: Number of required cross correlation calculations for neuronal networks up to 2000 neurons in size.

Figure 10-2 shows that a data set of 2000 neurons requires some 2,001,000 cross correlations to be performed. Each of these can in turn contain hundreds or thousands of data points (neuron spiking events). In contrast the number of calculations required for the smaller data sets traditionally used with the VISA software (approx. 200 spike trains) was 20,100 cross correlations. Hence a growth of 10x the number of recorded spike trains had led to nearly a 100x increase in the computational load measured in terms of the number of cross correlations to perform. The growth in computational load is then further compounded by the increase in recording length and the expectation of further growth in the number of recorded spike trains as technology advances.

To meet the explosive computation demands the new implementation of i-Grid adopts two approaches:

1. Raw spike train data is pre-processed into a set of highly compact cross correlation data composed of normalised bin counts. Once compiled significant system memory resources can be freed with the i-Grid visualisation operating from the normalised bin data.
2. The pre-processing algorithm has been designed from the ground up to function as either:
 - a. A multi-threaded program that exploits all of the available compute cores of the executing system OR
 - b. Exploits the Java implementation of the Message Passing Interface (MPI) to distribute its operation across a high performance computer cluster (HPC). The MPI implementation used is MPJ Express (Shafi & Jameel, 2006).

Each of the above approaches attempts to minimise the memory footprint of the i-Grid data model (item 1) or maximise the utilisation of the systems computing power (item 2). The supporting technology in the form of MPJ Express does introduce an external dependency were other visualisations in the VISA suite operate on a standard Java install. This limitation was accepted as necessary to scale the i-Grid visualisation to meet increasing computational demands.

10.1.2 MPJ Express as an implementing environment for i-Grid's cross-correlation algorithm.

MPJ Express is an implementation of the Message Passing Interface (MPI) standard developed for distributed computing using a high performance computer cluster (a HPC cluster). The Java language itself offers no official MPI bindings however various attempts at creating such bindings by the academic community exist. Technologies considered for developing the cross-correlation algorithm on a HPC included:

1. The HP Java Project (Carpenter, 2007): The original implementation for bridging the Java – MPI gap. It was rejected as it sacrificed the code portability of Java between platforms. Essentially a wrapper around a C implementation of MPI accessed via Java's Native Interface (JNI) technology it would have required the C library to be recompiled for each platform. Nevertheless this project did define the mpiJava interface that has become a de facto standard used in other MPI – Java projects
2. Open MPI (*Open MPI: Open Source High Performance Computing*, 2014): this implementation of the MPI standard provides a Java interface primarily at the request of the Hadoop development community. Development team is composed of many leading industry companies however it was rejected as the Java bindings are not formally part of the project and will be included in future releases only so long as there is active demand from the Hadoop community.

Ultimately MPJ Express was selected for use based on its unique feature of not being limited to use on high performance clusters. Instead the implementation offered two modes permitting use on laptop / desktop systems outside of a HPC cluster as well as a true HPC mode. In the first instance Java threads replace the individual compute cores of a HPC cluster. These threads are then scheduled by the operating system to exploit however many

compute cores the local machine offers. In the full HPC mode the more classic distribution of tasks across the cluster's compute nodes occurs. MPJ Express serves as an abstraction layer that hides the details of the operating mode from the code allowing code written to the mpiJava interface to operate seamlessly between multi-core and HPC cluster systems.

10.1.3 Implementation issues of i-Grid's cross-correlation algorithm.

Figure 10-3 below provides a visual representation of the cross correlation algorithm as it has been implemented in the revised i-Grid design.

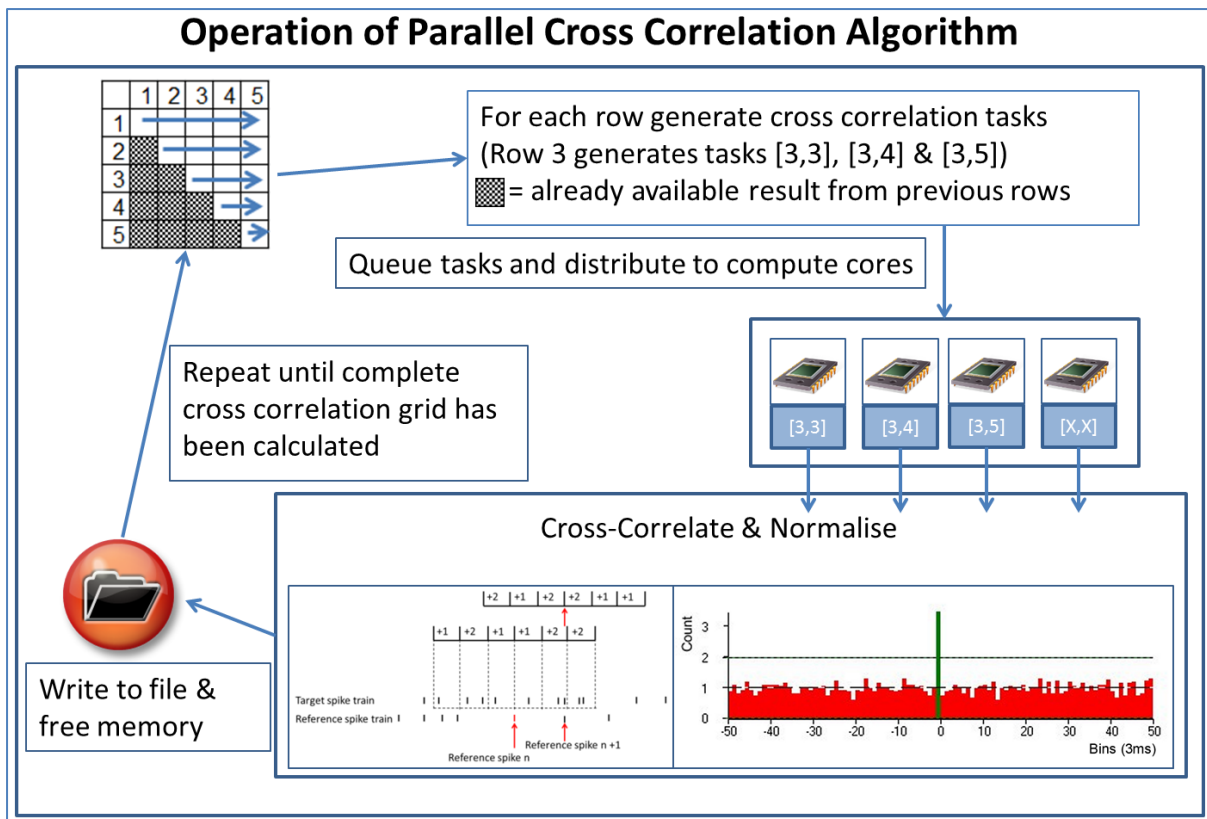


Figure 10-3: Operation of i-Grids MPJ Express implementation of the cross correlation algorithm

Development of the cross correlation algorithm passed through several prototypes. The primary development issue was the need to accommodate execution on both a HPC and more regular research machines. It became impossible to predict the resources that would be available at run time (memory was a key issue) yet a key development goal was that the algorithm should adapt to the execution environment. Writing a scalable algorithm that could process large data sets in both the HPC and classic desktop / laptop scenario proved challenging. The HPC, of course, had access to far more memory resources than the typical desktop system and the third prototype revised its approach so that the memory load actually reduced as the algorithm executed. This proved the key to scaling to large data sets on systems with more limited memory resources. A naive implementation would simply generate a cross correlation task and assign it to a task queue for completion. This is highly inefficient in terms of memory as both the large raw spike train recording and the processed cross correlation results must be held in memory simultaneously. Rather than attempting to hold all this data in memory at the same time the algorithm processes cross correlation for each "row" of the i-Grid representation.

Figure 10-3 describes the final implementations operation using a small data set for clarity. A data set of 5 spike trains is presented as a 5 x 5 grid for processing and the number of required computations per Figure 10-2 in such a grid is 15. Initially all five (5) spike trains must be loaded into memory and the first “row” of data (5 spike trains) generates a task queue of five cross correlation tasks. The queued tasks a distributed to processing cores (HPC cluster mode) or to worker threads (multicore mode). Completed cross correlation tasks are returned and written out to disk. Each complete cross correlation task generates the nested hash map structure shown in Figure 10-4. The hash map structure is readily storable as a JSON string and it is compressed to this format before being stored to disk.

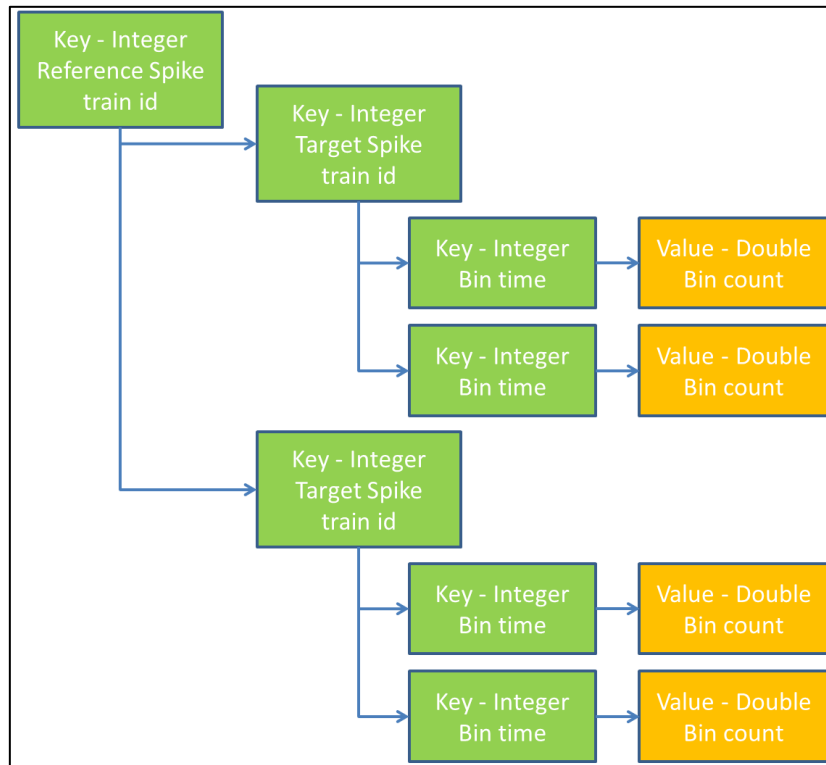


Figure 10-4: Cross correlation results as a nested hash map structure

Once stored the cross correlation is deleted from memory along with the spike train that was cross correlated with its self as it will not be involved in further computation. While on the small scale of a 5 x 5 grid this removal frees a relatively small amount of memory the impact is very significant for larger grids and long spike train recording times. Unlike earlier implementations which saw an explosion in memory use in line with Figure 10-2 this implementation actually reduces memory load as the calculations proceed. Of course the memory burden has been off loaded to the hard disk drive and long term data storage. However these systems provide considerably more storage capacity than RAM. This lifts the threshold in terms of the number of spike trains that can be cross correlated while making the limiting factor for computation the easily expandable available data storage capacity (as opposed to the more fixed available RAM).

10.1.4 Algorithm performance – Lab PC vs High Performance Computing (HPC) cluster

The cross-correlation algorithm has been tested on both a standard lab PC and on Plymouth University's high performance computer cluster (HPC). The performance of the algorithm in each environment is detailed in the relevant sub-section below.

A series of test datasets with the neuron count ranging between 50 and 2000 neurons were generated using a Neural Network Simulator (Borisyuk, 2002). Table 10-1 details the dataset sizes and expected number of cross correlations for each test run.

Neuron Count	Cross Correlation Tasks
50	1275
100	5050
150	11325
200	20100
300	45150
400	80200
500	125250
1000	500500
2000	2001000

Table 10-1: Test dataset sizes and number of cross correlations to be performed.

For each dataset the neural simulator generated a two minute simulated recording of spike train data from the network with millisecond resolution for spiking events. Each recording therefore covers a 120,000 millisecond time period.

10.1.4.1 Lab PC Cross-Correlation algorithm performance

The specification for the lab computer used to test the algorithm is as follows:

Item	Description
Processor	Intel Core i7-2600 CPU @ 3.40GHz
Installed Memory	16.0GB
Operating System	Windows 7 Enterprise (64 bit) Service Pack 1

Table 10-2: Lab PC Specification

The primary measure of performance will be the number of cross correlations completed per a second. It is expected that smaller datasets will generate a lower rate of cross correlations per a second as the initialisation and in memory management of data will be a significant portion of the total time spent performing the cross correlation task. For larger dataset the cross-correlations per second rate is expected to rise before become fairly stable as the overhead becomes a smaller percentage of the total compute time. The exact point at which this stabilisation occurs will depend on the hardware used and number of compute cores available. Hence as dataset sizes grow a point of diminishing returns will be reached after which the computation rate becomes static and the total time to complete a run will depend primarily on the number of spike trains being cross correlated. The Intel i7 CPU core used in this test contains four cores which exploit Intel's hyper threading technology to offer eight virtual compute cores.

The results generated by testing the cross-correlation algorithm in this environment are detailed in Table 10-3 below:

<u>Neuron Count</u>	<u>Cross Correlation Tasks</u>	<u>Total Time (Sec)</u>	<u>Per second rate</u>
50	1275	21.84	58.37
100	5050	83.05	60.81
150	11325	172.93	65.49
200	20100	305.34	65.83
300	45150	680.24	66.37
400	80200	1201.63	66.74
500	125250	1872.54	66.89
1000	500500	7445.04	67.23
2000	2001000	29744.07	67.27

Table 10-3: Lab PC Cross-Correlation rates for various sized datasets

As expected the performance of the lab PC plateaued with datasets in excess of 150 spike trains. For dataset sizes of 150 spike trains or less the average number of cross correlation tasks completed per a second was 61.56. This contrasts with the 200 – 2000 spike train range where the completed cross correlation tasks per a second was 66.72. The average across the entire range was 65 cross-correlations per a second.

10.1.4.2 Plymouth University High Performance Computing (HPC) clusters performance

To demonstrate that the cross-correlation algorithm fully utilises the computational resources made available to it the tests performed on the lab PC were then repeated using the high performance computing cluster at Plymouth University. The test run was performed on the universities Fotcluster1(Mills, 2013). Fotcluster1 is a 376 core distributed-memory cluster which comprises of:

1. A ViglenHX425Hi HPC combined head and storage node, plus 46 compute nodes.
2. Each compute node is a single U Viglen HX224i, equipped with Dual Intel Xeon E5420 (Harpertown) Quad Core 2.50Ghz processors and 8 GB of memory per motherboard.
3. Each node is connected via a 3com 3870 - 48 port network switch.
4. A total of 10TB of storage capacity is available.

For the purpose of this experiment a total of 32 compute cores were made available representing a '4x' increase in compute cores compared to the lab PC system. For the algorithm to have scaled successfully it needs to show approximately a 3x - 4x increase in performance. Note a direct 4x improvement is not expected since the overhead from managing the compute cluster is noticeably larger than that of a multi-core CPU operating on a single motherboard. Table 10-4 sets out the results achieved with the university HPC:

<u>Neuron Count</u>	<u>Cross Correlation Tasks</u>	<u>Total Time (Sec)</u>	<u>Per second rate</u>
50	1275	10.51	121.37
100	5050	29.04	173.89
150	11325	55.78	203.04
200	20100	92.85	216.47
300	45150	196.48	229.80
400	80200	339.09	236.51
500	125250	525.23	238.47
1000	500500	2031.16	246.41

<u>Neuron Count</u>	<u>Cross Correlation Tasks</u>	<u>Total Time (Sec)</u>	<u>Per second rate</u>
2000	2001000	8039.12	248.91

Table 10-4: HPC Cross-Correlation rates for various sized datasets

As expected the increased availability of compute cores in the HPC results in a marked increase in performance. Recall that the same algorithm is executing in both situations without any re-coding so any increase in performance must arise from utilisation of the HPC's increased resource availability. Table 10-5 contrasts the performance between the desktop and HPC environments:

Neuron Count	Cross Correlation Tasks	Desktop	HPC	HPCx Faster
50	1275	58.38	121.31	2.08
100	5050	60.81	173.90	2.86
150	11325	65.49	203.03	3.10
200	20100	65.83	216.48	3.29
300	45150	66.37	229.79	3.46
400	80200	66.74	236.52	3.54
500	125250	66.89	238.47	3.57
1000	500500	67.23	246.41	3.67
2000	2001000	67.27	248.91	3.70

Table 10-5: Contrast between desktop and HPC performance.

As anticipated the majority of HPC results placed firmly in the middle of the 3 – 4x faster range. The increased overhead from managing the cluster was significant only up to the 150 spike train dataset size. Additionally while the performance of the desktop system plateaued sharply at this point the HPC continued to improve its performance. Nevertheless it was clearly plateauing itself by the time the 2000 spike train dataset was reached. The more gradual onset of diminishing returns in the HPC environment becomes clearly visible if the cross correlation per second rate is charted (see Figure 10-5). Of course the primary benefit of employing a HPC is the ability to compute cross correlations for considerably more spike trains in less time. The increased compute rates of the HPC lead to a direct reduction in the time required to analyse a data set. In the experiments case the desktop required 8h 15min 44.07 seconds to fully process the 2000 spike train data set (2001000 cross correlations) whereas the same data set was processed by the HPC in 2h 13min 59.12seconds.

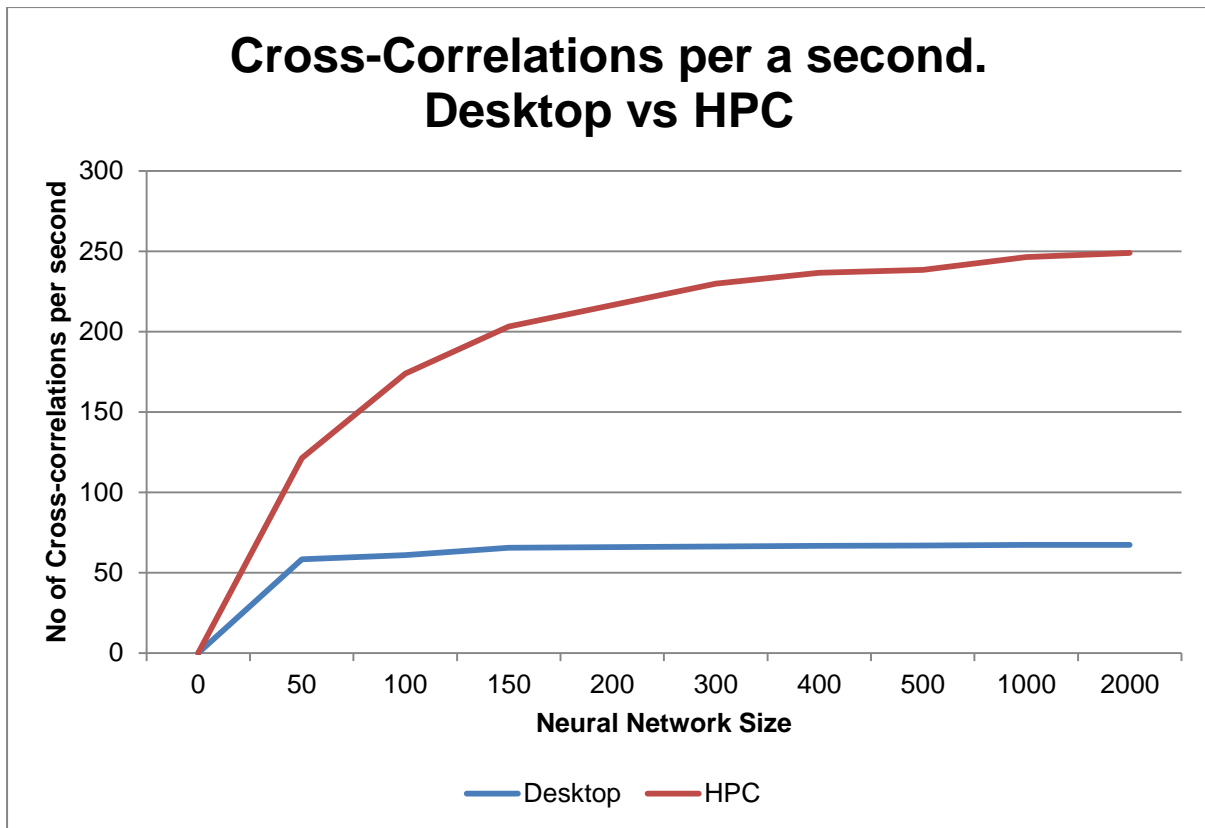


Figure 10-5: Cross-correlations per second vs neural network size in desktop and HPC environments

With the cross-correlation operation completed it is now necessary to use these calculations to infer the structure of the neural network that generated the recorded data set.

10.2 Identifying interconnected neural clusters

At its most basic level a neural network is a set of interconnected neurons. These interconnections form clusters of co-operating / communicating neurons which act as data processors. The ultimate goal for i-Grid is to identify these clusters of interacting neurons from the recordings of their firing activity. Cross-correlation provides a measure of the “strength” of the connection between two recorded neurons. A strong cross-correlation signal indicates that the neurons are likely to be connected and form part of a cluster of neurons that work together to process data. Having compiled the cross-correlation data it is now necessary to identify these clusters.

Clustering of data sets can be done in a variety of different ways but the identification of clusters based on connectivity is known as connectivity based clustering / hierarchical clustering. The classic visualisation associated with a hierarchical clustering is the dendrogram. Previous work has shown that iGrid can effectively identify neural network structure (Stuart, Walter & Borisyuk, 2005) but one of its primary drawbacks was its inability to scale to larger data sets. In this case the grid presents so much detail that it becomes visually overwhelming to the user or simply too dense to be effectively displayed. Clearly then for a cross correlation grid to become effective Shneiderman’s Information Visualisation mantra should be applied. This requires the introduction of a means for the user to “overview” the data set and to filter the items included on the grid. The grid itself then becomes the element that delivers the “detail on demand”. In this implementation of the cross correlation

grid the “overview” of the data set will be provided by an interactive dendrogram that groups and displays spike train clusters. The user’s interaction will allow the filtering of individual clusters or parts of clusters into / out of the “detailed” cross correlation grid.

The clustering algorithm selected as the foundation for the dendrogram overview is a hierarchical agglomerative clustering solution using complete linkage clustering. The metric employed to determine the distance between clusters is the Euclidean distance between the most significant peaks of the two spike trains normalised cross-correlogram. Alternative linking strategies were considered (single linkage and average linkage) but complete linkage proved the most effective at identifying the connectivity structure of the simulated neural network.

To understand how the selected clustering algorithm function works the terms agglomerative clustering and complete linkage need to be clearly defined:

- **Agglomerative clustering** is a recursive process where larger clusters are formed by the merging of smaller clusters. At each iteration the algorithm merges the two “closest” clusters to form a new cluster. The “closest” clusters are determined by applying a linkage algorithm to determine a “distance” between clusters
- A **linkage algorithm** is used to determine how closely two clusters are related. This involves both the selection of a metric to measure closeness and a means of determining that metric for clusters of multiple data sets.
 - The selected metric for this implementation of iGrid is the strength of the peak normalised cross correlation bin between two spike trains.
 - **Complete linkage** is used in which each spike train in each of the two clusters under consideration for merger provides a peak cross-correlation value. The largest cross-correlation peak determines how “close” the clusters are with larger peak values indicating greater closeness.
 - Other linkage methods were considered but proved less effective in identifying connectivity. The considered methods were:
 - **Single linkage** where a single spike train within each cluster would be taken as representing the cluster. The cross-correlation between these two would be used as a measure of closeness.
 - **Average linkage** where the average value of all cross-correlation peaks between spike-trains in the two clusters is taken as a measure of closeness.

Logically the process of clustering can be continued until all spike trains have been grouped into a single cluster. This would however be inappropriate as connectivity cannot be inferred between neurons where the spike train’s peak cross-correlation bin does not exceed the Brillinger threshold (Brillinger, 1979). The result of clustering can therefore be represented as a series of dendrograms and un-clustered spike trains which did not show a significantly strong correlation signal to be clustered together.

The final output of the cross correlation and clustering algorithm is two JSON encoded data files:

1. The detailed cross-correlation results encoding all cross-correlation bins for all recorded spike trains as per Figure 10-4. This data will be used to produce the classical iGrid representation.
2. A clustering file containing a series of dendrograms (and un-clustered spike trains) which will be used to provide both an overview of the dataset and a means to control filtering of spike trains into / out of the detailed iGrid view.

These two files and the original spike train recordings themselves are passed to the iGrid visualisation module.

10.3 Scaling the iGrid Visualisation

Addressing the pure computational challenges to cross-correlate spike train data using a cross platform program that scales to use all available resources is challenging. These issues are however primarily technical in nature and can be overcome with time and effort. The final goal however is to make the data available to the user to explore using visual analytics where the majority of the “heavy lifting” is performed by the users’ visual systems with only limited cognitive effort. In iGrid’s case the primary barrier to this is one of information overload. This is easily demonstrated by taking the 2000 spike train recording used in testing the cross-correlation algorithm and considering the iGrid display that would result from its visualisation.

Representing the 2,000 spike train data set as a cross-correlation grid would require the display of 4,000,000 individual grid cells. Typically a modern high resolution monitor operates at resolutions of 1920 x 1080 pixels. This provides a display of 2,073,600 pixels or 51.84% of a 4,000,000 cell grid. While this comparison clearly shows that such a grid cannot be displayed the selection of a single pixel to represent a data point is inappropriate because:

1. The modern high resolution monitor deliberately employs a resolution where the individual pixels cannot be appreciated as separate entities in the human visual system.
2. It would be impractical to dedicate the entire screen to the grid display as this would leave no room for the provision of interactive controls that would facilitate user control to overview, filter and extract selected detail from the visualisation.

To successfully allow the visual exploration of a spike train cross correlation dataset these issues must be addressed. The approach adopted in the implemented solution is to demote the grid representation from its position of providing the overview, filtering and detailed data delivery. Instead the grid will focus on the delivery of detailed data a role in which it excels. Responsibility for the provision of the overview and filtering of data will however be removed from the grid and re-allocated to a series of dendrogram visualisations. Each cluster generated in the data analysis phase will form a dendrogram providing the user with an overview of the identified clusters within the data set. Spike trains that exhibited no significant cross-correlation (and which are therefore not clustered) will be reported separately. The individual nodes of the dendrogram will be interactive allowing the user to expand or collapse individual dendrograms or sub-sections / sub-trees of a dendrogram. By this means filtering of the spike trains included in the iGrid cross-correlation plot will be achieved.

Despite the re-allocation of the overview and filtering roles to the dendrogram visualisation it is anticipated that the number of spike trains displayed will remain quite large in any significant sized data set. Instead of limiting the iGrid display to the available screen space a “viewport” approach was adopted where the grid visualisation is generated on a virtual screen space of potentially infinite extent (within the constraints of available system memory). This allows the visualisation to ensure that each cell of the cross correlation grid is rendered at a size that allows the human visual system to effectively process its relationship to other nearby spike trains. On screen presentation is via a viewport that shows a conveniently sized section of the rendered grid. Scroll bars and in viewport labelling is used to maintain the users overall awareness of position within the data set.

While iGrid itself provides a detailed visual representation of the relationships between the spike trains and spike train clusters it does not represent the “most detailed view available”. That of course is the cross correlation data itself that was used to generate the grid. This data is traditionally represented as a histogram plot of the cross-correlation bins generated from the raw spike train data. Previous implementations have provided these as pop-up graphs or via replacement of the iGrid cell with a glyph representing the major peaks of the histogram (those surpassing the Brillinger threshold). In each case the resulting histogram is visually small and difficult for the user to process without significant cognitive effort. The new implementation extends the viewport concept to the cross-correlation histogram. The iGrid viewport is dynamically resizable by the user allowing them to control the screen space dedicated to the iGrid representation. The remaining screen space is dedicated to the presentation of the cross-correlation data in its most basic form – the cross correlation histogram. This allows the user to directly control the display space for iGrid and the cross-correlation histogram. As the user explores the data set they will at different times assign different levels of importance to the iGrid vs histogram representations. The ability to visually resize the two viewports allows the user to visually place greater importance on one or both of the visualisations. Taken together these features implement the Visual Information-Seeking Mantra of overview first, zoom and filter with details on demand. Figure 10-6 and Figure 10-7 show the visual effect of this approach. In Figure 10-6 a data set of 20 spike trains has been rendered as a cross correlation grid with a supporting histogram. Greater visual emphasis has been placed on the entire data set as represented by the grid but the histogram shows an interesting feature – two significant cross correlation peaks. Figure 10-7 presents the same data but this time greater emphasis has been placed on the cross correlation histogram by enlarging the viewport to half the screen size. Despite this the correlation grid is still clearly visible, a vertical scroll bar has been added avoiding compressing the grid display to an unusable degree but still showing the interesting cross-correlation and the cluster of spike trains that it is part of.

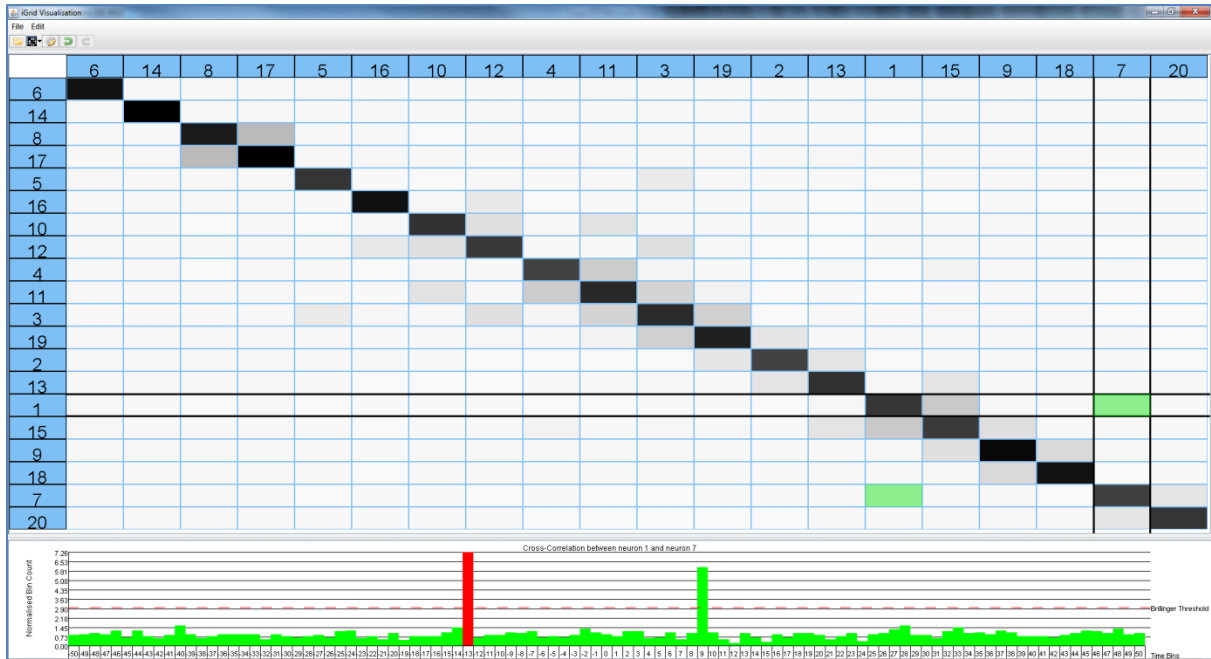


Figure 10-6: iGrid visualisation with cross correlation histogram for neuron spike trains 1 and 7. The red bar denotes the peak cross-correlation value used for the iGrid representation.

