ROUTING SIMULATION OF BRAIN NETWORK TOPOLOGY

by

Jason V. Ma

Bachelor of Science, Ryerson University, 2003

A thesis

presented to Ryerson University

in partial fulfillment of the
requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Ontario, Canada, 2014

©Jason V. Ma 2014

AUTHOR'S DECLARATION FOR ELECTRONIC SUBMISSION OF A THESIS

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

I understand that my thesis may be made electronically available to the public.

Routing Simulation of Brain Network Topology

Master of Science 2014

Jason V. Ma

Computer Science

Ryerson University

Abstract

The incessant search to understand human cognitive functions has led to the hypothesis that the brain works similar to a packet switched network such as the Internet [28]. In this thesis, I have developed a top-down simulator of brain-like networks which uses probability routing to route data and a distance vector routing algorithm [21] to propagate feedback to varying depths. I investigate the impact of the feedback depth on routing table metrics. The results indicate that important performance metrics are affected by the feedback depth of the routing algorithm but also, to a large extent, by the topological features of such networks [17, 44]. The results indicate feedback depths from 25 to 30 fill the routing table most efficiently in terms of routing table fill percentage, routing table fill time and packet rejection ratio. There is also a strong correlation between the macaque monkey brain and sparse topologies.

Acknowledgements

I would like to thank my research supervisor Dr. Vojislav Misic for his mentor ship and support over the past two years. Without his help, encouragement and expertise in networks and software engineering, this thesis would not be possible. I would like to thank my thesis defense committee members Dr. Jelena Misic and Dr. Isaac Woungang for their time and expertise to evaluate my research.

Dedication

This thesis is dedicated to my parents who have always encouraged me to become a lifelong learner.

Contents

<i>Declaration</i>	iii
<i>Abstract</i>	v
<i>Acknowledgements</i>	vii
<i>Dedication</i>	ix
<i>List of Tables</i>	xiii
<i>List of Figures</i>	xv
1 Introduction	1
1.1 Background	1
1.2 Network Topology Concepts	3
1.3 Routing Protocols	7
1.4 Problem Statement	8
1.5 Proposed Solution	9
1.6 Thesis Contribution	9
1.7 Thesis Organization	10
2 Related Work	11
2.1 Unidirectional Routing	11
2.2 Probabilistic Routing	13
2.3 Topologies	15
3 Implementation	17
3.1 Simulation Model	17
3.2 Architecture	18
3.3 C++ Implementation	26
3.4 Algorithm Analysis	28

4	Evaluation	31
4.1	Topologies	32
4.2	Fill Time Summary	39
4.3	Routing Tables	41
4.4	Fill Time Breakdown	58
4.5	Congested Nodes	66
4.6	Packet Latency	79
5	Analysis and Discussion	83
6	Conclusion and Future Work	87
	References	95

List of Tables

1.1	Notation for Rich Club Coefficient [22, 65, 43] Equations 1.1 and 1.2. . .	5
1.2	Notation for Equation 1.4 and 1.5.	6
1.3	Notation for Equation 1.7.	7
4.1	Normalized Small World Metrics	39
4.2	Best feedback depth for each data set based on fill time, percentage filled and rejection ratio. Some data sets have two scores listed for comparison since the second best is very close to the best score.	40
4.3	Comparison of peak percentage routing table filled between topologies. .	42
4.4	Average of each packet type dropped across all feedback depths based on topology.	57
4.5	Comparison of average properties between neurons with high rejection ratio against the same topology with the high rejection neurons excluded.	78
5.1	Comparison of metrics for each topology's optimal feedback depth from Table 4.2. Best value with the exception of random topology and cocomac with TT disabled, for each column highlighted in bold.	85

List of Figures

1.1	Two neurons connected via synapse, there is one axon per neuron. Picture taken from [5].	2
1.2	A (top): Rich club network which is more interconnected than a random network shown in A (bottom). B (right): Rich club network nodes [31]. .	4
1.3	Local clustering coefficient of node A is the number of edges amongst its neighbouring nodes (green edges) divided by the maximum number of possible edges amongst the neighbouring nodes. In this example $c_A = 4/10 = 0.4$ [37]	5
3.1	Message sent to node 4 along with feedback returned. This figure does not depict the additional message packets sent/received by each neuron to support unidirectional routing of feedback messages by Chen <i>et al.</i> [21] mentioned later.	19
3.2	EDBF for P2P content search using keywords [38]. As the distance of the search term from the content host increases, less information is advertised. Eventually, the advertisement decays until it is indistinguishable from the noise. The noise is caused by hash collisions due to the decreasing number of bits in the hash used to represent the keyword.	20
3.3	Data flow diagram for routing data packets.	21
3.4	Data flow diagram for routing feedback message packets.	23
3.5	Each neuron runs inside a thread. Each neuron consists of three main components, input queue, probability-based routing engine which decides which type of routing to use based on congestion and routing cache. . . .	25

3.6	Message packet used to carry data or feedback messages, contains support to track the neurons visited, as well as structures used for unidirectional feedback messages including TO/FROM tables and Source Routing. . . .	26
3.7	When the 6th packet is pushed onto the stack, it triggers the batch removal of two packets from the tail.	28
4.1	Comparison of mean normalized rich club coefficient between the four data sets.	32
4.2	Comparison of mean normalized rich club coefficient between transpose topologies.	33
4.3	Comparison of mean normalized cluster coefficient between the original data sets. Some values do not have a normalized clustering coefficient and are thus blank.	34
4.4	Comparison of mean normalized cluster coefficient between transpose data sets. Some values do not have a normalized clustering coefficient and are thus blank.	35
4.5	Comparison of mean normalized cluster coefficient between the sparse and cocomac data sets. There is a close match between these two for most neurons.	36
4.6	Comparison of mean normalized cluster coefficient between the lattice and cocomac data sets. Overall the lattice has a higher CC across most neurons due to its structure.	37
4.7	Comparison of mean normalized cluster coefficient between the random and cocomac data sets.	38
4.8	Fill time between different topologies up to 30% of probability routing table.	39
4.9	Table fill percentage vs. feedback depth comparison between topologies .	41
4.10	Cocomac topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.	43
4.11	Cocomac topology with TO table (TT) routing enabled/disabled. (a) Comparison of routing table fill percentage and packet drop. (b) Average queue size comparison.	45
4.12	Cocomac (TT routing disabled). Breakdown of packet types dropped. . .	46

4.13 Sparse topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.	47
4.14 Lattice topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.	49
4.15 Random topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.	50
4.16 Cocomac (transposed) topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.	52
4.17 Lattice (transposed) topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size. . .	54
4.18 Sparse (transposed) topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size. . .	56
4.19 Normalized routing table fill time for (a) cocomac, (b) cocomac (TT enabled/disabled).	59
4.20 Normalized fill time for (a) cocomac (transpose), (b) sparse topologies. .	61
4.21 Normalized fill time for (a) sparse (transpose), (b) lattice topologies. . .	63
4.22 Normalized fill time for (a) lattice (transpose), (b) random topologies. . .	65
4.23 Cocomac: Comparing rejection ratio, ratio of packets received and cluster coefficient.	66
4.24 Cocomac: Comparing rejection ratio, queue size ratio and compression ratio.	67
4.25 Lattice: Comparing rejection ratio, ratio of packets received and cluster coefficient.	68
4.26 Lattice: Comparing rejection ratio, queue size ratio and compression ratio.	69
4.27 Sparse: Comparing rejection ratio, ratio of packets received and cluster coefficient.	70
4.28 Sparse: Comparing rejection ratio, queue size ratio and compression ratio.	71
4.29 Cocomac (transpose): Comparing rejection ratio, ratio of packets received and cluster coefficient.	72
4.30 Cocomac (transpose): Comparing rejection ratio, queue size ratio and compression ratio.	73
4.31 Lattice (transpose): Comparing rejection ratio, ratio of packets received and cluster coefficient.	74

4.32 Lattice (transpose): Comparing rejection ratio, queue size ratio and compression ratio.	75
4.33 Sparse (transpose): Comparing rejection ratio, ratio of packets received and cluster coefficient.	76
4.34 Sparse (transpose): Comparing rejection ratio, queue size ratio and compression ratio.	77
4.35 Comparison between topologies for packet ratio received by each neuron.	79
4.36 Comparison between transpose topologies for packet ratio received by each neuron.	80
4.37 Comparison between topologies for average packet transit time to each neuron.	81
4.38 Comparison between transpose topologies for average packet transit time to each neuron.	82

Chapter 1

Introduction

For each era, the cutting edge technology of the time was used as a metaphor to propose how the brain works. Due to plumbing advances in the 1600s, French mathematician Descartes thought the nervous system was similar to a plumbing network. As telephones appeared in the late 1800s and early 1900s, some thought the brain worked similar to a telephone switchboard. Today's modern technology is the Internet, thus Graham and Rockmore theorize the brain works similar to a packet switched network like the Internet. Graham and Rockmore indicate the Internet metaphor for the brain may be incomplete and unable to address all of the brain's functionality [28]. Piccinini and Bahar [48] argue the brain is its own type of computer which is neither analog nor digital.

Bassett *et al.* [18] and Sporns *et al.* [59, 57, 56] studied the brain structure to indicate the brain is a complex network. However, they do not fully understand how the brain functions become apparent based on its neuronal structure [59]. This thesis focuses on the impact of routing metrics when implementing a probability routing protocol on the macaque brain topology.

1.1 Background

The brain is made of neurons and connected by axons to carry signals between neurons. Figure 1.1 shows two neurons connected via a synapse. A synapse is a tiny space between the dendrites of two neurons. The neurons communicate by jumping this tiny space via the use of chemical messengers [11]. The axon is wrapped in myelin which acts as an

insulator to allow signals to travel faster than without myelin [11].

Figure 1.1: Two neurons connected via synapse, there is one axon per neuron. Picture taken from [5].

Neuroscientists use different methods of mapping the brain to determine brain connectivity. The traditional method is to inject dyes into different brain regions and to wait for the dye to be propagated to another brain region. The disadvantage is the brain must be sliced open later for analysis and thus not suitable for humans [37].

A non-invasive technique is to use high resolution magnetic resonance imaging (MRI) to identify functional brain regions by recording a time series of activity which is blood oxygen level dependent when the patient performs a simple activity such as finger tapping. Afterward, a matrix can be created based on correlations in the time-series data [37]. There are other approaches like measuring the electromagnetic field of the brain using electroencephalography (EEG) or magnetoencephalography (MEG). All three methods MRI, EEG and MEG can be combined to minimize each method's deficiencies in order to create a correlation matrix. With MRI there is good spatial resolution, but poor temporal resolution. With EEG and MEG, there is good temporal resolution, but poor spatial resolution [60].

Once a connectivity matrix has been created, the brain network can be modeled using a graph. The graph vertices (nodes) represent the neurons. The edges/links connecting the nodes represent the axons [37] connecting the neurons. After the brain is modeled as a graph, network science can be applied to analyze the brain's network properties and a simulation can be run.

