AN AGENT BASED MODELING APPROACH FOR THE EXPLORATION OF SELF-ORGANIZING NEURAL NETWORKS

A Thesis

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Master of Science

by

Timothy W. Schoenharl, B.S.

Greg Madey, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
April 2005

AN AGENT BASED MODELING APPROACH FOR THE EXPLORATION OF SELF-ORGANIZING NEURAL NETWORKS

Abstract

by

Timothy W. Schoenharl

In this thesis we present the ABNNSim toolkit for the simulation of biologically inspired neural networks. This work applies the Agent Based Modeling paradigm to the simulation of biological neural networks, allowing rapid development of models, easy addition of features and a richness of expression that is not available with other tools. The focus of the ABNNSim toolkit is modeling neural networks at the network level. This view leads to some loss of fidelity over other computational neuroscience tools, but allows larger, system-level phenomena to be explored. This is motivated by discoveries in the area of complex networks, in their analysis and prevalence in biological systems. These networks have characteristics that are desirable for biological systems: They are resilient in the face of failure, efficient (both in terms of communication cost and speed of propagation) and can be constructed using simple local rules. This thesis describes several methods of developing complex network topologies in neural networks using pruning. Throughout the project the Agent-Based Modeling paradigm has been valuable as a vital tool for speeding the development and deployment of simulations of this type.

DEDICATION

For Ping, who continually threatened to beat me with her rolling pin if I didn't get this finished.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Greg Madey, for his guidance and support of this project.

Thanks to John Korecki for his work on software development and testing and maintenance of the Sourceforge project site.

This work was supported by a Schmitt Foundation Fellowship.

CONTENTS

FIGURES

TABLES

CHAPTER 1

INTRODUCTION

1.1 Overview

This thesis presents the ABNNSim framework, a toolkit for computational neuroscientists to explore developmental changes in biological neural networks. ABNNSim fills a niche in the computational neuroscience realm. It lies at a level of abstraction above Neural Modeling toolkits such as NEURON [32] and GENESIS[72] but below that of Topographica[6]. ABNNSim also brings the concepts and analysis tools from Complex Network Analysis to evaluate developing network topologies. Finally, ABNNSim uses the Agent-Based Modeling approach, allowing researchers to explore neural network development by starting with models of neurons. This work has been presented several times throughout its development at academic conferences[59][60][58][61].

1.2 Complex Systems

The work in this thesis is best understood in the context of complex systems. In the words of Alfred Hubler[36]:

A Complex System is a System with:

- Large Throughput of Energy, Information, Force, etc, through a well designed boundary

- Many Parts That Form Emergent Structures (e.g. Fractals, Chaos, Neural Networks, Genetic Algorithms, Cellular Automata)

Complex systems display interesting interactions between the elements of the system, be they molecules in a flow-system that interact to form turbulence, or plants and animals in a food web.

Complex systems are often described as existing on the boundary between order and chaos. They are clearly not ordered, but there is a discernible structure to them. Complex systems are important to this discussion because many interesting systems, for example knowledge networks in sociology, food webs in ecology, protein interaction networks in cell biology and neural networks in neurobiology, can be best described and understood as complex systems.

Two of the best tools available for understanding complex systems are simulation techniques using the Agent-Based Modeling approach and methods of network analysis. Agent-Based Modeling simulations allow researchers to study the effects of complex interactions. Complex network analysis, by abstracting away the players in a system and instead viewing them as nodes in a network, allows researchers to analyze the large scale structure of complex systems. This approach enables researchers to view the effects of interaction, characterize important elements and to describe the overall structure of the system.

1.2.1 Complex Networks

For centuries mathematicians have used graphs to represent networks from the real world. Euler was likely the first to do this when he formulated his solution to the "Bridges of Königsberg" problem [73]. Regular graphs (where each vertex has an identical degree) have been used to represent idealized or artificial systems.

In the last century, mathematicians worked extensively with random graphs[8]. These graphs are constructed in an iterative process whereby each node in the network is visited and is given a probability to connect to a randomly chosen node in the network. Random graphs were compelling in that they offered an interesting approximation to the networks found in the real world. However, in the 1990s it became apparent that these graphs did not characterize many interesting real-world networks. Despite their random appearance, networks like the World Wide Web and the actor collaboration network exhibit some underlying nonrandom structure.

Two groups of researchers working on the analysis of complex networks independently discovered several important characteristics for classifying networks and provided methods of generating networks with these characteristics. Watts and Strogatz, at Cornell University, found that complex networks can be characterized by two topological measurements: clustering coefficient and characteristic path length[71]. Barabasi, Albert, et al at the University of Notre Dame discovered that networks like the world wide web are characterized by a "scale-free" distribution of links. This work has had an impact in nearly every area of science, from sociology to molecular biology[41].

Watts and Strogatz, in [71], analyze the actors collaboration network, the neural network of the nematode *C. elegans* and the power grid of the western United States. Their research demonstrates that not only do these diverse networks display similar topological characteristics, but they also are significantly different than random graphs of the same size. Watts and Strogatz introduce two measurements, characteristic path length and clustering coefficient, that they use to characterize graphs. The characteristic path length is the "longest shortest path" between two nodes in the network[70]. The clustering coefficient describes quantitatively the degree to which a node's neighbors are interconnected. Averaging the clustering coefficient over all nodes in the network yields the average clustering coefficient. Networks with a high clustering coefficient and low characteristic path length are referred to as "Small World" networks[71], a term borrowed from Stanley Milgram's sociological studies conducted in the 1960s[51].