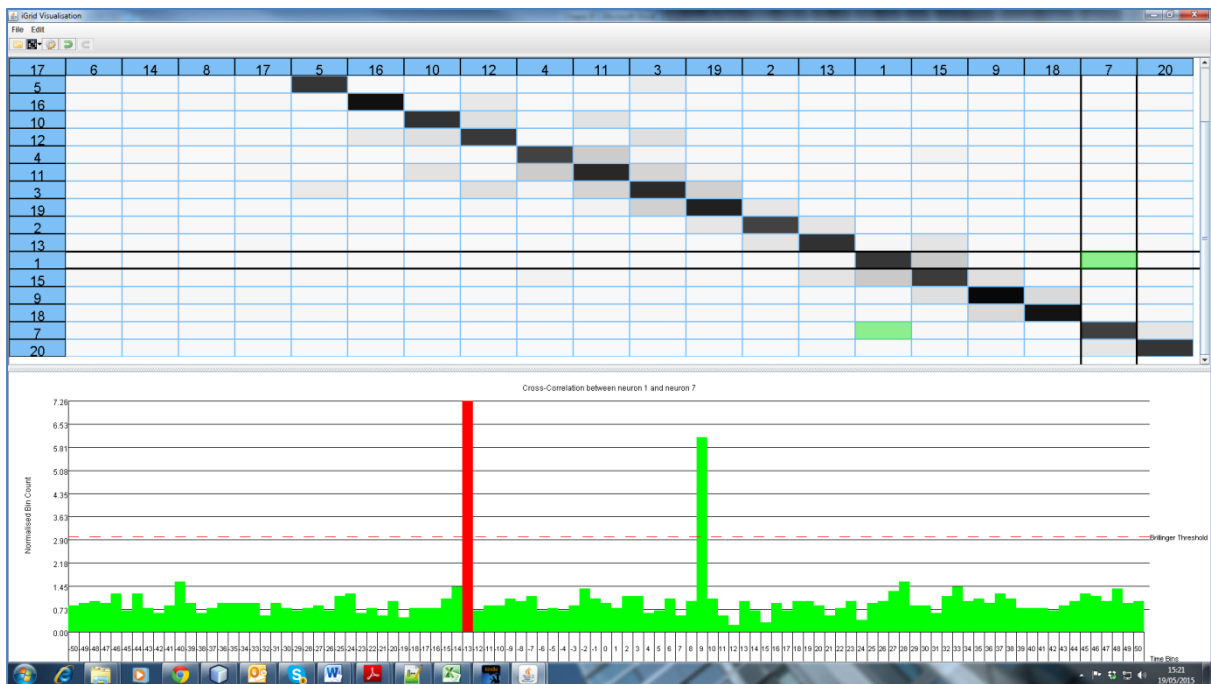


Figure 10-7: Resized iGrid visualisation & cross correlation histogram placing greater emphasis on the detailed histogram while retaining awareness of its place in the correlation grid. The red bar denotes the peak cross-correlation value used for the iGrid representation.

10.4 The Dendrogram Visualisation

The second component of the revised iGrid implementation is the dendrogram display. This display will serve as both an overview of the data set and a means of filtering it before rendering of the cross correlation grid. At its most basic a dendrogram is a directed graph (also known as a digraph) composed of nodes (or vertices) and edges (Bondy & Murty, 1976). The connecting edges between nodes have direction and navigation between nodes

is restricted to the edges direction. Together these are arranged in a tree structure with filtering being achieved by expanding / collapsing various branches of the tree. Directed graphs can be further categorised as cyclic or acyclic based on whether navigation through the tree along the directed edges permits a return to a previously visited node / vertex (cyclic if it does or acyclic if it does not). Whether a cyclic or acyclic dendrogram will arise is dependent on the clustering algorithm selected. In the case of an agglomerative clustering algorithm an acyclic dendrogram will arise as spike trains and clusters of spike trains merge to form ever larger groupings. The implemented clustering algorithm terminates when no significant cross correlation peak between the remaining clusters exists. These unrelated clusters then form the root nodes of potentially several directed acyclic **rooted trees** or arborescence's (Gordon & McMahon, 1989). The final presented dendrogram is, therefore, technically defined as an acyclic directed multi-tree – or forest - in graph theory. The agglomerative clustering approach also imparts the properties of a **k-nary tree** with $k = 2$. Such trees are known in computer science as binary trees and have been used for the efficient storage, searching and sorting of data (Garnier & Taylor, 2009). Indeed this concept was presented in Chapter 8 as an efficient data structure for the storage and manipulation of long duration neuron spike train recordings. K-nary trees have also seen regular use as a visualisation tool especially in computer science. The dependencies between software packages in a modern large-scale object orientated program should be visualised as a directed acyclic graph (DAG) – the acyclic dependencies principle (ADP) (Martin, 1996). The inability to visualise the software applications structure as a directed graph free of cycles is indicative of a poor high level design.

Clearly then the dendrogram as an acyclic directed graph is a well-studied area and in implementing the dendrogram for iGrid the developers first sought for an existing visualisation framework which could be leveraged to provide the foundation for iGrid's overview and filtering dendrogram. The key requirements were:

1. The framework must integrate with the existing visualisation components and the underlying data model.
2. The cross-platform support requirements of the project must be respected
3. Ideally the framework would provide base implementations for:
 - a. Vertices / Nodes
 - b. Directed Edges
 - c. The formation of rooted tree structures (K-nary trees)
 - d. The management of multi-tree structures as a single "forest"
 - e. Presentation of (a)-(d) as a directed acyclic graph (DAG)
4. The framework would have to be extensible so that it could be adapted to the spike train data as required.
5. Has a permissive free software license that imposes minimal restrictions on re-distribution

No framework was found that provided all of the required features however the developers selected the Java Universal Network / Graph framework (the JUNG framework) as the foundation on which the dendrogram visualisation should be built (Madadhain J. et al., 2005). The reasoning for this choice was as follows:

1. The JUNG framework is implemented in the Java language facilitating integration of the existing Java based data model and the developed iGrid & cross-correlation histogram views.
2. The Java language provides the cross platform support needed in a manner consistent with the rest of the project.
3. JUNG was developed using abstract graph theory as its guide. Extensive use was made of Java Generics when modelling nodes / vertices and edges. This permitted the development of various wrapper, adapter and decorator classes (Gamma et al., 1994) to be developed that allowed components of the existing data model to become components of the JUNG graph.
4. A suitable foundation graph class (DirectedSparseGraph) is pre-implemented.
5. The JUNG framework is distributed under the permissive Berkeley Software Distribution (BSD) license.

Despite meeting many of the criteria a number of issues remained unaddressed by the JUNG framework.

1. While the concept of a directed graph was supported the framework did not include any native support for acyclic graphs.
2. As a consequence of (1) no layout manager existed for rendering a directed acyclic graph as a dendrogram. Indeed no dendrogram layout manager existed at all with directed graphs being rendered as “networks” with no consideration given to presentation as dendrogram trees with “roots” and “leaves”.
3. While being designed from conception to be extensible (open source) and adaptable (via Java generics) the framework is not particularly well documented with largely incomplete online Java doc. This lead to extensive code reviews of online implementation examples and a steep learning curve.

Production of the final iGrid overview and filtering dendrogram therefore involved a significant development effort that included:

1. the extension of the JUNG frameworks DirectedSparseGraph to support a “forest” of directed acyclic rooted tree’s
2. The extension of JUNG’s base tree classes to support acyclic rooted trees
3. The implementation of a JUNG dendrogram layout manager that supports rendering multiple dendrograms to provide the base visualisation for the forest.
4. The development of various wrapper, adapter and decorator classes to bridge between the visualisation framework, underlying data model and the iGrid visualisation that would be filtered by the users interactions with the dendrogram.
5. Creation of a visual encoding for representing two different structures within the dendrogram. The visible leaves of the dendrogram trees represented the raw spike train data that would be included on the iGrid visualisation. The other nodes / vertices represented clusters of spike trains and at higher levels of the dendrogram clusters of clusters of spike trains.

10.4.1 The dendrograms visual encoding scheme

While most of the issues faced were purely technical in nature the selection and implementation of a suitable visual representation of the spike train data set is critical to the success of the dendrogram visualisation. The glyphs used and the meaning attached are summarised in Figure 10-8.




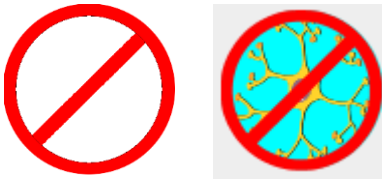
Glyph	Meaning and selection rational
	<p>Denotes a single spike train recording in the raw dataset. Visually a single neuron easily identifiable from soma and dendrite representation.</p>
	<p>Denotes a cluster of spike trains. Visually a series of cell like bodies grouped into a single entity. The primary role of the dendrogram will be to filter the spike trains included in the iGrid visualisation. Therefore this glyph may appear in two states:</p> <ul style="list-style-type: none"> A. Collapsed with all constituent spike trains filtered out of the iGrid visualisation. B. Expanded with constituent spike trains filtered in or out of the iGrid visualisation based on their own states. <p>These states provide coarse grained filtering of the data set by including or excluding entire clusters of spike train recordings.</p>
	<p>The “selected” glyph identifies nodes of the dendrogram tree that will be affected by the next user operation. A semi-transparent icon (A) that can be merged with other glyphs such as the individual spike train (B) or the cluster icon (C) to denote user selection.</p>
	<p>Denotes a single spike train recording in the raw dataset that has been excluded from the iGrid visualisation regardless of the status of any cluster it is part of. This permits finer grained filtering control which allows the filtering of individual cluster members. Variation of the “No Entry” traffic symbol to denote that the iGrid cross-correlation entry has been removed from the visualisation.</p>

Figure 10-8: iGrid dendrogram overview visual encoding scheme

While the dendrogram is a “forest” of individual dendrogram trees it is presented as a series of views targeting individual components of the forest. Within the forest there are essentially two types of spike train collections:

1. A true cluster of multiple spike trains presented as a series of collapsible / expandable nodes that ultimately represent a set of interconnected neurons.
2. A “cluster” of one spike train that shows no significant correlation in its spiking pattern with any other spike train recording.

Given this it is logical to present three views into the dendrogram dataset:

1. The “forest view” which presents the spike trains that show significant cross correlations as dendrogram tree. Each cluster identified in the data set will form

- one tree in the forest. Collapsing / expanding nodes will allow that cluster or part of the cluster to be filtered out of / or into iGrid's primary display.
2. An "un-clustered view" presenting the spike trains that showed no significant correlation with the other members of the data set. These potentially less interesting spike trains can be filtered out of / or into iGrid's primary display as a group or individually.
 3. An "Overview" presenting all members of the data set. Un-clustered spike trains + root nodes of the forest trees form the top level of this view. The forest trees are then expanded / collapsed in a manner similar to navigating files and folders in a "windows explorer" style view. This the user to explore the data set as a whole to make decisions on filtering data.

10.4.2 The dendrogram "forest" view

The forest view is where the user will spend the majority of their time when overviewing the spike train clusters identified in the dataset. Figure 10-9 shows a small dataset of 20 spike trains. A total of two major clusters have been identified and presented in order of complexity.

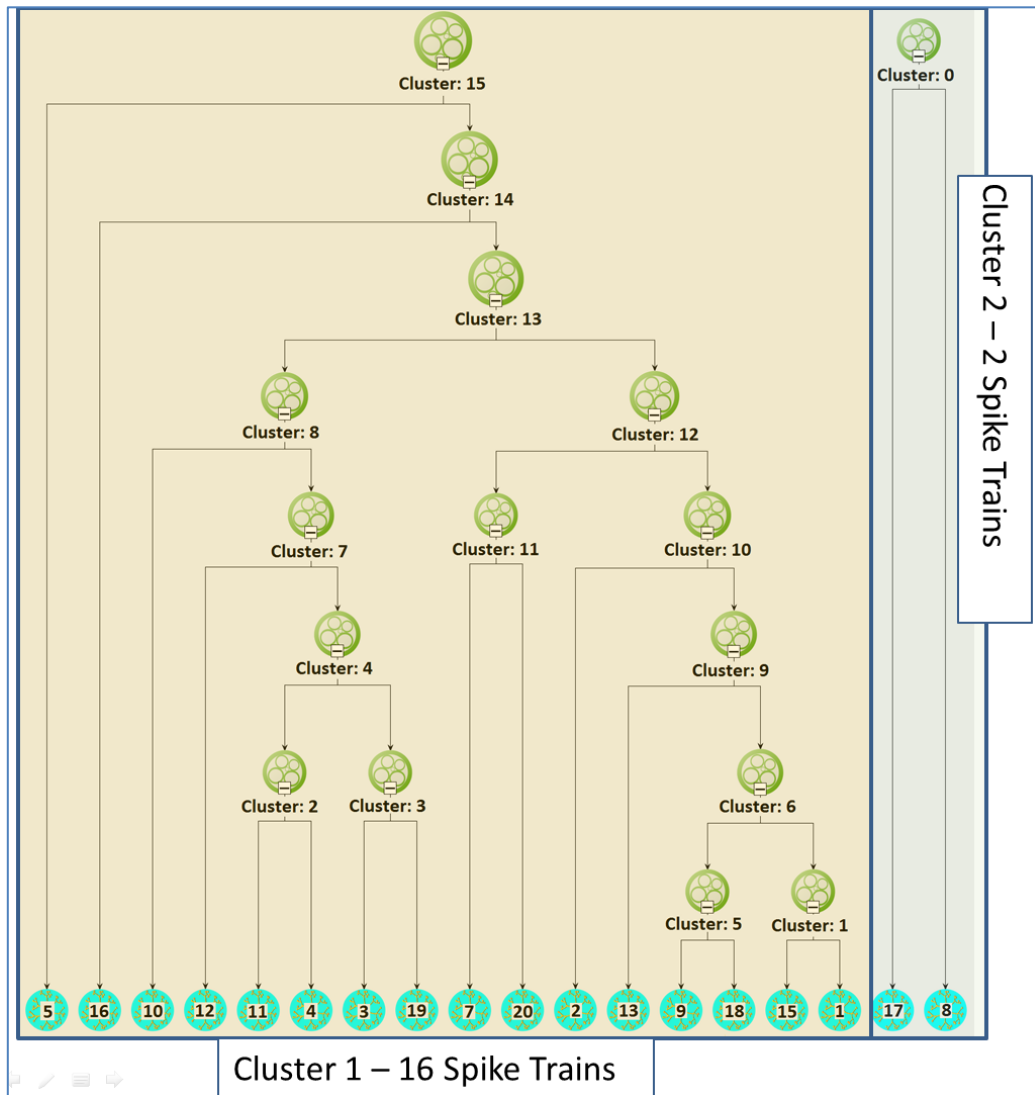


Figure 10-9: A "forest" composed of two trees

Cluster 1 is by far the more complex, containing 16 of the data sets 20 spike trains (80% of the recorded data). The result is an 8 level rooted tree with a large number of sub clusters. Figure 10-10 shows some of those sub clusters; spike trains 4 and 11 form “cluster 2” while 3 and 19 forms “cluster 3”. Both of these clusters show a strong cross correlation signal between their members creating “cluster 4”.

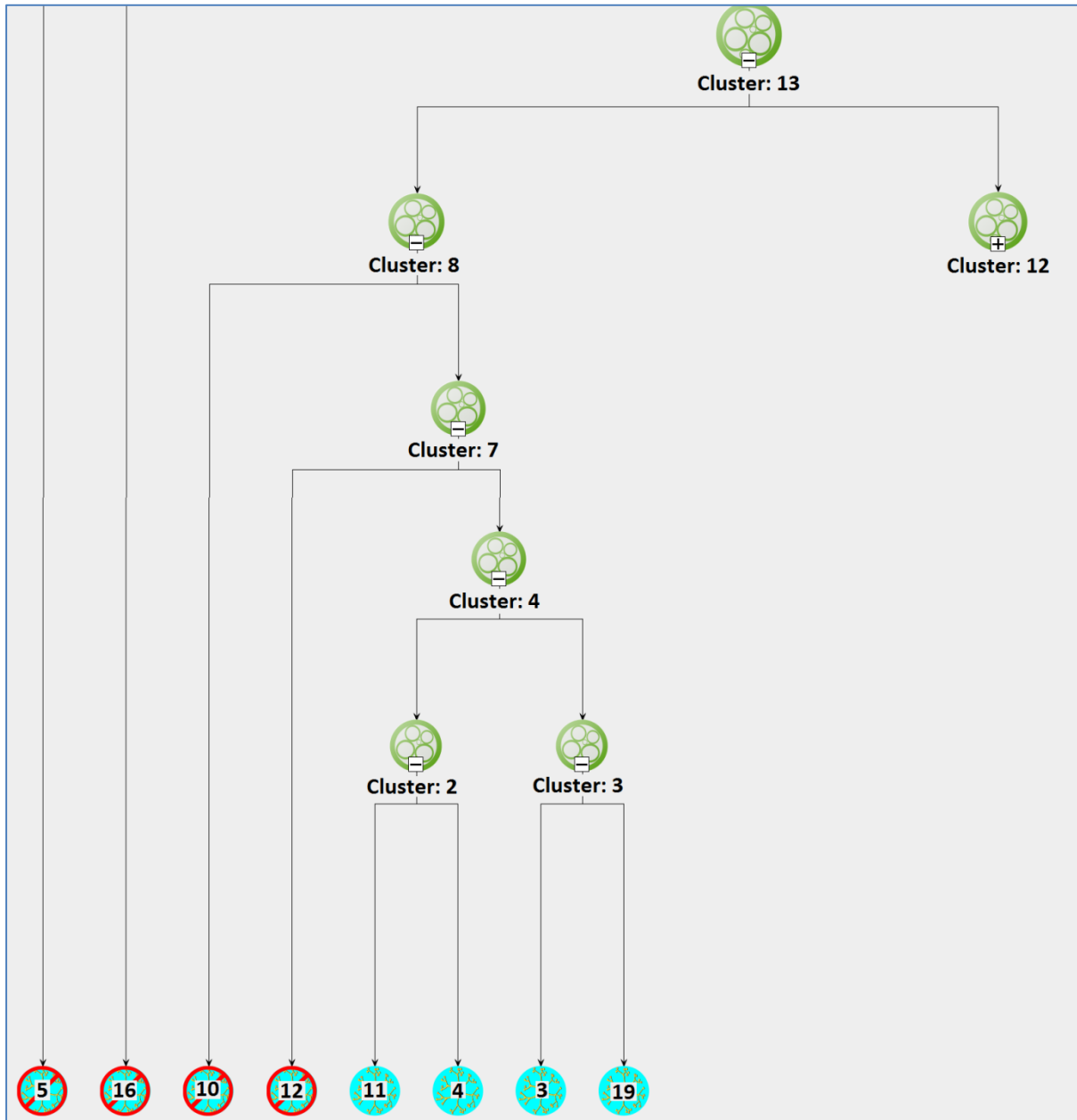


Figure 10-10: Spike train dendrogram collapsed and filtered to show a sub-cluster

In addition to the four clustered spike trains we can see that progressively weaker (but still significant) cross-correlations exist with spike trains 12, 10, 16 and 5. Initially these have been filtered out to focus on the stronger relationship between spike trains 4, 11, 3 and 19. Figure 10-11 presents the filtered iGrid for these four spike trains. Clearly strong relationships are expected between spike trains 4 and 11 and spike trains 3 and 19. Inspection of Figure 10-11 also shows a strong signal between spike trains 3 and 11. This signal is clearly visible in the cross-correlogram (see Figure 10-12) with spike train 11

recording activity at 10ms in advance of spike train 3. The normalised bin count of 12.329 indicates a strong correlation and hence it can be inferred that the connection between the two sub clusters is via spike train 3.

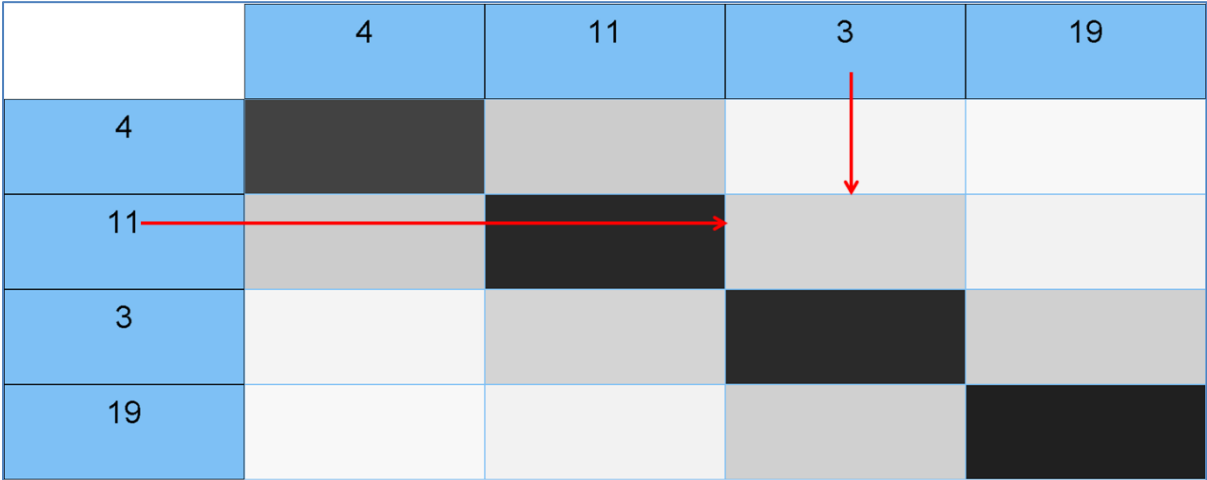


Figure 10-11: Filtered iGrid for spike trains 4, 11 and 3, 19

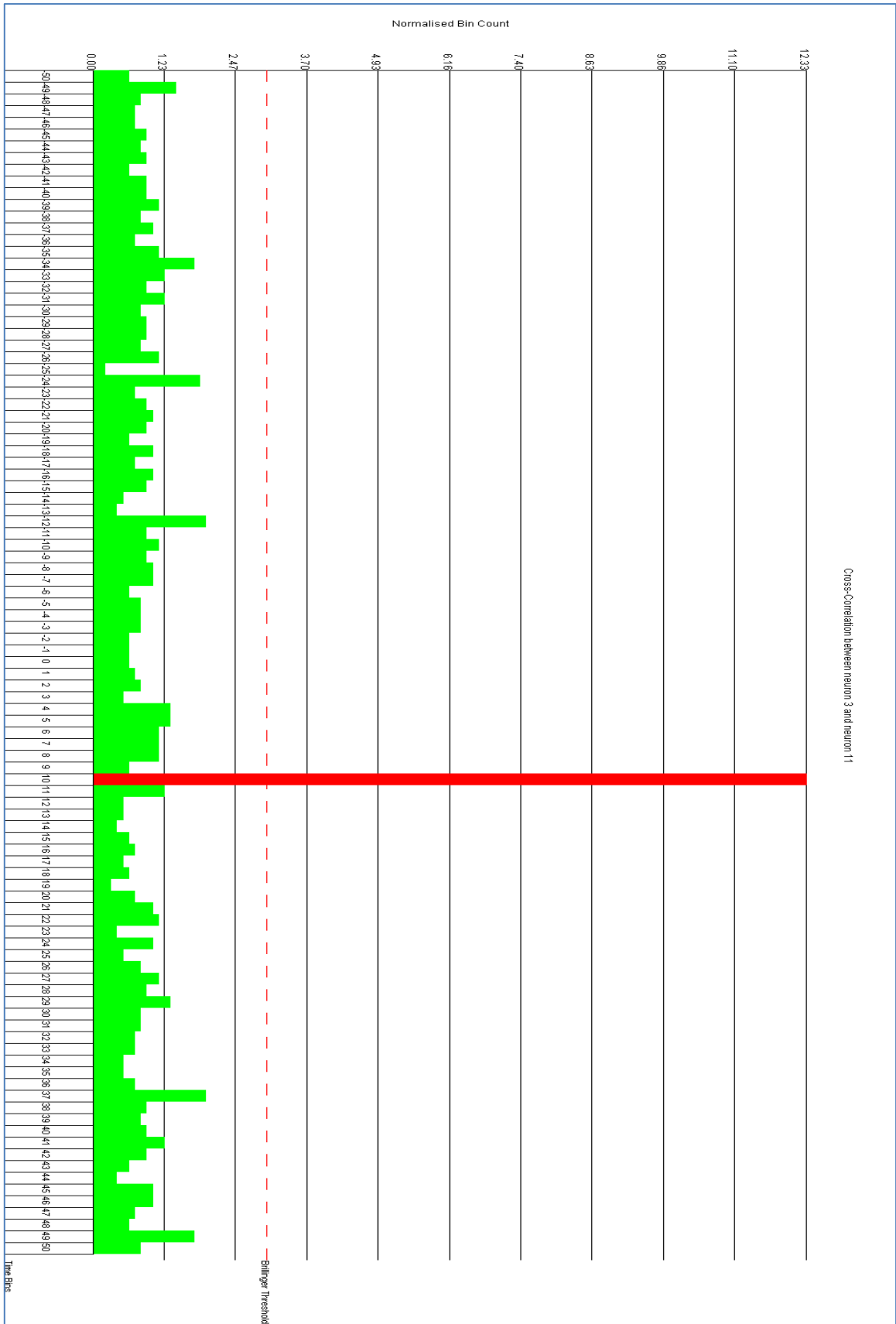


Figure 10-12: Cross-correlation chart for spike trains 3 and 11

The analysis has again been completed entirely using the researcher’s visual cortex while applying the Visual Information Seeking Mantra of Overview first, Zoom and Filter, Details on demand. The result is to identify that the spike trains 3, 4, 11 and 19 form the structure shown in Figure 10-13. Since a simulated neural network was used to test the implementation this structure can be contrasted with the actual network shown in Figure 10-14. While the analysis has been focused on only 4 of the spike trains in two sub clusters the connections have been successfully mapped for the section of the network examined.

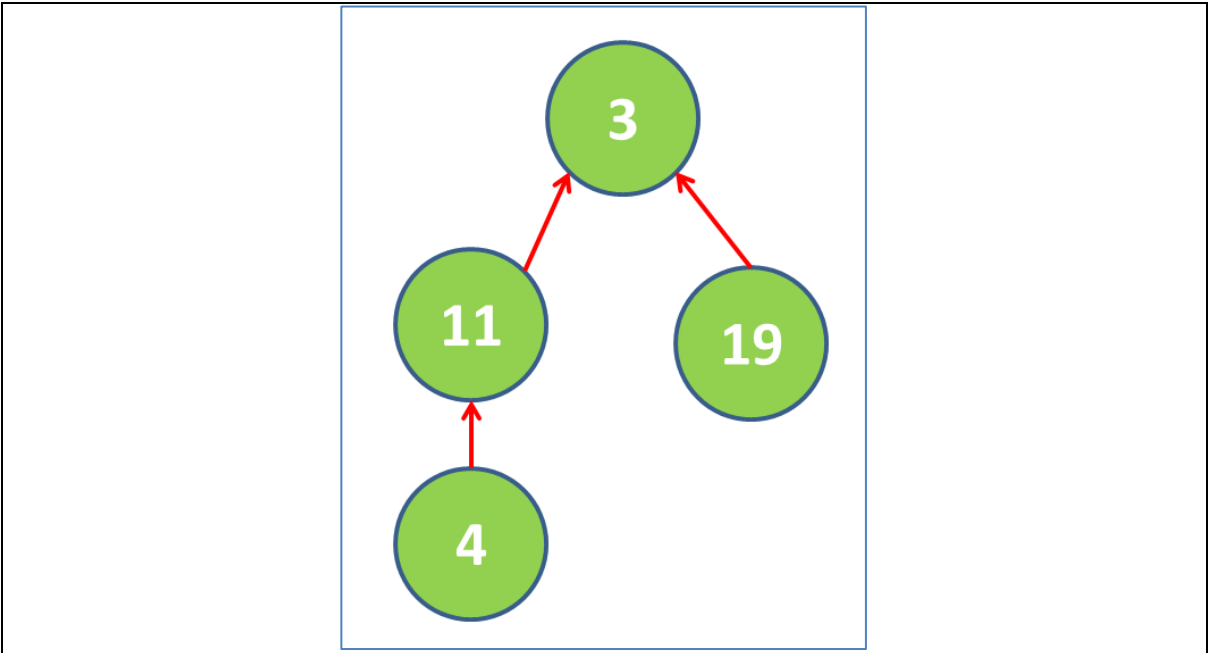


Figure 10-13: Predicted inter-neuron connections

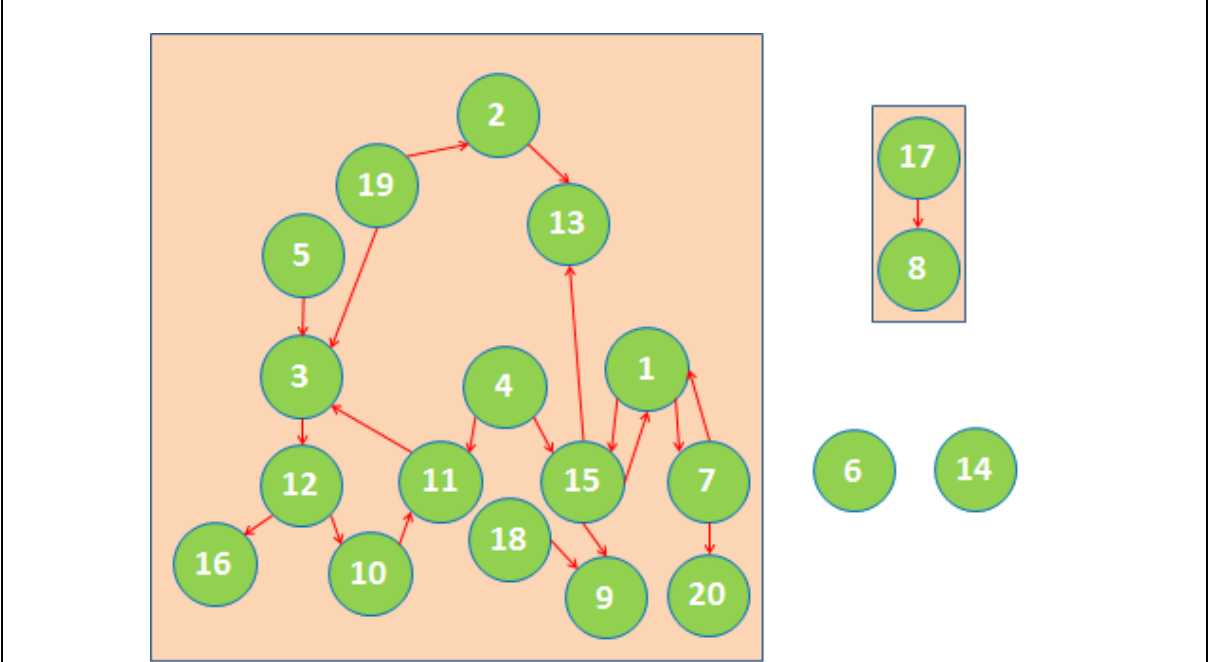


Figure 10-14: Actual simulated neural network connections

The next step would be to expand the analysis to take in the other clustered spike trains which were filtered out in Figure 10-10 (spike trains 5, 10, 12 and 16). As the actual

network structure is available in Figure 10-14 it is also possible to work in the other direction and make predictions about what the visualisations should show. For example I have not connected spike trains 11 and 19 in the first stage analysis. The actual network contains an indirect connection between these nodes via spike trains 19 to 3 to 12 to 10 to 11. Hence the prediction that a weak cross correlation signal should exist between spike train 11 and 19 can be made (while connected the connection is very indirect). Examination of the iGrid representation in Figure 10-11 does seem to indicate a weak signal. If the network structure was unknown this weak signal would be the logical point to begin expanding the network map. Figure 10-15 presents the cross-correlogram for spike trains 11 / 19 which shows a normalised peak of 3.6386 at -1ms – weak compared to the peak of 12.329 seen between spike trains 3 and 11 but still significant.