1.2 Network Topology Concepts

Network science is the study of networks (graphs). It is based on mathematical graph theory, where each network is represented using a mathematical graph. A vertex represents a node and connections between nodes are called edges or links. Network science has been applied to characterize artificial (e.g. power grid) and natural systems (e.g. brain) as networks. Network science has been applied to analyze the topology of the brain. For example, when neuroscientists applied graph metrics, they noted a decrease in network path length and increased clustering in networks related to world-fluency negatively impacted Alzheimer patients' perception [16].

This section reviews some metrics and terminologies used for network analysis. The terms below have been paraphrased or reproduced from Baronchelli *et al.* [16].

- Assortativity: The tendency for high degree nodes to connect to other high degree nodes. The opposite is disassortativity which high degree nodes tend to connect to low degree nodes.
- Clustering coefficient: Measures probability that two vertices are neighbours of each other if they already share a common neighbour vertex.
- Compression Ratio: The number of incoming degrees divided by number of outgoing degrees for a vertex.
- Connectome: The detailed “wiring diagram” of the neurons and synapses in the brain.
- Hubs: Vertices which have the largest number of connections compared to other vertices.
- Rich Club: A portion of the network consisting of hubs that have noticeably more connections amongst themselves than to the nodes outside.
- Scale-free networks: Networks with broad, heavy-tailed degree distribution and often approximated using the power-law $P(k) \sim k^{-\gamma}$.
- Small-world effect: In a well connected network, it will take only a few hops to reach the destination compared to the total network size.

- Small-world topology: A portion of the network characterized by short path lengths and clustering coefficients larger than 1.
- Transitivity of a network: Tendency for two nodes in network to be connected if they share a common neighbour. Measured by the clustering coefficient.
- Tree: A network with minimum number of edges in which it is possible to get from any given node to any other given node. In graph theoretic terms, this is a fully connected graph without cycles.

Highly connected nodes are called a rich club. It is more highly interconnected network than suggested by measuring the node degree alone. Figure 1.2 displays an example. Rich clubs are detected based on the Rich Club Coefficient (RCC) in Equation 1.1 [22, 65, 43]. RCC measures the tendency of two network hubs to be connected. Equation 1.2 is used, since Equation 1.1 assumes a graph is bidirectional instead of unidirectional. Table 1.1 lists the notation. The numerator represents the number of actual edges amongst nodes with degrees $>k$. The denominator represents the maximum number of connections possible if every node with degree $>k$ was connected to all other nodes with degree $>k$. Thus $\phi(k)$ is the ratio of actual connections versus the theoretical maximum number of connections amongst vertices (nodes) with degree $>k$.

Figure 1.2: A (top): Rich club network which is more interconnected than a random network shown in A (bottom). B (right): Rich club network nodes [31].

$$\phi(k) = \frac{2E_{>k}}{N_{>k}(N_{>k} - 1)} \quad (1.1)$$

$$\phi(k) = \frac{E_{>k}}{N_{>k}(N_{>k} - 1)} \quad (1.2)$$

Table 1.1: Notation for Rich Club Coefficient [22, 65, 43] Equations 1.1 and 1.2.

$V_{>k}$	Set of vertices with degree larger than k
$N_{>k}$	Number of vertices in the set $V_{>k}$
$E_{>k}$	Number of edges amongst $V_{>k}$

However, $\phi(k)$ must be normalized against a random network since nodes with a high degree will be more connected than nodes with a lower degree. The normalization is performed by taking a copy of the network and randomly switching the endpoints of two random edges to preserve the degree distribution. Equation 1.3 is the revised RCC. It is the current network's RCC divided by the randomized network's RCC [43].

$$\rho_{ran}(k) = \frac{\phi(k)}{\phi_{ran}(k)} \quad (1.3)$$

The clustering coefficient (CC) determines the probability that two nodes with a common neighbour are connected, also known as the transitivity of the network [16, 56]. Figure 1.3 illustrates the concept of CC. Equation 1.4 is the local CC (c_i). Equation 1.5 is the global clustering coefficient (GCC) which is the average of CC for all nodes in the network [16].

Figure 1.3: Local clustering coefficient of node A is the number of edges amongst its neighbouring nodes (green edges) divided by the maximum number of possible edges amongst the neighbouring nodes. In this example $c_A = 4/10 = 0.4$ [37]

Equation 1.4 has to be modified for unidirectional network since the original formula assumes each edge is bi-directional. Let c_i be the ratio of actual connections amongst neighbouring nodes compared to the theoretical maximum connections. c_i must be normalized against a random network similar to the method used to normalize RCC. Equation 1.6 is the normalized CC [17]. *Throughout the rest of this thesis, any mention of clustering coefficient (CC) will refer to the normalized cluster coefficient in Equation 1.6, and any mention of RCC will refer to the normalized RCC.* Section 4.1, Table 4.1 lists the GCC for all the datasets.

$$c_i = \frac{e_i}{k_i(k_i - 1)} \quad (1.4)$$

$$\langle c \rangle = \frac{1}{N} \sum_{i=1}^n c_i \quad (1.5)$$

$$\gamma = \frac{c_i}{c_{i(random)}} \quad (1.6)$$

Table 1.2: Notation for Equation 1.4 and 1.5.

c_i	Clustering coefficient for node i
$c_{i(random)}$	Randomized clustering coefficient for node i
e_i	Actual number of edges between neighbours of node i
$k_i(k_i - 1)$	Theoretical maximum number of edges between neighbours of node i
N	Number of nodes (neurons)

Equation 1.7 is known as the Average Shortest Path (ASP) or Minimum Path Length [37, 17]. Path length measures the shortest path in terms of minimal number of edges traversed between two nodes of interest. ASP averages the path lengths between all nodes in a graph. Table 1.3 provides the notation for ASP.

$$\text{Average Shortest Path} = \frac{1}{N(N-1)} \sum_{i,j} d(i,j) \text{ with } i \neq j \quad (1.7)$$

Table 1.3: Notation for Equation 1.7.

N	Number of nodes
i	Source node
j	Destination node
$d(i, j)$	Minimum number of edges between node i and j

Small-world network topology is characterized by high clustering ratio with short path lengths between distant nodes. These properties have been found to exist in social networks, power grids and cellular networks. Small-world networks have $CC > 1$ and the normalized ASP ($\lambda = ASP/ASP_{random}$) to be around 1 [17].

1.3 Routing Protocols

Current work on the analysis of the performance of brain-like networks shows important performance metrics such as mean transit time, throughput, and the like. However, these results were obtained by using simple random routing between the nodes (neurons) [44]. It is of interest to see the impact of selected routing mechanisms such as feedback depth on the performance metrics such as time to fill the routing table, percentage of routing table filled and packet rejection rates. To compound routing complexity, unlike common communications networks, tractable brain models such as CoCoMac [14] contain unidirectional links. Thus, an additional unidirectional routing protocol must be implemented to route feedback packets. In traditional communications networks, each communications link is bi-directional, thus feedback packets can be returned to the source using the same path traversed by the data packet. During run-time, extra communications overhead is also incurred by the additional unidirectional routing protocol.

Routing protocols can be broadly categorized into three types [35, 39, 42]:

- Proactive (Table driven): Sometimes known as link state routing. Maintain up to date information via flooding which sends periodic messages regarding a node's routes to every node in the network. Routing information usually stored in a table. A topology change causes an update to be sent to its neighbours. The advantage is there is no delay in finding a new route. Disadvantage is the additional overhead

of periodically sending updates to other nodes and possibility of network congestion. Examples include Optimized Link State Routing (OLSR) and Destination Sequenced Distance Vector (DSDV).

- Reactive (On demand): Routes are discovered when needed. Use distance-vector algorithms. Examples include Ad hoc On Demand (AODV) and Temporarily Ordered Routing Algorithm (TORA). Usually trade increased latency to find new routes for reduction in control messages sent throughout the network.
- Hybrid: Mix of proactive and reactive. One example is Zone Routing Protocol (ZRP).

The routing protocols can be further classified based on additional factors such as single or multiple channel communications, node hierarchy (e.g some nodes may only respond to routing control messages) and so on.

Reactive protocols scale better in terms of low bandwidth and storage in networks with light traffic and low mobility. Ideal workload for proactive routing are dense networks where low routing delay is important and communication is spread across a large number of nodes [35, 24].

This thesis uses three routing protocols. The first method is a reactive protocol. It uses a probability routing table to route a data packet based on a feedback packet received if the message reaches the destination neuron. The second method is a proactive distance vector routing protocol by Chen *et al.* [21] to route feedback packets over unidirectional links. We refer to this method as *TT routing*. More details on TT routing in Chapter 2 Related Work, Section 2.1 Unidirectional Routing and Chapter 3, Section 3.2 Architecture. The third method is random routing, it is used if probability routing is unavailable for data packets or if TT routing is unavailable for feedback packets.

1.4 Problem Statement

Assuming the brain is a packet switched network, we do not know the impact of network topology on feedback of signals sent via unidirectional neuron connections. We investigate various factors including how quickly each neuron's routing table is filled, packet rejection ratio and percentage of routing table filled when the feedback depth is varied.

1.5 Proposed Solution

Implement a top-down simulation which provide probabilistic feedback to populate each neuron's routing tables with the following functionality:

- Obtain input from a file containing an adjacency matrix.
- Simulation must be configurable in terms of number of nodes to simulate, number of packets to generate, size of neuron queue, feedback depth, time to live for certain feedback packets and number of threads to generate packets.
- Destination neuron sends feedback to each neuron traversed by the packet when it successfully receives a packet.
- Feedback routing must work in an unidirectional network.
- Collect metrics such as rejection ratio, packet transit time, type of packet dropped, etc... needed for analysis.

The simulation is evaluated using connectivity topology from a macaque monkey, lattice, sparse and random topologies to determine the impact of feedback depth on routing tables.

1.6 Thesis Contribution

This thesis makes two contributions:

1. Implementation of a multi-threaded neuron simulator with the functionality listed in Section 1.5.
2. Investigate the impact of network topology on neuron's routing table when varying feedback using the neuron simulator from item number 1 above. A subset of macaque monkey brain and additional topologies were used as input to the neuron simulator.

1.7 Thesis Organization

Chapter 2 reviews the related work including the brain as a network, unidirectional routing, probabilistic routing and brain topology. Chapter 3 reviews simulation model, implementation details including the hardware, software and architecture. Chapter 4 reviews the results of the simulator. Chapter 5 analyzes and discusses the results. Chapter 6 is the conclusion and future work.

Chapter 2

Related Work

2.1 Unidirectional Routing

There are multiple routing protocols which support unidirectional routing. The methods below [47, 32, 27, 45, 21] rely on flooding thereby increasing the network congestion. Some methods reduce the overhead by using unicast instead of multicast or other methods.