Watts and Strogatz present a method for constructing "Small World" networks in [71]. The method involves starting with a regular lattice, then iteratively deforming it by randomly rewiring edges. This method, although it produces the correct result, most likely does not adequately reflect the way in which "Small World" networks form in the real world.

Barabasi et al began studying the topology of the internet. Their research showed that the distribution of links follows a power law[2]. This was surprising as researchers expected that the links would be distributed according to a Poisson distribution, as in random graphs[8]. This discovery lead Barabasi and his collaborators to propose and refine a mechanism for forming representative networks.

1.3 Neural Networks

The computational power of the brain is not due solely to the computational power of a single neuron, but the coherent connection of billions. Accurate models of a brain must begin with an accurate model of an individual neuron, however, it does not suffice to merely wire them together in an arbitrary fashion.

1.3.1 Biological Neural Networks

Initial research into biological neural networks is attributed to William James at the turn of the century [39]. Throughout the latter half of the 20th century there has been an interesting interplay between biological and artificial neural network research.

The main focus of this work is on topology, so I will focus on relevant research, however, the interested reader is directed to canonical literature on the subject to learn more[3][30]. Historically research efforts focused on either the more accurate modeling of individual neurons or the description of large scale regions of the brain, mainly due to the technological limitations of research equipment. In recent years, technological advances such as high resolution Functional MRI and more accurate lab techniques have enabled researchers to study neural networks in greater detail.

It has been argued by Segev in [63] that human DNA lacks the expressive capability to contain an exact map of the brain. It is clear that on some level the brain must self organize. According to Segev [63] although the large-scale (above 1 cm) the brain is largely deterministic, the small-scale (below 1mm) structure of the brain appears to be random. Thus, if we accept that there is some organization on the small scale and it is not defined by the DNA, then it must be the result of self-organization.

One proposed mechanism of self-organization is that of pruning. It has been documented that the mammalian brain undergoes massive pruning of axons, synapses and neurons, beginning before birth and continuing until puberty[14]. Approximately half of the neurons and one third of the synapses are pruned before adolescence. This pruning process is not random, but seems to act in a manner resembling Hebbian learning[14]. Hebbian learning can be summarized in the following manner: Given two neurons, A and B, where A has a connection to B, if neuron A fires and in doing so causes neuron B to fire, then the connection between A and B is strengthened. Chechik and Meilijson [17] suggest that *neuronal regulation*, a biological mechanism that maintains post-synaptic membrane potential, may play a part in the synaptic pruning process. Neuronal regulation complements the action of Hebbian learning, preventing a runaway strengthening of the connections affected by Hebbian learning.

1.3.2 Artificial Neural Networks

Artificial neural network research was inspired by discoveries in biological neural networks. Applications of artificial neural networks include classification, clustering, function approximation, financial modeling[64], pattern recognition[7] and nonlinear optimization[19]. Artificial neural networks are ideal for situations where the system must be able to generalize from a set of known data. The power and utility of ANNs has been affected by three factors:

1. The sophistication of the neuron model.
2. The speed and effectiveness of the learning rule.
3. The topology of the network.

Early work used the perceptron, a simplistic neuron model. Later research groups explored different threshold functions, namely the sigmoid, that were more conducive to error based learning methods. Neuron models developed along with learning rules. The first effective learning rule was Backpropagation, a method pioneered by the PDP Research group led by Rumelhart and McClellan[56]. Recently there has been an explosion of learning rules, many suited to very specific areas.

The methods of wiring together artificial neural networks are varied and depend heavily on the intended application. Neural networks are often given a standard input layer, hidden layer, output layer design, where each layer is fully wired to the following layer. These networks are referred to as Feed-forward networks, referring to the single direction of information flow[3].

1.4 Agent-Based Modeling and Simulation

Agent-Based Modeling is a paradigm where a simulation is built in a bottom-up manner by specifying the individual components of a system and their interactions. Aggregate behavior is observed as an emergent property of the interactions of the agents and their environment. An emergent property of a system is a property that arises from the interactions of entities in the system. ABM is often contrasted with traditional, equation based approaches that model systems in a top-down manner.

In his article "Agent-based modeling: Methods and techniques for simulating human systems"[9], Eric Bonabeau lists 5 criteria for situations that benefit from the use of Agent Based Modeling techniques:

- When interactions between the agents are complex, nonlinear, discontinuous or discrete.
- When space is crucial and the agents' positions are not fixed.
- When the population is heterogeneous, when each individual is (potentially) different.
- When the topology of interactions is heterogeneous and complex.

- When agents exhibit complex behavior, including learning and adaptation.

Clearly biological neural networks meet most, if not all, of these criteria. One focus of this work is on measuring changes in the topology of the network as a result of pruning. Due to the sheer size and range of behaviors to be simulated (physical, chemical, electrical), an equation based model is inadequate. Equation based models would have difficulty capturing the complex interactions and merging the various simulated behaviors, whereas Agent-Based Models are well-suited to this domain.