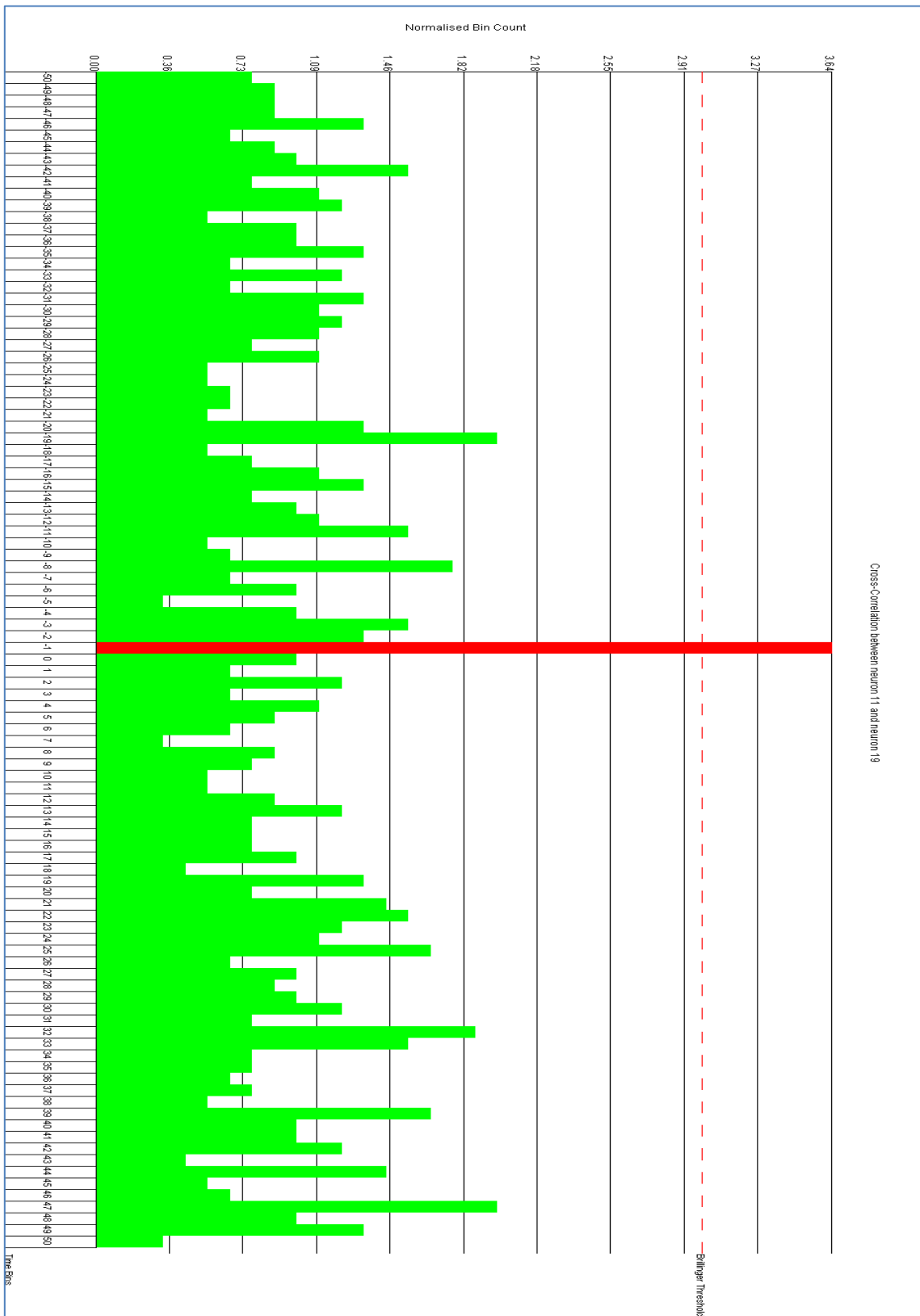


Figure 10-15: Cross-correlogram for spike trains 11 and 19 showing the weak but statistically significant signal (i.e. in excess of the Brillinger threshold).

Introducing the previously excluded spike trains generates the iGrid representation shown in Figure 10-16.

	5	16	10	12	4	11	3	19
5	■							
16		■						
10			■					
12				■				
4					■			
11						■		
3							■	
19								■

Figure 10-16: iGrid visualisation for the second step of neural network mapping

Here it is possible to see the all of the spike trains involved in the 19 to 3 to 12 to 10 to 11 chain and to observe that they show significant activity between each other. Examination of their cross-correlations would indeed permit the mapping of the 19 to 3 to 12 to 10 to 11 spike train connection.

In this manner it is possible to rapidly identify both direct and indirect connectivity between spike trains in a data set using a purely visual examination of the dendrogram forest and its supporting views.

10.4.3 The dendrogram “un-clustered view”

Inevitably in any data set some of the recorded spike trains will show no significant cross correlation with other members. In this case the selected agglomerative clustering algorithm will produce a “cluster” of only a single spike train. Using the previous example data set this was true for spike trains 6 and 14 (see Figure 10-14). While of course they are a part of the dataset it is most likely that the user will want to filter these out of the display as they should not form a part of any connectivity map. Rather than present these with clustered spike trains by creating a one spike train tree in the dendrogram forest these spike trains have been provided with a dedicated view. This allows the user to work with these items as a collection in their own right (even though they do not form a rooted tree). The most common decision will be to make a filtering decision (in / out) for this group. A viewport is again used to present this view immediately to the right of the primary dendrogram forest. Figure 10-17 shows this group with the individual spike trains filtered out of iGrid as they were in the previous example.



Figure 10-17: The "Un-clustered view" with the spike trains filtered out from iGrid

10.4.4 The dataset's "Overview" view

The presentation of the dataset as a forest of rooted trees provides an overview when no part of the tree has been collapsed. However as sections are hidden it is possible that the user will lose awareness of all the relationships within the dendrogram trees. This is, of course, expected as the point of collapsing these sections is to ease the cognitive load on the user. However no matter how far the user "drills down" into the dataset a time will come when they wish to return to the "higher" overview. Maintaining awareness of the overall dataset structure and the user's position within it is important. To permit this a windows explorer style view independent from the filtering performed by the primary forest view is provided. This view permits the user to examine the clustering applied to any portion of the dataset without changing the configuration of the filters applied to the iGrid view. Primarily, it provides a planning tool where the user can identify candidate spike trains for inclusion in the next stage of the analysis. Figure 10-18 shows the fully expanded explorer tree view for the test dataset used in the previous examples.

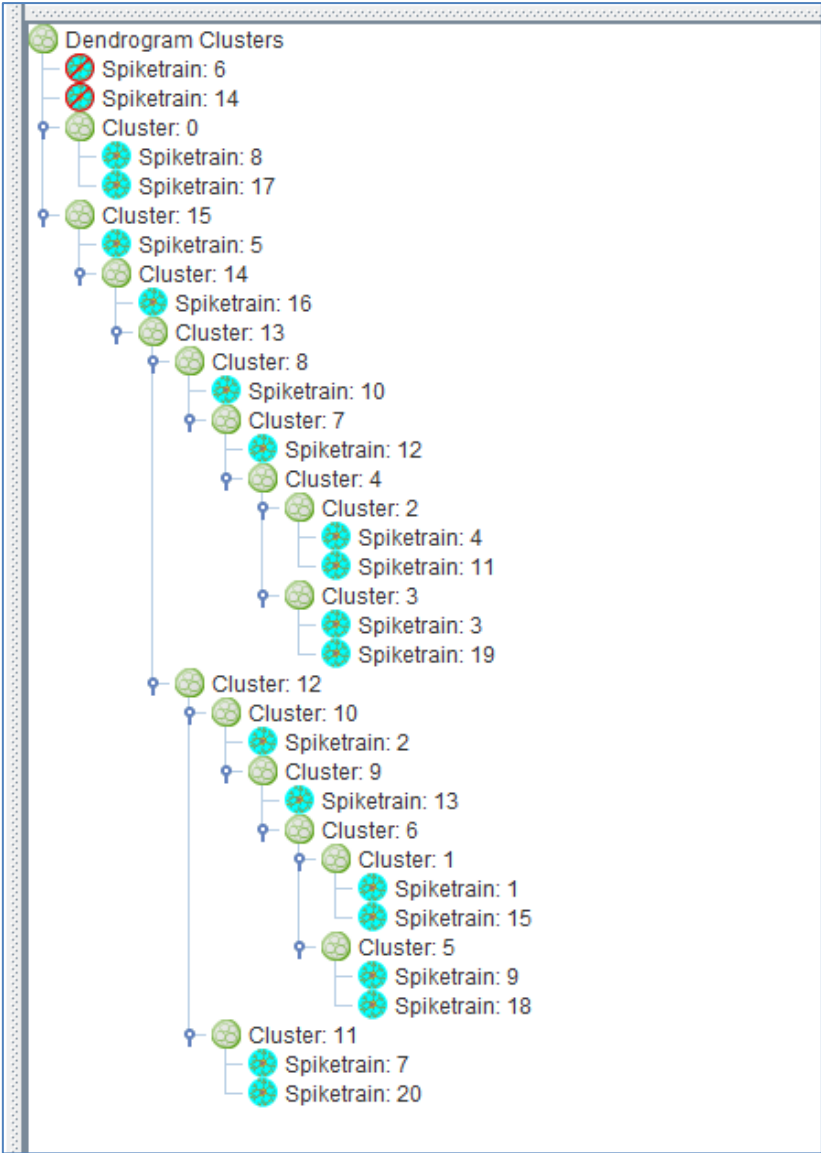


Figure 10-18: Fully expanded overview showing the test data sets structure.

Chapter 11

The i-Animate Visualisation

“microelectrode noun. A minute electrode; especially: one that is inserted in a living biological cell or tissue in studying its electrical characteristics”

Summary

This chapter describes the i-Animate visualisation which provides a visual representation of the multi-electrode array used to simultaneously record multiple spike trains from a biological sample.

11 Overview of the i-Animate Visualisation

Within neuroscience the primary means of recording and studying the electrical signals exchanged between interconnected neurons has been through arrays of microelectrodes (MEA's) inserted into a biological sample (either *in-vitro* in a laboratory setting or *in-vivo* where the array is inserted into a living biological organism). Data processing within a neural network depends on the connections between individual neurons. Hence it is connected clusters of neurons that form the data processing centres of biological organisms. In chapter 10 the i-Grid visualisation provided a mathematical (i.e. statistical) method for the identification of connectivity and clustering within a spike train sample. However as technology advances and it becomes possible to record thousands of individual spike trains in real time another, more visual based approach becomes possible. It has long been known that large groups of connected neurons (i.e. clusters) can exhibit many different patterns of activity when they are stimulated. Examples include:

- Waves of activity that pass across the recorded cluster(s).
- Inhibition where the activation of one region suppresses activation in another region.
- Simultaneous activation where the activation of one region triggers activation in another region.
- Long range connection neurons which “forward” activity in one region to what appears at first to be a completely different region of the neural network.

Historically recording these forms of activity between individual neurons has not been possible. On the coarser level of neuron clusters however the signal grows strong enough to record. Recordings of “brain waves” in humans was achieved by Hans Berger in 1924 (Millet, 2002; Swartza & Goldensohn, 1998), the inventor of electroencephalography, and are known as EEG's. Through the application of this science it has been possible to map the regions of the human brain and their associated functions.

Advances in MEA technology have recently allowed the recording of large numbers of neural spike trains in sufficient quantities to apply these techniques at the finer scale of inter-neuron connectivity. Stimulation of a neural network can be achieved in many different ways and is possible in both *in-vitro* and *in-vivo*. Photo sensitive neural networks such as a retina can be stimulated with light. Chemical stimulation is another common practice. Electrical stimulation is possible through a recording MEA (though this is rare, most MEA's are passive recording devices). In some cases stimulation may not be necessary as the neural network exhibits spontaneous activity (for example spontaneous activity is always present when recording from an immature piece of the central nervous system). This immediately suggests a visualisation in which each recording electrode “glows” when it detects a

spiking event and then “cools” over a short time period. Collectively by manipulating the “cooling” period it becomes possible to see clusters of activating neurons. The i-Animate visualisation provides the means to generate this representation of a spike train data set.

11.1 Goals of the i-Animate Visualisation

The i-Animate visualisation was created to meet the following key goals.

- I. To create a visualisation where the human visual system is used to identify interconnected neurons and clusters as an alternative to mathematically based cross correlation.
- II. The increased length of modern spike train recordings facilitates (i) as the human visual system is more sensitive to identification of changes over time. This provides another tool for exploration of the data set.
- III. Visualisation of the MEA grid provides a means to employ heat maps to exploit the human visual system. This facilitates the visual identification of active regions of the array and provides another means of identifying spike trains worthy of further analysis in i-Raster or i-Grid.

11.2 Basic implementation details

Figure 11-1 shows the primary interface to the i-Animate display with the key features labelled.

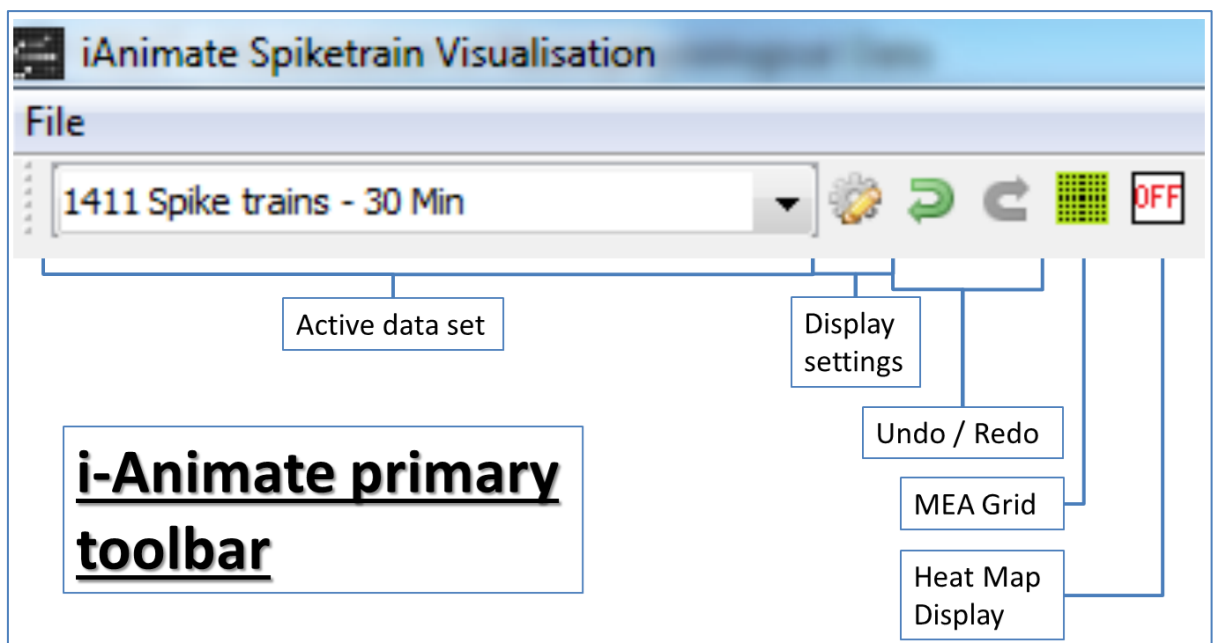


Figure 11-1: Primary i-Animate controls

In addition a set of simple animation controls are provided to play / pause the animation of the MEA grid activity and accelerate / decelerate the rate of playback. The File menu provides the option to save the current activity display. Finally long recordings can be time filtered using a timeline bar. Figure 11-2 details these controls.

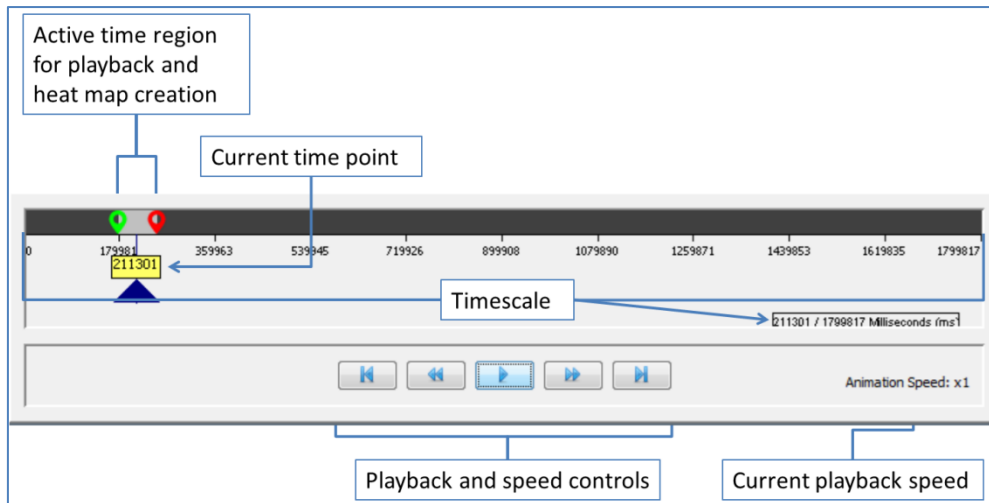







Figure 11-2: i-Animate time and time filtering controls

The central area of the i-Animate display is given over to the representation of the MEA grid that recorded the simultaneous spike trains. As the animation plays this area shows the ever shifting pattern of electrode activation in response to spike detection. Here it becomes possible to observe neural activity as it spreads from neuron to neuron and electrode to electrode. Two primary overlays can be applied to this area to present additional information:

- I. The MEA Grid – Accessed via the toolbar  button this overlay highlights each electrode which recorded spike train data over the course of the recording. It is used to identify those areas of the MEA grid that show activity.
- II. The heat map – Accessed via the toolbar  button the heat map overlay operates in one of three modes:
 - a. Off / Deactivated –  – The initial mode were the electrode activity animation plays without any heat map data being overlaid.
 - b. Grayscale Heat Map –  – In this mode each cell of the MEA Grid is assigned a greyscale value reflecting the total amount of spiking activity recorded in the active time sequence. Grayscale is the preferred operating mode for the heat map as it has been shown that the human visual system is most sensitive to changes in luminance rather than changes in hue.
 - c. Colour Heat Map –  – Despite having known drawbacks colour heat maps such as the classic rainbow heat map remain a popular. Therefore a colour heat map mode is also provided.

It is difficult to present information extracted from an animation in a static context. As previously observed it is the change over time that the human visual system is sensitive to rather than static displays. Figure 11-3 to Figure 11-6 attempts to show a classic sequence of spike train activity being triggered and how this spreads as a wave to three clusters of neurons that ‘process’ the stimulation. The screenshots show major events in the following order:

Time point	Comment	Diagram
203770 ms	Initial cluster of activation is visible	Figure 11-3
207990 ms	Wave front expanding away from initial activation	Figure 11-4
211198 ms	Wave front has split into three clusters of activity	Figure 11-5
215906 ms	Activity ceases in the three clusters	Figure 11-6

Table 11-1: Spike train activation in response to neural network stimulation

The total time period from initial activation of the neural network to its return to a quiescent background state is 12.136 seconds. Only the key static screenshots can be presented on paper but the animation can be verbally described.

- I. At 203.77 seconds into the spike train recording a ‘burst’ of activity begins in the MEA region highlighted in Figure 11-3
- II. Over the next 4.22 seconds a ‘wave’ of neural spiking events forms and begins to spread out through the network. This wave begins to break into three clusters of activity (Figure 11-4).
- III. Over the next 3.208 seconds the wave breaks up into three distinct clusters that exhibit highly elevated spiking frequency as these regions ‘process’ the stimulation event (Figure 11-5).
- IV. Finally over the last 4.708 seconds the spiking frequency in the activated neural clusters falls off and returns to levels characteristic of the network before the stimulating event (Figure 11-6).

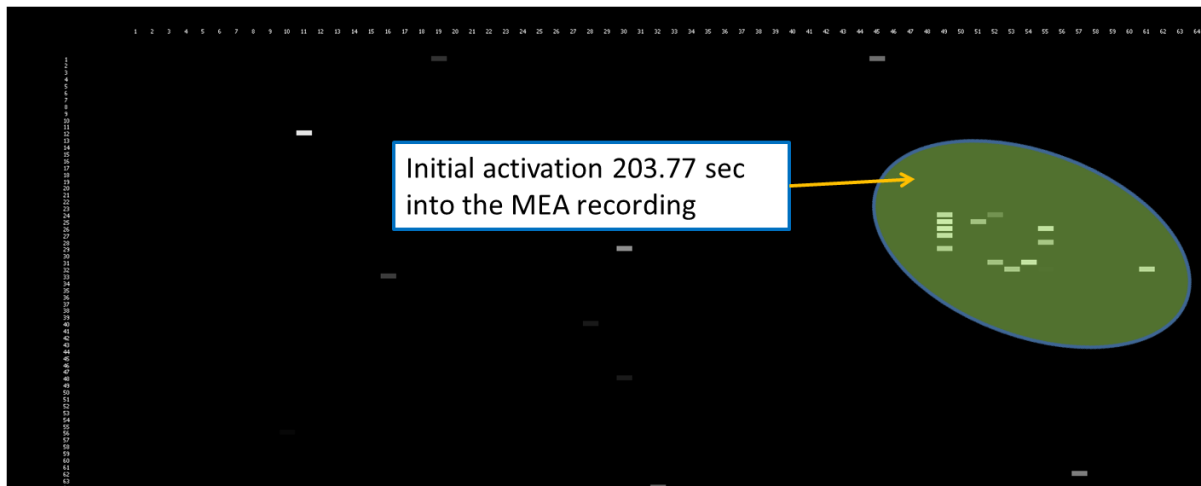


Figure 11-3: Stimulation event triggers initial response from the neural network

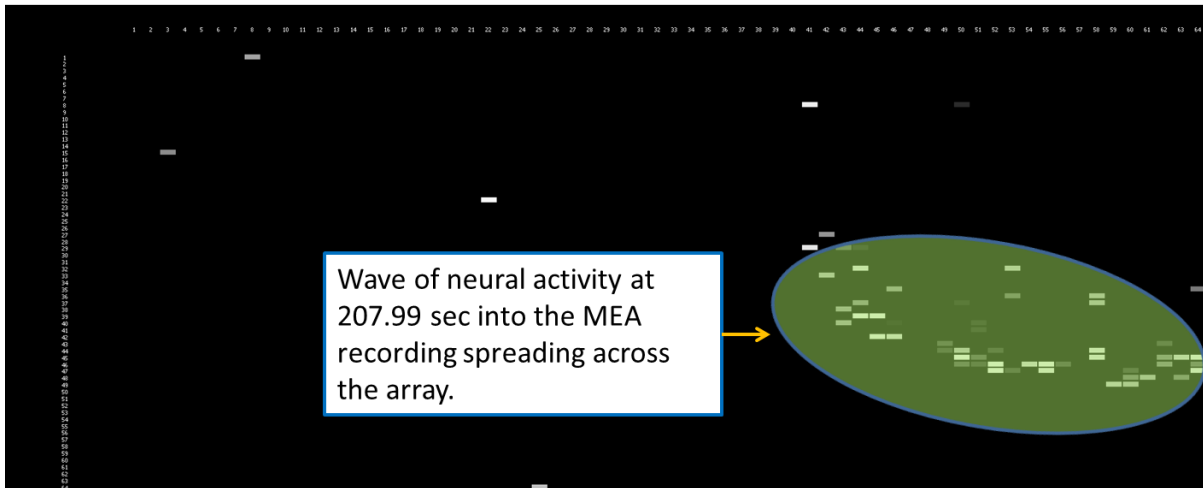


Figure 11-4: A wave of neural activity has formed and is spreading through the network

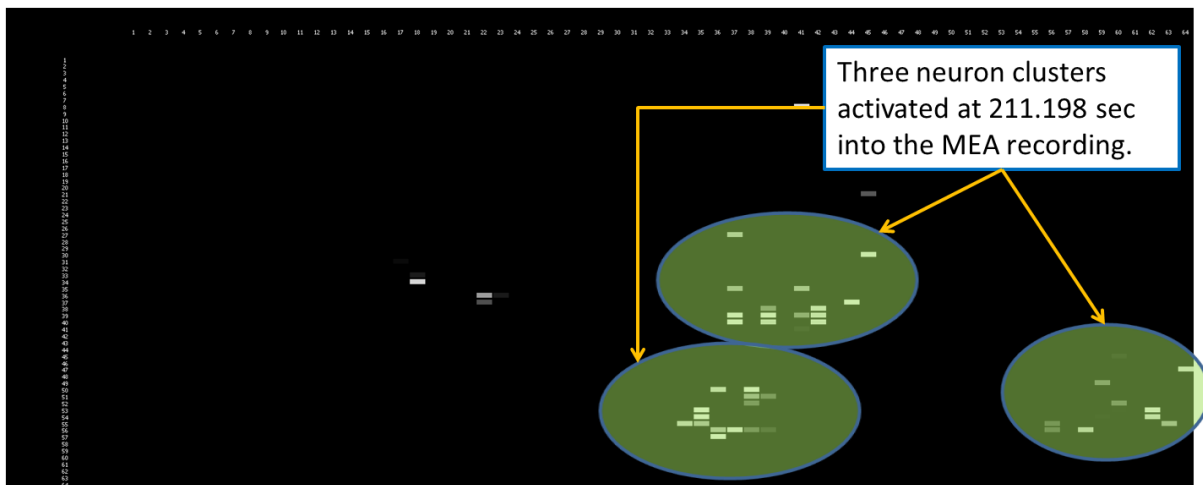


Figure 11-5: The wave's passage activates three neuron clusters which exhibit a period of intense spiking activity

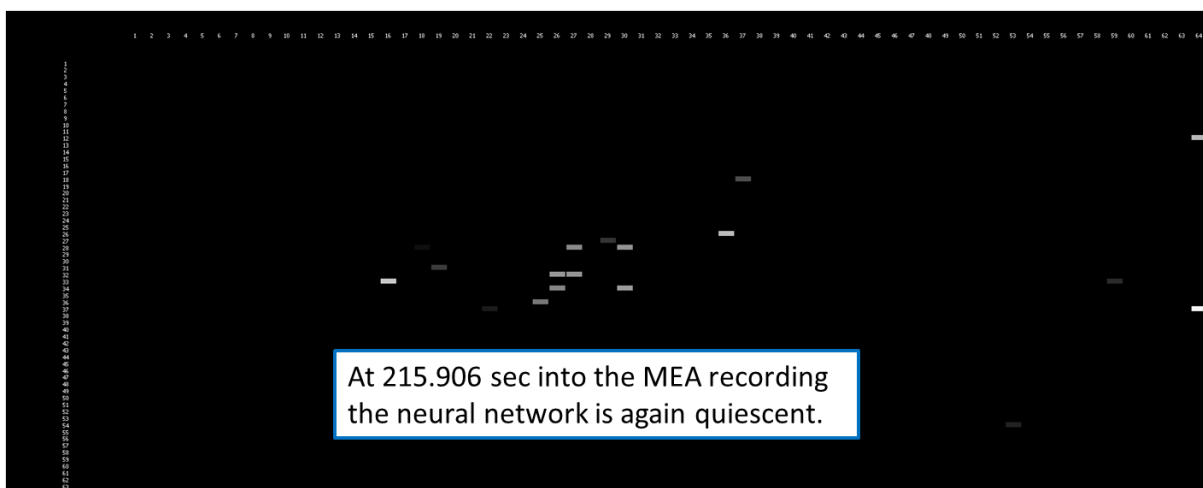


Figure 11-6: By approximately 12 seconds after the stimulating event the nextwork has returned its unstimulated state

11.3 The information overlay options in i-Animate.

While i-Animates primary display is focused on the presentation of changes in spike patterns and frequency a set of graphical overlays provides additional information. The MEA grid overlay provides a quick means to identify the active areas of the recording array were spiking events where detected. The collection of heat map overlays provides a means to identify the most active areas both over the entire recording or time filtered segments of it.

11.3.1 The MEA grid overlay.

This overlay provides a rapid means to identify the areas of the multi-electrode array that detected spiking events during the course of the recording. In Figure 11-7 the wave front seen in Figure 11-4 has been overlaid with the MEA grid. The wave is clearly moving through the most active region of the network over the course of the recording.

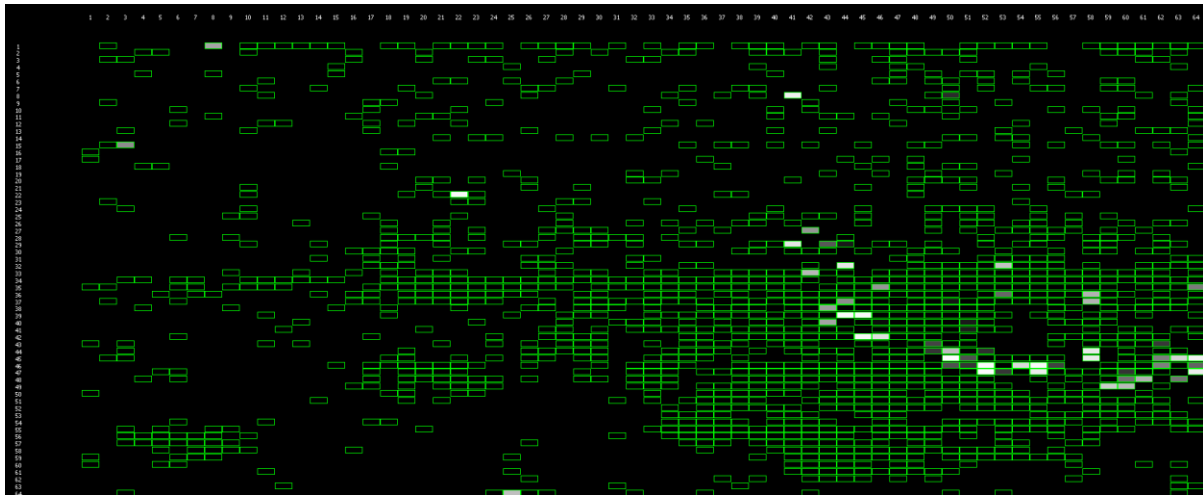


Figure 11-7: A wave of neural spiking activity moving through the neural networks most active region.

The MEA grid provides the highest level overview of the recording. It is useful to rapidly identify relatively isolated clusters of potentially interconnected neurons such as the groups highlighted in Figure 11-8 below. However in regions where spiking activity is high across many neurons it becomes impossible to visually identify cluster with this high level overview. This drawback is addressed through the introduction of heat maps (see next section).