Dynamic Source Routing (DSR) does not support unidirectional links, but extended DSR (eDSR) does. It uses two way flooding between the source and destination to discover a route. The source broadcasts RREQ (Route Request) to all its neighbour nodes. Each neighbour node forwards the RREQ to its neighbours if it has not already received the RREQ and appends its own address to the RREQ. Once RREQ reaches its destination, a back channel for the RREP (Route Reply) is discovered via flooding as well. Piao and Chang [47] improve eDSR to create Transmission Power Based Source Routing (TPSR) so each node is aware whether its neighbouring nodes are bi-directional or unidirectional during the route discovery phase. TPSR has lower end to end packet delay compared to DSR and eDSR. Disadvantage of TPSR is each node periodically broadcasts HELLO messages to determine which links are bidirectional or unidirectional. Nodes on unidirectional link are unable to reply to the HELLO message. The HELLO messages may increase network congestion.

Higaki [32] create Loop Based Source Routing (LBSR) which combines one flooding with multiple unicast messages. It finds a loop between the source and destination nodes concurrently instead of discovering them separately as in DSR. A loop is defined as a path

in a graph which has no repeated node. A loop starts and ends at the same node. Loop discovery is accomplished by sending a LREQ message which is eventually forwarded back to the source from the destination node. Their study concludes LBSR broadcasts less messages than DSR.

Gerla *et al.* [27] modify the On-Demand Multicast Routing Protocol (ODMRP) to adapt it for unidirectional networks called ODMRP-ASYM. ODMRP-ASYM exhibits marginal performance degradation compared to ODMRP in environments containing many unidirectional links. It also relies on broadcasting for loop discovery. Periodic flooding is used to update routes and group membership information.

Morino *et al.* [45] create FOCUS2, a flooding based protocol which limits the flooding area when building a return back channel. It reduces the number of control packets by 50% compared to DSR flooding.

Chen *et al.* [21] create a distance vector protocol. Each node contains a set of FROM tables (*FT*) and TO tables (*TT*). *FT* contain entries on how other nodes can reach the target neuron. *TT* is used as the actual routing table to determine the next hop based on destination. *FT* are periodically broadcast to each neighbouring node which uses it to update its own *FT* and *TT*. The disadvantage is the periodic *FT* broadcasts may congest the network.

Liu *et al.* [41] propose the Loop-Based Clustering Routing (LBCR), an on-demand protocol. The nodes are partitioned into loop clusters. Each cluster contains information regarding its members, thus a node can easily determine whether a message should be routed to one of the members or to an external cluster. This method reduces overhead of broadcasting RREQ and RREP messages. They compare LCBR against UANR [49] and unidirectional distance routing by Chen *et al.* [21] to conclude their method is generally better in terms of storage and communications complexity. Su *et al.* [61] also base their routing scheme on clusters, but use a multi-hop acknowledgment scheme.

Perlman *et al.* [46] rely on the network layer routing information to unicast acknowledgments from the destination to sender. Normally the receiver would attempt to broadcast its response to the sender. Their paper does not focus on methods used by network layer to support unidirectional networks, but they mention previous research such as tunneling and modifying the routing protocol.

AODV-EUDA works around Ad Hoc On Demand's (AODV) limitation for unidirectional links by not using them to transmit data. Shrestha *et al.* [55] extend AODV-EUDA

so unidirectional links are used for data transmission. It works by electing a monitor node during route discovery which has bidirectional links with the source and destination. The sender unicast the message to its neighbouring nodes who attempt to forward the message to the destination. Upon reaching the destination, the reply is sent via the monitor node. In some cases the monitor node is unavailable so they revert back to AODV-EUDA behaviour.

Bao and Garcia-Luna-Aceves [15] create a new routing algorithm called Unidirectional Link-State protocol (ULP). ULP adapts the Link Vector Algorithm (LVA) [25] for unidirectional links. Each node in LVA updates its neighbours the links it uses to reach a destination and the status (link up/down). This information is used to determine the network topology so a routing algorithm be run to get the next hop.

2.2 Probabilistic Routing

Bhorkar *et al.* [19] design the learning routing scheme called d-AdaptOR based on transmission success probabilities using a stochastic routing scheme.

Cui *et al.* [23] create a probabilistic multipath routing protocol for multiple flows. Each flow finds disjoint multiple routes from source to destination to avoid going through nodes neighbouring other routes. The route with the largest effective bandwidth is chosen. Route discovery is based on DSR which sends RREQ messages to collect information on each of the links.

Ziane and Mellouk [66] propose an Adaptive Mean Delay Routing (AMDR). It explores the network to use the MAC layer protocol IEEE 802.11 to estimate each node's mean delay. This information is sent back as feedback to update a probability routing table to increase/decrease the probability for a destination.

Xie *et al.* [64] use Wardrop routing which is also known as user-optimal routing. Each node receives latency updates from neighbouring nodes which are used to update its own routing delay estimation to a destination. No mention whether this can be applied to unidirectional routing.

Methods [30, 62, 53] are reactive protocols which do not require pre-exploration of the network to establish routing probabilities. However each have their own advantages and disadvantages since some of them rely on flooding for communication.

Gupta [30] make use of probabilistic function in Delay Tolerant Networks (DTN) architectures such as underwater or deep space using the Bundle protocol. They propose Bundle Fault Tolerance Based Probabilistic Routing scheme (BFTBPR). When node A comes into contact with node B they compare their probabilities for a destination. If A decides to route a packet through B, A increases its probability.

Tan and Munro [62] modify the epidemic algorithm which relies on broadcasting to create Adaptive Probabilistic Epidemic protocol (APEP). They reduce the number of collisions from flooding by including information regarding estimated known neighbours so each receiving node can decide whether to re-broadcast the message.

Sandalidis *et al.* [53] use swarm intelligence from artificial ants following a pheromone (scent) trail based on Schoonderwoerd *et al.* [54] model. Each ant is analogous to a message and routed based on a probabilistic routing table. Normally ants are routed using the highest probability in the routing table, but their approach uses the probability to randomly route a message in order to avoid congestion. When an ant reaches its destination the routing table probability is increased/decreased based on the age of the ant. Advantage is there is no need to send messages for route discovery, thus reducing network congestion.

Replication based routing use a "store and forward" methodology for networks with intermittent or unpredictable connectivity. Flooding is used to increase the probability of message delivery at the cost of network congestion. One example of a replication based routing scheme is Epidemic. Proadhan *et al.* [50] introduce a quota based routing protocol called ProbRoute to limit message flooding. A probability metric is used to determine which node to forward each message instead of blindly forwarding the message as in a conventional quota based routing protocol.

Athanasopoulou *et al.* [13] probabilistically route a packet by modifying the back-pressure routing algorithm using movement of fictitious packets. The back-pressure algorithm [62] has each node maintain one queue for each of its neighbouring nodes to track congestion. Routing is based on the congestion information provided by the neighbours. Its main drawback is one queue must be maintained for each destination or flow.

The probability routing used for this thesis was inspired by Kumar *et al.*. They use probabilities to search for keywords using a peer to peer search protocol. The probability for each search term or keyword was decayed exponentially and advertised to each neighbouring node. Thus a random walk through the network would quickly converge

upon the node if it is nearby. Further details in Chapter 3, Section 3.2 Architecture.

2.3 Topologies

With the exception of the papers by Misic *et al.* [44] and Graham and Rockmore [28], there is very little work on brain routing [29]. Most of the papers below focus on applying network science to analyze the topology of the brain.

Bassett and Bullmore [17] apply the small-world network model to the brain. Small-world properties consist of high local clustering ratios with short path lengths. The small-world model is ideal since it allows for high global and local efficiency when processing the information. The sparse connectivity also reduces wiring costs in the brain [17]. Hilgetag *et al.* [34, 33] and Sporns *et al.* [58] also find small world properties in the cat and macaque brains.

Colizza *et al.* [22] quantify rich clubs for different types of networks including biological, transportation and the internet. McAuley *et al.* [43] scrutinize real world physical, social and biological networks for rich club characteristics. Harriger *et al.* [31] detect rich club organization in the macaque cerebral cortex. Heuvel *et al.* [63] detect rich clubs in the human connectome.

Humphries *et al.* [36] propose some metrics to measure the extent to which a network has small-world and scale free properties.

Misic *et al.* [44] examine the communication efficiency on the macaque monkey brain. They conclude that the macaque brain network resembles a rich club, that rich club regions handle more traffic than non rich club regions and the network can dynamically shape traffic flow for workload balancing.

Sporns *et al.* [56] apply network analysis to gain an understanding of how cognition arises from the brain.

This thesis considers four topologies, the cocomac [14] macaque monkey brain topology, sparse topology, lattice and random topologies. The sparse topology was included because Bassett and Bullmore [17] indicate there is strong empirical evidence the brain is sparsely connected. The first three topologies are transposed to create three additional topologies.

The next Chapter 3 Implementation, reviews the simulation model, architecture,

C++ implementation and algorithm analysis.

Chapter 3

Implementation

3.1 Simulation Model

There are two approaches to creating a simulator. The first method is the bottom up approach where individual neurons, synapses, neuron signal propagation, millivolt threshold to activate/reset a neuron, etc.. are simulated. Examples of this method include the Brian spiking simulator [1], Virtualbrain [40] and IBM's C2 cortical simulator which simulates 1.6 billion neurons and 8.87 trillion synapses using a supercomputer [12]. Nengo [8] allows the user to quickly create groups of neurons and to form connections between them to build large scale cognitive models.

The second method takes a top down approach using agents. Examples include Simbrain [7], a neural network simulator that allows the user to interface a neural network to an artificial sandbox to observe its behaviour.

The second approach was taken, since the goal was to observe the impact of feedback on routing table metrics using different topologies. The top down approach also helps to reduce complexity during implementation. Initially, the multi-agent frame work Flexible Large-scale Agent Modeling Environment [6] was considered, but was unsuitable due to limited support for arrays [4].

3.2 Architecture

Figure 3.1 displays the overview of the simulation system. Each neuron is run as a thread. We use the term neuron and node interchangeably. We also use the term packet and message interchangeably. Data packets are routed between the neurons until they reach their destination. Upon reaching the destination, the neuron sends feedback to x number of previous neurons the packet traversed. We refer to this as the feedback depth (fbDepth). fbDepth is configurable by the user. Each feedback message contains a probability multiplier which decreases as the distance from the destination increases.

In Figure 3.1, a message is sent from node 0 (n0) to n4 via n2. n4 sends feedback to the last x nodes visited so they can update their routing tables. Since paths are unidirectional, feedback messages traverse a different path back $n4 \rightarrow n1 \rightarrow n2 \rightarrow n3 \rightarrow n0$. In some cases, each feedback message can take a return path different from other feedback messages. Neurons closer to the destination receive a higher probability (1.5) which decreases as the distance from n4 increases. n2 receives a feedback multiplier (fbMult) of 1.6, n3 receives fbMult=1.38 and n0 receives fbMult=1.10 since it is furthest away from the destination. fbMult is used at each neuron to increase the probability of the routing entry. New routing entries begin with the probability of 0.10 and multiplied by fbMult. Once a probability routing entry reaches the maximum of 1.0, a message is returned to the sender to cease sending further fbMult packets in the future.