1.5 Research Goals

The goals of this research are the following:

- Develop a Neural Network research tool that allows computational neuroscientists to explore topological development.
- Create the tool so that it is extensible and powerful.
- Demostrate the extensibility of the tool through a series of case studies.
- Explore the types of network topologies that can be developed using local rules in a biologically-inspired neural network.
- Replicate similar work on the effects of local rules on network topology.
- Explore the effects of pruning in the development of global network topology.

In this Thesis, we synthesize the various research areas presented above into a coherent model for the purpose of examining the effects of pruning on global network topology and testing the flexibility and extensibility of a model built using concepts from Agent-Based Modeling.

CHAPTER 2

BACKGROUND

This work intends to synthesize knowledge from several disparate domains. The overarching goal of creating a new tool for studying the development of self-organizing neural networks necessitates drawing elements from the study of biological neural networks, current thinking in artificial neural networks, simulation techniques from Agent-Based Modeling and, in order to evaluate the research, tools from complex network analysis.

The organization of this Chapter is as follows: We begin with summarizing recent trends in complex network analysis. This is done first to introduce fundamental concepts and to justify the need for using complex network analysis tools in the evaluation of ABNNSim simulations. As will be shown shortly, several types of complex networks, specifically Small-World networks and Scale-Free networks, are often found in biological systems that are characterized by growth and self-organization. Second, we examine a selected set of work from neuroscience that is relevant to our discussion, namely work related to growth and development in biological neural networks. Third, we present historical trends and current work on artificial neural networks, including neuron simulators from the computational neuroscience field. Finally, we present the field of Agent-Based Modeling, with background and justification for its usefulness in the modeling of neural networks.

2.1 Complex Networks

Here we introduce the field of Complex Network Analysis, discussing the metrics, construction and prevalence in the real world. The field became popular in the mid-1990s, as computational power increased and appropriate datasets became more available. Complex Network Analysis has come to fundamentally change thinking on topics as diverse as the structure and formation of the world-wide web and the layout of biological neural networks.

2.1.1 Metrics

There are numerous approaches to the analysis of complex networks. These approaches can be divided into two main areas: classification based on degree distribution and classification based on local or global statistics. Methods of classification based on degree distribution usually create a histogram of the degree of the nodes in the network, then using techniques from statistics, identify this with a known distribution. Examples of distributions commonly used include the normal, log-normal, exponential and scale-free. Methods of classification based on network statistics involve computing some value based on a network property (shortest path, connectivity among a node's neighbors), and either comparing it to networks of a known type with the same number of nodes, or normalizing the value and comparing it to reference values.

2.1.1.1 Small Worlds

Watts and Strogatz present two metrics for describing complex networks. These are the clustering coefficient (C) and the characteristic path length (L)[71]. The clustering coefficient, in the qualitative sense, is a measure of how connected each node's neighbors are, averaged over all the nodes in the network. The characteristic path length is the average of every shortest path in the network.

The formula for calculating the clustering coefficient is given in 2.1. For each node i in the set of nodes N, there are n_i neighbors, connected to i by an edge. Given n_i neighbors, there are at most $\frac{n_i(n_i-1)}{2}$ edges between them. For each node in the network, count the number of edges among its neighbors and divide by the number of possible edges. Average the resulting values to find the clustering coefficient.

$$\frac{1}{N}\sum_{i=1}^{N}\frac{\sum_{j,k\in G_i}\epsilon_{j,k}}{\frac{n_i(n_i-1)}{2}} \text{ where } \epsilon_{j,k} = \left\{ \begin{array}{c} 1 \text{ for } e_{j,k} \in G_i \\ 0 \text{ for } e_{j,k} \notin G_i \end{array} \right\} \tag{2.1}$$

Calculating the characteristic path length of a graph is more involved. The formal definition of L is the median of the means of all shortest paths in the network. To calculate L, (taken from [70]) for each vertex $v \in G$, compute $d(v, j)$ $\forall j \in G$ where $d(v, j)$ is the distance between vertex v and vertex j. Next, compute $\bar{d}_v$ for each v. Finally, the characteristic path length L is the median of the set of all $\bar{d}_v$.

Both the characteristic path length and the clustering coefficient are normalized by dividing the computed value by the value for a similar size regular lattice. For a more detailed discussion of these measurements, consult [71].

Watts and Strogatz analyzed several complex networks and demonstrated that they could be differentiated from random graphs by their normalized L and C values. A random graph of the same size (both number of nodes and edges) will have a similar but lower L value and a significantly lower C value. The value of this cannot be overstated. It shows that there are demonstrable differences in the structure of complex networks and random graphs. Moreover it suggests the inadequacies of using random graphs to model complex networks.

Graphs with low L values and high C values are considered "Small Worlds" graphs.

2.1.1.2 Scale Free Networks

Barabasi et al took a different approach to analyzing complex networks. They drew from their experience measuring the topology of the internet and analyzed the link structure of the resulting graph. A histogram of the link structure, organized by frequency, displayed a power law distribution. This was surprising, given that a similar random graph would produce a Poisson distribution[2]. Again, random graphs, which had been used in simulations of the internet, were found to be inadequate for the representation of complex networks.