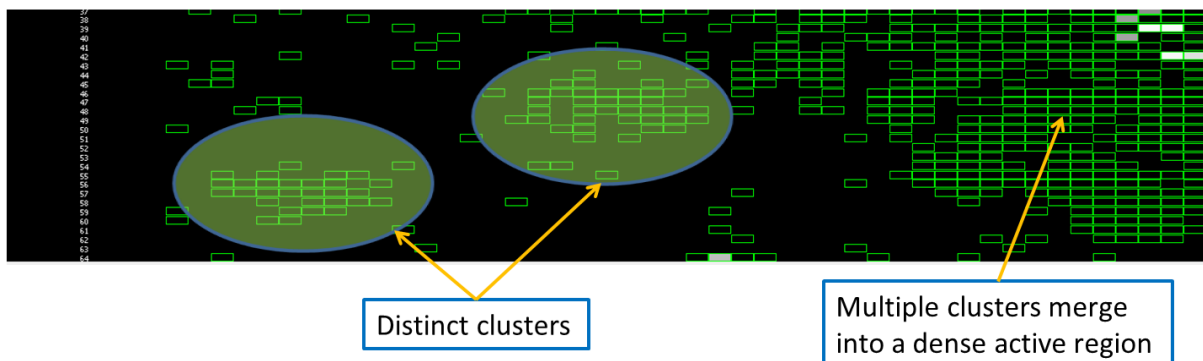


Figure 11-8: MEA Grid showing isolated clusters of active neurons and densely packed regions of active clusters.

It should be noted that the apparent gaps or “black” areas does not imply an absence of neurons or even that the neurons did not produce spiking activity. One of the primary challenges of making MEA recordings is to ensure good electrical connectivity across the tissue sample being recorded from. These regions may simply indicate areas of poor connectivity between the sample and the recording electrodes.

11.3.2 The Heat map overlay selection.

To provide more detailed information on the recorded spiking activity (particularly in recordings with densely packed active clusters) i-Animate provides a series of time filterable heat map overlays. These heat maps permit a detailed visual examination of recorded spiking activity both for the entire recording or if time filtered for a segment of it. The heat map is rendered atop the display of individual spiking events and by default is 30% transparent permitting both displays to be viewed together. The user may right click the heat map toolbar button to adjust the transparency level from 0% to 100%. This provides options to view the heat map overlay with or without the spiking event animation and when viewing both to find a comfortable level for the user.

A selection of heat map algorithms are provided as no single algorithm has proven consistently useful for all types of data (or even the different data sets of the same data type). An analysis of common heat map choices in the field of scientific visualisation showed that the rainbow colour map (see Figure 11-9) was by far the most used (Borland & Taylor II, 2007). This heat map colour space was the default used by 8 out of 9 visualisation toolkits examined by Borland and Taylor. An analysis of the published IEEE Visualisation papers showed it was used 51% of the time (Moreland, 2009a).



Figure 11-9: The classic rainbow colour map. (Moreland, 2009a)

Despite its dominance research has demonstrated that the rainbow colour map is almost certainly a poor choice for almost all forms of scientific data visualisation (Moreland, 2009a; Ware, 2012). The majority of the rainbow colour maps failings can be traced to the fact that it is “based on the colours of light at different wavelengths, the rainbow colour map’s design has nothing to do with how humans perceive colour” (Moreland, 2009a). The key failings identified by Moreland are:

1. Colours do not have a **natural ordering** unlike grayscale colours. “Perceptual experiments show that although a test subject with no prior training will always order grayscale colours in order of luminance (in one direction or the other), the test subjects will order rainbow colours in numerous different ways” (Moreland, 2009a; Ware, 2012).
2. The degree of **colour change perceived across the rainbow map is not a constant**. As can be seen in Figure 11-9 there is little change in colour across the blue, green and red areas while the colour change becomes far more rapid in the cyan and yellow areas producing a distinct “band” in these sections. Called Mach bands these visual artefacts obfuscates real data and can hide important data (Borland & Taylor II, 2007).
3. **Viewers with colour deficiencies cannot distinguish many colours considered “far apart” in the rainbow colour map**. Roughly 5% of the population cannot

distinguish between red and green in the classic rainbow colour scheme. More generally users with colour deficient vision often cannot distinguish colour “far apart” on the spectrum. This leads to severe miss-interpretation were radically different data values are perceived as being the same (Light & Bartlein, 2004).

To address these failing Moreland proposes that a colour map should result in the generation of heat maps that have the following key features:

1. The map yields images that are aesthetically pleasing.
2. The map has a maximal perceptual resolution.
3. Interference with the shading of 3D surfaces is minimal.
4. The map is not sensitive to vision deficiencies.
5. The order of the colours should be intuitively the same for all people.
6. The perceptual interpolation matches the underlying scalars of the map.

While points 2-6 are grounded firmly in scientific reasoning Moreland considers point 1, that the heat map should be aesthetically pleasing, to remain important as users will often select the visual tool to use based on its appearance.

Given Moreland’s criteria i-Animate provides three heat map colour schemes:

1. A **greyscale heat map** – This scheme naturally meets the majority of Moreland’s criteria with the exception of point 1 in that it does not appear particularly aesthetically pleasing.
2. A **rainbow heat map** – Despite its demonstrated shortcomings the rainbow heat map remains popular and its omission would be seen by some as a failure. Hence it has been included despite the known drawbacks. While it satisfies point 1 of Moreland’s criteria it fails to meet the other criteria.
3. Moreland proposes that a **diverging colour heat map** that moves from “cool to warm” colour regions is most suited to scientific visualisation. The cool to warm colours introduce to the divergent colour map a “natural” ordering of colours; Moreland argues that in all other respects the classic divergent colour map meets his criteria. Moreland’s divergent colour map is i-Animate’s third heat map option.

Figure 11-10 shows the heat map colour scheme recommended by Moreland and Figure 11-11 shows its application to geographic data as Moreland presented it in his paper to illustrate its effectiveness along with the RGB values that normalised values (0 to 1) would map too.

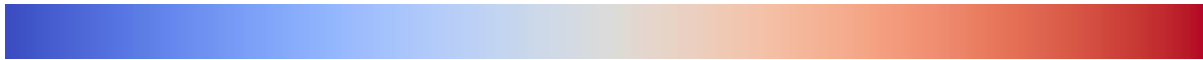


Figure 11-10: Divergent colour map for scientific visualisation

Normalised scalar values to RGB mapping				Geo-spatial data visualised using Moreland's colour scheme			
Scalar	Red	Green	Blue	Scalar	Red	Green	Blue
0.0	59	76	192	0.53125	229	216	209
0.03125	68	90	204	0.5625	236	211	197
0.0625	77	104	215	0.59375	241	204	185
0.09375	87	117	225	0.625	245	196	173
0.125	98	130	234	0.65625	247	187	160
0.15625	108	142	241	0.6875	247	177	148
0.1875	119	154	247	0.71875	247	166	135
0.21875	130	165	251	0.75	244	154	123
0.25	141	176	254	0.78125	241	141	111
0.28125	152	185	255	0.8125	236	127	99
0.3125	163	194	255	0.84375	229	112	88
0.34375	174	201	253	0.875	222	96	77
0.375	184	208	249	0.90625	213	80	66
0.40625	194	213	244	0.9375	203	62	56
0.4375	204	217	238	0.96875	192	40	47
0.46875	213	219	230	1.0	180	4	38
0.5	221	221	221

Figure 11-11: Moreland's colour space applied to geo-spatial data along with its RGB colour mapping for normalised scalar values. (Moreland, 2009b)

It is hoped that by providing a wide choice of colour schemes for the heat map function i-Animate will provide sufficient choice to all users while encouraging the use of the most effective scientific techniques.

11.3.3 Implemented heat maps in the i-Animate visualisation.

This section presents the three implemented heat map colour schemes and the associated i-Animate render for the control data set of 1411 spike trains recorded over 30 minutes.




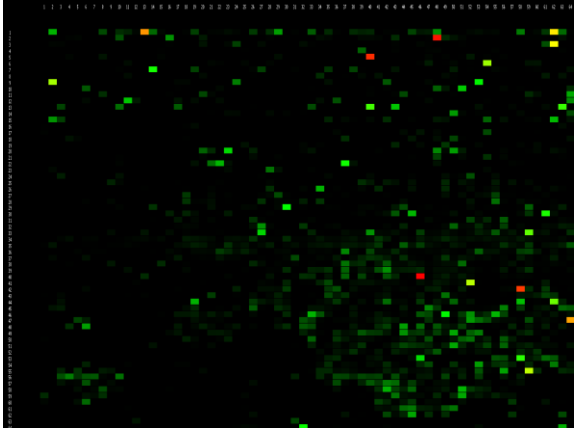

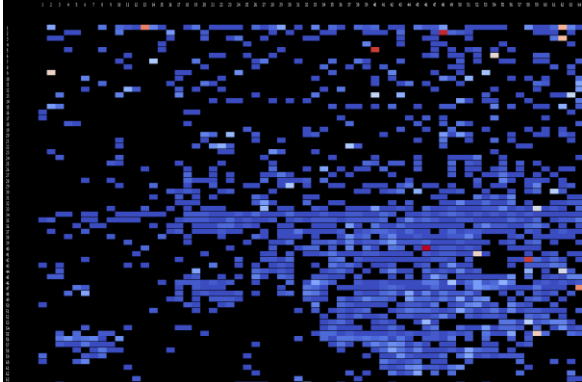
<u>Heat map colour scheme</u>	<u>Heat map applied to control data set</u>
Grayscale heat map 	
Rainbow heat map 	
Divergent colour map – Moreland 	

Table 11-2: I-Animate's heat map colour schemes in action

As can be seen from Table 11-2 the three implemented heat maps provide markedly different views of the same data set. In each view it is clear that there are key, highly active spike trains. The grayscale colour scheme is the least effective of the three with little structure apparent beyond the most active recording electrodes. The rainbow heat map reveals more structure and permits the identification of active regions that are likely to form clusters in any cross correlation analysis. Moreland's divergent colour scheme is however markedly more effective with many of the clusters that merged to form densely packed active areas becoming visible.

Chapter 12

Evaluation and the Way Forward

Summary

This chapter evaluates what has been achieved through this work and where the research may progress in the future.

12 Conclusions

This chapter reviews the research and the contributions it has made, there is a discussion of future work, and finally, the achievements in the context of the original research question.

12.1 Contributions

This section reviews the various contributions made by the thesis's software project. The software draws from the three primary fields of neuroscience, software engineering and data visualisation uniting aspects of all three to produce an extensible software tool for the analysis neural spike train recordings.

12.1.1 The Visualisation Studio (i-Pipeline)

The visualisation studio exploits the “pipelining” introduced by dataflow / visual programming languages to create pipelines of data processing activities. These pipelines process raw data into a form suitable for visualisation. The modern computer increasingly delivers its computational power through multi-core processors. To fully utilise this power a modern software application must be written in a parallel form with units of work being performed by multiple compute cores. The pipelines produced by dataflow programming are amenable to parallelisation permitting efficient execution of data processing activities in a multi-core environment.

Visual programming of a data processing pipeline also permits a researcher to rapidly introduce, remove or re-order data processing activities to make the processed data set more amenable to analysis. Visual analysis of the final data set is the preferred approach and the visualisation studio provides tools to facilitate big data analytics.

While this research has demonstrated the effectiveness of the visualisation studio using neuroscience data the framework itself is generic and can be applied to almost any field of research.

This research has also served to demonstrate that the researcher's typical desktop / laptop computer has the potential to be utilised far more effectively. A great deal of its computational power is wasted when traditionally developed software is used for data processing and analysis. This is most clearly demonstrated by the visualisations produced which now handle thousands and not hundreds of spike trains while remaining highly interactive.

12.1.2 The Neural Science Problem Domain Library (i-Pipeline)

The creation of the Visualisation studio's problem domain layer demonstrates how developers and domain experts working together can create libraries of algorithms and visualisations. The libraries creator has been left almost completely free to create a data representation for their problem domain. Any data analysis algorithm in the problem domain can be “wrapped” into a Visualisation studio

process. The process wrapper ensures the simple deployment and incorporation of the algorithm into a Visualisation studio dataflow pipeline.

Recognition of the modern trend of delivering increased computational power through a multi-core architecture allows algorithms and visualisations to fully exploit available compute resources. The implemented problem domain layer demonstrates this by writing its most computationally expensive algorithm (cross-correlation) in a way that adapts to available resources. Resources may range from a limited two core laptop, through a 4-6 core desktop system to a full HPC compute cluster.

Visualisations that apply the techniques of visual analytics have been re-engineered to also exploit the delivery of increased compute power through multiple cores. This has allowed visualisation previously limited to presenting hundreds of spike trains to present thousands while remaining highly interactive. To permit visual exploration of these large data sets Ben Shneiderman's Visual Information-Seeking Mantra of "*Overview first, zoom and filter, then details-on-demand*" has been applied. The final result is a significant improvement in the amount of data that can be processed and effectively visualised on the typical researcher's computer.

12.1.3 The i-Raster Visualisation

Somerville's i-Raster visualisation (Somerville et al., 2011) has been re-engineered. Its many spike train sorting algorithms are now available not only within the visualisation but as data pre-processing operations in the pipeline. Paralysis of the visualisations code has been the key to expanding its ability to present thousands rather than hundreds of spike trains. Grouping of spike train data has been used to provide an overview of the larger data set and a burst sort and grouping algorithm introduced. Interactive zoom and filtering tools permit the visual examination of detail in the spike train data set. Time filtering permits the detailed examination of a large data set and the generation of multiple (smaller) data sets.

12.1.4 The i-Grid Visualisation

Stuart's i-Grid visualisation (Stuart, Walter & Borisyuk, 2005) was originally used to assist researchers in visually identifying clusters of inter-connected neurons from their spike train recordings. The clustering technique of the original i-Grid has been expanded to become the foundation of a new "overview" – the cluster dendrogram. This overview serves to provide a means to zoom and filter i-Grid's display, while maintaining a complete overview of the data set.

The computationally expensive component of creating an i-Grid visualisation is the cross-correlation process. This algorithm has been re-coded to fully exploit the delivery of compute power through multi-core systems. In addition it has been re-written to use Message Passing Java (MPJ) allowing it to effectively utilise high performance compute (HPC) clusters. All of this is wrapped into a Visualisation studio pipeline process that allows even this complex algorithm to be rapidly used in any data processing pipeline.

12.1.5 The i-Animate Visualisation

I-Animate is a new visualisation that creates a representation of the modern (large) multi-electrode array used to simultaneously record spike trains. An animation of the recorded neuron spiking events over time is providing allowing the researcher to visually identify potentially connected neurons. Overall and time filtered views of the electrodes recorded activity levels are available through a selection of heat maps. Available heat maps range from the classic (but visually ineffective) rainbow heat map to Moreland's "Diverging Colour Map for Scientific Visualization".

12.2 Future Development

The Visualisation Studio and the implemented neuroscience problem domain library represent a significant change in the way data is processed and visualised. However modern technology still provides avenues through which greater gains can be realised. This section examines some of these avenues to advance the data processing elements and performance of the Visualisation Studio.

12.2.1 Exploitation of high performance GPU hardware

As described in chapter 4 techniques to code applications in a manner that makes full use of the modern computers multiple compute cores has lagged far behind the hardware's ability to deliver increased performance. This research has insisted that the application generated should make full use of multiple compute cores as a means of increased performance delivery. This has been taken to its furthest extreme with the computationally intensive pairwise cross-correlation calculations on which the i-Grid visualisation is based. The algorithm as implemented can be run on any computer supporting an MPJ installation from the humble laptop to the HPC cluster without modification. The algorithm will determine the available compute cores; assign cross-correlation operations and process results completely independently from the user. Most users will not have the benefit of access to a HPC cluster at will. The modern graphics processing unit (GPU) has already revolutionised the field of interactive data visualisation. Now the computing power of the graphics card is being opened up to accelerate scientific, analytics, engineering, consumer, and enterprise applications. This provides even a simple laptop with access to what is effectively a mini-HPC cluster. Programmatic access to GPU's is now available through application programming interfaces (API's) such as:

- NVIDIA's Compute Unified Device Architecture (CUDA),
- Khronos groups Open Computing Language (Open CL) and the
- OpenMP Architecture Review Board's Open Multi-Processing (OpenMP) API

The Visualisation Studio is a cross platform Java based application entirely capable of using these tools either through Java bindings or by invoking native code from Java using the Java Native Interface (JNI). This opens the possibility of using these mini-HPC clusters, present in almost all modern computers, to perform complex and computationally intensive data analysis in addition to their more

traditional roles in producing interactive displays. The Visualisation Studio could also optionally implement core components to run in such an environment. The parallel execution engine would be a prime candidate for such a conversion.

12.2.2 Distributed Computing and Cloud Computing

This research has focused on the production of an application deployed to the researcher's desktop. However modern technology affords other options such as distributed or cloud computing as a means to bring increased compute power to bear on a problem. The Apache Hadoop framework is an open source Java application to facilitate the distributed processing of very large data sets. Interfacing the Visualisation Studio with the Hadoop framework would offer access to HPC scale compute power even in the absence of a HPC or GPU option.

12.2.3 Application to other problem domains

Finally while the field of neuroscience is used in this research as the primary problem domain the developed Visualisation Studio application has been designed from the ground up as a general solution that can be applied to many different problems. Science and nature are replete with problems that are “embarrassingly parallel”. Such problems are well suited to the hardware and software tools (such as the Visualisation Studio) that are now emerging. Alternate problem domains and computationally intensive tasks which could be the subject of an implementation of the Visualisation Studio's problem domain layer might include:

- Financial analysis and reporting.
- Event simulation in particle physics.
- Ensemble calculations of numerical weather prediction.
- Genetic algorithms and many other evolutionary computing techniques.
- Brute-force searches in cryptography.
- Computer simulations comparing many independent scenarios, such as climate models.
- Fluid Mechanics

12.3 Conclusion

This research began by asking the question “How can Software Engineering and Visual Analytics be applied to aid the general analysis of scientific data and specifically current neural spike train data?” The development and testing of the Visualisation Studio application has shown that:

- Software tools that rely on delivering compute power from a single, ever faster; CPU can only provide limited interactive data visualisations.
- Software tools that embrace the delivery of compute power through multi-core hardware and a parallel programming model can offer greatly enhanced interactive data visualisations.

- Practical parallel programming models can emerge from dataflow programming's pipeline model and the creation of a visual programming language (VPL) for the problem domain under study.
- These VPL programs can tackle complex data analysis task while being efficiently and efficiently executed on modern multi-core computer systems.
- Interactive data visualisations can manage the resulting "big data sets" even on limited desktop hardware. This is achieved by combining Software Engineering with the techniques of Visual Analytics. In the case of this research the amount of data visualised increased by a factor of 10 from hundreds to thousands of spike trains.
- In future the most effective research will combine the efforts of a multi-disciplinary team to produce valuable results. At the core of these teams will be a problem domain expert and a software engineer.

Appendix 1

Appendix 1: Creating a new iPipeline toolbox process

Summary

This appendix serves as a guide for problem domain developers looking to build their own implementation of the iPipeline problem domain layer. It reviews the creation of a new data processing algorithm ready for inclusion into a workflows directed graph.

1 Creating iPipeline Toolbox Processes

The primary interface between iPipelines 'thin' framework layer and the 'thick' problem domain layer is the iPipeline toolbox. This graphical user interface (GUI) component provides user access to the problem specific data processing algorithm's and visualisations. The iPipeline framework supports extension of the problem domain layer allowing developers to encapsulate new data processing algorithms and add them to the visual programming language (VPL). In this appendix the process for encapsulating a new data analysis algorithm for use in either the input, data processing or output stages of the VPL. The most common of these three is the creation of a new data processing algorithm; therefore the appendix focuses on this task. Additional requirements to implement an input or output process will be detailed at the end.

1.1 Data processing algorithm to segment data by a time window

For the purposes of this appendix a problem domain algorithm requires implementation. Spike trains are a time ordered sequence of spike events and for this appendix an algorithm will be implemented that divides a single spike train into multiple time segments. Typically this would be used to divide a spike train into smaller segments and work with the sub-set of data. An application might be to take a single recording, divide it into say ten pieces and then feed this into a burst sort algorithm. This will apply the burst sort algorithm to ten different points in the original spike train. If the burst sorted segments shows repeating patterns, i.e. the same neurons fire in sequence, a possible connection can be inferred. Figure 1-1 illustrates the concept of dividing a spike train into multiple time windows each forming a new data set.

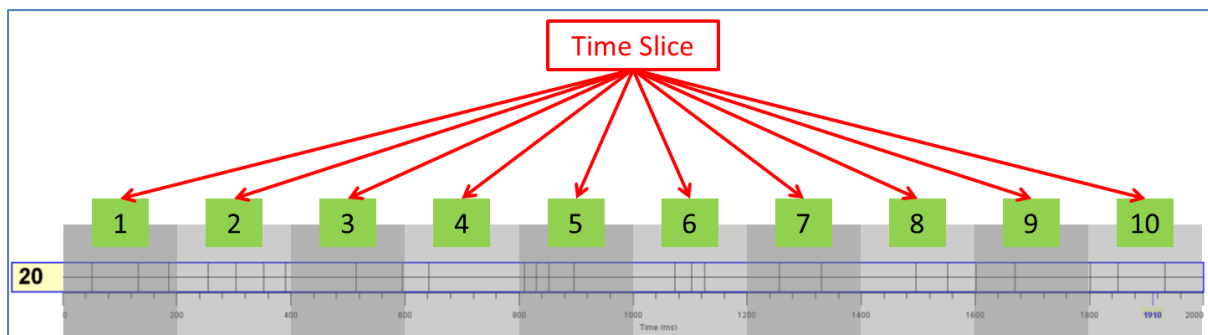


Figure 1-1: Dividing a single spike train recording into ten data sets using a 200ms time window

It is a common technique when analysing neural recordings to divide the spike train into time windows and examine each for recurring or repeating patterns. More complex implementations are possible but the purpose of this appendix is to review how the algorithm is created and integrated into the problem domain layer of iPipeline. The algorithm will therefore be kept relatively simple to avoid obscuring the development process with unnecessary detail.

1.2 Planning the algorithm

Chapter 6 section 6.2.1 outlined the points of contact between the iPipeline framework and the problem domain algorithm(s). Each of them will need to be considered when implementing the time segmenting algorithm. In summary the points that must be considered were:

1. The problem domain data model
2. Definition of permitted connections
3. Process settings panel(s)
4. Problem domain specific visual elements
5. The iPipeline toolbox

Each of these sections will be examined in turn and the required resources and Java code created.

1.2.1 How the problem domain data model is modified by this algorithm

The first and most important task is to identify the changes that the algorithm will make to the problem domains data model as it ‘flows’ through the VPL’s directed graph. In the case of time segmenting algorithm it can expect to receive at least one and possibly many INeuronAssembly objects. In addition to at least one set of raw data the user will have to provide at least one parameter in the form of the length of the time window. In addition it would seem sensible to allow the user to assign a new name to each generated sub-set. This should offer the option to include the time period covered from the original data set so that the user can maintain context. Additionally the original undivided data set should be preserved allowing the user to display an ‘overview’ of the entire data set and then switch to specific time segments when more detail is required. This applies Shneiderman’s information seeking mantra of overview first, zoom / filter and details on demand to generate a visually explore able data set.

1.2.2 Permitted connections to / from the time segmenting algorithm

The time segmenting algorithm is a classic data processing task with two potential sources of data. It can either:

1. Receive a dataset loaded directly from disk or other storage medium.
2. Receive a dataset from data processing algorithms earlier in the directed dataflow graph.

Equally there are two possible outputs to which it may be connected:

1. It may deliver the data sets it generates to a visualisation (output) module or
2. The data sets may serve as the input to further data processing modules.

1.2.3 Process settings panel(s)

The content required for the user controlled settings can be listed as:

1. A time window value
2. A scheme for naming the generated sub-sets

The time window size will be relatively simple to design – a single double value and a unit of measure. The user should not have to enter the time value in the same units used to measure spike times. Conversion between units should be automatic. The developer should also provide a sensible default value. In this case most spike trains are measured in milliseconds so this is a sensible unit to use. The ideal default time value is more difficult as it depends on the data set, task being performed and desired number of sub-sets. For this implementation a time window of 200ms has been selected to generate ten sub sets.

The implemented process will offer two naming conventions. In either case the user will be asked to provide a name string to which a second component is appended. The two options for appending to the name string will be:

1. The period of time covered by the time window that generated the sub set or
2. A numeric value that increments by one for each generated sub set.

In case one the user might provide the name string “Raw Data” and generate a set of names such as “Raw Data 0ms - 200ms”, “Raw Data 201ms - 400ms”, “Raw Data 401ms - 600ms” up to “Raw Data 1801ms - 2000ms” for a two second recording. In case two the time value is replaced with a numeric integer i.e. “Raw Data sub set 1”, “Raw Data sub set 2” up to “Raw Data sub set 10”.

1.2.4 Creating visual elements

An iPipeline process requires at least three visual elements to be created. These are:

1. A large 50 x 50 pixel graphic used to represent the process as part of the dataflow directed graph. This image should visually represent the processes operation on the data set. In the case of the time segmenting algorithm a representation similar to Figure 1-1 will be created showing a segmented spike train.
2. A small 16 x 16 pixel graphic being a scaled down version of (1) will be needed to represent the process in the iPipeline toolbox. This image will be composed with the ▲ process symbol to create the toolbox entry for the new process. Normally this can be generated as a 1/3rd scale version of (1).
3. The process settings panel should be designed to include the elements identified in section 1.2.3. A mock up is shown in Figure 1-2 that addresses all the required elements that the user must define.

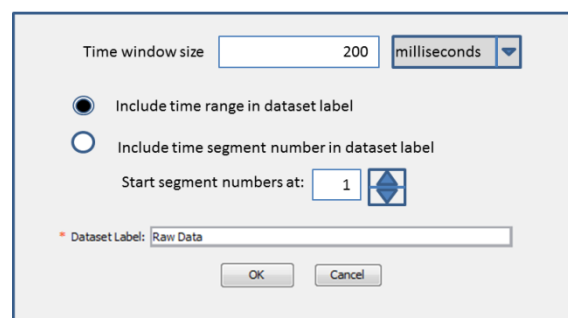


Figure 1-2: Time segmenting process setting panel (mock up)

The final step will be to combine the

1.2.5 The iPipeline Toolbox

The iPipeline toolbox obtains its contents from a Java JAR archive that contains all problem domain specific code. Essentially this is the developer produced ‘thick’ problem domain layer of iPipeline. Once the process implementation is complete the JAR archive must be re-built to include the new process. The process may be summarised as:

1. Implement all required code in the **ProcessLibrary.Processes** package of the iPipeline project.
2. Create a Java JAR archive named ‘**ProcLib.jar**’ that contains all sub-folders of the iPipeline projects ProcessLibrary folder.
3. Deploy this Java archive along with the iPipeline executable. Optionally you may hide the file so it is not visible.

At start-up iPipeline will scan its executing folder for the ProcLib.jar archive and build the contents of its toolbox based on the processes and other resources in the archive. Essentially the archive can be thought of as the problem domain layer of the iPipeline implementation. Replacing the ProcLib.jar archive allows the problem domain layer to be changed, i.e. from neuroscience to financial analysis. Obviously creating a new problem domain layer involves considerable work by a domain expert and coders but once done reconfiguring to a new problem domain is relatively simple.

For developer convenience the process of preparing the ProcLib.jar archive can be automated. Simply add the functionality to the ProcessLibrary.Processes package and execute the LibCreate.bat file to automatically construct the JAR archive.

1.3 Implementing a skeleton data analysis process

This section details the steps required to create a new data analysis process. Examples of the produced code are shown using the time segmenting algorithm. After adding a new class to the ProcessLibrary.Processes package have it extend the BaseProcess class. To create a new data analysis process the following setup work is required:

- Create the code that allows the new process to be deployed from the toolbox. It would not be normal to modify this code except to change the declaration of the variable being added to the new class's name. The relevant points at which to make the change are shown **highlighted in yellow**. The standard code snippet is shown in Figure 1-1:

```
public Action getAddWidgetAction() {
    return new AddAction();
}

private class AddAction extends AbstractAction {

    @Override
    public void actionPerformed(ActionEvent e) {
        if (MouseDualClickDetector.isDoubleClick()) {
            PipeControl objControl = PipeControl.getInstance();
            if (null != objControl) {
                TimeSegmentingProcess widgetToAdd = new TimeSegmentingProcess();
                objControl.addProcess(widgetToAdd);
            }
        }
    }
}
```

Table 1-1: Standard code snippet to add a new process to the iPipeline virtual desktop

- Create the processes standard set of constructor methods. This involves the creation of the four constructors shown in Figure 1-2. As before the only change that is required is to the name of the class **highlighted in yellow**.

```
//Constructors
public TimeSegmentingProcess() {
    super();
}

public TimeSegmentingProcess(String strNewName) {
    super(strNewName);
}

public TimeSegmentingProcess(JDesktopPane objNewDesktop) {
    super(objNewDesktop);
}

public TimeSegmentingProcess(String strNewName, JDesktopPane objNewDesktop) {
    super(strNewName, objNewDesktop);
}
```

Table 1-2: 'Standard Process' constructors' code snippet

With the above steps completed the new process can be instantiated by the iPipeline framework and added to a dataflow directed graph. The next steps implement the details of the algorithm.


- The base process class provides protected class attributes which should be initialised to define the connectivity and visual appearance of the process widget. This will require the creation of the visual icons used to represent the process. A template method with the signature `protected void initProcess()` is provided by the base process class. This should be overridden and customised for the algorithm being implemented. Table 1-3 shows the overridden method with the minimal code which should be included for all processes.

```
protected void initProcess() {
    //The GUI Widget for this class should always be this text.
    this.setProcessName("Time Segment");
    //Setup Icon for parameters (Time segmenting Icon)
    this.icnDisplay = new
    ImageIcon(getClass().getResource("/ProcessLibrary/Icons/TimeSegLarge.png"));
    this.icnSettingsCog = new
    ImageIconAnimated(getClass().getResource("/ProcessLibrary/Icons/TimeSegLarge.png"));
    //Specify Connections
    this.icnParent = IconFactory.getInstance().getConnectorIcon(ConnectorType.CON110);
    this.icnChild = IconFactory.getInstance().getConnectorIcon(ConnectorType.CON011);
    //Setup Menu Display Name and ImageIcon
    this.strDisplayName = "Time Segment";
    //Setup Other Attributes here
}
```

Table 1-3: Minimal process initialisation code for the `initProcess` template method.

The initialisation code achieves three tasks:

1. Provides a process and a display name. The process name is used on the process GUI widget to identify the process while the `strDisplayName` attribute is shown in the toolbox. Normally these are the same but this is not required, it may be useful to provide a longer more descriptive name for the toolbox.