Equation 3.1 displays the formula used to calculate fbMult. n is the neuron connectivity degree from the destination. Thus the first neuron one degree from the destination with a feedback depth of 3 is $fbMult = \log(3 + 1 - 1) = 1.099$. The formula for fbMult was chosen after some experimentation to ensure it scales well over 243 neurons. For values of $fbMult = \log(2) = 0.69$ and $fbMult = \log(1) = 0$ this would decrease the probability, but when comparing these results with a different formula which would add 1 if $fbMult = \log(1)$ or $fbMult = \log(2)$, there was no difference in routing table fill percentages.

$$\log(fbDepth + 1 - n) \tag{3.1}$$

The probability feedback was inspired by Kumar *et al.* [38]. They implement a peer to peer search (P2P) protocol to search for content using keywords by creating a hash for

Figure 3.1: Message sent to node 4 along with feedback returned. This figure does not depict the additional message packets sent/received by each neuron to support unidirectional routing of feedback messages by Chen *et al.* [21] mentioned later.

each keyword at each node. Each node then advertised its hash to neighbouring nodes in the P2P group. The advertisement was decayed via time using an Exponential Decay Bloom Filter [20] by reducing the number of bits in the hash used to represent the search term. This is shown in Figure 3.2. For each network node, the further away a term was, the lower probability of finding the keyword. To search for a term, a random walk is performed. Once the search is near a node with the search term, it is quickly routed toward the node containing the content to be retrieved. Rhea *et al.* [52] develop a similar P2P search system using an attenuated Bloom filter. In our approach, instead of using EDBF, we use Equation 3.1 for decay.

Figure 3.5 illustrates the main components of each neuron. The input signals are buffered in a LIFO manner, the same method as a neuron. There is only one input queue acting as a stack. When the stack is full, the oldest signal at the tail is removed [44].

Normally, returning feedback to a neuron should not be difficult if the links between neurons are assumed to be bi-directional. However, the macaque network topology contains unidirectional links. The unidirectional network can be overcome by implementing TT routing [21]. The trade-off is additional complexity in implementing TT routing,

Figure 3.2: EDBF for P2P content search using keywords [38]. As the distance of the search term from the content host increases, less information is advertised. Eventually, the advertisement decays until it is indistinguishable from the noise. The noise is caused by hash collisions due to the decreasing number of bits in the hash used to represent the keyword.

along with extra overhead (e.g. additional network congestion) when running this protocol.

In TT routing, each node contains a set of FROM tables (FT) and TO tables (TT). FT contain entries on how other nodes can reach the target neuron. TT is used as the actual routing table to determine the next hop based on destination. FT are periodically broadcast to each neighbouring node which uses it to update its own FT and TT . The FROM tables are enclosed inside broadcast packets. The disadvantage is the periodic FT broadcasts may congest the network.

This thesis implements three routing protocols:

- Probability routing: Uses feedback received if message successfully reaches destination to increment probability in routing table entry for destination. This is the primary method to route data packets and the secondary method to route feedback if TO table routing is unavailable.
- TO table routing: Since network is unidirectional, returning feedback to the sender may take a different path. Chen *et al.* unidirectional routing protocol was implemented [21]. This is the primary method to route feedback packets.

- Random routing: Used if the primary or secondary method to route data or feedback packets are unavailable.

The routing engine routes data messages using the Probability Routing Table (PBRT) or random routing if there is no PBRT entry. *From here on, we will refer to PBRT as **routing table** for brevity.* The data packet is routed by choosing the next hop with the highest probability based on the destination. There is additional functionality to route messages based on a score which is the probability divided by the cost of the link. Score is not used since our adjacency matrices do not have a weight for each edge. During initialization, routing table is empty except for neurons which it is directly connected to. Figure 3.3 shows the flow diagram to route data packets. The algorithm is very simple. If there is a probability routing table entry available with probability >0.05 it is used, otherwise it is routed to a random neuron. The probability of 0.05 was arbitrarily chosen.

Figure 3.3: Data flow diagram for routing data packets.

By default, feedback messages are routed via a back channel using TT routing by Chen *et al.* [21], which included a Time To Live (TTL) field for each entry in the FT

and TT tables. This thesis did not implement a TTL for each entry in the FROM and TO tables, because our experiments test static network topologies.

When routing feedback, if the congestion for the next hop exceeds a specified threshold, a Bernoulli trial will be run to determine whether to use TT routing or an alternate method. If the trial fails, the alternate method checks to see if there is a valid routing table entry, if not it resorts to using random routing. Feedback packets are not cached in the routing cache. Note feedback uses the same routing table as data packets if this method is chosen. Depending on the number of neurons in the network, congestion and feedback depth, there are routing table entries which will not be populated.

To reduce the number of feedback and broadcast messages routed, once the probability for a destination reaches the maximum of 1.0, a message will be sent to the originator node to cease further feedback. The number of broadcast messages are dynamically reduced as the TT routing table is filled by running a Bernoulli trial of various probabilities. For example, if the TO table is less than 50% full, a Bernoulli trial will be run with probability 0.02 of success to send out FROM tables. With TT more than 50% full, the probability is decreased to 0.0175. Sending out a FROM table triggers the receiving neuron to update its own FROM and TO tables.

Figure 3.4 is the flow diagram for routing feedback messages followed by the pseudo code listing. The algorithm checks to ensure there is a valid TT routing entry for the destination, then checks the destination's queue congestion to ensure it is below the arbitrary threshold of 70%. If neither of these conditions is met, it starts a Bernoulli trial to decide whether to use TT routing, otherwise it will try probability routing, followed by random routing.

Figure 3.4: Data flow diagram for routing feedback message packets.

Algorithm 1 is the pseudo code for Figure 3.4. It ensures a TT routing entry is available and queue congestion is less than 0.70 on line 1. At line 4, a Bernoulli trial is run with a 50% success/failure rate to determine whether to try TT routing again if this option is available. Otherwise it will try using probability routing (line 7) or random routing (line 10) if the former is unavailable. Both the congestion threshold of 0.70 and Bernoulli trial’s success probability were arbitrarily chosen.

Congestion control checks the queue size of the next neuron it will route to and adds a delay of 5 milliseconds for each queue entry used. There is no congestion control when using random routing. When each data packet is routed, its next hop is stored inside the routing cache.

The routing cache is implemented as a linked list with a limit of four possible next hops for each destination. The cache removes the lowest probability entry for destination x when the cache limit is exceeded. When a feedback packet is received, the routing

Algorithm 1 Feedback Routing

```

1: if congestion < 0.70 && TTAvalable then
2:   routeTT
3:   goto exit:
4: else
5:   BernTrial = genBernTrial(0.50)
6:   if TTAvalable && BernTrial == true then
7:     routeTT
8:   else if ProbAvalable then
9:     routePb
10:  else
11:    routeRandom
12:  end if
13: end if
14: :exit

```

history contained inside the packet is checked to determine which next hop was chosen in the past by the receiving neuron. The corresponding cache entry has its probability multiplied by $fbMult$. The updated entry is synchronized between the routing cache and routing table to ensure the routing table always contains the highest probability for each destination. It is possible two messages going to the same destination would be routed differently based on different feedback.

Each neuron has a random service delay before it can process an incoming packet. The delay varies from 0 to 45 milliseconds and is generated using a uniform distribution with the Mersenne Twister random number generator.

Figure 3.5: Each neuron runs inside a thread. Each neuron consists of three main components, input queue, probability-based routing engine which decides which type of routing to use based on congestion and routing cache.

Figure 3.6 displays the contents of the message packet used to send data and feedback messages. VisitHistory structure is used to prevent messages from being stuck inside a cycle in the network. If a message has visited neuron x more than two times, the packet is dropped. Structures related to source routing are needed for TT routing. Source routing is functionality which stores the neurons to traverse inside the message packet.

Figure 3.6: Message packet used to carry data or feedback messages, contains support to track the neurons visited, as well as structures used for unidirectional feedback messages including TO/FROM tables and Source Routing.

3.3 C++ Implementation

The simulation was written using the newest 2011 standard of the C++ programming language because it includes native support for multi-threading and data structures without the need to use third party libraries [3]. It is also known as C++11. Older standards required the use of third party libraries for multi-threaded support. Since the simulator needs to simulate many neurons, performance was a priority. Thus a compiled language was chosen instead of an interpreted language.

C++'s condition variables were used for communication to notify another thread of an event. For example, when a new message is pushed onto a neuron's stack it will wake up the receiving thread to process the message. Mutexes were used instead of atomic types to serialize access to concurrent data due to simpler implementation.

The class `msgStk_t` is the packet stack used by each thread to implement the message queue. It is based on the C++ list container class with additional `push()` and `pop()` methods to simulate the LIFO behaviour of a stack. The initial design of the class would delete a message once the queue size is exceeded by one. However this was changed to a batch system where the delete is deferred until the queue size has been exceeded by 20. 20 or 1% of maximum neuron queue size was chosen after running a quick test to check CPU usage. Figure 3.7 displays an example of the stack internals. The artificial maximum size (`qArtSize`) is 3 packets, with batch removal triggered when `qSize` exceeds 5 packets. `qArtSize` is the maximum queue size specified by the user when they run the simulation. When the 6th packet is pushed onto the stack from the head, two packets are removed from the tail (e.g. `tail`, `tail-1`). After batch removal, the actual queue size is 4. When two more packets are inserted, batch removal will be triggered again to remove two more packets (`qRem=2`).

The batch removal feature significantly reduced CPU consumption. The class `msgStk_t` also collects metrics regarding the size of the queue and types of packets dropped.

The routing cache was implemented using a list data structure in the class `rtCache`. `rtCache` permits a maximum of four next hop entries per destination. For example, if there are 242 neurons, there would be up to $4 \times 242 = 968$ entries in the routing cache.

Time stamps to measure time taken to fill a routing table were implemented using dynamically allocated arrays in the class `rtTimer`. It contains methods to create and store the increment text to display e.g. (10%, 20%, etc...) and C++'s `time_point` class data. The class constructor permits varying increments to be used. For this thesis ten percent increments were used, but it can use other increments chosen by the user such as 20%, 25%, etc...

Metrics were implemented for each neuron to collect information such as number of data message packets dropped, routed via probability routing, routed via alternate methods, average number of hops taken to reach destination, etc... Metrics were stored in the class `neuronMetrics` by each neuron thread and then returned to the main calling thread where the information was aggregated. The main thread produces several comma

Figure 3.7: When the 6th packet is pushed onto the stack, it triggers the batch removal of two packets from the tail.

delimited (.csv) files containing metrics reported from individual threads and a summary of all results. The .csv file format permits import into a spreadsheet.