2.1.1.3 Others

In response to Watts and Strogatz, Latora and Marchiori created similar metrics based on the harmonic mean distance [48]. The global harmonic mean distance behaves like L, the characteristic path length, whereas the local harmonic mean distance behaves similarly to $\frac{1}{C}$. The authors felt that L and C, while useful for characterizing abstract graphs, could not handle graphs that were not connected. The global and local harmonic mean distances work correctly regardless of whether a graph is connected or not. They also will provide meaningful insight on metrical graphs, ie graphs that are embedded in a metric space [48]. Finally, the computation of the global and local measures is done in a similar manner, leading to a unified description of the network from local and global perspectives.

The formula to calculate the global harmonic mean distance is:

$$D_{global}(G) = \frac{N(N-1)}{\sum_{i,j \in G} \frac{1}{d_{i,j}}} \tag{2.2}$$

Similarly, the average local harmonic mean distance can be computed in the following manner:

$$D_{local}(G) = \frac{1}{N} \sum_{i=1}^{N} \frac{N(N-1)}{\sum_{j,k \in G_i} \frac{1}{d_{j,k}}} \tag{2.3}$$

In this case, G_i is the subgraph of nodes connected to i.

These values are normalized by dividing by $D_{global}(0)$ and $D_{local}(0)$ respectively, where 0 represents a regular lattice. In order to show that their measurement provided results consistent with those of Watts and Strogatz, Latora and Marchiori [48] produced a graph similar to Figure 2.1. This graph is itself a replication of the results of Watts and Strogatz, namely Figure 2 of their 1998 Nature paper[71]. In this graph, $\frac{D_{global}}{D_{global}(0)}$ and $\frac{D_{local}(0)}{D_{local}}$ are plotted against the rewiring probability of a circular graph. We plot $\frac{D_{local}(0)}{D_{local}}$ because the D_{local} metric roughly corresponds to $1/C$, as explained in [48]. It is noteworthy to mention that the x-axis of the graph is on a logarithmic scale between 0 and 1. For values of $p > 0$ the global harmonic mean distance rapidly drops, while the average local harmonic mean distance remains relatively stable. Graphs with high average local harmonic mean distance and low global harmonic mean distance are characterized as "Small Worlds" graphs, as discussed above.

2.1.2 Prevalence of Complex Networks in the Real World

Much of the research above was a result of the dissatisfaction with random graphs as a representation of the complex networks found in the real world. Re-

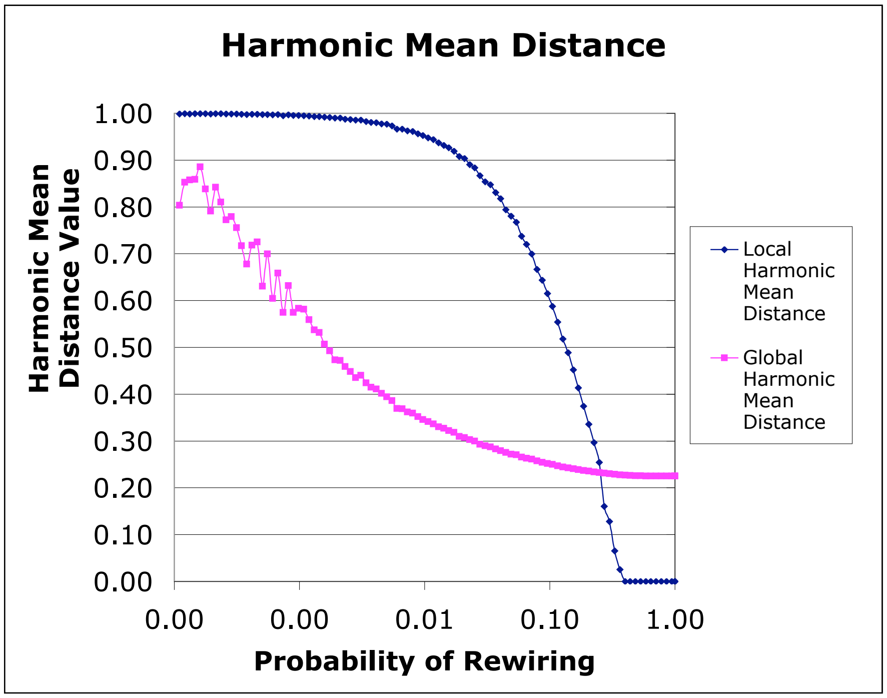


Figure 2.1. Global and Local Harmonic Mean Distance as a function of the rewiring probability.

searchers in fields spanning the sciences encounter systems that are most easily understood as networks. In this section we present in more detail some examples of these systems.

2.1.2.1 Constructs - Examples of Man-made Networks

The canonical example of a man-made network is the Internet. A graph of the Internet can be made by letting each system (computer, router, etc.) be a node in the network and each physical connection (fiber optic cable, Cat 5 cable, etc) be an edge. Similarly, the World-Wide Web is a related but distinct network. The attendant graph would represent an individual web page as a node and a hyperlink between pages as an edge.

Other man-made entities that can be represented as graphs are power grids and the network of airports and airline routes. Both of these examples have been analyzed by Barabasi[5] and Watts[70]. The power grid can be modeled intuitively as a graph. A graph of the airports and airline routes would represent airports as nodes and a route between airports as an edge.