2. Defines two icons used by the process widget to represent the process. The first of these (icnDisplay) is used when all parameters have passed validation. The second (icnSettingsCog) is used when the parameters fail validation. It will be composed with the rotating cogs icon and displayed as a visual prompt that this process requires its parameters to be correctly configured. Usually the animated cogs are sufficient to denote the need for the user to re-configure the parameters. However this provides the option to go further and modify the underlying graphic. In the case of this process no modification is needed so the same graphic is used.
 3. Defines the types of process that may connect to this process on an input arc and on an output arc of the dataflow directed graph. Permitted types are defined by the information processing cycle. These are input, processing and output processes. An enumeration provides a true / false flag used to define permitted connections. The ConnectorType.CON110 used for the processes input arcs means that input and data processing processes may connect to this process but not output processes. Similarly the ConnectorType.CON011 means that this processes output arcs may lead to either another data processing process or an output process.
- To complete the initialisation of the visual elements it is necessary to instantiate the graphic that will be used in iPipelines toolbox to represent the process. This image will be composed with the  process symbol to create the toolbox entry for the new process. Again a standard code snippet customised to the appropriate graphic is used as shown in Table 1-4:

```

@Override
public ImageIcon getProcessIcon() {
    if (null == this.icnMenuIcon) {
        //Load Image resource from Jar Archive
        JarResources jar = new JarResources("ProLib.jar");
        Image objImage =
Toolkit.getDefaultToolkit().createImage(jar.getResource("ProcessLibrary/Icons/TimeSegSmall.png"));
        this.icnMenuIcon = new ImageIcon(objImage);
    }
    return this.icnMenuIcon;
}

```

Table 1-4: Loading the customised toolbox image for the new process

- One visual component is still missing at this point – the settings panel for the process. As the settings panel is specific to the process for now simply create a class that extends from Java's JPanel. This will produce an empty JPanel which can be customised later. As a naming convention use the name of the process that will own the panel followed by 'SettingsPanel'. In the case of the process being developed this becomes TimeSegmentingSettingsPanel.
- With the empty panel created it must now be attached to the owning process. Override the method JPanel setupParamWindow() to match the following code snippet:

```

protected JPanel setupParamWindow() {
    TimeSegmentingSettingsPanel jpnDisplay = new
    TimeSegmentingSettingsPanel(this);
    jpnDisplay.addCancelActionListener(this);
    jpnDisplay.invalidate();
    return jpnDisplay;
}

```

Table 1-5: Creation of process settings panel for the time segmenting process

- The blank settings panel is now attached to its owning process. Of course at the moment it does nothing so the next step is to define a process specific group of settings. Obviously most of the code for a settings panel will be specific to its associated process. However the code that manages the parameters created and makes them available to the associated process is fairly standard. It still requires customisation to handle the specific parameters but the abstract concept of managing parameters is expressed through the ISettingsPanel interface. The developer should implement this interface after designing the settings panel so that it can exchange parameters with the owning process. Creating a settings panel is therefore a two stage process:
 - Implement the JPanel GUI components and associated input validation logic. This step is highly specific to the process being developed so only general guidance can be given. Nevertheless the considerations for the time segmenting algorithm's settings are reviewed below as an example of the process.
 - Implement the ISettingsPanel interface. This is the more general step and more specific guidance can be given. Nevertheless there remain points at which the implementation must be customised to the algorithms specific needs.

1.3.1 Time Segmenting algorithms settings panel GUI creation

Table 1-6 shows the time segmenting settings panel implemented in the Netbeans GUI designer with its initial defaults:

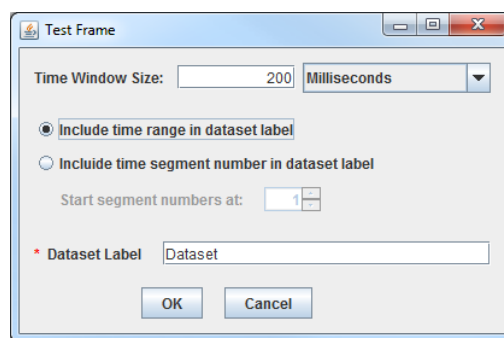


Table 1-6: Time segmenting settings panel GUI implementation

For the time segmenting algorithm the following business rules have been implemented:

1. Default time measure is Milliseconds with the option to change to either seconds or microseconds.

2. The default time window size is 200 which will create 5 datasets for every second of spike train recording. Defining this is the aim of the settings panel and it is expected that most users will be changing the values for item (1) and (2).
3. The radio buttons allow the user to adopt one of schemes for the generated datasets:
 - a. Append the time period covered by the dataset to the dataset's label (name) or
 - b. Append a sequential number to the dataset's label (name). The number must be a positive integer greater than 0.
4. A dataset label or name is required for clear identification of the dataset when it is visualised. The default label "Dataset" is supplied and if the user blanks this field or fills it with whitespace and no other characters the default will be restored.

The selected defaults provide an immediately executable process though the user should review at least items (1) and (2) to ensure their suitability for the data set being analysed.

1.3.2 Time segmenting algorithm's ISettingsPanel interface implementation

The implementation of the ISettingsPanel interface provides the link between a process and its associated configuration settings. The settings are required to be managed in a generic manner as the iPipeline framework cannot predict the number or types of the individual parameters. This is achieved through the ParameterManager which provides a container that can manage a list of key / value pairs. Each parameter is stored as a Parameter object which uses a string as the identifying key and any 'value'. The value element is any type of data such as integer, double, string, enumeration or object. The current parameter collection is maintained by the process to which the settings panel belongs but the panel can retrieve the collection via the ISettingsPanel interface.

The ISettingsPanel interface and documentation is shown in Table 1-7. A discussion of the implementation for each of the five methods follows.

```

/**
 * Interface implemented by a class that extends JPanel to serve as a settings window
 * for a process
 * @author Roy
 */
public interface ISettingsPanel {

    /**
     * Accessor method to retrieve the current collection of parameters from
     * the owning process
     * @return An IParameterManager interface to the parameter collection
     */
    public IParameterManager getCurrentParameters();

    /**
     * This method takes the current list of parameters from the settings panel's
     * owning IProcess and uses it to update the various fields / GUI
     * components on the panel
     */
    public void refreshParameters();

    /**
     * Adds an action listener which will respond to a left click on the OK
     * button. The process which owns the settings panel will normally be the
     * listener
     * @param objNewListener - Any ActionListener
     * @return boolean true if the new listener was added, false otherwise.
     */
    public boolean addOkButtonActionListener(ActionListener objNewListener);

    /**
     * Adds an action listener which will respond to a left click on the CANCEL
     * button. The process which owns the settings panel will normally be the
     * listener
     * @param objNewListener - Any ActionListener
     * @return boolean true if the new listener was added, false otherwise.
     */
    public boolean addCancelButtonActionListener(ActionListener objNewListener);

    /**
     * This method test the current settings shown on this settings panel and
     * determines if they are valid for the processing algorithm. Invalid settings
     * will prevent execution of the attached process. The process widget will
     * show a set of rotating cogs indicating that the user must correctly
     * configure the settings panel before use.
     * @return boolean true if the current settings are usable by the processing
     * algorithm, false otherwise.
     */
    public boolean hasValidSettings();
}

```

Table 1-7: The ISettingsPanel interface

1.3.2.1 Method 1: `public IParameterManager getCurrentParameters();`

The implementation for this method is standard across all settings panels and provides an accessor for the owning processes parameter collection. The standard code is shown in Table 1-8.

```
private IProcess objParent = null;

public IParameterManager getCurrentParameters() {
    return this.objParent.getParameters();
}
```

Table 1-8: The `getCurrentParameters()` implementation

Note that although the `objParent` is uninitialized; it will be set by the `iPipeline` framework when creating the settings panel. See Table 1-5 for the initialisation code the guarantees this.

1.3.2.2 Method 2: `public void refreshParameters();`

The purpose of this method is to synchronise the currently displayed parameter values on the settings panel with those store by the panels owning process. Obviously the code will be customised for each GUI component but the general structure for each parameter should follow this pseudo code:

1. From the parent process obtain the list of parameters and search for the required parameter
2. If the parameter is found extract its value and set the relevant component on the settings panel GUI to reflect the value
3. If the parameter is not found initialise the settings panel GUI component to a sensible default value.

Table 1-9 provides sample code showing the processing for the dataset label parameter. This code can be extended to process any parameter. Note that the developer is responsible for assigning meaningful parameter names. As before the points where a developer will need to make changes for the specific parameter being coded are highlighted in yellow.

```
public void refreshParameters() {
    //Display Name update
    IParameter objDispNameParam =
this.objParent.getParameters().getParameter("DisplayName");
    if (null != objDispNameParam) {
        if (objDispNameParam.getParamaterValue() instanceof String) {
            String strNameValue = (String) objDispNameParam.getParamaterValue();
            this.txtDSName.setText(strNameValue);
        } else {
            this.txtDSName.setText("Dataset");
        }
    } else {
        this.txtDSName.setText("Dataset");
    }
}
```

Table 1-9: Sample code to refresh the dataset label / name parameter

1.3.2.3 Method 3: *public boolean addOkButtonActionListener(ActionListener objNewListener);*

All settings panels should feature an 'OK' and 'Cancel' button which following the Java convention an ActionListener object should be provided that encapsulates the relevant code. The parent process usually defines this listener, indeed it often IS the listener. As such this accessor method allows an external object to associate an ActionListener with the 'OK' button. This is standard default code that can in most instances be copy / pasted.

```
public boolean addOkButtonActionListener(ActionListener objNewListener) {
    boolean blnWasSet = false;
    if (null != objNewListener) {
        this.btnOk.addActionListener(objNewListener);
        blnWasSet = true;
    }
    return blnWasSet;
}
```

Table 1-10: Standard accessor to add an action listener to the settings panel 'OK' button

1.3.2.4 Method 4: *public boolean addCancelActionListener(ActionListener objNewListener);*

All settings panels should feature an 'OK' and 'Cancel' button which following the Java convention an ActionListener object should be provided that encapsulates the relevant code. The parent process usually defines this listener, indeed it often IS the listener. As such this accessor method allows an external object to associate an ActionListener with the 'Cancel' button. This is standard default code that can in most instances be copy / pasted.

```
public boolean addCancelActionListener(ActionListener objNewListener) {
    boolean blnWasSet = false;
    if (null != objNewListener) {
        this.btnCancel.addActionListener(objNewListener);
        blnWasSet = true;
    }
    return blnWasSet;
}
```

Table 1-11: Standard accessor to add an action listener to the settings panel 'Cancel' button

1.3.2.5 Method 5: *public boolean hasValidSettings();*

This method provides the parent / owning process with a means of validating that the current configuration of parameters on the settings panel can be used to execute the algorithm. If the algorithm can be successfully completed with the current settings then this method should return true. If any setting is invalid, for example a time window size of 0 in the time segmenting algorithm, the method should return false prompting the iPipeline framework to visual indicate that the process needs further user configuration.

The code for this method obviously depends on the number of parameters and types of validation required so it is impossible to give a standard code snippet for it. However as an example Table 1-12 contains example code to validate that the user has provided a dataset name for the new dataset created by the processing algorithm.

```

public boolean isValidSettings(){
    boolean blnResult = false;
    //Ensure that a valid name exists for the combined dataset
    String strName = this.txtDSName.getText();
    //Ensure a non-null & non-empty string exists
    if(null != strName && !strName.isEmpty()){
        //Strip whitespace and ensure length at least 1
        strName = strName.replace(" ", "");
        if(0 < strName.length()){
            blnResult = true;
        }
    }
    return blnResult;
}

```

Table 1-12: Sample implementation to validate that the dataset name has been configured

1.3.3 Responding to OK / Cancel button events on the settings panel

Every settings panel should include an 'Ok' and 'Cancel' button for the user to accept / reject changes to the settings. Two methods are provided to respond to the users 'Ok' and 'Cancel' button events allowing the developer maximum flexibility in responding. The developer may choose to respond via either the settings panel itself or via its parent process object. By default the code snippets given so far have programmed the parent process to respond to the 'Cancel' buttons click event. For most processes the action taken is simply to hide the settings panel without updating the process settings. This is usually delegated to the parent process and the code required is reviewed in section 1.3.4 when integrating the settings panel with its parent process. The 'Ok' button response is highly dependent on the actual parameters added to the settings panel. As such its behaviour is usually defined by the settings panel itself following the principle of encapsulating data and related behaviours into the same class. The 'Ok' button handler's primary task is to scan all user inputs and package them into the parent / owning processes ParameterManager object. It should also invoke the parent processes 'Ok' handler if the developer has any process level code that should execute when the process settings code is updated. The code snippet in Table 1-13 should be sufficient to perform these tasks:

```

private void btnOkActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    //Scan new settings and store them to owning process
    this.packageSettingsIntoManager();
    //Invoke any parameter change handlers on the parent process
    if (null != this.objParent) {
        if (this.objParent instanceof ActionListener) {
            ((ActionListener) this.objParent).actionPerformed(evt);
        }
    }
}

```

Table 1-13: Standard code to process a parameter update on the settings panel

The packageSettingsIntoManager() method from the Table 1-13 code snippet performs the work of scanning the settings panel for user changes and updating the parameters in the parent / owning processes ParameterManager object. Obviously no standard code can be given for this as it will be based on the problem domain and the required parameters to

control the process being implemented. The pseudo code for this method can however be clearly stated as follows:

1. Ensure that the SettingsPanel has a parent / owner process
2. For each parameter on the settings panel
 - a. Assign a **parameter name** (this should match the name used in the refreshParameters() method; see code snippet in Table 1-9).
 - b. Extract the **parameter value** from the settings panel data entry field.
 - c. Get the parent process parameter manager and call the setParameter method passing the **name, value** pair.

As an example of how this method should be implemented the time segmenting algorithm's implementation is shown in Table 1-14.

```
private void packageSettingsIntoManager() {
    if (null != this.objParent) {
        if (!this.txtDSName.getText().equalsIgnoreCase("")) {
            this.objParent.getParameters().setParameter("DisplayName",
this.txtDSName.getText());
        }
        SpikeTimescale timeScale = (SpikeTimescale) this.cbxTimescale.getSelectedItem();
        this.objParent.getParameters().setParameter("TimeWinScale", timeScale);
        TimeSegmentingMode mode = TimeSegmentingMode.APPEND_TIME_PERIOD;
        if (this.radBtnTimeSegmentNumber.isSelected()) {
            mode = TimeSegmentingMode.APPEND_SEQUENCE_NUMBER;
        }
        this.objParent.getParameters().setParameter("TimeSeqMode", mode);
        SpinnerModel model = this.spnSegStart.getModel();
        if (model instanceof SpinnerNumberModel) {
            SpinnerNumberModel numModel = (SpinnerNumberModel) model;
            Number number = numModel.getNumber();
            Integer startSeqNo = number.intValue();
            this.objParent.getParameters().setParameter("TimeSeqNo", startSeqNo);
        }
        try {
            Integer winSize = Integer.parseInt(this.txtTimeWin.getText());
            this.objParent.getParameters().setParameter("TimeWinSize", winSize);
        } catch (NumberFormatException ex) {
            //Time window size could not be set revert to default of 200
            this.objParent.getParameters().setParameter("TimeWinSize", 200);
        }
    }
}
```

Table 1-14: Example implementation for the packageSettingsManager() method

1.3.4 Integrating the settings panel with the parent process

With the settings panel completed it must now be integrated with its parent or owning process. The panel has already been created using the code in Table 1-5. The process has the option of responding to the settings panels 'Ok' and 'Cancel' button behaviours. Usually the cancel operation simply hides the settings panel and the ok option does not need process specific code. Table 1-15 gives the code snippet needed to implement this default behaviour.

```

protected void processParamAction(ActionEvent e) {
    //Process Ok action
    if (e.getActionCommand().equals("OK")) {
        this.paramOKClick(e);
    }
    //Process Cancel action
    if (e.getActionCommand().equals("Cancel")) {
        frmParams.setVisible(false);
    }
}

private void paramOKClick(ActionEvent e) {
    //Implement any process specific code to execute when the process parameters
    //are changed / updated
    //Close parameters window
    frmParams.setVisible(false);
}

```

Table 1-15: Default code to respond to settings panel changes

At this point the settings panel has been successfully integrated with its owning parent process. The new process itself is almost complete; all that remains is to implement the process core algorithm.

1.3.5 Implementation of process core algorithm

The base process class provides a single protected method which the problem domain developer must override and implement with the algorithm that performs the data processing. The method signature is:

```
protected boolean processAction(Object[] objData);
```

This is of course the concrete implementation of the algorithm encapsulated into the process object. As such it is impossible to define any standard code as it depends wholly on the problem domain being studied, the data available and the desired analysis to be performed. However it is possible to give some general advice and guidance to problem domain developers.

Almost all implementations of this method should begin by compiling a list of the process(es) attached as a parent to this process. These will be the processes delivering a dataflow token along the input arcs of the directed dataflow graph. The base process implementation provides this list in the form of the `vecParents` vector variable. From each of these processes a `IProcessingResult` object can be obtained by calling the `getResult()` method. The retrieved `IProcessingResult` represents the dataflow token delivered by the input arc and will contain one or more instances of the domain specific data model. It is recommended that the problem domain developer writes a static `unpackData()` and `packData()` method that can extract the list of data models from the `IProcessingResult` envelope or package a data model list into the envelope. The `IProcessingResult` essentially stores data as name value pairs allowing almost any data model, no matter how complex, to be stored. As an example Table 1-16 shows the structure adopted in the implemented neural analysis model:

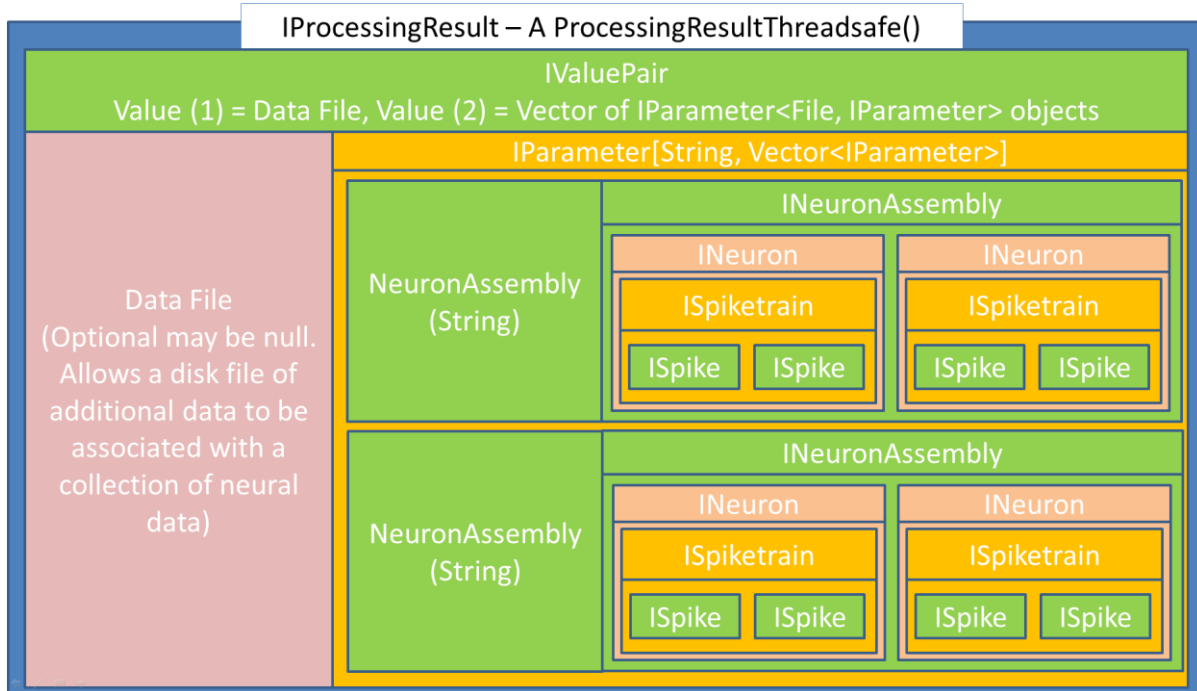


Table 1-16: IProcessingResult Dataflow token with neural model embedded

Chapter 7 provides a detailed discussion of the problem domain data model adopted for neural data analysis. For our purpose here the problem domain static pack method accepts a list of `INeuronAssembly` objects in the form of a vector and returns an `IProcessingResult` object with the Table 1-16 structure. Equally the static unpack method unwraps this structure and retrieves the original vector of `INeuronAssembly` objects.

Once the dataset has been extracted from the dataflow token the processing algorithm is applied to each delivered dataset in turn. In the case of the time segmenting algorithm illustrated in this appendix this involves the creation of additional neuron assemblies. Each assembly contains the same set of neurons with the spike trains modified to include only those spikes that fall within a given time range.

At the end of processing a vector of new / modified datasets is produced which is once more packaged into an `IProcessingResult` object that serves as the output arc dataflow token for this process.

In summary then the standard pseudo code for implementing the `processAction` method becomes:

1. Obtain vector of parent `IProcess` objects.
2. For each parent process retrieve the `IProcessingResult` dataflow token by calling `getResult()` on each process.
 - a. For each retrieved result. Unpack the problem domain specific data model from the `IProcessingResult` dataflow token.
 - b. For each dataset delivered apply the algorithm being implemented to the delivered data.
 - c. Add the modified / new dataset to a vector of problem domain data models.
3. Pack completed vector of problem domain models into `IProcessingResult` and store into the processes protected attribute `objResultSet`. This places the

IProcessingResult onto the processes output arc(s) to trigger further processing of the directed workflow graph.

1.3.6 Introducing the new process to iPipeline’s problem domain

At this point a fully functional process has been created. This process must now be deployed to iPipeline’s problem domain layer. This process is essentially automated with its own executable batch file which builds the required Java jar file, the **ProcLib.jar** file. Nevertheless it is worth reviewing how this is achieved to better understand the application structure.

As was noted in Chapter 6 iPipeline is divided into two ‘layers’ a thin framework layer and a thick problem domain layer. When deployed these ‘layers’ take the form of two Java jar files that should be deployed together. Table 1-17 and Figure 1-3 detail which jar corresponds to which layer and shows them deployed and ready for execution.

iPipeline Layer	Java Jar File
Framework Layer	PipelineTestbedV021.jar
Problem Domain Layer	ProcLib.jar

Table 1-17: iPipeline ‘Layer’ Jar files

Name	Date modified	Type	Size
lib	16/04/2014 12:15	File folder	
PipelineTestbedV021.jar	16/04/2014 12:17	Executable Jar File	11,682 KB
ProcLib.jar	05/06/2014 10:12	Executable Jar File	1,075 KB
README.TXT	16/04/2014 12:15	Text Document	2 KB

Figure 1-3: A deployed iPipeline framework with process library

The toolbox of processes presented to the application user depends on the processes contained in the ProcLib.jar. This approach maintains a distinct separation between the two layers and allows iPipeline to switch to a new problem domain simply by providing a different ProcLib.jar file that contains the new problem domain processes.

The iPipeline source files provide a package named ‘ProcessLibrary’ which the developer may use to add and organise the problem domain specific code. Step 1.2.5 instructed you to place your new process into the ‘ProcessLibrary.Processes’ package. It is this package that is used to produce the ProcLib.jar file. The ProcessLibrary package and its sub packages for the neural analysis problem domain layer are shown in Figure 1-4. The only required sub package is the Processes package. The other packages simply assist in organising the various problem domain elements. The ‘ProcessLibrary.Processes’ package must be present however as it is from here that the iPipeline framework will draw the processes used to populate its process toolbox making them available to the user. As a practical note it makes sense for problem domain developers to use additional packages to organise their work, for example the data model implementation and the visualisations implementations.

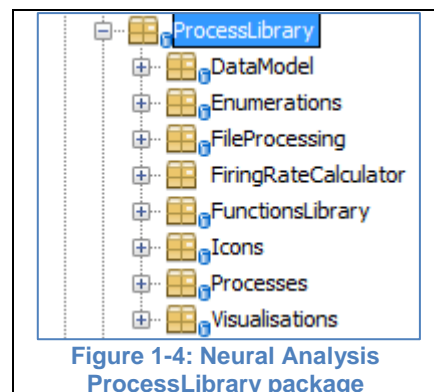


Figure 1-4: Neural Analysis ProcessLibrary package

The creation of a new problem domain library is achieved by compiling the 'ProcessLibrary' package into its own jar file. A simple two-step process will achieve this goal:

1. Build the iPipeline project so that the compiled classes are placed into the projects build\classes\ProcessLibrary\ folder. This is their normal destination for a Netbeans project.
2. Execute the batch script shown in Figure 1-5 to generate the ProcLib.jar file containing only the problem domain layer code.

To use any specific problem domain jar file simply copy / paste it to the same directory as the Framework Layer jar, PipelineTestbedV021.jar, as shown in Figure 1-3.

```
1 set LIBCLASSES=.\build\classes\ProcessLibrary\  
2 mkdir .\ProcessLibrary  
3 xcopy %LIBCLASSES%*. * .\ProcessLibrary\ /S  
4 jar cf ProcLib.jar .\ProcessLibrary  
5 RMDIR /S /Q .\ProcessLibrary  
6 set LIBCLASSES=
```

Figure 1-5: Batch script to generate the ProcLib.jar file from the compiled code.

Appendix 2

Appendix 2: Glossary of Terms

Summary

This appendix serves as a guide to various technical terms used throughout the thesis and across the three fields of Software Engineering, Visualisation and Neuroscience.

Term / abbreviation	Description
CPU	Central Processing Unit (CPU) - the part of a computer that performs logical and arithmetic operations on the data as specified in the instructions. (HarperCollins, 2014)
Hadoop	Apache Hadoop is an open-source software framework written in Java for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. (Apache, 2014)
JSON	JavaScript Object Notation (JSON) - is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language. (ECMA, 2013)
JUNG	Java Universal Network / Graph framework is a software library that provides a common and extendible language for the modelling, analysis, and visualization of data that can be represented as a graph or network. It is written in Java, which allows JUNG-based applications to make use of the extensive built-in capabilities of the Java API, as well as those of other existing third-party Java libraries. (Madadhain J. et al., 2005)
Machine Learning	A field of study that gives computers the ability to learn without being explicitly programmed. (Samuel, 1953)
MapReduce	MapReduce is a programming model and an associated implementation for processing and generating large data sets with a parallel, distributed algorithm on a cluster. (Dean & Ghemawat, 2004)
MEA	Multi-Electrode Array (MEA) - are devices that contain multiple plates or shanks through which neural signals are obtained or delivered, essentially serving as neural interfaces that connect neurons to electronic circuitry. (Taketani & Baudry, 2006)
MPI	Message Passing Interface (MPI) - is a language-independent communications protocol used for programming parallel computers. Both point-to-point and collective communication are supported. MPI "is a message-passing application programme interface, together with protocol and semantic specifications for how its features must behave in any implementation." (Gropp et al., 1996) MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today. (Sur, J.Koop & Panda, 2006)

Term / abbreviation	Description
MPJ	Message Passing Java (MPJ) - An open source Java message passing library that allows application developers to write and execute parallel applications for multicore processors and compute clusters/clouds. (Shafi & Jameel, 2006)
NoSQL	A variety of non-relational databases that are used for handling huge amounts of data in the multi-terabyte and petabyte range. Rather than the strict table/row structure of the relational databases that are widely used in all enterprises, NoSQL databases are field oriented and more flexible for many applications. They are said to scale horizontally, which means that inconsistent amounts of data can be stored in an individual item/record (the equivalent of a relational row). The "not" SQL designation comes from the SQL language used to query a relational database. (Freedman, 2016)
OOP	Object Orientated Programming (OOP) - is a programming paradigm based on the concept of "objects", which may contain data, in the form of fields, often known as attributes; and code, in the form of procedures, often known as methods. A distinguishing feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated (objects have a notion of "this" or "self"). In OO programming, computer programs are designed by making them out of objects that interact with one another. (Kindler & Krivý, 2005)
Thread safe	Thread safety is a computer programming concept applicable in the context of multi-threaded programs. A piece of code is thread-safe if it only manipulates shared data structures in a manner that guarantees safe execution by multiple threads at the same time. (Oracle, 2015)
UML	The Unified Modelling Language (UML) – The Object Management Groups (OMG) standard for specifying, visualising, and documenting models of software systems, including their structure and design. (OMG, 2005)
VISA	The Visualisation of Inter Spike Associations project that developed the original i-Raster and i-Grid visualisations. Funded by the Engineering and Physical Sciences Research Council (EPSRC). (Stuart, Walter & Borisyuk, 2003)
VPN	Virtual Private Network (VPN) - extends a private network across a public network, such as the Internet. It enables users to send and receive data across shared or public networks as if their computing devices were directly connected to the private network, and thus are benefiting from the functionality, security and management policies of the private network. (Mason, 2001)

Appendix 3

Appendix 3: Publications

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

THE BACKGROUND AND IMPORTANCE OF EXPLOITING MULTIPLE CORES: A CASE STUDY IN NEUROPHYSIOLOGICAL VISUALIZATION

Roy Tucker, Nigel Barlow and Liz Stuart

The Visualization Lab, School of Computing & Mathematics, University of Plymouth, Plymouth, UK
roy.tucker@plymouth.ac.uk(contact author), nigel.barlow@plymouth.ac.uk, liz.stuart@plymouth.ac.uk

Keywords: Concurrency, Parallel Computation, Multithreading, Massive datasets, Visualization

Abstract: *This paper reviews the history and current use of multi-threading in software development. Over the last decade, there has been a complete paradigm shift in computer hardware. Currently, increased computing power is supplied using an increasing number of core processors. This paradigm shift has led to the development of higher level programming constructs and frameworks in many popular languages. Therefore, it is clear that both the hardware and software of the future are going to be based heavily upon concurrency. Despite this, software developers are still resistant to embrace multi-threading. This resistance is understandable due to the complexity of coding concurrency. However, this paper proposes that Concurrency is no longer an option but a necessity. In conclusion, the paper presents a case study based on the visualization of large quantities of Neurophysiological data. This application was developed using the Java concurrency framework.*

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

1 Introduction

During the early days of Computing, there was a considerable difference between the operating speed of CPUs and the speed of connected peripherals. Back then, CPU time was expensive, thus it was highly inefficient to pause the execution of a program to enable the execution of a slower input/output peripheral.

The first attempt to address this problem was the LEO III (Lyons Electronic Office) developed in 1961. LEO III employed the first multiprogramming system [1] which enabled a batch of programs to be loaded into the CPU simultaneously thereby essentially queuing for CPU time. In this system, the first program would execute until it reached an instruction which required the use of a peripheral device. At this point, the context of the first program would be stored thereby enabling the next program to execute. Subsequently, the use of CPU time was maximised. The main limitation of this system was that it required multiple programs to maintain this level of CPU usage. Nowadays this would be recognised as a problem of granularity, the executing units were too large to maximise the CPU usage.

The limitations of multiprogramming became obvious as computer systems moved from batch processing to interactive use. Multiprogramming was not capable of delivering well designed systems. One of the key benchmarks for well-designed systems was the set of "Golden rules" defined by Shneiderman [2]. These rules were, and still are, widely adopted throughout the industry. They emphasise the importance of providing "informative feedback" to the user.

Initial attempts to provide interactive feedback to users involved the co-operation of software developers. This was known as co-operative multitasking. This relied on developers ensuring their applications yielded CPU time to other applications.

Initially, this approach was successful. The earlier versions of Windows (prior to Windows 95) and the Mac operating system (prior to MAC OS X) employed co-operative multitasking [3, 4]. Whilst co-operative multitasking was increasingly deployed as a multitasking solution, another option called pre-emptive multitasking evolved.

In contrast, pre-emptive multitasking paradigm provides slices of CPU time to each of the executing processes. Effectively, this enforces the sharing of the CPU time. This implicit guarantee of CPU time enables developers to provide user with well- designed systems capable of proving "informative feedback" quickly. In 1969, this approach was selected for use in the UNIX operating system. It is standard in UNIX and its derived operating systems [5].

By the mid 1990's, Microsoft Windows had adopted the pre-emptive multitasking system incorporating it into both Windows NT and Windows 95. Apple Inc. followed suit with the MAC OS 9.x, released in October 1999.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

2 The Pre-emptive Model

In the pre-emptive model, processes are separated into two categories:

- I/O Bound processes where the total computation time to complete the process is limited by the speed at which data can be requested and delivered [6]. Typically processes that make long read / write operations to disk would be I/O Bound and
- CPU Bound processes where the total computation time to complete the process is limited by the operating speed of the CPU. Typically a process that primarily 'crunches numbers' will be CPU bound.