The Valgrind [9] set of tools along with GNU debugger was used to debug data races, deadlocks and memory corruption in the multi-threaded neuron simulation. The simulation was compiled and run on Ubuntu Linux Server 12.0.4 using GCC compiler version 4.8.1 with 64GB RAM.

3.4 Algorithm Analysis

Complexity analysis was used to determine how data set size change would affect the resource usage. Most of the routing algorithms and classes used are of complexity $O(n)$, where n is the number of neurons. The classes used to support TT routing and probability routing require $O(n)$ amount of storage space, one for each neuron. Data structures such as stack and list use mostly $O(1)$ complexity operations. There is one small discrepancy, the GCC compiler used implements `std::list::size()` of complexity $O(n)$ [2] due to a bug,

whereas the C++11 standard indicates it should be $O(1)$ [10]. The feedback depth and routing cache call these functions, but the impact is minimal if the feedback depth is not too large.

Equation 3.2 calculates the number of messages broadcast to support TT routing of feedback messages. Each neuron would replicate an incoming message and send it to all its directly connected neighbours. i is the neuron number from 0 to maximum number of neurons m . D is the number of outgoing degrees (edges) from the neuron i , P_{genpkt} is the probability of generating a packet and W is the number of times the neuron will wake up per minute to make a broadcast. The worst case complexity is $O(n^2)$.

The neighbours receiving the broadcast can continue to rebroadcast the message, however the current implementation limits the broadcast to one hop instead of multiple hops to limit the number of messages routed through the network.

Equation 3.3 lists the number of messages routed when sending feedback to n neurons with a feedback depth of f . The feedback depth can be from 1 to maximum number of neurons m . L represents the path length taken for each feedback message which varies from 1 to m as well. The worst case complexity would also be $O(n^2)$.

$$\text{Messages broadcast} = \sum_{i=0}^m (n_i) D_i P_{genpkt} W \quad (3.2)$$

$$\text{Messages routed} = \sum_{f=1}^m n_f (L_f) \quad (3.3)$$

Chapter 4

Evaluation

Four data sets consisting of adjacency matrices were used to evaluate the impact feedback depth on how quickly the probability routing table was filled. The data sets are macaque monkey (cocomac), lattice, randomized and sparse topologies. All data sets consist of 4090 edges and 242 nodes and were from [44]. The cocomac data set originates from the CoCoMac database which contains results from hundreds of research papers published from tract (neural pathway) tracing studies done on the macaque monkey brain [14]. The random data set is an average of three randomly created data sets using a Mersenne twister random number generator with a uniform distribution.

The simulation uses a two dimensional adjacency array for each topology to determine whether an edge exists from a node to another node. For example, if there is an edge from node 5 to node 73, it checks `adjArray[5][73]` to determine if there is a 1 (edge exists) or 0 (no edge exists). To determine if there is an edge from node 73 to node 5 it would check `adjArray[73][5]`. To further investigate the impact of topology, three additional data sets were derived by transposing the adjacency matrices of the cocomac, lattice and sparse data sets. Each topology was transposed by swapping the adjacency array's co-ordinates so incoming edges become outgoing edges and vice versa. For example, `adjArray[5][73]` becomes `adjArray[73][5]`.

Packets were generated using a Poisson distribution to vary the arrival time [44, 26]. The arrival time varies between 0 to 45 milliseconds. Before processing a packet, each neuron has a random service delay which varies from 0 to 45 milliseconds and an uniform distribution. For both arrival time and service delay, two separate independent Mersenne

twister random number generators were used.

Section 4.1 reviews characteristics of the topologies used in the simulation. Section 4.2 presents an summary of the findings taking into consideration the factors percent routing table filled, packet drop and routing table fill time. Section 4.3 presents detailed information regarding percent routing table filled and packet drop information. Section 4.4 breaks down the routing table fill time, Section 4.5 reviews some congested nodes and Section 4.6 examines the transit time of packets.

4.1 Topologies

This section presents some properties of the topologies used to run the simulation.

Figure 4.1: Comparison of mean normalized rich club coefficient between the four data sets.

Figure 4.1 displays the mean RCC for the group of vertices whose degrees are greater than $>k$. $\rho_{ran}(k) > 1$ over a range of k indicates a rich club exists in the network [63].

Figure 4.2: Comparison of mean normalized rich club coefficient between transpose topologies.

Figure 4.1 compares the mean RCC between all four data sets. As k increases, the number of nodes in each group decreases. The lattice data set's RCC is the most stable of the four data sets since most of its nodes have more than approximately 55 degrees. There is a close correlation between the cocomac and sparse data sets. The random data set does not have any neurons with degree $k > 42$ so the RCC drops to zero.

With the transpose topologies in Figure 4.2, the cocomac, lattice, sparse and random transpose topologies begin to diverge at $k > 19$. There is no close match between the transpose sparse and cocomac topologies. The lattice maintains RCC close to 0.90 for $k > 52$, but the transposed version maintains approximately the same RCC value up to k

>37. All transpose data sets have a RCC higher than the random data set.

The non-transposed data set maintains an overall RCC higher than the transposed data set over a larger range of k .

Figure 4.3: Comparison of mean normalized cluster coefficient between the original data sets. Some values do not have a normalized clustering coefficient and are thus blank.

Figure 4.4: Comparison of mean normalized cluster coefficient between transpose data sets. Some values do not have a normalized clustering coefficient and are thus blank.

Figure 4.3 compares the mean CC between all four data sets. There is a close correlation between the cocomac and sparse data sets. Figure 4.5 and 4.6 provide a clearer comparison between cocomac against the sparse and lattice data sets. Due to the lattice's more consistent structure in terms of in degrees and out degrees, its cluster coefficient is mostly consistently higher than the other topologies. The random topology has the lowest CC since it is a random structure.

Figure 4.3 indicates most neurons in the cocomac data set have $CC > 1$ which is one of the indications of a small world network. The other indication is an Average Shortest Path (ASP) (refer to Section 1.2, Equation 1.7) of approximately 1 [17].

The Global Clustering Coefficient (GCC) (refer to Section 1.2, Equation 1.5) is the average of mean normalized cluster coefficient across all nodes in the data set. The cocomac’s GCC is 3.30, while the sparse is 3.03. The lattice has GCC of 5.34, while the random network is 1.02. These are also listed in Table 4.1.

Figure 4.4 compares the transpose version of the data sets. There is a similar result. The transpose lattice structure is consistently higher than the other data sets. There is a similarity between the cocomac and sparse data sets.

Figure 4.5: Comparison of mean normalized cluster coefficient between the sparse and cocomac data sets. There is a close match between these two for most neurons.

Figure 4.6: Comparison of mean normalized cluster coefficient between the lattice and cocomac data sets. Overall the lattice has a higher CC across most neurons due to its structure.

Figure 4.7: Comparison of mean normalized cluster coefficient between the random and cocomac data sets.

Table 4.1 compares the ASP and clustering coefficients against Bassett and Bullmore’s results [17]. GCC is a close match between the cocomac and sparse data set. The cocomac data set has a GCC within 10% of Bassett and Bullmore’s results for the Macaque cortex.

Table 4.1: Normalized Small World Metrics

Data set	Average Shortest Path	Global CC	Source
Macaque visual cortex	1.04	1.47	Bassett & Bullmore [17]
Macaque whole cortex	1.17	3.06	Bassett & Bullmore [17]
cocomac	2.68	3.30	This thesis.
sparse	2.68	3.03	This thesis.
lattice	2.79	5.34	This thesis.
random	2.51	1.02	This thesis.

4.2 Fill Time Summary

Figure 4.8: Fill time between different topologies up to 30% of probability routing table.

Figure 4.8 displays the time to fill the probability routing table to 30% for each data set when varying feedback depth. Table 4.2 summarizes the results by displaying the ideal fbDepth which provides the fastest fill time while taking into consideration the packet rejection ratio and percentage of neurons whose routing table was filled to 30%. A lower score is better for the fifth column (Time / (Fill:Drop)) of Table 4.2.

For the cocomac data set, fbDepth=30 is the fastest. For cocomac with TT disabled, it is fbDepth=10. The sparse data set at fbDepth=25 with fill time of 557 min. is very close to cocomac with 556 min. at fbDepth=30. Surprisingly, the random topology performs exceptionally well by a very wide margin of 55 minutes compared to 404 minutes for cocomac topology with TT disabled, followed by the transposed cocomac (476 min.).

Table 4.2: Best feedback depth for each data set based on fill time, percentage filled and rejection ratio. Some data sets have two scores listed for comparison since the second best is very close to the best score.

Data set	Feedback Depth	Fill Time (min.)	Fill:Drop ratio	Time / Fill:Drop
cocomac	15	581.9	0.609	955.31
cocomac	30	556.4	0.608	915.5
cocomac (transpose)	20	478.7	0.678	705.9
cocomac (transpose)	30	476.15	0.688	691.9
cocomac (TT off)	10	404.4	1.01	399.3
sparse	25	557.6	0.628	888.0
sparse	40	577.82	0.624	925.1
sparse (transpose)	5	562.2	0.62	913.2
sparse (transpose)	30	574.2	0.62	932
lattice	25	1048.5	0.522	2006.6
lattice	40	1093.5	0.516	2118.9
lattice (transpose)	5	1244.4	0.74	1691.6
lattice (transpose)	30	1049.2	0.74	1414.9
random	15	<i>55.82</i>	<i>59.64</i>	<i>0.94</i>

4.3 Routing Tables

There are two routing tables used. The first one is the probabilistic routing table, the second is the TO table (TT) used to optionally route feedback depending on the queue congestion of the next hop neuron. We refer to the probabilistic routing table as *routing table* for brevity. This section compares the fill percentage of the routing table when the feedback depth is varied. Section 4.4 provides a detailed breakdown of how quickly the routing table was populated.

Figure 4.9 displays the percentage of routing table filled compared to feedback depth across the different topologies. The random topology performs the best, followed by the cocomac topology with TT disabled, then transpose cocomac.

Figure 4.9: Table fill percentage vs. feedback depth comparison between topologies

Figure 4.10 (a) shows the percentage of routing table entries which are filled along with the percentage of packets dropped as the feedback depth is increased. The TO

table and FROM table are additional tables used by each neuron to support feedback routing via a back channel. For all topologies except random, fbDepth from 15 to 40 yield the highest percentage of routing table filled. The random topology has the highest fill percentage close to 45% at fbDepth=15. Peak percentage filled for each data set shown in Table 4.3.

The transposed lattice topology exhibits strange behaviour since it peaks at fbDepth=5 and 30. The sparse exhibits inverse behaviour compared to its transposed version at fbDepth greater than 10.

Table 4.3: Comparison of peak percentage routing table filled between topologies.