2.1.2.2 Social Systems

There are several examples of complex networks in social systems. Watts and Strogatz analyzed the Actor's collaboration network[70], where each actor is a node in the network and edges represent the two actors working together on a film. A similar collaboration network is found in academic research, where two authors have a link between them if they wrote a paper together. Mathematicians have considered this type of network for years, specifically in order to determine a person's Erdős number. This value is the number of links in the collaboration network that it takes to go from a given person to Paul Erdős, an eminent Hungarian mathematician. Barabasi analyzed the Mathematicians collaboration network in [5].

Network analysis of social networks is prevalent in the context of sociology[62]. Friendship networks, advice networks and collaboration networks are examples of these. An interesting example of a collaboration network is the Open Source Software Developer's network[24][74]. Here nodes represent software developers, and two developers share an edge if they have worked on the same software project. (These networks can also be bipartite graphs, with developers in one group and projects in another. It can be shown that there is a transformation from the bipartite graph to a developer-only graph, however this is not reflexive.)

2.1.2.3 Biological Systems

Several biological systems can be understood in terms of complex networks. Barabasi has done some very interesting work analyzing the metabolic pathways in cells as a network[41]. At a much larger scale, food webs can be instructive in the complex predator-prey relationships in an ecosystem. In this case, each species is represented by a node, and an edge represents an "eats" or "eaten-by" relationship.

Perhaps one of the most interesting applications of network analysis to biology is the study of the connectivity of neurons in a neural network. Scientists have analyzed the connectivity of the neural network in the nematode *C. Elegans* [71][48]. Biologists have also begun looking at the connectivity of the mammalian brain with projects such as CoCoMac[43].

2.1.3 Algorithms for Network Construction

This section introduces some of the methods of creating complex networks.

2.1.3.1 Watts-Strogatz and Barabasi

The method presented by Watts and Strogatz creates a small-world graph by tuning a regular graph[71]. First, start with a graph that is a ring lattice, where each node connects to its k nearest neighbors. Then deform the graph in the following manner: Step through the graph, visiting each node. At each node, rewire an edge with probability p. The value of p determines the structure of the resulting graph. For $p = 0$, the graph is the unchanged lattice. For $p = 1$, the graph is essentially random. For values of $0 < p < 1$, the graph may resemble a small-world network. For small values of p, the characteristic path length drops quickly, while the clustering coefficient remains near its original value. Networks that exhibit a high clustering coefficient and low characteristic path length are considered to be Small Worlds networks. Figure 2.2 presents a graphical description of the relationship between the rewiring probability p, the Clustering Coefficient and the Characteristic Path Length.

The Watts-Strogatz method is easily understood and provides a useful computational tool, as it allows researchers to create graphs with the desired characteristics. However, the construction of these graphs is decidedly artificial. Clearly some other mechanism is responsible for their development in the natural world.

Barabasi in [5] provides a method for "growing" a scale-free network. He adapted his method from the one given by Erdős and Renyi for constructing random graphs. The original method works in the following way: Create N nodes, where N is the intended size of the finished network. For each pair of

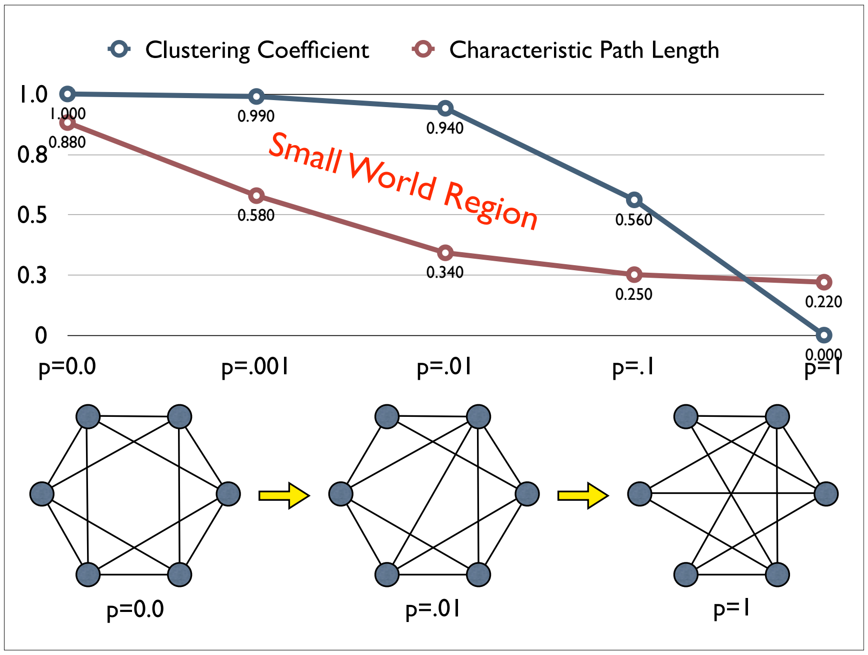


Figure 2.2. Graphical depiction of Relationship Between Rewiring Probability p, Clustering Coefficient and Characteristic Path Length

vertices, i, j add an edge with probability p. Barabasi first modified the random graph algorithm so that it added nodes iteratively. Each time a node was added, it created all of its edges immediately, in the same manner as the random graph. The new node went through the list of existing nodes, but instead of adding nodes with a fixed probability, the probability value varied. Each node was assigned a fitness value uniformly at the beginning of the algorithm. Every time the node is given a new incoming edge, its fitness value is increased. Thus, as a node gets more links, its fitness value increases, creating a positive feedback loop.