This blocking mechanism enables the CPU time being consumed by these "waiting" processes to be re-allocated to processes in the CPU bound category. This continues until an interrupt signals to the I/O bound process that the blocked process can proceed.

As this pre-emptive model evolved, programmers began to develop applications as a collection of interacting (co-operating) processes. In turn, this raised the issue of how to efficiently share data between multiple processes. As originally conceived, a process ran in its own protected memory space isolated from other processes. However, this was not conducive to data sharing. The solution was that co-operating processes should share the same memory space. This approach would become known as multi-threading with multiple 'threads of execution' sharing a single processes memory space.

2.1 The paradigm shift from Moore's Law to Amdahl's law

Moore's Law states that the power of computer processing would double approximately every two years. Since its formulation in 1965, this law has provided a reliable guide to the growth of computing power. This predictable growth in computing power has been quietly exploited by software engineers worldwide. Until now, developers have enjoyed ever faster performance from their software simply by updating to newer hardware. However, it is proposed [7] that Moore's Law cannot be sustained as physical limitations for miniaturisation are encountered. Conversely, it is argued that advancing technologies will enable the law to survive far into the future [8].

Regardless of these opposing opinions, developers must consider how current and future technologies will deliver computing power. It is clear that hardware manufacturers have moved to the idea of delivering computer power through multi-core systems. Almost all forms of computer now employ multi core hardware as standard. This ranges from the mobile phone through to desktop computers of major research projects. Whilst considering highly compute intensive operations such as weather forecasting, computing power is now delivered by massively parallel super computers, grid computing and the opening of graphics processing cores for general non-graphical use (for example using Nvidia's CUDA) is becoming more widespread in research. Yet, it is still not enough.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

Consequently, another law is now dominating the current expansion of computer processing and throughput. This is Amdahl's law which describes the performance increases that can be achieved through the parallelisation of software. Essentially a program is divided into two portions, the parallelisable and the sequential components. Additional compute cores will improve the execution speed of the parallel component of the program but these will have no effect on the sequential component [9].

2.2 Recent developments in hardware

In the last decade computer power has expanded based on the development of multithreaded execution architectures as well as the delivery of additional power using multicore systems. However, this increased processing capability can only be realised when software developers change their programming styles to exploit this new paradigm. Fundamentally, the latest hardware advances are completely dependent on the understanding and adoption of these new techniques by software developers. David Stewart CEO of CriticalBlue and chairperson for the Multicore Programming Practices (MPP) working group comments on this situation stating that "There's capability in (multicore) platforms which is not being utilized or not being optimized by the software development community" [10].

Developers are often deterred due to the difficulty of using threads and locks to control access to shared memory. Goetz [11], a key developer of the Java languages Concurrency Framework, states "writing correct programs is hard; writing correct concurrent programs is harder." Indeed, this style of programming has "a well-deserved reputation for introducing bugs that are difficult to find and fix." [10]. Even Apple Inc. dismisses it stating that "the dominant model for concurrent programming - threads and locks - is too difficult to be worth the effort for most applications." [12]

Despite the difficulties Goetz notes that "threads are the easiest way to tap the computing power of multiprocessor systems". Furthermore, "as processor counts increase, exploiting concurrency effectively will only become more important" [11]. Therefore if the true power of multicore systems is to be realised, it is essential that the next generation of developers will have the tools and the training to address the difficulties of concurrent programming.

2.3 Recent developments in software

Over the last few years several mainstream programming languages have evolved to address the tools needed for concurrent programming. These tools provide concurrency frameworks that allow developers to work with abstract concepts rather than the lower level threads and locks.

This trend is likely to continue. Intel is currently experimenting with the Intel Array Building Blocks (Intel® ArBB) framework that will integrate with standard C++ applications without any compiler specific extensions [13]. This would remove a major compatibility hurdle to developing C++ parallel code.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

Table 1 shows a selection of the major coding languages that have released a concurrency framework over the last decade. Stack-less Python represents a significant fork of the language that has enjoyed commercial success and development support (via CCP Games Inc). The Java Concurrency Utilities merged a concurrency framework with cross platform support to produce a flexible and widely deployable solution. Microsoft's Task Parallel Library went a step further seeking the same goal for a whole range of languages supported by the .NET framework. While all of these were high level languages, Intel have addressed the lower level C++ language. Note that they too are now experimenting with a new more general solution than the existing Threading Building Blocks (TBB).

The development of frameworks to remove the complexity of task and thread management is essential to promote multi-threaded computation and ensure it is accessible to the software development community. This is very helpful. However, software engineers must also be able to design, debug, validate and optimise code.

Tools to support all these area of software development are at an early stage of their development. Areas such as debugging and validation are challenging. Patterson [14] states that multi-threaded code is well known to be notoriously difficult to debug and validate. The non-deterministic behaviour of concurrent code and the dangers of deadlock as well as race conditions are well understood. Despite these challenges, the tools to detect such errors during development are only now beginning to emerge.

For example, it was 2008 when Intel announced the development of a dedicated package of software engineering tools called the Intel® Parallel Studio suite [15]. The package was released to the development community in May 2009 [16]. With such tools only just beginning to emerge most developers remain uncertain how to validate and debug parallel code beyond repeated testing and code inspections. Issues of code optimisation, identification of parallelisable code and its long term maintenance are seldom considered by most developers.

3 Future Trends

Over the next decade, parallelism in software development will become more important both in research and in commerce applications. Indeed, the development company, CCP based in Iceland, is already crediting its commercial success to Stackless Python [17]. With the preference for multicore architectures firmly entrenched with manufacturers, the demand for software to exploit its power is unavoidable. However, this raises one of the primary issues, namely that of training software developers to exploit the hardware capability. Many of the current generation of programmers seldom consider parallel execution of code. This is understandable as it has mirrored the underlying hardware (subject to processor time slicing providing an illusion of parallel execution).

However, the next generation of developers must embrace the harder task of parallel code development as 'standard practice'. To achieve this training programs both in universities and industry must to be updated to emphasize parallel coding principles and to introduce the

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

software engineering tools that will support the design, implementation, validation and tuning of parallel programs.

3.1 Power and Environmental issues

Power has traditionally been seen as being in abundant supply, but this view is changing. The emergence of cloud computing has led to the formation of warehouse sized data centres that “are now consuming more energy than heavy manufacturing in the United States” [18]. Subsequently, this has led to political pressures (in light of carbon emission targets and taxes) to reduce energy consumption and waste.

Table 18: Concurrency Frameworks for major coding languages

Language	Frame-work	Re-lease Date	Source
Java	JSR 166: Concurrency Utilities	09/04	[19] Lea (2004)
.Net Framework (v4)	Task Parallel Library (available in all .Net languages)	04/10	[20] Microsoft (2010)
Python	Stack-less Python	01/00	[21] Tismer, (2000)
C++	Intel® Threading Building Blocks	08/06	[22] Reinders (2007)

One of the primary benefits of the multicore architecture is that such systems “feature more even power density and do not show dramatic temperature peaks” [23]; nevertheless “the power consumption of the basic multicore component is critical to its cost and operation” [18].

4 Case Study: Visualization of large Neurophysiological datasets

This case study of the VISA (Visualisation of Inter Spike Associations) software demonstrates that concurrent programming is already a necessity in current applications. This software is the main output of an established research VISA project at the Visualization Lab, University of Plymouth. The main aim of this research is to develop a new approach to the analysis of experimental data in neurophysiology based on the use of modern computer science techniques such as graphical engineering, visualization and virtual reality. This new approach will provide neuroscientists with an interactive environment within which to explore their data sets. The primary focus of this research is on the analysis of multi-dimensional spike train datasets.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

4.1 VISA Project Goals

Due to increasing size of data, the VISA software was completely redeveloped to fully exploit the Java concurrency framework. It was essential for the software to maintain throughput in order to achieve three main project goals:

- To enable the Neurophysiologists to process exponentially larger datasets than earlier versions in order to manage current and future demand.
- To develop the software in a language that offered cross platform compatibility to enable easy distribution
- To target a “typical researchers computer” with the option to execute with increased efficiency and speed on more powerful systems.
- To introduce a workflow based interface loosely based on “visual programming languages”. This workflow interface would enable users to control the ordering of data pre-processing operations. This would provide users with greater control over the types and ordering of pre-processing operations. Ultimately the goal is to enable users to create their own pre-processing code modules to fine-tune their workflows in the future

4.2 Spike train data

In general, a neuron accumulates electrical stimulus, from other neurons coupled to it, until some internal threshold is reached. Once its threshold is reached, the neuron initiates an action potential. When a neuron initiates action potentials over time, we say that the neuron is firing. Note that action potentials are more commonly referred to as spikes and a series of these spikes over time is known as a spike train. Spike train data is one of the main types of data collected during Neurophysiological experimentation.

Spike train datasets are records of the activity of a collection of neurons under investigation. It is well established that information is encoded in this data. In the VISA project, the spiking frequency and thus, inter-spike-intervals carry information. Therefore, research is focused on the analysis of multidimensional spike train data to uncover information about the synchronisation of spike trains and the connectivity of neurons.

4.3 Quantity of Data

VISA is currently in its third edition. The first two development cycles of the VISA project incorporated the development of a cross platform tool [24, 25].

When the project began, laboratories were typically recording from at most 64 electrodes simultaneously. This was deemed to be a very large amount of data and it was recorded using an 8 by 8 multi-electrode array. Currently, due to recent improvements in electrode hardware, Neurophysiologists are now able to routinely capture data using 4096 electrodes simultaneously. For example, the Plexon Array [26], can now record data for over 30 minutes at a sampling frequency of 7.702 kHz when matched with appropriate hardware.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

With typical trials lasting anywhere from a few milliseconds to tens of minutes, datasets sizes have vastly increased. For example: During an experiment when recordings are taken from all 4096 electrodes sampling data every millisecond, a 30 minute trial would result in a data file of 10.43 MB with approximately 1 million data points.

4.4 Introduction to VISA

The VISA³ interface is effectively a visual programming language which enables the user to create a workflow.

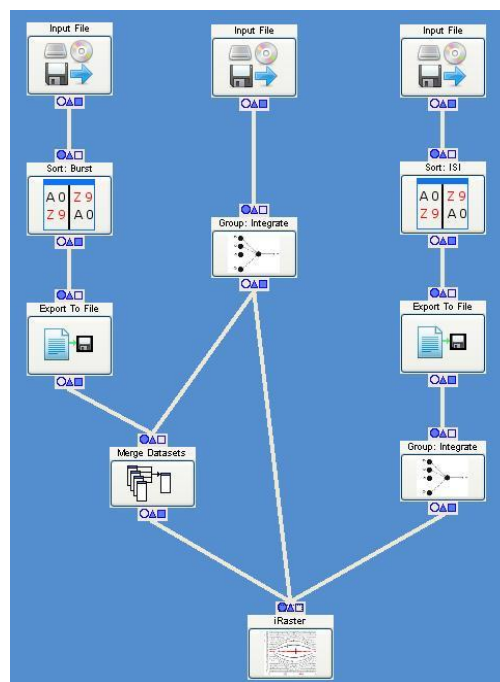


Figure 6: Directed Graph of a VISA³ Visual Program

A workflow is a set of processes joined together. It is understood that such a programming model is naturally parallelisable; processes are able to execute as soon as all inputs are available.

A typical VISA³ workflow is shown in Figure 6. This workflow shows 12 processes on the workflow interface. Note that these processes are selected by the user from a Toolbox and dragged on to the interface. These processes have parameters set by the user and they are connected together into the workflow as shown by dragging the mouse between process “ends”. This workflow is dealing with three input files, two of which subsequently sorted and then exported to an external file. Eventually, all this data is visualised using the iRaster visualization.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

The iRaster visualization enables the Neurophysiologists to investigate their data interactively using a traditionally-based raster plot representation. In addition to the functionality delivered by classical raster plot visualization, iRaster also provides the user with the following core functionality: data zooming methods, multiple views of data with view synchronization, and the contextual labelling of spike train identifiers and time points. iRaster also enables users to interactively and dynamically remove spike trains or time periods and to zoom in onto specific time periods to look at spike trains in greater detail. The software also provides an extensive list of spike train reordering functions. The majority of software tools currently available provide the usual static raster plots that are merely snapshots of the spike train data. In contrast, iRaster enables the user to interactively navigate through and directly manipulate spike train data, providing a dynamic experience of the data.

4.5 Concurrency in the VISA Interface

The key benefits of this interface are apparent. Scientists are not confronted with the need to learn a text based interface, the interface is designed to be intuitive.

The structure of the workflow, which may exploit parallel execution, is also apparent. If you refer to Figure 6, three sections of the workflow that could execute in parallel are obvious. Each process, such as the Import File process or the "Merge Datasets" process, represents a processing activity that may commence execution as soon as its inputs are available. The connections between various processes show the precise flow of data within the program.

Within the Java language, parallel execution is achieved through the concurrency framework. It operates as follows:

A 'pool' of threads each capable of executing a code module is created. Note this applies only to code modules that implement the concurrency frameworks 'Callable' interface. Pool size is dynamically determined based on the processors available on the executing machine but aims to optimise for CPU bound tasks. It is reasonable to assume that most tasks in VISA³ will fall into this category.

At the beginning of execution the application determines all paths, through the directed graph, that comply with the requirements of the information processing cycle. Specifically this ensures that some input is received (usually from a data file), some processing is performed and finally some output is generated (typically a data visualisation module is triggered).

Each processing node of the directed graph is implemented as a "binary latch" [11]. This latch acts a synchroniser which simulates a gate that can only be opened once. Until the conditions for opening the gate are met, all threads reaching the latch will be unable to proceed. The latch will be converted into the terminal state once these conditions are met. In the case of VISA³, the conversion requires all preceding operations to have completed and delivered a dataset to the waiting process. When the gate opens, all datasets from previous nodes are delivered to the next node and processing begins on that thread.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

The datasets flowing through the directed graph are required to implement the VISA³ interface `IProcessingResult`. Note that `IProcessingResult` is designed to wrap any data structure. Therefore, classes implementing this interface serve as the 'tokens' that drive the dataflow processing [27] and are placed onto the nodes input/output paths (triggering the 'latches' as required).

In addition to the development of a workflow interface, the Java Concurrency Framework was further leveraged to improve performance of the iRaster visualisation. As described in section 4.4, this visualisation provides an interactive raster plot of the spike trains.

The re-engineering of the visualisation was critical to maintain interactivity, due to the increase in dataset size. The original implementation typically worked with several hundred spike train recordings simultaneously whereas the new version works with thousands. Note that scientists are particularly keen to embrace this software as their traditional means of analysis do not scale.

4.5.1 Exploiting multiple-cores in VISA

Maintaining the responsiveness of an interactive application while filtering and processing this amount of data is challenging. Given this complexity, effective use of computing power in the rendering and display of the raster plot is essential. Recall that it has already been established that future increases in computer power will be delivered through multiple compute cores. The Java Concurrency Framework provides a natural means to access and manage this increased computing power in the future.

To exploit multiple cores, the VISA³ data model was redesigned as a set of thread safe classes that could be shared across multiple executing tasks in the Concurrency Framework. The iRaster visualization process was designed using the model, view, controller design pattern with the synchronisation of shared data occurring in the model. Subsequently, the rendering could easily be adapted for parallel execution.

This required a concurrent task that accepted a screen area and spike train to render within the area. Therefore, rendering the display focuses on the individual components of the data model that are to be shown as well as generating a collection of concurrently executable tasks. A latch is again employed to ensure the various tasks complete before the final display is presented. The raster chart is then cached with re-rendering occurring only as data is filtered into/out of the display. Interactive components, such as selection highlights, scale sliders, are sequentially rendered over the cached chart.

4.6 Future VISA3 Development plans

The workflow interface and the iRaster visualisation have clearly demonstrated that parallel execution of data pre-processing and visualisation activities can scale the application to 2000+ data recordings each with thousands of data points while maintaining responsiveness. There are now two primary tasks. The first task is the conversion of the iGrid visualisation

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

[24] to use the Java Concurrency Framework to enable the new larger datasets to be visualized in this way. This will enable the popular visualization iGrid to exploit multicore processors, thus enabling the visualization of new, larger datasets. This is challenging as iGrid is based on the production of numerous cross-correlograms. These calculations become so numerous that it is likely to be necessary to move to general-purpose computing on graphics processing units (GPGPU) model to ensure the user requirement for responsiveness are delivered.

The second task is to deploy the iRaster software more widely. The CARMEN Project is developing a 'workflow' system similar to the VISA interface [28] for processing neural recordings. In principle, this should be directly connectable to the iRaster visualisation. Whilst it requires conversion to a client-server model to support deployment on the CARMEN hardware, it will be offered as a downloadable client to view data stored in the CARMEN repository. The intention is to make this software freely available for all non-commercial use.

5 Conclusions

Multi-Threaded code originally arose from the need to manage long running tasks without the executing application becoming unresponsive to the user. However, over the last decade a new use of this technique has arisen. This is attributed to the fact that hardware manufacturers now deliver increased computing performance through multiple compute cores. Threads have been used as a natural way of writing applications that fully exploit delivery of computing power. Nevertheless, this application of threads falls outside their original purpose. Therefore, its use has been limited by a lack of software tools and developer training.

Now that hardware manufacturers have committed themselves to multi-core hardware systems, the software developer must acknowledge that to continue to benefit from ever faster hardware, the way in which they write programs must change.

Developers are beginning to accept this need to change and this is supported by the availability of development tools such as Intel® Parallel Studio suite and the addition of concurrency frameworks to major programming languages.

Despite the availability of these support tools, threading and concurrency continue to be seen as advanced concepts with training continues to lag behind the deployment of hardware capable of executing concurrent code.

In the next few years, this must change with the next generation of developers being trained to expect their code to execute in a concurrently on multi-core hardware. In many research fields multi-core hardware will be combined with clustering and grid technologies (cloud computing) to dramatically increase data processing throughput. In business, every employee can expect to be using multi-core devices as standard (from desktop PC to mobile phones and tablets). The developers of today graduate with a firm understanding of object orientated development; the developer of tomorrow will need to add concurrent programming

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

and the experience of a concurrency framework to these skills if they are to meet the needs of business tomorrow!

The value of these skills has been demonstrated in the case study of the VISA³ project. In VISA³, the number and length of neural spike train recordings that need to be processed by the software has been scaled up by a factor of ten. Finally the application of a workflow based interface to the VISA³ software demonstrates that visualising a concurrent application in this manner may offer a means to effectively design and debug concurrent software.

6 References

[1] ARIS, J., HERMON, P., LAND, F. & CAMINER, D. 1997. L.E.O.: The Incredible Story of the World's First Business Computer McGraw-Hill.

[2] SHNEIDERMAN, B. & PLAISANT, C. 1998. Designing the user interface: Strategies for effective human-computer interaction Addison Wesley.

[3] MICROSOFT. 1995. Windows 95 Architecture Components Windows TechNet [Online]. Available: <http://technet.microsoft.com/en-us/library/cc751120.aspx> [Accessed 14/03/2012].

[4] APPLE. 2001. Threading Architectures - Technical Note TN2028 [Online]. Apple Inc. Available: https://developer.apple.com/legacy/mac/library/#technotes/tn/tn2028.html#/apple_ref/doc/uid/DTS10003065 [Accessed 26/03/2012 2012].

[5] AIKAT, D., STEPNO, B., CHERNOFF, E., MANNING, M., ROBINSON, W. & HUGHES, T. 1995. The Digital Research Initiative - What is UNIX [Online]. Chapel Hill: University of North Carolina. Available: <http://www.ibiblio.org/team/intro/unix/what.html> [Accessed 05/03/2012 2012].

[6] CORPORATION, Intel. 2008. What does it mean to be I/O Bound. Intel Corporation.

[7] DUBASH, M. 2005. Moore's Law is dead, says Gordon Moore [Online]. Techworld. Available: <http://news.techworld.com/operating-systems/3477/moores-law-is-dead-says-gordon-moore/> [Accessed 22/09/2011].

[8] KRAUSS, L. M. & STARKMAN, G. D. 2004. Universal Limits on Computation [Online]. Available: <http://arxiv.org/pdf/astro-ph/0404510v2.pdf> [Accessed 13/03/2012 2012].

[9] HILL, M. D. & MARTY, M. R. 2008. Amdahl's Law in the Multicore Era. *Computer - IEEE Computer Society*, 33-38.

[10] MYSLEWSKI, R. 2009. The multicore future, and how to survive it - Avoiding the proprietary extensions trap [Online]. San Francisco. Available: http://www.theregister.co.uk/2009/08/25/multicore_developments/ [Accessed 05/03/2012 2012].

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

[11] GOETZ, B., PEIERLS, T., BLOCH, J., BOWBEER, J., HOLMES, D. & LEA, D. 2006. *Java Concurrency in Practice*, Pearson Education.

[12] APPLE. 2009. Grand Central Dispatch - A better way to do multicore [Online]. Apple Inc. Available:
http://www.ctestlabs.org/hughes_multicore/documents/GrandCentral_TB_brief_20090608.pdf [Accessed 26/03/2012 2012].

[13] GHULOUM, A., SHARP, A., CLEMONS, N., TOIT, S. D., MALLADI, R., GANGADHAR, M., MCCOOL, M. & PABST, H. 2010. A Flexible Parallel Programming Model for Multicore and Many-Core Architectures [Online]. Intel Software and Services Group. Available:
<http://drdobbs.com/cpp/227300084> [Accessed 14/03/2012 2012].

[14] PATTERSON, D. A. & HENNESSY, J. L. 2008. *Computer Organization and Design: The Hardware/Software Interface* Morgan Kaufmann.

[15] INTEL. 2008. News Fact Sheet [Online]. Intel Corporation. Available:
http://download.intel.com/pressroom/kits/events/idffall_2008/IDF_Day2_FactSheet.pdf [Accessed 13/03/2012 2012].

[16] INTEL. 2009. Intel PR Chip Shots [Online]. Intel Corporation. Available:
<http://www.intel.com/pressroom/chipshots/archive.htm#052609a> [Accessed 13/03/2012 2012].

[17] PETURSSON, H. V. 2011. Stackless Python Applications [Online]. Python Software Foundation. Available: <http://www.stackless.com/wiki/Applications> [Accessed 14/03/2012 2012].

[18] BLAKE, G., DRESLINSKI, R. G. & MUDGE, T. 2009. A Survey of Multicore Processors. *IEEE SIGNAL PROCESSING MAGAZINE*, 26-37.

[19] LEA, D. 2004. JSR 166: Concurrency Utilities [Online]. Java Community Process Program. Available: <http://jcp.org/en/jsr/detail?id=166> [Accessed 06/03/2012 2012].

[20] MICROSOFT. 2010. Parallel Programming in the .NET Framework [Online]. Microsoft Developer Network. Available: [http://msdn.microsoft.com/en-us/library/dd460693\(v=vs.110\).aspx](http://msdn.microsoft.com/en-us/library/dd460693(v=vs.110).aspx) [Accessed 06/03/2012 2012].

[21] TISMER, C. 2000. Stackless Python 1.0 + Continuations 0.6 [Online]. Available:
<http://mail.python.org/pipermail/python-dev/2000-January/001835.html> [Accessed 26/03/2012 2012].

[22] REINDERS, J. 2007. *Threading Building Blocks Outfitting C++ for Multi-core Processor Parallelism* O'Reilly Media.

[23] MONCHIERO, M., CANAL, R. & LEZ, A. G. 2008. Power/Performance/Thermal Design-Space Exploration for Multicore Architectures. *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, 19, 666-681.

Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.

[24] STUART, L., WALTER, M. & BORISYUK, R. 2005. The correlation grid: analysis of synchronous spiking in multi-dimensional spike train data and identification of feasible connection architectures. *Biosystems*, 79, 223-233.

[25] SOMERVILLE, J., STUART, L., SERNAGOR, E. & BORISYUK, R. 2011. iRaster: A novel information visualization tool to explore spatiotemporal patterns in multiple spike trains *Journal of Neuroscience Methods*, 194, 158-171.

[26] PLEXON 2006. MEA Workstation - System for recording and analyzing microelectrode arrays. In: INC, P. (ed.) Online. Dallas: Plexon Inc.

[27] DENNIS, J. & ROBINET, B. 1974. First version of a data flow procedure language. *Programming Symposium*. Springer Berlin / Heidelberg

[28] SMITH, L. S. 2010. CARMEN - Code Analysis, Repository & Modeling for E-Neuroscience [Online]. CARMEN Consortium. Available: <http://www.carmen.org.uk/> [Accessed 29/03/2011 2011].

A SCALING CROSS PLATFORM TOOL FOR THE ANALYSIS OF NEUROPHYSIOLOGICAL DATA

Roy Tucker, Saman Gunaratne, Nigel Barlow and Liz Stuart

The Visualization Lab, School of Computing & Mathematics, Plymouth University, Plymouth, UK
roy.tucker@plymouth.ac.uk(contact author), nigel.barlow@plymouth.ac.uk, liz.stuart@plymouth.ac.uk

Keywords: Concurrency, Parallel Computation, Multithreading, Massive datasets, Information Visualization, Visual analytics

Abstract: This paper describes the development of a cross platform cross correlation and clustering application for neural spike train recordings that will scale seamlessly from use on a researchers PC to a high performance computing cluster (HPC). Clusters of neurons are identified from the neural recordings using a hierarchical agglomerative clustering algorithm applied to the cross correlation data. Finally a cross correlation grid is used to visualise the neural clusters with the user navigating through the dataset using a dendrogram depicting the identified clusters. The cross correlation algorithm is an “embarrassingly parallel problem” that is scaled to cope with a large number of neurons through exploiting multiple compute cores both in the setting of a researchers PC or an HPC. This scaling is achieved using MPJ Express a Java implementation of the Message Passing Interface (MPI).

1. Introduction

The study of neurons as the functional component of the nervous system began in 1873 when Camillo Golgi developed a staining technique that made them visible to researchers using microscopes [1]. Combined with the work of Santiago Ramón y Cajal these scientists laid the foundation for the “neuronal doctrine”. At its core this doctrine asserts that the brain is composed of individual units that contain specialized features such as dendrites, a cell body, and an axon. Since this discovery neural science has developed to study how cognition, memory and the higher brain functions emerge from networks of interconnected neurons. Communication between neurons in a “neuronal network” is by electro-chemical means and it is the study of these electro chemical signals that form the basis of neural scientists’ investigations into the emergent properties of neuronal networks.

2. Neurophysiological Data

While cognition, memory and the higher brain functions are highly complex, the neuron itself is a relatively simple cell dedicated to the production and transmission of electrical signals. A neuron

accumulates electrical charge from other neurons that have connected to it. Once the accumulated charge reaches a threshold the neuron generates an action potential and discharges. When a neuron discharges its stored electrical energy it is said to ‘fire’ and modern recording hardware can record these electrical discharges’ over time. A single discharge event is usually referred to as a spike while a recording over time of these spikes is termed a spike train. Spike trains form the basic data for neural science research into both the connections between neurons and the data being transmitted within a neuronal network. Figure 7 shows a typical spike train recording for three neurons over 500ms (700ms – 200ms). Here the horizontal plot denotes the spiking of the neuron over time with a vertical line.

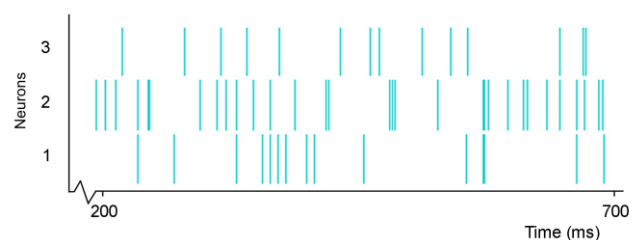


Figure 7: An example of a typical spike train recording for three neurons over a period of 500ms

It has been shown that, each neurons spiking event is essentially identical [2] and therefore the individual spikes do not seem to transmit or carry information. Rather the information is encoded in the timing and sequence of spikes. This has encouraged researchers to focus on the frequency (or inter spike intervals) and the synchrony between spike trains in their attempt to decode the information being transmitted. Research has demonstrated that synchrony between spike trains is of great importance for information processing in the brain [3].

2.1. Coupling of neurons

The synchronisation between any two neuron spike trains is dependent on the coupling that exists between the neurons within the neuronal network that they are part of. Researchers have identified two types of coupling:

- i. Direct coupling and
- ii. Indirect coupling.

Both types are illustrated in Figure 7.

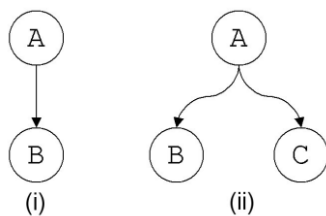


Figure 8: An example of (i) direct synaptic coupling and (ii) common input coupling

In case (i) neuron A delivers an electrochemical signal directly to neuron B while in case (ii) neurons B and C share a common input from neuron A. In each case the resulting spike train recordings will show a strong synchronisation (or correlation) between A and B / C while B and C will show a weaker correlation reflecting their shared input from A.

The study of the correlation between multiple spike trains offers the opportunity to map the underlying neuronal network structure and interpret the neural code being used to transmit

information, but faces a considerable computational challenge when its sheer scale is considered.

The human brain is estimated to contain approximately one hundred billion neurons (10^{11}) with each neuron on average maintaining 7000 connections to other neurons. Even with the most modern equipment we lack the ability to simultaneously record the spike trains for this many neurons as well as the computational tools to analyse the resulting data. Nevertheless the study of smaller neuronal network structures, in detail, is possible through the study of synchrony and correlations between spike trains.

Much data has been recorded in the form of multi-dimensional spike trains where the spiking behaviour of an assembly of neurons is simultaneously recorded while being subjected to a stimulus. A common first step in the analysis of this data is the preparation of a cross correlation analysis.

3. Cross Correlation

The process of cross correlation aims to provide a consistent mathematical basis to measure the degree of synchrony between multi-dimensional spike trains. Focused on comparing a pair of spike train recordings (and hence a pair of neurons) its output is a cross correlogram such as that shown in Figure 9.

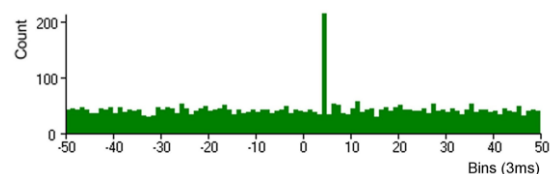


Figure 9: Example cross-correlogram for two connected neurons

Here we see a strong correlation in spike events at the +5ms position indicating that the two spike trains analysed show a highly synchronous spiking pattern separated by 5ms. Such a chart is generally referred to as a Pair-wise cross correlogram.

3.1. Pair-wise cross-correlogram

The creation of a cross correlogram is essentially a two-step procedure involving 'binning' the spike events of the two spike trains into time bins and then preparing a histogram of the number of spike events in each time bin.

The two selected spike trains are assigned the roles of reference spike train and target spike train and the binning parameters of bin size and window size are decided upon (for Figure 9 these where a bin size of 3ms and a window size of 100ms). The binning process involves positioning each spike event on the reference spike train at the centre of the time window and calculating the number of spikes of the target spike train that fall into each bin as illustrated in Figure 10.

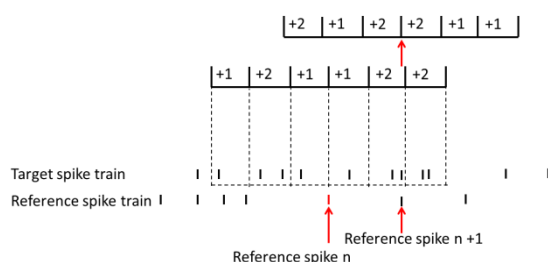


Figure 10: An example of a Cross Correlogram calculation using a six-bin window

The result for each bin is summed with the result from all other spike events on the reference spike train to generate the data for the histogram plot. If the spiking data exhibits a temporal correlation between the two spike trains a statistically significant peak will be seen on the histogram plot.