Topology	Feedback Depth	Percentage Filled
cocomac	30	37.5
sparse	25	37.8
lattice	40	28.9
random	15	44.27
cocomac (transpose)	30	39.5
sparse (transpose)	30	37.4
lattice (transpose)	30	30.9
cocomac (TT disabled)	10	39.8

The percentage of routing table filled for most topologies is independent from feedback depths 5 to 40. There is a small improvement increasing the feedback depth from 3 to 5. This seems to confirm findings from Ramasubramanian and Mosse [51] where reverse back channels from 2 to 3 hops have good connectivity. The total percentage of packets dropped also remains steady in this interval as well. There is no significant marginal benefit when using feedback depths larger than 5. For all topologies, the percentage of packets dropped due to queue full versus other reasons such as hops exceeded remained steady and did not exceed 4%. Thus a breakdown of packet types dropped due to queue full was performed for each topology.

Figure 4.10: Cocomac topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.

Figure 4.10 (b) provides a breakdown on the type of packet being dropped. The majority of packets dropped are broadcast (BCAST) and data packets. RT_UP packets are used to tell a neuron to update its routing table. RT_UP (routing table update) can also be categorized as a feedback packet, but a separate metric was created to determine the percentage of these packets being dropped. The drop rate for RT_UP packets is close to zero. BCAST packets are the FROM tables periodically broadcast. The average queue size was approximately less than 50% full.

The TT tables used to support the routing of feedback packets through the back channel was disabled to gauge its effect on routing table fill percentages. In theory, disabling TT routing should have a negative impact on routing table fill percentage, since each feedback message would be randomly routed. With TT disabled, there is a higher fill percentage as in Figure 4.11 (a), lower average queue size (Figure 4.11 (b)) and much faster fill time (Figure 4.19). The packet rejection ratio takes a dive from around 60% to 40%. TT routing relies on periodic broadcasts of FROM tables, which also trigger an exchange of TO tables. The lower packet rejection rate and queue size point to TT routing causing network congestion due to its periodic broadcasts of FROM tables.

(a)

(b)

Figure 4.11: Cocomac topology with TO table (TT) routing enabled/disabled. (a) Comparison of routing table fill percentage and packet drop. (b) Average queue size comparison.

Figure 4.12: Cocomac (TT routing disabled). Breakdown of packet types dropped.

(a)

(b)

Figure 4.13: Sparse topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.

The sparse data set's results in Figure 4.13 are similar to the cocomac data set from Figure 4.10 in terms of routing table filled and dropped packet ratio. The low percentage of TO table used to route feedback has minimal effect on the routing table even though it is significantly smaller at fbDepths of 10, 20 and 30. The TO table routing will be used if there is less than 70% queue congestion. It is possible due to congestion, the feedback is being randomly routed most of the time and thus a drop in the TO table does not have much effect on the probability routing table being filled.

The ratio of packet types dropped is also similar to cocomac. The quickest routing table fill time to 30% from the sparse data set is almost identical to the cocomac data set. See Section 4.2, Table 4.2 for comparison.

The lattice data set in Figure 4.14 has a drop percentage similar to the cocomac and sparse data sets. However its packet drop percentage is slightly lower at around 55% compared to close to 60% for cocomo and sparse data sets.

Figure 4.14: Lattice topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.

(a)

(b)

Figure 4.15: Random topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.

Figure 4.15 (a) shows a dramatic decrease in the number of total packets dropped to almost zero. For other topologies the packet drop rate is normally in the 40 to 60% range. The average queue size is much higher at around 80% versus around 50-60% for other topologies.

Figure 4.16: Cocomac (transposed) topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.

The routing table fill rate in Figure 4.16 (a) hovers close to 40% vs. mid to upper 30% for non-transpose cocomac.

(a)

(b)

Figure 4.17: Lattice (transposed) topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.

In Figure 4.17 (a) the routing table fill percentage has two peaks, one at fbDepth=5 and another at fpDepth=30. The percentage of packet type dropped in Figure 4.17 is similar to cocomac.

Figure 4.18: Sparse (transposed) topology. (a) Routing table fill percentage and packet drop. (b) Breakdown of packet types dropped and average queue size.

With the sparse (transpose) topology, the peak routing table percentage filled are at fbDepth=5 and 30 with approximately 37% filled.

For all topologies, approximately 96 to 99% of packets dropped were caused by queue full. The other causes for packet drop include number of hops exceeded, invalid data and other types of packet errors. The average queue size remained close to 50% for all topologies except random. Table 4.4 summarizes the average breakdown for each packet type dropped due to queue full across all feedback depths. There is not much variation between the different topologies.

Table 4.4: Average of each packet type dropped across all feedback depths based on topology.

Topology	Data	Feedback	RT_UP	Broadcast
cocomac	37.93	5.57	0.17	56.46
lattice	36.29	4.19	0.21	59.51
sparse	39.02	5.85	0.21	55.53
cocomac (trans)	39.29	6.33	0.21	54.38
lattice (trans)	36.30	4.27	0.18	59.38
sparse (trans)	35.96	5.60	0.18	58.44
cocomac (TT disabled)	83.20	16.75	0.48	0
random	10.45	6.87	0.05	83.17

For all topologies, the percentage of TT routing table filled seems to have little or no effect or on the probability routing table. Since TT routing is only used when the next hop's queue is less than 70% congested, it is possible the Bernoulli trial is being run frequently to determine whether to use an alternate routing method such as random routing. The average queue size is between 50-60% for all topologies. The packet drop rate for most topologies with the exception of random and cocomac (TT disabled) is around 50-60%.

4.4 Fill Time Breakdown

This section displays the time taken for the routing table to be filled to 10%, 20%, etc... Feedback depth levels at 25 or 30 provide the lowest fill times. Normalization was performed since not all members reached each fill percentage. For example out of 242 neurons, only 235 may have 10% of their routing table filled. For the subsequent fill level of 20%, only 190 neurons may have reached this level. For all figures in this section, feedback depth of 1 was excluded since the numbers were too small to be accurate.

(a)

(b)

Figure 4.19: Normalized routing table fill time for (a) cocomac, (b) cocomac (TT enabled/disabled).

Figure 4.19 compares the normalized time taken to fill the routing table to 40%. The result for feedback depth of 1 was insignificant and thus does not appear on the graph. In Figure 4.19 (a) fastest fill times to 40% are fbDepth=15 (2017 min.) and 30 (3121 min.). In (b) there is a negative impact with TT enabled. There is a 17-45% reduction in fill times after disabling TT.

(a)

(b)

Figure 4.20: Normalized fill time for (a) cocomac (transpose), (b) sparse topologies.

Figure 4.20 (a) cocomac transpose has quickest fill time to 40% at fbDepth=30 (1008

min.) closely followed by fbDepth=20 (1012 min.). In figure 4.20 (b) reaches 30% at fdDepth=15 (585 min.), closely followed by fbDepth=25 (557 min.)

(a)

(b)

Figure 4.21: Normalized fill time for (a) sparse (transpose), (b) lattice topologies.

In Figure 4.21 (a), at 30%, fdDepth=5 (562 min.) is closely followed by fbDepth=30

(574 min.). With 4.21(b), fbDepth=25 provides the best fill time of 1048 minutes. Increasing the feedback depth to 40 marginally increases the fill time to 1093 minutes.

In Figure 4.22 (a) fbDepth=40 provides the quickest fill time at 1121 minutes.

Figure 4.22: Normalized fill time for (a) lattice (transpose), (b) random topologies.

4.5 Congested Nodes

Figure 4.23: Cocomac: Comparing rejection ratio, ratio of packets received and cluster coefficient.

Figure 4.24: Cocomac: Comparing rejection ratio, queue size ratio and compression ratio.

Figure 4.25: Lattice: Comparing rejection ratio, ratio of packets received and cluster coefficient.

Figure 4.26: Lattice: Comparing rejection ratio, queue size ratio and compression ratio.

Figure 4.27: Sparse: Comparing rejection ratio, ratio of packets received and cluster coefficient.

Figure 4.28: Sparse: Comparing rejection ratio, queue size ratio and compression ratio.

Figure 4.29: Cocomac (transpose): Comparing rejection ratio, ratio of packets received and cluster coefficient.

Figure 4.30: Cocomac (transpose): Comparing rejection ratio, queue size ratio and compression ratio.

Figure 4.31: Lattice (transpose): Comparing rejection ratio, ratio of packets received and cluster coefficient.

Figure 4.32: Lattice (transpose): Comparing rejection ratio, queue size ratio and compression ratio.

Figure 4.33: Sparse (transpose): Comparing rejection ratio, ratio of packets received and cluster coefficient.

Figure 4.34: Sparse (transpose): Comparing rejection ratio, queue size ratio and compression ratio.

Table 4.5: Comparison of average properties between neurons with high rejection ratio against the same topology with the high rejection neurons excluded.

Topology	Rej. Ratio	Queue Ratio	Packet Recv. Ratio	Transit Time	Global Cluster Coeff.	Comp. Ratio
High Rej. cocomac	1.58	0.95	0.20	15.81	2.18	1.39
cocomac	0.20	0.39	0.16	23.79	3.62	1.49
High Rej. sparse	1.58	0.93	0.19	12.99	2.19	1.39
sparse	0.20	0.39	0.17	23.02	3.49	1.49
High Rej. lattice	1.63	0.94	0.16	11.94	2.91	2.05
lattice	0.18	0.39	0.15	16.35	6.61	1.40
High Rej. sparse (trans)	1.69	0.94	0.22	18.94	2.13	1.44
sparse (trans)	0.20	0.42	0.20	27.78	3.58	1.70
High Rej. cocomac (trans)	1.71	0.93	0.20	18.81	1.89	1.51
cocomac (trans)	0.20	0.42	0.20	30.28	3.36	1.68
High Rej. lattice (trans)	1.57	0.93	0.14	11.94	3.02	1.23
lattice (trans)	0.18	0.41	0.16	16.35	6.43	1.72

Table 4.5 compares the average metrics from the high rejection group against the low rejection group. Each topology is split into two groups, the first group with rejection rates equal or greater than 1, and the low rejection rate group with rejection rate less than 1. The last column is the compression ratio.

The cocomac and sparse topologies have almost identical properties for their high rejection rate neurons. The high rejection rate for sparse (transpose) topology is also close to the cocomac and sparse topologies in terms of queue ratio, packet receive ratio, cluster coefficient and compression ratio.

Examining the queue ratio and packet receive ratio, the queue of the high rejection neurons is almost double the average of the low rejection group. Since the queue is close to being full, the packet receive ratio is also lower than the low rejection group. The CC of the high rejection group is also lower than the low rejection group.

Since the CC measures how closely a neuron's neighbours are interconnected, a lower CC may reduce the throughput of packets. When a neuron is congested, it will likely

select a random link to route the packet through. With less links to select from, the chance of congestion increases.