Barabasi found that this method, although producing scale-free networks, could not reproduce some of the behavior seen in evolving networks in the real-world. The problem was that in this method nodes added first had a much higher chance of becoming a hub than nodes added later in the algorithm. And contrary to that result, there were numerous cases in real-world networks where a node added much later to the network ended up with more links than older nodes.

Barabasi modified his method to account for this by adding a fitness value to each node. The rate at which nodes receive new links in a network is proportional to its fitness. The addition of fitness to the model accounts for such phenomena as the rise of Google in the world wide web [5].

2.1.3.2 Local Models

Cancho Sole et al [11] present a method for generating scale-free network topologies as a result of a network level optimization procedure. The authors introduce the notion of network density, defined as in Equation 2.5. Here ρ is related to $< k >$, the average degree of each of the n nodes. They present Equation 2.4 as the energy function for optimization, which is a linear combination of the density and normalized vertex distance. By rewiring a random network and varying λ, the optimization process yields graphs of varying degree distributions. As λ increases in the range $0 \leq \lambda \leq 0.25$, the resulting networks exhibit random, exponential and finally scale-free link distributions. As λ increases above 0.25, the resulting networks display a star topology.

$$E(\lambda) = \lambda d + (1 - \lambda)\rho \tag{2.4}$$

$$\rho = \frac{< k >}{n - 1} \tag{2.5}$$

2.2 Biological Neural Networks

Here we present selected topics from recent work in neuroscience. The focus here is on neurons and neural networks from a network development perspective.

2.2.1 Reductionist Approach: Neurons

Currently neuroscientists have highly sophisticated models of the behavior of neurons. These models simulate the behavior of the neuron by representing it as a series of compartments. A neuron can be modeled from thousands of compartments to just one, with each compartment governed by cable equations. Cable equations were initially derived for modeling the behavior of undersea telegraph cables, and their application to the modeling of neurons is straightforward. Each compartment represents a different physical component of the neuron, with appropriate cable equations related to the number of connections to the compartment and the leakage current through the cell membrane.

2.2.2 Networks

The nematode *C. Elegans* has been systematically researched since it was proposed as a model organism in the 1960s.The research relevant to our current discussion is the characterization of the neural network of *C.Elegans*. The network contains only 302 neurons (in the hermaphrodite form) and it is wired in a similar fashion across individuals. The data for the wiring of the *C. Elegans* neural network is available online [68]. As mentioned above, analysis of the structure of the neural network of *C. Elegans* has shown that it displays a Small-World topology.

Significant research has also explored the connectivity of the cortical regions of the macaque brain[43]. Researchers have compiled a database of all work on the macaque brain, and there exists a simulator that is intended to simulate regions of the macaque brain using the connectivity information as described in the database[12].

2.2.3 Network Formation: Possible Mechanisms

There have been several clues as to network formation that have emerged from Neurobiology. An interesting result is described in the work of Chalup [14]. Studies conducted on rhesus monkeys demonstrated a significant loss of neurons, axons and synapses between birth and adolescence. The major events in this process are: (Adapted from [14])

1. Rapid growth of the number of neurons before birth.
2. Rapid growth of the number of axons before birth.
3. Slow synaptogenesis which starts before birth and continues until puberty.

4. Very rapid axon loss starting at birth.

5. Slow apoptosis of neurons until puberty.

6. Slow synapse elimination starting in the middle of the critical phase (between birth and puberty) and continuing through the remaining life.

The process is characterized by a rapid overgrowth of the network, followed by pruning. Pruning is an important mechanism in neural development and can be seen as a way of reducing the size of the network while maintaining the overall computational ability of the network[3]. Neurons have high energy requirements, thus organisms will seek to limit the size of the neural network in order to reduce its energy consumption.

Chechik et al describe a local mechanism that is useful in topological development, called Neuronal Regulation[17]. Neuronal Regulation is a complementary mechanism to Hebbian learning.

2.3 Artificial Neural Networks

Artificial neural network research is a vast field that shares its roots with neuroscience, but has diverged significantly in recent decades. Lately there has been less work in creating biologically plausible models of neurons and neural networks and more focus on using neural networks as an engineering tool.

2.3.1 Historical Perspective

A detailed account of the history and development of the field of artificial neural networks is beyond the scope of this thesis. However, there are certain highlights that bear mentioning. The field owes much to the pioneering work of

D. O. Hebb, who formulated the first learning rule for neural networks and which now bears his name[31]. Also of importance is the work of McCulloch and Pitts, who, in 1943, described the behavior of a biological neuron in electrical terms and created the first artificial neurons[50].

Various neuron models are presented in Table 2.1. The table presents various characteristics of artificial neural network and neuron models. The most important of these are described in greater detail below.