3.2. Peak significance

A histogram plot with a strong 'peak' as seen in Figure 9 provides a good indication of a temporal correlation between the two neurons spiking patterns. It is however possible to obtain false positives due to the high spiking frequency of a spike train. To address this issue the cross correlogram data is usually normalised using the Brillinger normalisation and confidence interval [4].

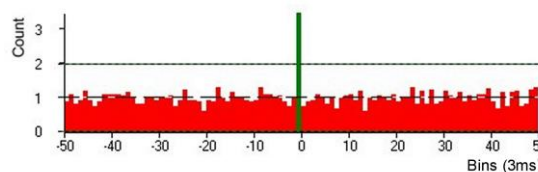


Figure 11: Example Brillinger normalised cross-correlogram with confidence interval

The Brillinger technique gives each bin a mean value of one allowing meaningful comparisons to be made between spike trains of different lengths. A confidence interval based on the data is calculated and any peak lower than this interval is considered to be purely random while peaks greater than this interval denote significant correlation.

Figure 11 shows a Brillinger normalised cross-correlogram with the mean spike bin size of one and the confidence interval shown. While a Brillinger normalised cross-correlogram provides a reliable indication of neuronal coupling it remains focused on the connectivity between two neurons while a functional neuronal network will include many more than two neurons. Inevitably the pairwise cross-correlogram is not a suitable visualisation for identifying connectivity within any meaningfully sized neuronal network.

3.3. The Cross-Correlation Grid (iGrid)

This deficiency was first addressed by Walter, Stuart and Borisyuk who created a compact visualisation known as iGrid [5] to provide a visual overview of a large set of cross-correlogram data.

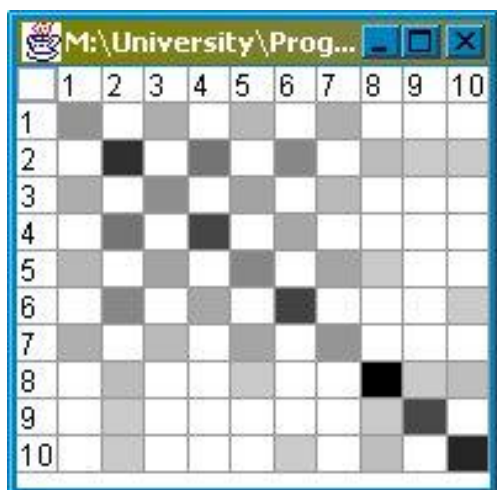


Figure 12: Example Correlation Grid showing only significant peaks (bin size 2ms, window size 100)

In this visualisation the significant peaks in the cross correlogram data (and their relative strength) are encoded using grey-scale squares with white indicating no significant peak and black the largest peak in the grid. In Figure 12 we see a cross-correlation grid for a network of 10 neurons representing some 55 individual cross-correlograms. The initial iGrid implementation was further improved upon by introducing a clustering algorithm to re-order spike trains within the grid. In Figure 13 we see the same cross-correlation grid presented in Figure 12 but this time re-ordered using the clustering algorithm.

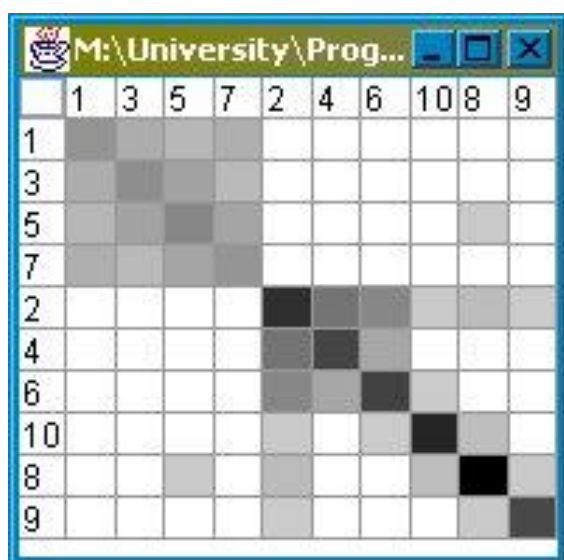


Figure 13: Example Correlation Grid, showing only significant peaks and clustered (bin size 2ms, window size 100)

The two clusters of connected neurons in the test dataset are now clearly visible with neurons 1, 3, 5 & 7 forming the first cluster while 2, 4 & 6 form the second. The remaining neurons 8, 9 & 10 show no strong correlation with either cluster. This does indeed mirror the interconnections in the test neuronal network which is shown in Figure 14.

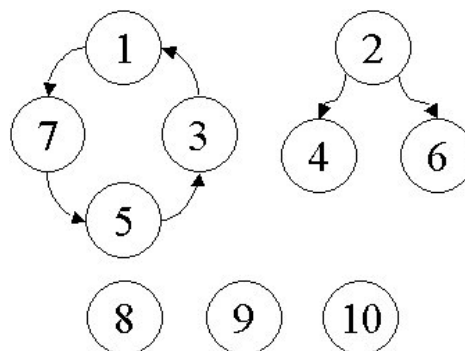


Figure 14: Neuron assembly for test data set

Clearly then the cross correlation technique provides a useful means to extract the structure of a neuronal network from simultaneous recordings of that networks spiking behaviour. In turn knowledge of the network structure permits the researcher to identify which neurons are generating/receiving signals in response to the applied stimulus.

3.4. Drawbacks to the cross correlation grid

While useful the cross correlation grid does present a number of problems to the researcher. Most noticeably it still suffers from scalability issues when applied to larger datasets in two ways:

- The computational workload in terms of cross correlation generation and normalisation rapidly grows as neuronal network size increases and
- The available screen space to physically display a cross correlation grid is limited and beyond approx. 100 neurons the visual benefit of the grid is lost to the human eye.

In examining these points we note that the number of cross-correlograms that must be generated and

normalised for a neuronal network of size n is given by the equation:

$$\frac{n^2 + n}{2}$$

This function is graphically shown in Figure 15, for neuronal networks of, up to, 2000 neurons. A neuronal network of 2000 neurons would require the production and normalisation of 2,001,000 pairwise cross correlations. Current recording hardware for multi-dimensional spike trains can already routinely record neuronal networks in the 1500 – 2000 neuron range [6], using Multi-Electrode Arrays (MEA's). It is anticipated that a rapid growth in recording sizes will occur over the next several years so even a 2000 neuron neuronal network recording will soon be seen as relatively small. In addition to the growth in recording size in terms of physical neurons recorded the time over which the recordings are made has also been growing with durations of 30+ minutes or more becoming increasingly common [7]. This has resulted in a marked growth in the number of recorded spike events (or data points) to be processed at the cross-correlations binning stage further leading to an increased computational workload in creating the pairwise cross-correlation data.

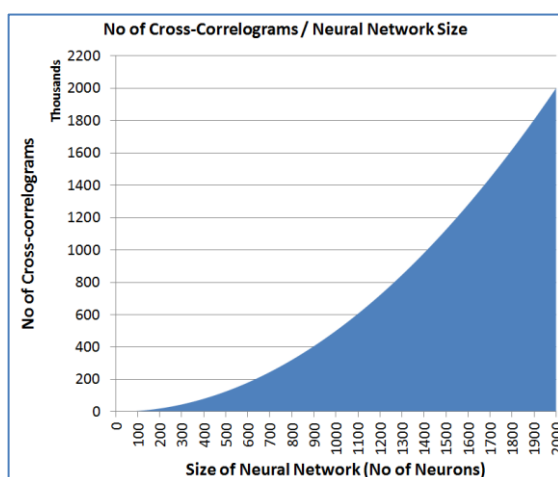


Figure 15: Number of required cross correlation calculations for neuronal networks up to 2000 neurons in size.

Finally the growth in recorded neurons has also made it impossible to visually display the

correlation grid in any way that allows meaningful conclusions to be extracted. This can be simply demonstrated by contrasting the screen pixels available with the required grid size. The modern monitor runs with a typical resolution of 1920 x 1080 pixels providing a visible grid of 2073600 in which the display is rendered. However the visualisation of a 2000 neuron cross-correlation grid will require at a minimum 4,000,000 pixels even if one pixel represented a single grid entry. Thus, in practice, it is impossible to visually extract a useful overview of a dataset from a display where a pixel must represent 1.929 data points (4000000 / 2073600).

A final drawback of these scaling issues in the iGrid visualisation is the loss of interactivity with the user. The original iGrid implementation allowed users to interact with the grid, viewing individual cross-correlograms and re-ordering the grid as needed. The growth in dataset size often slows these interactions to the point where the benefit to the user is lost.

Giving these issues of increasing computational load and expanding dataset sizes we must conclude that iGrid as originally implemented, while useful, does not scale well for application to the datasets produced by modern recording hardware.

4. Parallelism

4.1. Scaling iGrid's computational elements

Addressing the scaling issue of the iGrid analysis tool requires:

1. Managing the computational burden of cross-correlation analysis so that the goal of an interactive tool that can be used on a 'typical' standard PC.
2. Providing a new visualisation element to provide a meaningful 'overview' of the data set.

The need to address point one provided an opportunity to employ parallel computation to create a data model that would support the

Tucker, R., Gunaratne, S., Barlow, N. & Stuart, L. (2014) 'A Scaling Cross Platform Tool for the Analysis of Neurophysiological Data'. *International Journal of Computer Application*, 3 (4). pp 41-56.

interactive iGrid tool. Generally computation architectures are categorised using Flynn's Taxonomy [8] into the following four architectures:

1. Single instruction, single data stream (SISD)
2. Single instruction, multiple data streams (SIMD)
3. Multiple instructions, single data stream (MISD)
4. Multiple instructions, multiple data streams (MIMD)

The production of cross-correlogram data is a "Single instruction, multiple data streams" task where the computational algorithm is a constant (Single instruction) but the data to be operated on repeats (Multiple data streams, with each neuron pair within the dataset being a separate data stream). Therefore we can categorise the cross-correlation computation as a Single instruction, Multiple data stream (or SIMD) task. It has been noted that such tasks are naturally parallelisable but this was not exploited in the original iGrid implementation. Given the smaller data set sizes used for the original implementation this did not hinder the application however as the application scales this becomes a significant bottleneck.

Modern computing hardware delivers performance through parallel computation (Multi-Core computers and the increasing use of GPU's for parallel computation) [9]. Additionally the delivery of computing power 'in the cloud' is now emerging as a new approach to creating computationally intensive applications that can draw on additional computation power as needed. This growth in the options for parallel processing of large data sets presented a variety of options that were considered for scaling the computational portion of the iGrid software. In summary the considered options were:

1. A dedicated processing cluster available to researchers over the web (in the form of the Carmen Project's Virtual Laboratory [10]).
2. A cloud based solution such as Amazon Web Services that caters for big data analysis.

3. A distributed computing / cluster computing solution such as that offered by the Apache Hadoop project.
4. An implementation of the algorithm utilising the MPI – Message Passing Interface allowing execution on any computing cluster that supports this standard.
5. An implementation that exploits the compute capabilities of GPU's now being opened up by frameworks such as CUDA and OpenCL which are particularly suited to this type of problem.

Our previous work on the VISA project had created a Java based data model suitable for representing neural spike trains and the developer undertook to extend this model to support the cross-correlation and clustering required for the iGrid software. Initial development was aimed at the Carmen Project's Virtual Laboratory which provided a service based framework that readily supported the existing Java implementation. The service based architecture wraps a Java executable that performs the cross-correlation and clustering analysis. The ultimate approach adopted was to generate from the wrapped executable a file a cross-correlation data file and a file of clustering records derived from the cross correlations. This approach separated the naturally parallelizable component of the analysis (the cross-correlation) from the serial clustering process. Parallel processing within the Java executable was achieved through the Java Concurrency Framework and the use of multiple threads to exploit multiple cores. To cluster the cross correlation results we selected a hierarchical (or connectivity based) agglomerative clustering solution using complete linkage clustering. The metric used to determine the distance between clusters was the Euclidean distance between the most significant peak of the cross correlogram after Brillinger normalisation. Consideration was given to single linkage and average linkage clustering and these remain as options in the deployed CARMEN service but complete linkage provided the most accurate identification of the underlying neural structure in our simulated

Tucker, R., Gunaratne, S., Barlow, N. & Stuart, L. (2014) 'A Scaling Cross Platform Tool for the Analysis of Neurophysiological Data'. *International Journal of Computer Application*, 3 (4). pp 41-56.

networks. Both the cross correlation and clustering files were encoded as JSON strings allowing portability of results between processing systems.

While the CARMEN service was successfully implemented it did not realise the speed benefits hoped for several reasons. Most noticeably the availability of solely 8 compute cores coupled with the overhead of the service wrapper meant a speed increase could not be realised. Despite this it served as a useful proof of concept that indicated a more flexible system was required. Furthermore encoding the processing results into JSON strings provided a portable cross platform format that allowed re-coding of the actual analysis implementation without affecting integration with the iGrid software.

The next phase of development involved identifying a technology that would:

1. Preserve the benefits of the cross platform nature of Java while
2. Operate in a timely manner in a cluster computing environment.

In this case the developers looked to the widely supported Message Passing Interface standard (MPI) to provide portability between many high performance computing clusters [11]. MPJ Express was selected as the implementation of MPI as this provided support for operation not only in the 'Cluster Configuration' typical of many high performance computing clusters but also offered a multi-core mode that allowed multiple threads to replace the cluster nodes. The use of this mode greatly improves the portability of the MPI solution to the cross-correlation generation and neural cluster identification as it permits the same MPI implementation of these functions to migrate seamlessly from a user's laptop / desktop to a high performance cluster computer [12]. This did involve re-coding the cross correlation and clustering algorithms' to support the MPI standard but fortunately the use of JSON strings for encoding results limited the overall changes needed to these two areas.

The MPI implementation of the cross-correlation and clustering algorithms was initially developed on a hyper-threaded 4 core desktop system (providing access to 8 concurrent threads) running an Intel Core i7-2600 CPU @ 3.40 GHz. It was subsequently migrated to the University of Plymouth High Performance Computer (Fotcluster 1) [13]. This cluster is a 104 core distributed memory cluster composed of a ViglenHX425Hi HPC combined head and storage node plus 12 compute nodes employing a mixture of Dual Intel Xeon Quad Core E5620 @ 2.4Ghz processors and Dual Intel Xeon Quad Core E5310 @ 1.6Ghz processors. In each case the MPJ Framework provided the executing environment for the MPI implementations and the speed & performance of the computationally intensive cross-correlation and clustering algorithms can be seen in the results section.

4.2. Implementing the Algorithm

Similar to all programs being parallelized, these cross correlation and clustering programs will contain both a serial and a parallel element. Initially, it is necessary to decompose the problem into steps that (i) can and (ii) cannot be executed in parallel [14]. Both the cross correlation and clustering parts of this algorithm are open to parallelization. However, the cross correlation portion of the algorithm is a naturally parallelisable problem (also known as an embarrassingly parallel problem), whilst the clustering process is considerably more complex to implement in parallel. Fortunately the hierarchical agglomerative clustering algorithm, which uses complete linkage, for a 2000+ neuronal network is not a computationally expensive process (although as data size increases, it may become so) [15]. The majority of the computational workload is in the calculation and normalisation of the 2,000,000+ pairwise cross correlations that must be derived for a 2,000+ neuronal network. The efficient parallelisation of this component of the algorithm is all about the effective load balancing across the compute cores. Ideally, the algorithm should ensure all cores are working at maximum

Tucker, R., Gunaratne, S., Barlow, N. & Stuart, L. (2014) 'A Scaling Cross Platform Tool for the Analysis of Neurophysiological Data'. *International Journal of Computer Application*, 3 (4). pp 41-56.

capacity [16]. The natural division of the work is into single pairwise cross correlation calculations between two spike train recordings.

Theoretically, it is possible to apply a finer grained division of the work. This would involve parallelisation of the binning operation. RT: for each reference spike trains spiking event could be used. Whilst theoretically possible, this option is not feasible due to the large number of threads that would be generated and the need to co-ordinate the movement of data between them. Note that each reference spike train could include 1000+ individual spiking events. Since, this algorithm needs to execute efficiently whether it is executing on a laptop, a desktop PC or a HPC, the finer grained parallelisation could not be adopted. Thus, the implemented solution is based on a coarser grained, individual cross correlation approach that can be effectively deployed across all hardware.

It should be noted that in the future, the use of laptop and desktop GPUs for general computation may lead to a re-evaluation of the coarse grained task parallelism approach. Thus, this algorithm would be adapted to exploit the thousands of potential GPU cores made available as such hardware and corresponding software becomes available [17].

Note that the cross correlation of neuron 1 (reference neuron) with neuron 10 (target neuron) is identical to the cross correlation of neuron 10 with neuron 1. Thus, the grid is symmetrical.

Within the algorithm, each row of the cross correlation grid is processed individually one after the other. After processing a row of the correlation grid, the data regarding that reference neuron is no longer required. For example: after processing row 1 of the grid, all data regarding to the reference neuron 1 is no longer required in the current task. Exploiting this property of the correlation grid enables the algorithm to release memory as it executes. During testing, this memory management was essential when executing on a laptop/desktop.

In the solution a queue of pairwise cross correlation tasks are assigned to each available compute core. As each row of the grid is processed these queues receive a decreasing number cross correlation tasks. Note that queued tasks are delivered to the compute cores via the message passing interface as the cores complete previous work. Thus, when all queues are eventually emptied, the cross correlation calculations for that row are complete and the data of the reference neuron is released from memory.

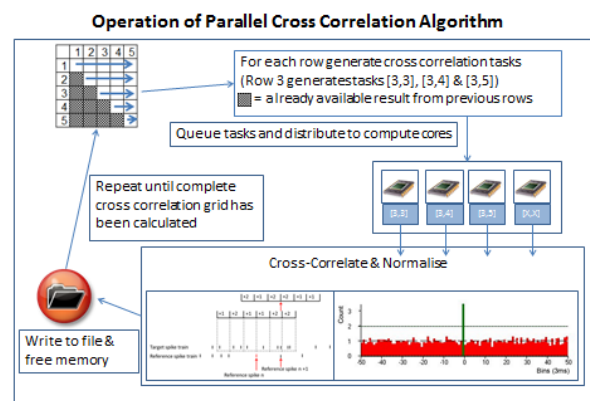


Figure 16: Overview of the parallel cross correlation algorithm

The concurrent operation itself consists of deriving the cross correlation bins seen in Figure 10 and the Brillinger normalisation process shown in Figure 11. The resultant cross correlations are stored in memory until a complete row of the grid has been processed. When the row is complete, the bin data for this row is saved to a file. Thus, the corresponding memory is released so that the next row of the grid can be processed. Note that the cross correlation data is encoded as JSON strings in order to maintain system portability. Figure 16 depicts the complete process for a grid size of 5.

Once this compute intensive part of the algorithm is complete, multi-core processing queues are shut down and serial execution resumes for the clustering calculation. The clustering calculation uses an agglomerative, hierarchal clustering technique with complete linkage. This technique (see Figure 11) is a bottom-up clustering approach that clusters items based on the height of the

highest peak in the cross correlogram [15]. The user has the option to set the value of a 'confidence level' which is used to define whether any cross correlation peak is significant. Peaks are deemed to be significant when they are higher than this confidence level [4].

This is sufficient to define the relationship between any two 'clusters' when both clusters include a single spike train. In cases where the cluster contains more than one spike train, the cross correlation of each spike train in the first cluster and every spike train in the second is examined. Once again the most significant peak found denotes the strength of the correlation between the two clusters (this is known as 'Complete Linkage' as all cross correlations are considered when selecting the most significant peak). Each repetition of the clustering algorithm merges the two most closely correlated clusters in the dataset. This results in a new cluster at each iteration until no more significant peaks exist between the remaining clusters. Thus, clustering is complete and the generated clusters are saved to a file. In conjunction with the original spike train data file, the cross correlation and clustering files are input to the iGrid visualisation.

4.3. Scaling the visualisation elements of iGrid

The correlated and subsequently clustered dataset is input to the iGrid visualisation tool. Once loaded, the tool produces the default initial cross correlation grid plot, as well as, a dendrogram plot of the data clustering (refer to Figure 17). Figure 17).

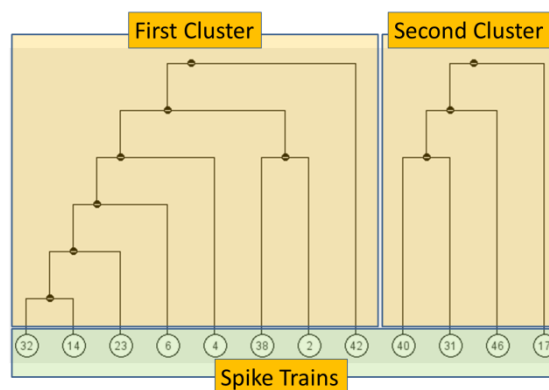


Figure 17: Dendrogram plot of the first two spike train clusters in a 50 neuron network

The dendrogram is used as navigation tool due to the vast quantity of data. The dendrogram is used to select spike trains for inclusion in the iGrid plot.

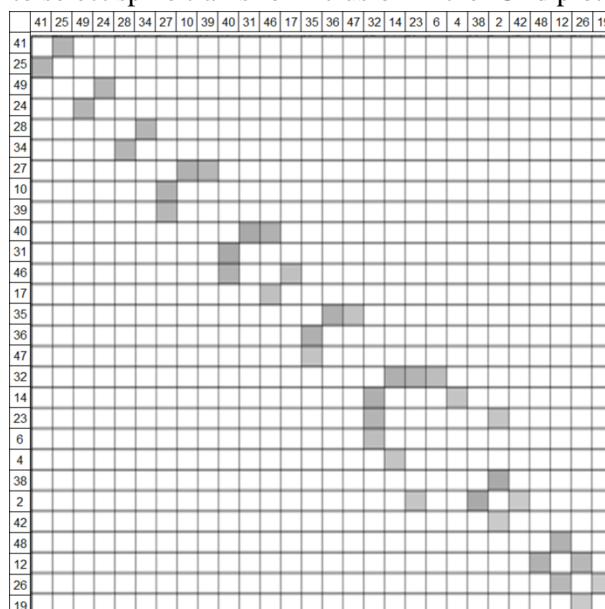


Figure 18: iGrid showing significant cross-correlation peaks in a 50 neuron test dataset

This means that the iGrid provides a detailed view of specific related spike trains. Figure 18 shows an iGrid plot generated from our test data of 50 neurons recorded over 30 minutes. The spike train plot has been ordered to group highly correlated spike train patterns together, thus revealing the neuron clusters.

Error! Reference source not found. shows the associated dendrogram for the same 50 neuron dataset. Note that the dendrogram identified 28 spike trains that show significant correlation in

their spiking pattern. These spike trains are connected in eight clusters. The remaining 22 spike trains were uncorrelated (and have been filtered out to improve clarity).

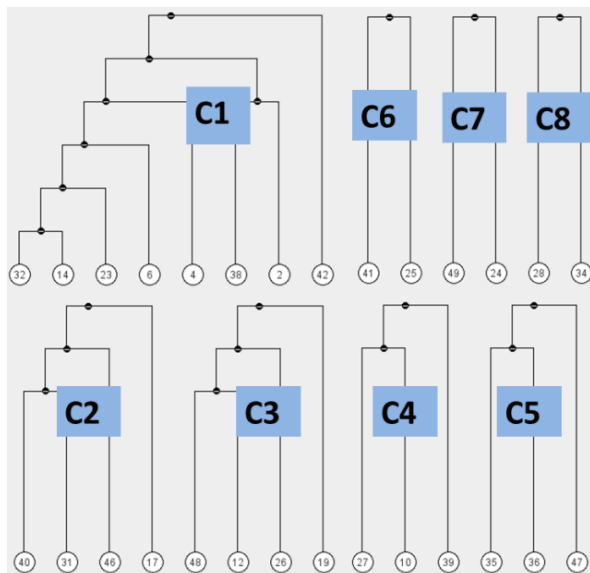


Figure 19: Dendrogram for 50 neuron test dataset

When this derived (guessed) topology is compared to the original (previously unseen) neuronal network, it was correct. The same eight clusters of correlated data are also apparent in the iGrid representation in Figure 18. It is essential that this system can scale-up. The results of testing its ability to scale-up follows. To measure its success testing will examine its performance under various data loads and across various hardware platforms.

5. Results – using MPJ Express and a High Performance Cluster

Performance of the scalable cross correlation algorithm has been tested on both a standard PC and on the High Performance Computing Cluster (HPC) at Plymouth University.

5.1 Test Datasets

Testing was performed using datasets which ranged in size from 50 neurons to 2000 neurons. The size, and resulting number of cross correlations performed, of each dataset is shown in Table 19.

Neuron Count	Cross Correlations
50	1275
100	5050
150	11325
200	20100
300	45150
400	80200
500	125250
1000	500500
2000	2001000

Table 19: Dataset sizes and number of cross correlations performed

It should be noted that the HPC is capable of processing larger datasets. The test data was limited only by the largest comparable dataset on a standard PC. Note that the 2000 neuron network took 8.26 hours to complete while consuming almost all available memory resources.

5.2 Standard PC Results

Testing of the cross correlation algorithm on a standard PC used the following hardware:

Item	Description
Processor	Intel Core i7-2600 CPU @ 3.40GHz
Installed Memory	16.0GB
Operating System	Windows 7 Enterprise Service Pack 1

The performance of the application was measured in terms of the number of completed cross correlations per second of operation. The results are given in Table 20.

Neuron Count	Cross Correlation tasks	Total Time (Sec)	Per second rate
50	1275	21.84	58.37
100	5050	83.05	60.81
150	11325	172.93	65.49
200	20100	305.34	65.83
300	45150	680.24	66.37
400	80200	1201.63	66.74
500	125250	1872.54	66.89
1000	500500	7445.04	67.23
2000	2001000	29744.07	67.27

Tucker, R., Gunaratne, S., Barlow, N. & Stuart, L. (2014) 'A Scaling Cross Platform Tool for the Analysis of Neurophysiological Data'. *International Journal of Computer Application*, 3 (4). pp 41-56.

Table 20: Total Cross Correlation time and per sec rates for a standard PC

5.3 HPC Results

The High Performance Computer cluster (HPC) at Plymouth University [13] was used to demonstrate the scalability of the software. For the purpose of this test 32 of the 104 compute cores were made available. This represents a '4x' increase in the number of available compute cores compared to the standard PC. Performance was measured by the number of cross correlations completed per second. Results are detailed in Table 21.

Neuron Count	Cross Correlation tasks	Total Time	Per second rate
50	1275	10.51	121.37
100	5050	29.04	173.89
150	11325	55.78	203.04
200	20100	92.85	216.47
300	45150	196.48	229.80
400	80200	339.09	236.51
500	125250	525.23	238.47
1000	500500	2031.16	246.41
2000	2001000	8039.12	248.91

Table 21: Total Cross Correlation time and per sec rates for HPC

5.4. Performance of scaling the algorithm

As expected the increased availability of compute cores in the HPC results in a marked increase in performance. Recall, that the same algorithm is executing in both situations without any re-coding. This demonstrates that the platform agnostic algorithm executes in both environments, Furthermore, it clearly shows that it also scales as additional resources are made available.

Performance is measured in terms of the number of cross correlations performed per second. Table 22 shows the result for a standard PC and the High Performance Cluster:

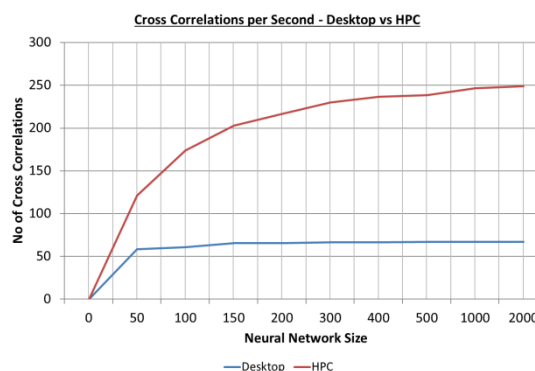


Table 22: Cross correlations per a second Desktop / HPC

As expected the performance of the algorithm flat lines based on the resources made available. The desktop system averaged 65 cross correlations per second with networks of 50 to 2000 neurons. Its performance remained close to this average for all networks of 150+ neurons.

As expected, the HPC makes greater resources available in terms of compute cores and this leads to a higher performance. The HPC averaged 212.76 cross correlations per second with networks of between 50 and 2000 neurons. However, there was a greater degree of variation in the averages.

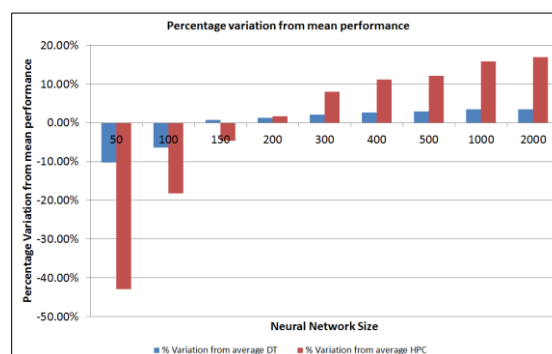


Table 23: Performance variation from mean calculation rate by neuronal network size

Note that the HPC performed better with larger datasets up until its performance flat lined. Smaller datasets executed on the HPC suffered from significantly poorer performance, in terms of the variation from the mean.

This is attributed to poor resource utilisation of the HPC's compute cores. The HPC reached its average performance when the size of the datasets

Tucker, R., Gunaratne, S., Barlow, N. & Stuart, L. (2014) 'A Scaling Cross Platform Tool for the Analysis of Neurophysiological Data'. *International Journal of Computer Application*, 3 (4). pp 41-56.

exceeded 200. The HPC continued to show a marked performance increase when the size of the datasets exceeded 1000. The percentage deviation from the mean performance for both the standard PC and the HPC are shown in Table 23.

6. Summary of Results

This paper has demonstrated this algorithm used to reveal the architecture of neuronal networks will scale seamlessly from standard PCs up to large HPCs. The application is supported by the hardware agnostic Java language and implements its solution using MPJ Express. MPJ Express is a widely used Java implementation of the Message Passing Interface standard. This enables parallel computation of the cross correlations to substantially improve both the performance and size of datasets that can be analysed. The computational performance of this application scales as additional compute cores become available. This enables the application to increase performance over time, as future delivery of computing power will be through the deployment of larger multi core processing systems (Amdahl's law) as well as scaling to exploit the power of current and future HPC environments.

7. Future work

This work has focused on providing an iGrid implementation in which the computationally intensive portions of the grid generation are performed using more modern techniques (parallel rather than serial code). The implementation has been re-engineered into a more flexible and reusable form, namely an MPJ Express implementation that scales seamlessly from the standard PC to a HPC. Note that this does not exhaust the potential for improving the performance of the iGrid analysis of simultaneously recorded spike trains.

It is anticipated that further performance increases can be realised by utilising the expanding field of General-purpose computing on graphics processing units (GPGPU). Exploiting this area would enable an increased dataset size to be

analysed by the iGrid application before the use of HPC technology becomes necessary. Accordingly it may be possible to realise further performance increases (in terms of speed and dataset size) through the application of CUDA and OPEN CL. These could be used to solve the computationally intensive cross correlation algorithm without needing to involve technologies beyond those common to a standard PC.