4.6 Packet Latency

Figure 4.35: Comparison between topologies for packet ratio received by each neuron.

Figure 4.36: Comparison between transpose topologies for packet ratio received by each neuron.

Figure 4.35 and 4.36 compare the ratio of packets received by each neuron. There is a strong correlation between the cocomac and sparse topologies and their transpose versions.

Figure 4.37: Comparison between topologies for average packet transit time to each neuron.

Figure 4.38: Comparison between transpose topologies for average packet transit time to each neuron.

Chapter 5

Analysis and Discussion

In Section 4.1 Topologies, the Rich Club Coefficient (RCC) for all topologies was higher throughout a range of k than the random topology. However, they were close but did not exceed 1. A $RCC > 1$ is an indication of a rich club [63].

The cluster coefficient similarity between the cocomac and sparse data sets confirms Bassett and Bullmore's findings, based on strong experimental evidence in which the brain is sparsely connected with a high clustering coefficient (CC) [17]. The lattice topology had a consistently higher CC throughout most of the neurons due to its inherent structure. The random topology had the lowest CC. The transpose topologies also exhibited similar behaviour.

In Section 4.2 Fill Summary, amongst all data sets taking into consideration fill time, fill to drop ratio and packet drop ratio, the cocomac data set with TT routing disabled performed the best. This indicates TT routing used to support unidirectional feedback routing has a significant negative impact on performance due to network congestion. Disabling TT routing improved performance by 15.2% against the best score for the cocomac data set. However the trade off is a significantly slower mean transit time which is 49.6% slower than the cocomac topology. The packet drop breakdown data from Figure 4.11 (a) confirms a 33% reduction in packet drop when TT routing was disabled. The packet breakdown figures from Section 4.3 also confirm a majority of packets (50-60%) dropped due to queue full were BCAST (broadcast) packets used to support TT routing.

In Section 4.3 Routing Tables, there is very little effect between the size of the TO table used to route feedback and the size of the probability routing table for cocomac

and sparse topologies. This could mean most of the feedback packets are being routed randomly instead of using TT routing due to congestion. The TO tables have no effect on the lattice topology. For all topologies, feedback depths from 3 to 5 provided a routing table fill percentage close to the peak provided at higher feedback depths. This confirms findings from Ramasubramanian and Mosse [51] where reverse back channels from 2 to 3 hops have good connectivity. Increasing the feedback depth greater than 5 provided marginal benefits.

For all topologies except random and lattice transposed, there was a correlation between the overall packets dropped and the percentage of routing table filled. If the packet drop percentage increased, then a higher percentage of routing table is filled. This is likely because more feedback packets are needed to be sent to increase the routing table fill percentage.

From Table 5.1, one advantage cocomac has over the transpose version is the mean transit time is approximately 22% faster. It is possible evolution chose the cocomac topology over transpose cocomac topology because signals can be sent faster and thus the monkey can react quicker. The global clustering coefficient (GCC) for cocomac is 8.6% higher than the transpose version.

When comparing cocomac against lattice, the lattice has a significantly reduced mean transit time at 23% less. The mean number of hops each packet took to reach its destination is 4.7-6.6% less. However, its overall score (Time / Fill:Drop) is the worst score with a low fill to drop ratio and high fill time much worse (refer to Section 4.2 Fill Time Summary, Table 4.2) than cocomac. The lattice's high GCC likely helps with lower transit times and mean number of hops traveled since each node is more closely interconnected.

The cocomac (transpose) provides the best overall score, with the exception of the random and cocomac with TT routing disabled. However the trade off is a much higher mean transit time 27.7% slower than cocomac.

It is likely evolution did not choose the lattice topology because the significantly lower mean packet transit times were not worth the trade off for having a low routing table fill to packet drop ratio. Evolution did not choose the transpose cocomac data set due to the longer mean transit time.

The sparse topology comes closest to cocomac because its receive ratio is equal and mean transit time is 4.7% faster than cocomac. The sparse topology's overall score is 28.3% better. For all metrics in Table 5.1, with the exception of the global cluster

coefficient and mean hops, the sparse topology provides equal or better performance. The real question becomes, why did nature pick cocomac over the sparse topology instead of vice versa? It is possible cocomac’s higher global clustering coefficient permits easier re-routing of packets if a neuron is unavailable due to congestion.

I hypothesize there are specialized workloads the cocomac topology handles better than the sparse topology that have not been captured by the current metrics. Possible workloads could include intermittent burst traffic and dynamic network topology changes due to congestion.

Table 5.1: Comparison of metrics for each topology’s optimal feedback depth from Table 4.2. Best value with the exception of random topology and cocomac with TT disabled, for each column highlighted in bold.

Data set	Receive Ratio	Mean Transit Time (seconds)	Global Cluster Coeff.	Mean Hops	Time/ Fill:Drop
cocomac	0.17	22.38	3.30	4.11	915.5
cocomac (trans)	0.20	28.58	3.04	4.03	691.9
cocomac (TT disabled)	0.18	33.48	3.30	4.11	399.3
lattice	0.15	15.81	5.43	3.84	2006.6
lattice (trans)	0.15	16.89	5.48	3.84	1414.9
sparse	0.17	21.32	3.04	4.06	888.0
sparse (trans)	0.20	26.33	3.08	4.09	913.2
random	0.15	21.43	1.02	4.41	0.94

In Section 4.5 Congested Nodes, Table 4.5, neurons with an overall rejection rate equal or greater than 1.0 were examined. They all have a higher queue ratio and packet receive ratio compared to the low rejection group which excludes the high rejection neurons. These neurons are queue bottle-necked because their queue ratio is more than double of the comparison group and close to being fully utilized. The queue bottle-neck and high rejection ratio are symptoms of a neuron unable to handle its packet throughput. For all topologies, the higher cluster coefficient correlates to a higher mean transit time. The

high CC may support the theory the neuron may be overwhelmed by traffic. It indicates the neuron is more highly interconnected and may receive more traffic than the average neuron. However each high rejection neuron has a compression ratio smaller than the low rejection group. Compression ratio measures the ratio of in degrees to out degrees of a node. The high rejection group should have a higher compression ratio to accept more incoming traffic, unless most of the traffic is coming in from only a few nodes. Thus it is unclear whether this theory pans out.

The same pattern of low transit time versus higher packet receive ratio for high rejection lattice compared to high rejection cocomac and sparse topologies appears again in Table 4.5. The high rejection transpose versions of cocomac and sparse result in higher transit times with higher packet receive ratios with the exception of lattice, when compared to their non transpose high rejection group.

Chapter 6

Conclusion and Future Work

The purpose of this thesis was to investigate the impact of various routing parameters on the performance of a small network that mimics the connections in a biological brain - assuming the biological brains can be modelled with a communications network. The brain simulator was implemented using a top-down model and programmed with the newest version of the C++ standard to leverage its native support for multi-threading and data structures.

The results indicate the cocomac data set is similar to the sparse data set. The ideal feedback depth to propagate feedback is 25 or 30 for all data sets when taking into consideration packet drop ratio, routing table fill time and percentage of routing table filled. If performance was based on routing table fill percentage alone, then the ideal feedback depth is from 3 to 5. Varying the feedback depth or data set did not have much effect on the ratio of the types of packets dropped. For all data sets and feedback depths, no more than 3.62% of packets dropped were caused by issues other than the queue being full. Disabling TT routing with the cocomac topology provided the best overall score (17-45% lower routing table fill time, 22% lower average queue size, 33% lower packet drop) outside of the random data set. It indicates TT routing negatively impacts performance.

The cocomac topology is better than lattice topology and equal to sparse in terms of packet receive ratio. The sparse topology is equal or better than the cocomac topology for all metrics in Table 5.1 except for a lower global clustering coefficient (GCC). It is possible the higher GCC allows for packets to be rerouted if a neuron is unavailable. We

can see why evolution chose the macaque brain topology over the lattice due to its much better signal reliability, but the decision is not as clear cut when choosing cocomac over the sparse topology.

In the future, additional investigation can be performed to determine why evolution chose the cocomac over the sparse topology. There may be special workloads such as intermittent burst traffic or dynamic network changes which are better handled by cocomac than sparse topology.

A decay function can be added to decrease the probability routing entry when a neuron is temporarily removed or congested. Various simulation parameters such as queue size, probability of success/failure for Bernoulli trials, congestion delay, congestion threshold, number of entries in routing cache, etc... can be modified to observe the impact on routing table metrics.

Graphics Processing Units (GPU) can be used to accelerate the simulation since the workload is mostly parallel. Another form of routing can be substituted for TT routing to determine its effect on performance.

References

- [1] Brian Spiking neural network simulator. <http://briansimulator.org>.
- [2] C++0x std::list::size complexity. http://gcc.gnu.org/bugzilla/show_bug.cgi?id=49561.
- [3] C++11 Overview. <https://isocpp.org/wiki/faq/cpp11>.
- [4] Dynamic array limit in message. http://www.flame.ac.uk/docs/user_manual.html.
- [5] *Encyclopaedia Britannica*. <http://www.britannica.com/EBchecked/topic/46342/axon>.
- [6] Flexible Large-scale Agent Modelling Environment. <http://www.flame.ac.uk>.
- [7] Simbrain. <http://www.simbrain.net>.
- [8] The Nengo Neural Simulator. <http://www.nengo.ca>.
- [9] Valgrind. <http://valgrind.org>.
- [10] Working Draft, Standard for Programming Language C++ (N3337). <http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3337.pdf>.
- [11] *The Human Body*. Arch Cape Press, 1989.
- [12] ANANTHANARAYANAN, R., ESSER, S., SIMON, H., AND MODHA, D. The Cat is Out of the Bag: Cortical Simulations with 10^9 Neurons, 10^{13} Synapses. In *High Performance Computing Networking, Storage and Analysis, Proceedings of the Conference on* (2009), pp. 1–12.
- [13] ATHANASOPOULOU, E., BUI, L., JI, T., SRIKANT, R., AND STOLYAR, A. Back-Pressure-Based Packet-by-Packet Adaptive Routing in Communication Networks. *Networking, IEEE/ACM Transactions on* 21, 1 (February 2013), 244–257.