The first neuron model of importance was the Perceptron, which is also referred to as a threshold neuron. The perceptron uses a response function to map its input to output. The response function for the Perceptron developed in concert with learning algorithms. The first response functions were simple all or nothing functions, however these could not be adapted with the tools of Calculus. This led to the use of the sigmoid function, a differentiable equation that was more suitable for optimization-based learning algorithms.

The Perceptron was used primarily in the context of feedforward networks, a type of network where neurons are arranged in layers. In each layer, neurons connect only to neurons in following layers, cycles are not permitted. The standard configuration for a feedforward network is 3 layers: an input layer, a hidden layer and an output layer.

The most recent advance in artificial neural networks is the development of the spiking neuron[34][46]. This model is significantly closer to what neuroscientists now understand about the behavior of biological neurons. In Perceptron networks, the activity level of the neuron is defined by the level of the output of the neuron. In biological neurons, the electrical output of the neuron is fixed and the activity level is related to the frequency of activation.

TABLE 2.1: ARTIFICIAL NEURAL NETWORK MODELS

Network Type	**Neuron Type**	**Topology**	**Learning Rule**	**Biological Plausibility**
Perceptron	Perceptron	Feedforward	Backpropogation	Low
SOM	Perceptron	Self-Organizing	Hebbian	Moderate
Hopfield	McCulloch-Pitts	Fully Connected	Hebbian	Moderate
SNN	STDP	Various	Rate-Based Hebbian	High

2.3.2 Similar Work

This work is related to neural network toolkits in the artificial and biological neural network realms. Its focus, however, is clearly directed towards the simulation of biological neural networks. It shares some concepts and motivations with works in both areas, and they are reviewed here.

2.3.2.1 Artificial Neural Network Toolkits

There exist several simulators for the development of neural networks. Most of these tools focus on the creation of feedforward neural networks for use in classification, optimization and regression. The most popular neural network simulators are presented and compared in Table 2.2.

The premiere simulator is the Stuttgart Neural Network Simulator [75]. This package is a large, comprehensive package that allows the specification of nearly every detail of network construction and training. The Stuttgart Simulator includes support for a large number of learning methods, such as Hebbian and backpropogation, and includes support for several types of pruning, Optimal Brain Damage[45], magnitude based, skeletonization[52] and others. It allows construction of traditional feedforward, ART[13] and Kohonen[42] networks. Despite its wealth of features and well-deserved reputation as the most comprehensive neural network toolkit, the Stuttgart Neural Network Simulator is not well suited to the task of simulating the growth and development of biologically inspired neural networks. Although it provides many of the required features, allowing arbitrary network topology, providing pruning algorithms and spiking neurons, it is limited by its ability to simulate network changes other than pruning during development and it is limited in the automated specification of complex network topologies.

The MatLab Neural Network Toolbox is an addition to the MatLab scientific modeling tool[22]. The package provides the ability to integrate feedforward and recurrent neural networks into MatLab models. Feedforward, Kohonen and recurrent networks (such as the Hopfield net) are supported, as well as numerous learning methods including variations of backpropagation and Hebbian. The Neural Network Toolbox is a powerful tool for engineers interested in using artificial neural networks, but it is not well suited to the realm of biologically inspired simulation.

JOONE is a neural network package developed entirely in Java[49]. It is an object oriented design that is intended to provide a comprehensive environment for developing artificial neural networks. JOONE supports feedforward neural networks and backpropogation and Kohonen learning rules. Due to its limited learning rules, lack of support for pruning and arbitrary network topology and the absence of spiking neurons, the JOONE package is not suitable for modeling biologically inspired neural networks.

The NEURObjects library [67] is a set of C++ classes that provide a framework for studying and using neural networks. The authors provide a class hierarchy that allows user-defined components to be added in place of the provided classes, allowing researchers to build a neural network to their precise specifications. The work is directed at researchers and engineers who are exploring the computational capability of feedforward neural networks or would like to incorporate a neural network based component into a software product. NEURObjects is limited in its ability to design networks. It is restricted to a ”layer-based” view of neural networks, and adding support to the tool to design and evaluate other network topologies would be prohibitively difficult. The limitations of this framework, while well designed, place it outside the domain of the biological neural network researcher.

There are several other artificial toolkits that are similar, the most important of these are the Artificial Neural Network Development Environment (ANNDE) [37] and Neural Network Objects (NNO)[44]. ANNDE presents an Object-Oriented Design of a neural network toolkit. It is focused on the development of neural networks for use in structural engineering applications, but the overall design of the toolkit makes it of interest. Neural Network Objects is a toolkit for developing object oriented artificial neural networks for use in application software.

2.3.2.2 Biological Neural Network Simulators

There are several neural network simulators from the biological neural network field that are also similar to ABNNSim. Among these are NEURON and Genesis. Both of these focus on developing extremely precise models of the behavior of a single neuron.

The premier toolkit for modeling biologically accurate neurons and neural networks is NEURON [32][33]. NEURON is designed to provide researchers with "biologically realistic models of electrical and chemical signaling in neurons and networks of neurons" [32]. This simulator models each neuron as a series of compartments, with different values for each component of the cell. NEURON is primarily aimed at simulations designed to calculate ionic concentrations and at simulations to compute the extracellular potential near the membrane.