The relatively fixed rate of calculations at 60-68 cross correlations per second on a standard PC also indicates that further optimisation of the MPI implementation may be possible as significant time is spent in communication. In this case a potentially more efficient distribution of data across compute cores could be further investigated. The existing implementation divides data into individual spike trains and delivers those spike trains to the compute cores. The drawback of this approach is that the considerable re-transmission of data. The alternative strategy would be to divide data into time slices with compute cores being responsible for calculating portions of the cross correlation. Subsequently, cores would sum up the resulting cross correlation bins for each time slice. This may result in more efficient transmission of data to cores at the cost of additional work to produce the final result. Whether such an approach will improve performance is under investigation.

8. References

- [1] NICHOLLS, J. G., MARTIN, A. R., WALLACE, B. G. & FUCHS, P. A. 2001. *From Neuron to Brain: A Cellular and Molecular Approach to the Function of the Nervous System*, Sinauer Associates.
- [2] ROBINSON D. editor 1998. *Neurobiology*. Springer, The Open University. ISBN: 3-540-63546-7
- [3] BORISYUK, R. M. & BORISYUK, G. N. 1997. Information coding on the basis of synchronization of neuronal activity. *Biosystems*, 40, 3-10.
- [4] BRILLINGER, D. R. 1979. Confidence intervals for the crosscovariance function. *Selecta Statistica Canadiana*, 5, 1-16.

Tucker, R., Gunaratne, S., Barlow, N. & Stuart, L. (2014) 'A Scaling Cross Platform Tool for the Analysis of Neurophysiological Data'. *International Journal of Computer Application*, 3 (4). pp 41-56.

- [5] WALTER, M., STUART, L. & BORISYUK, R. 2003. A Compact Visualisation for Neurophysiological Data. *Seventh International Conference on Information Visualization (IV'03)*.
- [6] PLEXON 2006. MEA Workstation - System for recording and analyzing microelectrode arrays. In: INC, P. (ed.) Online. Dallas: Plexon Inc.
- [7] BUZSAKI, G. 2004. Large-scale recording of neuronal ensembles. *Nature Neuroscience*, 7, 446-451.
- [8] FLYNN, M. J. 1972. Some Computer Organizations and Their Effectiveness. *Computers, IEEE Transactions on*, C-21, 948-960.
- [9] TUCKER, R., BARLOW, N. & STUART, L. 2012. The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2, 352-358.
- [10] GIBSON, F., AUSTIN, J., INGRAM, C., FLETCHER, M., JACKSON, T., JESSOP, M., KNOWLES, A., LIANG, B., LORD, P., PITSILIS, G., PERIORELLIS, P., SIMONOTTO, J., WATSON, P. & SMITH, L. 2008. The CARMEN Virtual Laboratory: Web-Based Paradigms for Collaboration in Neurophysiology *6th Int. Meeting on Substrate-Integrated Microelectrodes*.
- [11] Tennessee, U. o. (Sept 2009) *MPI: A Message-Passing Interface Standard Version 2.2*. Message Passing Interface Forum.
<http://www.mcs.anl.gov/research/projects/mpi/mpi-standard/mpi-report-2.0/mpi2-report.htm> (Accessed: 01/10/2013).
- [12] SHAFI, A. & JAMEEL, M. (2006) *MPJ Express Project*. <http://mpj-express.org/index.html> (Accessed: 01/10/2013).
- [13] University of Plymouth. (2013) *High Performance Computing - fotcluster1*.
<http://www.plymouth.ac.uk/pages/view.asp?page=33935> (Accessed: 29/10/2013).
- [14] Grama, A., Karypis, G., Kumar, V. & Gupta, A. (2003) *Introduction to Parallel Computing*. Harlow: Pearson Education Ltd.
- [15] Everitt, B. S., Landau, S. & Leese, M. (2001) *Cluster Analysis*. London: Hodder Headline Group.
- [16] Breshears, C. (2009) *The Art of Concurrency - A Thread Monkey's Guide to Writing Parallel Applications*. O'Reilly Media.
- [17] Kirk, D. (2010) *Programming Massively Parallel Processors: A Hands-on Approach (Applications of GPU Computing Series)*. Burlington: Elsevier Inc.

References

- Ackerman, W. B. (1982) 'Data Flow Languages'. *Computer*, 15 (2). pp 15-25.
- Adrian, E. D. & Zotterman, Y. (1926) 'The impulses produced by sensory nerve endings: Part II: The response of a single end organ'. *The Journal of Physiology*, (61). pp 151-171.
- Aikat, D., Stepno, B., Chernoff, E., Manning, M., Robinson, W. & Hughes, T. (1995) *The Digital Research Initiative - What is UNIX*. vol. 2012. Chapel Hill: University of North Carolina.
- Apache (2014) *What Is Apache Hadoop? The Apache Software Foundation*. <http://hadoop.apache.org/> (Accessed: 14/04/2014).
- Apple (2001) *Threading Architectures - Technical Note TN2028*. vol. 2012. Apple Inc.
- Apple (2009) *Grand Central Dispatch - A better way to do multicore*. vol. 2012. Apple Inc.
- ArchonMagnus (2015) *The Scientific Method as an Ongoing Process.svg*. Wikimedia Commons. https://en.wikipedia.org/wiki/Scientific_method#cite_note-Garland2015-1 (Accessed: 15/03/2016).
- Aris, J., Hermon, P., Land, F. & Caminer, D. (1997) *L.E.O.: The Incredible Story of the World's First Business Computer*. McGraw-Hill.
- Backus, J. (1977) 'Can Programming Be Liberated from the von Neumann Style? A Functional Style and Its Algebra of Programs', *ACM Annual Conference*. Seattle Communications of the ACM, pp. 613-641.
- Bayer, R. (1972) 'Symmetric binary B-Trees: Data structure and maintenance algorithms'. *Acta Informatica*, 1 (4). pp 290-306.
- Bertin, J. (1983) *Semiology of Graphics: Diagrams, Networks, Maps*. ESRI Press.
- Blanshard, B. (2016) 'Rationalism', in *Encyclopædia Britannica Online*. Encyclopædia Britannica Inc., Available at: <http://www.britannica.com/topic/rationalism>. (Accessed: 15/03/2016)
- Boehm, B. (1986) 'A Spiral Model of Software Development and Enhancement'. *ACM SIGSOFT Software Engineering Notes*, 11 (4). pp 14-24.
- Boehm, B. (2000) 'Spiral Development: Experience, Principles, and Refinements'. [in *SPECIAL REPORT CMU/SEI-2000-SR-008*. Pittsburgh, USA: Carnegie Mellon University Software Engineering Institute. 49. Available at: <http://www.sei.cmu.edu/reports/00sr008.pdf> (Accessed: Boehm, B.
- Bondy, J. A. & Murty, U. S. R. (1976) *Graph Theory With Applications*. Elsevier Science Ltd/North-Holland.
- Borisyyuk, R. (2002) *Neural Network Simulator*. [Computer Program]. The Visualisation Lab - University of Plymouth. Available at: http://www.tech.plymouth.ac.uk/infovis/LAB_Downloads.htm (Accessed: 11/01/2010)
- Borisyyuk, R. (2008) *Network Creator*. [Computer Program]. The Visualisation Lab - University of Plymouth. Available at: http://www.tech.plymouth.ac.uk/infovis/LAB_Downloads.htm (Accessed: 11/01/2010)
- Borland, D. & Taylor II, R. M. (2007) 'Rainbow color map (still) considered harmful.'. *IEEE Computer Graphics and Applications*, 27 pp 14-17.
- Brent, J. (1998) *Charles Sanders Peirce, Revised and Enlarged Edition: A Life*. Indiana University Press.

- Brillinger, D. R. (1979) 'Confidence intervals for the crosscovariance function'. *Selecta Statistica Canadiana*, 5 pp 1-16.
- Brock, J. F. (2001) 'THE OLDEST CADASTRAL PLAN EVER FOUND : The Catalhoyuk Town Plan of 6200 B.C', *42nd Australian Surveyors Congress*. Brisbane, Queensland, Australia, pp. 25-28.
- Brown, E. N., Kass, R. E. & Mitra, P. P. (2004) 'Multiple neural spike train data analysis: state-of-the-art and future challenges'. *Nature neuroscience*, 7 pp 456-461.
- Buzsaki, G. (2004) 'Large-scale recording of neuronal ensembles'. *Nature Neuroscience*, 7 (5). pp 446-451.
- Card, S. K., Mackinlay, J. & Schneiderman, B. (1999) *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann.
- Carpenter, B. (2007) *The HP Java Project*. Pervasive Technology Labs at Indiana University.
<http://www.hpjava.org/mpiJava.html>
(Accessed: 25/11/2014).
- Ceruzzi, P. E. (2003) *A History of Modern Computing*. MIT Press.
- Clark, B. D., Goldberg, E. M. & Rudy, B. (2009) 'Electrogenic Tuning of the Axon Initial Segment'. *Neuroscientist*, 15
- Connors, B. W. & Long, M. A. (2004) 'ELECTRICAL SYNAPSES IN THE MAMMALIAN BRAIN'. *Annual Review. Neuroscience*, 27 pp 393-418.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2009) *Introduction to Algorithms*. 3 edn. The MIT Press.
- Craft, B. & Cairns, P. (2005) 'Beyond Guidelines: What can we learn from the Visual Information Seeking Mantra?', *9th International Conference on Information Visualisation*. London Institute of Electrical and Electronics Engineers (IEEE).
- Creemers, R., Deaves, P., Feather, C., Lloyd, P. & Willsher, M. (2014) *The London Tube Map* Archive.
<http://www.clarksbury.com/cdl/maps.html>
(Accessed: 24/03/2014).
- Dale, H. (1935) 'Pharmacology and Nerve-endings (Walter Ernest Dixon Memorial Lecture)', *Proceedings of the Royal Society of Medicine*. London PubMed Central, pp. 319-332.
- Davis, A. L. & Keller, R. M. (1982) 'Data flow program graphs.'. *IEEE Compute*, 15 (2). pp 26-41.
- Dayan, P. & Abbott, L. F. (2005) *Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems*. MIT Press.
- Dennis, J. & Robinet, B. (1974) 'First version of a data flow procedure language Programming Symposium'. Springer Berlin / Heidelberg, pp 362-376.
- Deregowski, J. B. (1968) 'Picture recognition in subjects from a relatively pictureless environment.'. *African Social Research*, 5 pp 356-364.
- Dubash, M. (2005) *Moore's Law is dead, says Gordon Moore*. Techworld. Techworld.
- Duignan, B. (2016) 'Empiricism', in *Encyclopædia Britannica Online*. Encyclopædia Britannica Inc. Available at: <http://www.britannica.com/topic/empiricism>.
(Accessed: 15/03/2016)
- Einevoll, G. T., Franke, F., Hagen, E., Pouzat, C. & Harris, K. D. (2012) 'Towards reliable spike-train recordings from thousands of neurons with multielectrodes'. *Current Opinion in Neurobiology*, 22 pp 11-17.

- Feldman, R. P. & Goodrich, J. T. (1999) 'The Edwin Smith Surgical Papyrus'. *Childs Nervous System*, 6-7 pp 281-284.
- Finger, S. (2001) *Origins of neuroscience: a history of explorations into brain function*. USA: Oxford University Press.
- Floridi, L. & Sanders, J. W. (2004) *Levellism and the Method of Abstraction*. 43 pp. Available at: http://www.cs.ox.ac.uk/activities/ieg/research/reports/ieg_rr221104.pdf (Accessed: 11/04/2014).
- Flynn, L. J. (2004) *Intel Halts Development of 2 New Microprocessors*. The New York Times. <http://www.nytimes.com/2004/05/08/business/08chip.html?ex=1399348800&en=98cc44ca97b1a562&ei=5007> (Accessed: 11/04/2014).
- Flynn, M. J. (1972) 'Some Computer Organizations and Their Effectiveness'. *Computers, IEEE Transactions on*, C-21 (9). pp 948-960.
- Foster, I. (1995) *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley.
- Friendly, M. & Denis, D. J. (2001) *Milestones in the History of Thematic Cartography, Statistical Graphics and Data Visualization*. <http://www.datavis.ca/milestones/> (Accessed: 05/03/2014).
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*. 1 edn. Addison-Wesley Professional.
- Garnier, R. & Taylor, J. (2009) *Discrete Mathematics: Proofs, Structures and Applications*. 3rd edn. CRC Press.
- Georgopoulos, A. P., Schwartz, A. B. & Ketiner, R. E. (1986) 'Neuronal Population Coding of Movement Direction'. *Science*, 233 pp 1416-1419.
- Gerstner, W. & Kistler, W. M. (2002) *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. London: Cambridge University Press.
- Gerstner, W., Kreiter, A. K., Markram, H. & Herz, A. V. M. (1997) 'Neural codes: Firing rates and beyond', *Proceedings of the National Academy of Sciences of the United States of America*. National Academy of Sciences of the United States of America.
- Gibson, F., Austin, J., Ingram, C., Fletcher, M., Jackson, T., Jessop, M., Knowles, A., Liang, B., Lord, P., Pitsilis, G., Periorellis, P., Simonotto, J., Watson, P. & Smith, L. (2008) 'The CARMEN Virtual Laboratory: Web-Based Paradigms for Collaboration in Neurophysiology'. *6th Int. Meeting on Substrate-Integrated Microelectrodes*,
- Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D. & Lea, D. (2006) *Java Concurrency in Practice*. Pearson Education. 8th
- Gordon, G. & McMahon, E. (1989) 'A greedoid polynomial which distinguishes rooted arborescences'. *Proceedings of the American Mathematical Society*, 107 (2). pp 287-298.
- Graphical Programming*. (2013) National Instruments Corporation. <http://www.ni.com/gettingstarted/labviewbasics/dataflow.htm> (Accessed: 23/04/2014).
- Harrell, J. A. & Brown, V. M. (1992) 'The oldest surviving topographical map from ancient Egypt (Turin Papyri 1879, 1899 and 1969)'. *Journal of the American Research Center in Egypt*, 20 pp 81-105.
- Harris, R. L. (2000) *Information Graphics: A Comprehensive Illustrated Reference*. 1 edn. Oxford University Press.

- Hibbard, T. N. (1962) 'Some combinatorial properties of certain trees with applications to searching and sorting.'. *Journal of the ACM*, 9 (1). pp 13-28.
- Higginbotham, C. W. & Morelli, R. (1991) 'A System for Teaching Concurrent Programming', *SIGCSE '91 Proceedings of the twenty-second SIGCSE technical symposium on Computer science education*. New York Association for Computing Machinery (ACM), pp. 209-316.
- Hill, M. D. & Marty, M. R. (2008) 'Amdahl's Law in the Multicore Era'. *Computer - IEEE Computer Society*, pp 33-38.
- Hochberg, J. E. & Brooks, V. (1962) 'Pictorial recognition as an unlearned ability'. *American Journal of Psychology*, 75 pp 624-628.
- Hondius, J. & Purchas, S. (1607) *Designatio Orbis Christiani [World Map]*. Atlas Minor London.
- Hutchins, E. (1995) *Cognition in the Wild*. University Press Group Limited.
- Huxley, A. F. & Stämpfli, R. (1949) 'Evidence for saltatory conduction in peripheral myelinated nerve fibres'. *Journal of Physiology*, 108 (3). pp 315-339.
- Illingworth, V. (1997) *Oxford Dictionary of Computing*. Oxford Paperback Reference. 4 edn. Oxford University Press.
- Jarosz, Q. (2009) *File:Neuron Hand-tuned.svg*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Neuron_Hand-tuned.svg (Accessed: 10/03/2016).
- Johnson, R. B. & Christensen, L. B. (2010) *Educational research: Quantitative, qualitative, and mixed approaches*. 4th Edition edn. SAGE Publications.
- Johnston, W. M., Hanna, J. R. P. & Millar, R. J. (2004) 'Advances in Dataflow Programming Languages'. *ACM Computing Surveys*, 36 (1). pp 1-34.
- Keller, R. M. & Yen, J. (1981) 'A graphical approach to software development using function graphs', *IEEE Compton*. pp. 151-161.
- Khan, G. (1974) 'The semantics of a simple language for parallel programming.', *International Federation for Information Processing (IFPT) Congress*. Amsterdam, pp. 471-475.
- Khronos (2014) *OpenCL - The open standard for parallel programming of heterogeneous systems*. Khronos Group. <http://www.khronos.org/opencv/> (Accessed: 14/04/2014).
- Kirk, D. B. & Hwu, W.-m. W. (2012) *Programming Massively Parallel Processors: A Hands-on Approach*. 2 edn. Elsevier Science.
- Knuth, D. E. (1998) *Art of Computer Programming: Sorting and Searching*. 2 edn. vol. 3. Addison-Wesley Professional.
- Konda, M. (2011) *What's New in Java 7*. O'Reilly Media.
- Kosinski, P. R. (1978) 'A straightforward denotational semantics for non-determinate data flow programs', *5th ACM Symposium on Principles of Programming Languages 1978*. ACM Press.
- Laney, D. (2001) '3D Data Management: Controlling Data Volume, Velocity, and Variety.'. *Application Delivery Strategies*, 949
- Lea, D. (2004) *JSR 166: Concurrency Utilities*. vol. 2012. Java Community Process Program.

- Lewis, B. & Berg, D. J. (1995) *Threads Primer: A Guide to Multithreaded Programming*. Prentice Hall PTR.
- Light & Bartlein (2004) 'The End of the Rainbow? Color Schemes for Improved Data Graphics.'. *EOS, TRANSACTIONS, AMERICAN GEOPHYSICAL UNION*, 85 (40). pp 385-391.
- Loktofeit, C. (2013) *EVE Online Surpasses 500,000 Subscribers Worldwide*. CCP Games. <http://community.eveonline.com/news/news-channels/press-releases/eve-online-surpasses-500-000-subscribers-worldwide/> (Accessed: 11/04/2014).
- Macionis, J. J. & Gerber, L. M. (2010) *Sociology, Seventh Canadian Edition with MySocLab*. 7th Edition edn. Pearson Education Canada.
- Madadhain J., Fisher D., Smyth P., White S. & B., B. Y. (2005) 'Analysis and visualization of network data using JUNG'. *Journal of Statistical Software*, 10 (2). pp 1-35.
- Maps & Cartographic Information*. (2014) *University of Washington Map Collection*, University of Washington Libraries. <http://guides.lib.washington.edu/content.php?pid=123049&sid=1103146> (Accessed: 05/03/2014).
- Marey, E. J. (1878) *La methode graphique dans les sciences experimentales*. Paris:
- Martin, R. C. (1996) Granularity. *The C++ Report*, 8, (10) 57-62.
- Maunsell, J. H. & Van Essen, D. C. (1983) 'Functional properties of neurons in middle temporal visual area of the macaque monkey. I. Selectivity for stimulus direction, speed, and orientation'. *Journal of Neurophysiology*, 49 pp 1127-1147.
- Micheva KD, Busse B, Weiler NC, O'Rourke N & SJ., S. (2010) 'Single-synapse analysis of a diverse synapse population: proteomic imaging methods and markers.'. *Neuron*, 68 (4). pp 639 - 653.
- Microsoft (1995) *Windows 95 Architecture Components*. Windows TechNet. Microsoft Corporation.
- Microsoft (2010) *Parallel Programming in the .NET Framework*. vol. 2012. Microsoft Developer Network.
- Microsoft (2012) *VPL Introduction*. Microsoft Developer Network. <http://msdn.microsoft.com/en-us/library/bb483088.aspx> (Accessed: 22/04/2014).
- Millet, D. (2002) 'The origins of EEG'. *International Society for the History of the Neurosciences 7th Annual Meeting*. Los Angeles, California, USA: International Society for the History of the Neurosciences.
- Mills, P. (2013) *High Performance Computing - fotcluster1*. Plymouth University. <https://www.plymouth.ac.uk/your-university/about-us/university-structure/faculties/science-engineering/hpc/what-is-a-cluster> (Accessed: 29/10/2013).
- Moler, C. (1986) 'Matrix computation on distributed memory multiprocessors', *First conference on Hypercube Multiprocessors*. Knoxville, Tennessee Society for Industrial and Applied Mathematics.
- Moore, G. E. (1965) 'Cramming more components onto integrated circuits'. *Electronics*, 38 (8).
- Moreland, K. (2009a) 'Diverging Color Maps for Scientific Visualization'. *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II*. Las Vegas, Nevada: Springer-Verlag.

- Moreland, K. (2009b) 'Diverging Color Maps for Scientific Visualization (Expanded)'. *ISVC '09 Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II*. Las Vegas: Springer-Verlag, pp 92 - 103.
- Myslewski, R. (2009) *The multicore future, and how to survive it - Avoiding the proprietary extensions trap*. vol. 2012. San Francisco:
- National Oceanic and Atmospheric Administration, N. O. S. (2015) *Raster Navigational Charts: NOAA RNCs*. US Department of Commerce. <http://www.nauticalcharts.noaa.gov/mcd/Raster/> (Accessed: 17/07/2015).
- Neumann, J. V. (1945) *First Draft of a Report on the EDVAC*. University of Pennsylvania. Available at: <https://sites.google.com/site/michaeldgodfrey/vonneumann/vnedvac.pdf?attredirects=0&d=1> (Accessed: 08/04/2014).
- Nightingale, F. (1858) *Notes on Matters affecting the Health, Efficiency and Hospital Administration of the British Army*.
- Nightingale, F. (1859) *A Contribution to the sanitary History of the British Army during the late War with Russia*. London:
- Normann, R. A. (1993) *Utah array*. Wikimedia Commons.
- NVIDIA (2014a) *What is GPGPU Computing?* NVIDIA. <http://www.nvidia.com/object/what-is-gpu-computing.html> (Accessed: 14/04/2014).
- NVIDIA (2014b) *GPU Cloud Computing Service Providers*. <http://www.nvidia.com/object/gpu-cloud-computing-services.html> (Accessed: 14/04/2014).
- NVIDIA (2014c) *CUDA Parallel Computing Platform*. NVIDIA.
- http://www.nvidia.com/object/cuda_home_new.html (Accessed: 14/04/2014).
- Olshausen, B. A. & Field, D. J. (1996) 'Emergence of simple-cell receptive field properties by learning a sparse code for natural images'. *Nature*, 381 pp 607 - 609.
- Open MPI: Open Source High Performance Computing*. (2014) The Open MPI Team, Indiana University. <http://www.open-mpi.org/> (Accessed: 26/11/2014).
- Oracle (2004) *Concurrency Utilities Overview*. Oracle Corporation. <http://docs.oracle.com/javase/1.5.0/docs/guide/concurrency/overview.html> (Accessed: 11/04/2014).
- Oracle (2014) *Concurrency Utilities Enhancements in Java SE 7*. Oracle Corporation. <http://docs.oracle.com/javase/7/docs/technotes/guides/concurrency/changes7.html> (Accessed: 11/04/2014).
- Oxford English Dictionary*. (2014) Oxford University Press. <http://www.oxforddictionaries.com/definition/english/semiotics> (Accessed: 10/03/2014).
- Pearson, D., Hanna, E. & Martinez, K. (1990) 'Computer-generated cartoons'. in Barlow, H., Blakemore, C. and Weston, S.M. (eds.) *Images and understanding*. Cambridge: Cambridge University Press, 33 pp 46-60.
- Peirce, C. S. (1868) 'On a New List of Categories'. *Proceedings of the American Academy of Arts and Sciences*, 7 pp 287 - 298.
- Petursson, H. V. (2011) *Stackless Python Applications*. vol. 2012. Python Software Foundation.
- Pike, A. W. G., Hoffmann, D. L., García-Diez, M., Pettitt, P. B., Alcolea, J., De Balbín, R., González-Sainz, C., de las Heras, C.,

- Lasheras, J. A., Montes, R. & Zilhão, J. (2012) 'U-Series Dating of Paleolithic Art in 11 Caves in Spain'. *Science*, 336 (6087). pp 1409-1413.
- Potter, S. M. (2010) *File:MEAinHand.jpg*. Wikimedia Commons. <https://en.wikipedia.org/wiki/File:MEAinHand.jpg> (Accessed: 10/03/2016).
- Rappenglück, M. A. (1999) 'Palaeolithic Timekeepers Looking At The Golden Gate Of The Ecliptic; The Lunar Cycle And The Pleiades In The Cave Of La-TETe-Du-Lion (Ardèche, France) – 21,000 BP'. *Earth, Moon, and Planets*, 85-86 (0). pp 391-404.
- Redwood, J. (1976) *Reason, Ridicule and Religion: The Age of En'lightenment in England, 1660-1750*. Thames & Hudson Ltd.
- Reinders, J. (2007) *Threading Building Blocks Outfitting C++ for Multi-core Processor Parallelism*. O'Reilly Media.
- Remenyi, D. & Money, A. (2012) *Research Supervision for Supervisors and their Students*. Academic Publishing International. 2nd Edition
- Robinson, A. H. (1967) *The Thematic Maps of Charles Joseph Minard*. Imago Mundi. Imago Mundi, Ltd.
- Rocca, J. (2003) *Galen on the Brain: Anatomical Knowledge and Physiological Speculation in the Second Century Ad (Studies in Ancient Medicine)*. Boston: Brill Academic Pub.
- Roo, M. D., Klauser, P., Mendez, P., Poglia, L. & Muller, D. (2008) 'Activity-Dependent PSD Formation and Stabilization of Newly Formed Spines in Hippocampal Slice Cultures'. *Cerebral Cortex*, 18 (1). pp 151-161.
- Rothman, S. S. (2002) *Lessons from the living cell: the culture of science and the limits of reductionism*. New York: McGraw-Hill.
- Rutkowski, C. (1982) An introduction to the Human Applications Standard Computer Interface Part 1: Theory and Principles. *Byte Magazine*, 7, (10) 291-310.
- SAS (2014) *Big Data. What it is and why it matters*. SAS Institute Inc. http://www.sas.com/en_us/insights/big-data/what-is-big-data.html (Accessed: 14/04/2014).
- Sernagor, E. (2016) *Newcastle University Institute of Neuroscience*. Newcastle University. <http://www.ncl.ac.uk/ion/staff/profile/evelynesernagor.html#publications> (Accessed: 03/03/2016).
- Shafi, A. & Jameel, M. (2006) *MPJ Express Project*. <http://mpi-express.org/index.html> (Accessed: 01/10/2013).
- Shepherd, G. M. (1991) *Foundations of the neuron doctrine*. History of Neuroscience. Oxford: Oxford University Press.
- Shinar, D., Dewar, R. E., Summala, H. & Zakowska, L. (2003) 'Traffic sign symbol comprehension: a cross-cultural study'. *Ergonomics*, 46 (15). pp 1549-1565.
- Shneiderman, B. (1996) 'The eyes have it: A task by data type taxonomy for information visualizations'. *Proceedings of the IEEE Symposium on Visual Languages*, pp 336-343.
- Shneiderman, B. & Plaisant, C. (1998) *Designing the user interface: Strategies for effective human-computer interaction*. Addison Wesley. 4
- Smith, E. E. & Kosslyn, S. M. (2006) *Cognitive Psychology: Mind and Brain*. Pearson.
- Snijders, C., Matzat, U. & Reips, U.-D. (2012) "Big Data": Big Gaps of Knowledge in the

- Field of Internet Science'. *International Journal of Internet Science*, 7 (1). pp 1-5.
- Snow, J. (1855) 'On the Mode of Communication of Cholera'. London: John Churchill.
- Somerville, J. (2011) *The Exploration of Neurophysiological Spike Train Data using Visual Analytics*. University of Plymouth.
- Somerville, J., Stuart, L., Sernagor, E. & Borisyuk, R. (2011) 'iRaster: A novel information visualization tool to explore spatiotemporal patterns in multiple spike trains'. *Journal of Neuroscience Methods*, 194 (1). pp 158-171.
- Stevenson, A. & Waite, M. (2011) *Concise Oxford English dictionary*. 12th ed. / edited by Angus Stevenson, Maurice Waite. edn. Oxford: Oxford University Press.
- Stuart, L., Walter, M. & Borisyuk, R. (2003) 'Analysis of multi-dimensional Spike Trains using VISA'. *5th Neural Coding Workshop*. Aulla, Italy: September 2003 Proceedings of 5th Neural Coding Workshop. (Accessed: 14/04/2011).
- Stuart, L., Walter, M. & Borisyuk, R. (2005) 'The correlation grid: analysis of synchronous spiking in multi-dimensional spike train data and identification of feasible connection architectures'. *Biosystems*, 79 (1-3). pp 223-233.
- Sutherland, W. R. (1966) *The on-line graphical specification of computer procedures*. Massachusetts Institute of Technology.
- Swartza, B. E. & Goldensohn, E. S. (1998) 'Timeline of the history of EEG and associated fields'. *Electroencephalography and clinical Neurophysiology*, 106 pp 173–176.
- 'synapse' (2014), in *Encyclopædia Britannica*. Encyclopædia Britannica, Inc. Available at: <http://www.britannica.com/EBchecked/topic/578220/synapse#ref284813>. (Accessed: 27/03/2014)
- Taketani, M. & Baudry, M. (2006) *Advances in Network Electrophysiology Using Multi-Electrode Arrays*. Springer.
- Tasaki, I. (1939) 'THE ELECTRO-SALTATORY TRANSMISSION OF THE NERVE IMPULSE AND THE EFFECT OF NARCOSIS UPON THE NERVE FIBER'. *American Journal of Physiology*, 127 (2). pp 211-227.
- Tendler, J. M., Dodson, J. S., Fields, J. S. J., Le, H. & Sinharoy, B. (2002) 'POWER4 System Microarchitecture'. *IBM Journal of Research & Development*, 46 (1). pp 5-25.
- The Merriam-Webster dictionary*. (2004) 11th ed. edn. Springfield, Mass.: Merriam-Webster.
- Theunissen, F. & Miller, J. (1995) 'Temporal encoding in nervous systems: a rigorous definition.'. *Journal of Computational Neural Science*, 2 (2). pp 149-162.
- Tismer, C. (2000) *Stackless Python 1.0 + Continuations 0.6*. vol. 2012.
- Tucker, R., Barlow, N. & Stuart, L. (2012) 'The Background and Importance of Exploiting Multiple Cores: A Case Study in Neurophysiological Visualization'. *Proceedings of the 2012 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2 pp 352-358.
- Tufte, E. R. (2001) *The Visual Display of Quantitative Information*. 2 edn. USA: Graphics Press.
- Vignais, P. M. & Vignais, P. V. (2010) *Discovering Life, Manufacturing Life: How the experimental method shaped life sciences*. Berlin: Springer.

Walter, M. A. (2004) *Visualization Techniques for the Analysis of Neurophysiological Data*. University of Plymouth.

Ward, J. S. & Barker, A. (2013) 'Undefined By Data: A Survey of Big Data Definitions'. *arXiv* [Online]. Available at: <http://arxiv.org/abs/1309.5821> (Accessed: 14/04/2014).

Ware, C. (2012) *Information Visualization: Perception for Design (3rd Revised edition)*. Morgan Kaufmann.

Weeks, M. (2010) *Writing Code for Carmen Services*.

Wikipedia (2014) *File:Synapse diagram picture.jpg*. Wikimedia Commons. https://commons.wikimedia.org/wiki/File:Synapse_diagram_picture.jpg (Accessed: 10/03/2016).

Zotterman, Y. (1939) 'TOUCH, PAIN AND TICKLING: AN ELECTRO-PHYSIOLOGICAL INVESTIGATION ON CUTANEOUS SENSORY NERVES'. *The Journal of Physiology*, 95 (1). pp 1-28.