- [14] BAKKER, R., WACHTLER, T., AND DIESMANN, M. CoCoMac 2.0 and the future of tract-tracing databases. *Frontiers in Neuroinformatics* 6, 30 (2012), 1–6.
- [15] BAO, L., AND GARCIA-LUNA-ACEVES, J. Link-state routing in networks with unidirectional links. In *Proceedings. Eight International Conference on Computer Communications and Networks, 1999.* (1999), pp. 358–363.
- [16] BARONCHELLI, A., I CANCHO, R. F., PASTOR-SATORRAS, R., CHATER, N., AND CHRISTIANSEN, M. H. Networks in cognitive science. *Trends in Cognitive Sciences* 17, 7 (2013), 348–360.
- [17] BASSETT, D. S., AND BULLMORE, E. Small-World Brain Networks. *The Neuroscientist* 12, 6 (December 2006), 512–523.
- [18] BASSETT, D. S., AND GAZZANIGA, M. S. Understanding complexity in the human brain. *Trends in Cognitive Sciences* 15, 5 (2011), 200–209.
- [19] BHORKAR, A. A., NAGHSHVAR, M., JAVIDI, T., AND RAO, B. D. Adaptive Opportunistic Routing for Wireless Ad Hoc Networks. *IEEE/ACM Trans. Netw.* 20, 1 (February 2012), 243–256.
- [20] BLOOM, B. H. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications Of The ACM* 13, 7 (1970), 422–426.
- [21] CHEN, G., LAU, F. C. M., DU, P., AND XIE, L. Distance-vector routing protocols for networks with unidirectional link. In *International Conference on Parallel Processing Workshops 2001.* (2001), pp. 473–478.
- [22] COLIZZA, V., FLAMMINI, A., SERRANO, M. A., AND VESPIGNANI, A. Detecting rich-club ordering in complex networks. *Nature Physics* 2, 2 (February 2006), 110–115.
- [23] CUI, Y., HU, W., TARKOMA, S., AND YLA-JAASKI, A. Probabilistic routing for multiple flows in wireless multi-hop networks. In *Local Computer Networks, 2009. LCN 2009. IEEE 34th Conference on* (October 2009), pp. 261–264.
- [24] FORUZAN, B. A. *Data Communications and Networking, 2nd Edition.* 2001.

- [25] GARCIA-LUNA-ACEVES, J., AND BEHRENS, J. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications* 13, 8 (1995), 1383–1395.
- [26] GELENBE, E., AND FOURNEAU, J.-M. Random Neural Networks with Multiple Classes of Signals. *Neural Computation* 11 (May 1999), 953–963.
- [27] GERLA, M., LEE, Y.-Z., PARK, J.-S., AND YI, Y. On Demand Multicast Routing with Unidirectional Links. In *IEEE Wireless Communications and Networking Conference 2005* (2005), vol. 4, pp. 2162–2167.
- [28] GRAHAM, D., AND ROCKMORE, D. The Packet Switching Brain. *Journal of Cognitive Neuroscience* 23, 2, 267–276.
- [29] GRAHAM, D. J. Routing in the brain. *Frontiers in Computational Neuroscience* 8, 44 (2014).
- [30] GUPTA, S. An Adaptive and Efficient Data Delivery Scheme for DFT-MSNs (Delay and Disruption Tolerant Mobile Sensor Networks). In *2012 International Conference on Advances in Engineering, Science and Management (ICAESM)* (March 2012), pp. 99–104.
- [31] HARRIGER, L., HEUVEL, P. v., AND SPORNS, O. Rich Club Organization of Macaque Cerebral Cortex and Its Role in Network Communication. *PLoS One* 7, 9 (September 2012).
- [32] HIGAKI, H. LBSR: Routing Protocol for MANETs with Unidirectional Links. In *3rd IEEE International Conference on Wireless and Mobile Computing, Networking and Communications - WiMoB 2007* (October 2007), pp. 84–84.
- [33] HILGETAG, C. C., BURNS, G., O’NEIL, M., SCANNELL, J., AND YOUNG, M. Anatomical connectivity defines the organization of clusters of cortical areas in the macaque monkey and the cat. *Philosophical Transactions of the Royal Society Biological Sciences* 355, 1393 (2000), 91–110.
- [34] HILGETAG, C. C., AND KAISER, M. Clustered organization of cortical connectivity. *Neuroinformatics* 2, 3 (2004), 353–360.

- [35] HUHTONEN, A. Comparing AODV and OLSR Routing Protocols. In *Seminar on Internetworking, Sjkulla* (2004), pp. 26–27.
- [36] HUMPHRIES, M., GURNEY, K., AND PRESCOTT, T. The brainstem reticular formation is a small-world, not scale-free, network. *Proceedings of the Royal Society Biological Sciences* 273 (2005), 503–511.
- [37] KAISER, M. A tutorial in connectome analysis: Topological and spatial features of brain networks. *Neuroimage* 57, 3 (August 2011), 892–907.
- [38] KUMAR, A., XU, J., AND ZEGURA, E. Efficient and scalable query routing for unstructured peer-to-peer networks. In *IEEE Infocom 2005: The Conference on Computer Communications, Proceedings* (2005), vol. 1-4, pp. 1162–1173.
- [39] KUOSMANEN, P. Classification of Ad Hoc Routing Protocols. *Finnish Defence Forces, Naval Academy, Finland*. (2002).
- [40] LEON, P. S., KNOCK, S. A., WOODMAN, M. M., DOMIDE, L., MERSMANN, J., MCINTOSH, A. R., AND JIRSA, V. The Virtual Brain: a simulator of primate brain network dynamics. *Frontiers in Neuroinformatics* 7 (June 2013), 1–23.
- [41] LIU, G., DENG, Q., AND WANG, H. Loop-based clustering routing (LCBR) in wireless ad hoc networks with unidirectional links. In *Wireless Communications Networking and Mobile Computing (WiCOM), 2010 6th International Conference on* (2010), pp. 1–5.
- [42] MAQBOOL, B. B., AND PEER, M. Classification of Current Routing Protocols for Ad Hoc Networks - A Review. *International Journal of Computer Applications* 7, 8 (October 2010), 26–32.
- [43] MCAULEY, J. J., DA FONTOURA COSTA, L., AND CAETANO, T. Rich-club phenomenon across complex network hierarchies. *Applied Physics Letters* 91, 8 (2007).
- [44] MISIC, B., SPORNS, O., AND MCINTOSH, A. R. Communication Efficiency and Congestion of Signal Traffic in Large-Scale Brain Networks. *PLoS Computational Biology* 10, 1 (January 2014), 1–8.

- [45] MORINO, H., MIYOSHI, T., AND OGAWA, M. Unidirectional Ad Hoc Routing Protocol with Area-controlled Flooding Using Overheard Neighbor Node Information. In *Eighth International Symposium on Autonomous Decentralized Systems, Proceedings* (2007), pp. 519–525.
- [46] PEARLMAN, M., HAAS, Z., AND MANVELL, B. Using Multi-Hop Acknowledgements to Discover and Reliably Communicate over Unidirectional Links in Ad Hoc Networks. In *Wireless Communications and Networking Conference, 2000. WCNC. 2000 IEEE* (2000), vol. 2, pp. 532–537.
- [47] PIAO, J. J., AND CHANG, T. M. TPSR: Transmission Power Based Source Routing Protocol for Mobile Ad Hoc Networks with Unidirectional Links. In *2010 6th International Conference on Wireless Communications Networking and Mobile Computing (WiCOM)* (September 2010), pp. 1–4.
- [48] PICCININI, G., AND BAHAR, S. Neural Computation and the Computational Theory of Cognition . *Cognitive Science* 37 (April 2013), 453–488.
- [49] PRAKASH, R., AND SINGHAL, M. Impact of unidirectional links in wireless ad-hoc networks. *Proceedings of the DIMACS Workshop on Mobile Networking and Computing* (1999), 272–281.
- [50] PRODHAN, M. A. T., DAS, R., KABIR, M., AND SHOJA, G. Probabilistic quota based adaptive routing in Opportunistic Networks. In *Communications, Computers and Signal Processing (PacRim), 2011 IEEE Pacific Rim Conference on* (August 2011), pp. 149–154.
- [51] RAMASUBRAMANIAN, V., AND MOSSE, D. Statistical Analysis of Connectivity in Unidirectional Ad Hoc Networks. In *Proceedings. International Conference on Parallel Processing Workshops, 2002.* (2002), pp. 109–115.
- [52] RHEA, S. C., AND KUBIATOWICZ, J. Probabilistic location and routing. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings.* (2002), vol. 3, pp. 1248–1257.

-
- [53] SANDALIDIS, H. G., MAVROMOUSTAKIS, C. X., AND STAVROULAKIS, P. Ant based probabilistic routing with pheromone and antipheromone mechanisms. *International Journal of Communication Systems* 17, 1 (2004), 55–62.
- [54] SCHOONDERWOERD, R., BRUTEN, J., HOLLAND, O., AND ROTHKRANTZ, L. Ant-based load balancing in telecommunications networks. *Adaptive Behavior* 5, 2 (1996), 169–207.
- [55] SHRESTHA, D., AND KO, Y.-B. A New Routing Protocol in Ad Hoc Networks with Unidirectional Links. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 3741 LNCS (2005), 287–292.
- [56] SPORNS, O., CHIALVO, D. R., KAISER, M., AND HILGETAG, C. C. Organization, development and function of complex brain networks. *Trends in Cognitive Science* 8 (September 2004), 418–425.
- [57] SPORNS, O., AND KOTTER, R. Motifs in Brain Networks. *PLoS Biology* 2 (November 2004), 1910–1918.
- [58] SPORNS, O., TONONI, G., AND EDELMAN, G. Theoretical neuroanatomy: Relating anatomical and functional connectivity in graphs and cortical connection matrices. *Cerebral Cortex* 10, 2 (February 2000), 127–141.
- [59] SPORNS, O., TONONI, G., AND KOTTER, R. The Human Connectome: A Structural Description of the Human Brain. *PLoS Computational Biology* 1 (September 2005), 245–251.
- [60] STAM, C. J., AND REIJNEVELD, J. C. Graph theoretical analysis of complex networks in the brain. *Nonlinear Biomedical Physics* 1 (July 2007), 1–19.
- [61] SU, Y.-Y., HWANG, S.-F., AND DOW, C.-R. An Efficient Cluster-Based Routing Algorithm in Ad Hoc Networks with Unidirectional Links. *Journal of Information Science and Engineering* 24, 5 (2008).
- [62] TAN, S. K., AND MUNRO, A. Adaptive Probabilistic Epidemic Protocol for Wireless Sensor Networks in an Urban Environment. *16th International Conference on Computer Communications and Networks* (August 2007), 1105–1110.

-
- [63] VAN DEN HEUVEL, M. P., AND SPORNS, O. Rich-Club Organization of the Human Connectome. *The Journal of Neuroscience* 31, 44 (2011), 15775–15786.
- [64] XIE, H., QIU, L., YANG, Y., AND ZHANG, Y. On self adaptive routing in dynamic environments - an evaluation and design using a simple, probabilistic scheme. In *Proceedings of the 12th IEEE International Conference on Network Protocols 2004* (October 2004), pp. 12–23.
- [65] ZHOU, S., AND MONDRAGON, R. The rich-club phenomenon in the internet topology. *IEEE Communications Letters* 8, 3 (2004), 180–182.
- [66] ZIANE, S., AND MELLOUK, A. Inductive Routing Based on Dynamic End-to-End Delay for Mobile Networks. In *IEEE Global Telecommunications Conference (GLOBECOM 2010)* (December 2010), pp. 1–5.