The application domain of NEURON is that of single neuron and small neural network models[32]. The authors describe the upper limit of a NEURON model as being 10^4 neurons with 10^6 total synapses. These numbers are from 1997, so the

TABLE 2.2: NEURAL NETWORK SIMULATORS

Simulator	**Specify Topology?**	**Generate Complex Topology?**	**Pruning?**
Stuttgart NNS	Yes	No	Yes
MatLab NN Package	Yes	No	No
JOONE	Yes	No	No
NEURObjects	Yes	No	No
ANNDE	Yes	No	No
NNO	Yes	No	No

upper limit of current hardware is certainly higher, but we can reasonably claim that it is around 10^5 to 10^6 neurons. This number was arrived at by assuming that the size of the model is constrained by the size of main memory and noting that there has been a 1-2 order of magnitude increase in main memory since 1997. This limitation makes the modeling of certain domains impossible.

The intended research area of ABNNSim is in the exploration of topology change in neural development. This is a quite different area of application than that of NEURON. NEURON is simply not well suited to the exploration of activity induced topology change. Users can create network topologies by placing individual neurons, randomly wiring a network or reading a topology from a file[26]. The ability to read from a file suggests that it would be possible to develop a tool simply to handle topology formation. Although NEURON can easily provide an accurate description of the chemical and electrical behavior of the neurons and could be modified to provide more complex initial network topologies, it is not well suited for modeling changes in topology.

Another important neuron and neural network modeling toolkit is GENESIS [72][10]. GENESIS has a similar research domain as NEURON, that of the neurobiologist. It is designed to provide detailed simulations ranging in size from individual biologically representative neurons to networks of less detailed neurons. GENESIS, like NEURON, has a compartment-based model for simulating the electrical behavior of neurons. GENESIS is ideal for modeling the behavior of individual neurons and small neural networks. GENESIS can simulate networks of up to 60,000 neurons on a single commodity desktop machine with 1GB RAM [35]. As in NEURON, simulations in GENESIS are limited by the system's available memory. Simulations are limited to around 10^5 neurons, as in NEURON.

As the focus of GENESIS is on modeling the behavior of neurons and networks and not on their development, it is also not well suited to our task. The compartment-based model provides better fidelity and a more accurate representation of neuron function than the standard spiking neuron model in ABNNSim, but does not scale well. Larger networks can be built with the artificial neurons provided by GENESIS, but these are similar in their function to those provided by ABNNSim and thus offer no added accuracy. Additionally, the GENESIS simulator is designed for the simulation of the electrical and chemical modeling of neurons, and not with the development of networks. For researchers interested in exploring the processes involved in network formation, GENESIS is not a good fit.

GENESIS has a parallel version, PGENESIS, that runs on parallel machines[27]. PGENESIS is used to run single machine simulations in parallel, as well as large scale simulations that are distributed across multiple machines. It can run in a cluster environment that supports PVM, as well as several parallel architectures. PGENESIS gives researchers the ability to model larger scale simulations than those that can be modeled with GENESIS. Despite allowing larger simulations, PGENESIS does not contain the functionality necessary to model neural network development.

Catacomb2 is a neural network simulator that provides the ability to simulate regions of a Macaque monkey brain[29][12]. Catacomb simulates neurons and neural networks at a level of detail similar to NEURON and GENESIS, but has the distinct advantage of being able to create networks based on data from the CoCoMac database. As with NEURON and GENESIS, Catacomb2 is a tool for exploring the electrical and chemical behavior of individual neurons and small neural networks and is not well suited to the simulation of developing neural networks.

A very recent tool for modeling large-scale neural networks is Topographica[6]. It has not yet been released to the public, but its purpose and capabilities have been publicly announced. Topographica is intended to model cortical maps at a higher level than NEURON and GENESIS. Its stated purpose is to focus on the function and large-scale structure of biological neural networks. As it intended to model function in a biologically plausible manner, Topographica is of interest to our current research. Topographica does appear to be designed for functional, as opposed to developmental, simulations, which tempers its utility.

2.3.2.3 Neural Network Simulations

Szirtes, Lorincz and Palotai describe a neural network simulation[65], HebbNets, that is constructed of spiking neurons and learning happens via a Hebbian learning rule. Spike timing dependent plasticity (STDP) is an adaptation of Hebbian learning to spiking neurons, where the plasticity of the connection is related to the timing of spikes [66]. They present a design where the network is sustained by random input and over time develops structure. The authors demonstrate a range of topologies that can be formed by varying certain local rules in the simulation. HebbNets is compelling in that the global network topology is formed as a result of local rules. The authors analyze their results in terms of the harmonic mean distance of Latora and Marchiori and the scale-free analysis of Barabasi.

Segev and Ben Jacob present interesting work on the small-scale self-organization of neural networks in [63]. Their work explores the use of chemotactic signaling to direct the path of neuron growth cones during neural development. The authors demonstrate a simulation that incorporates their hypothesis. The simulations show that by simply following the gradients of chemicals in the medium, neurons can be directed to grow around obstacles and link together. The focus on the early formation of biological neural networks sets this work apart from that described here. The authors are focusing on network formation during a different developmental phase than that considered in ABNNSim. Also, their work does not incorporate the effect of electrical signaling on network development. The work described in Chapter 4 can be seen as complementary to the work of Segev and Ben Jacob.

2.4 Agent Based Modeling and Simulation

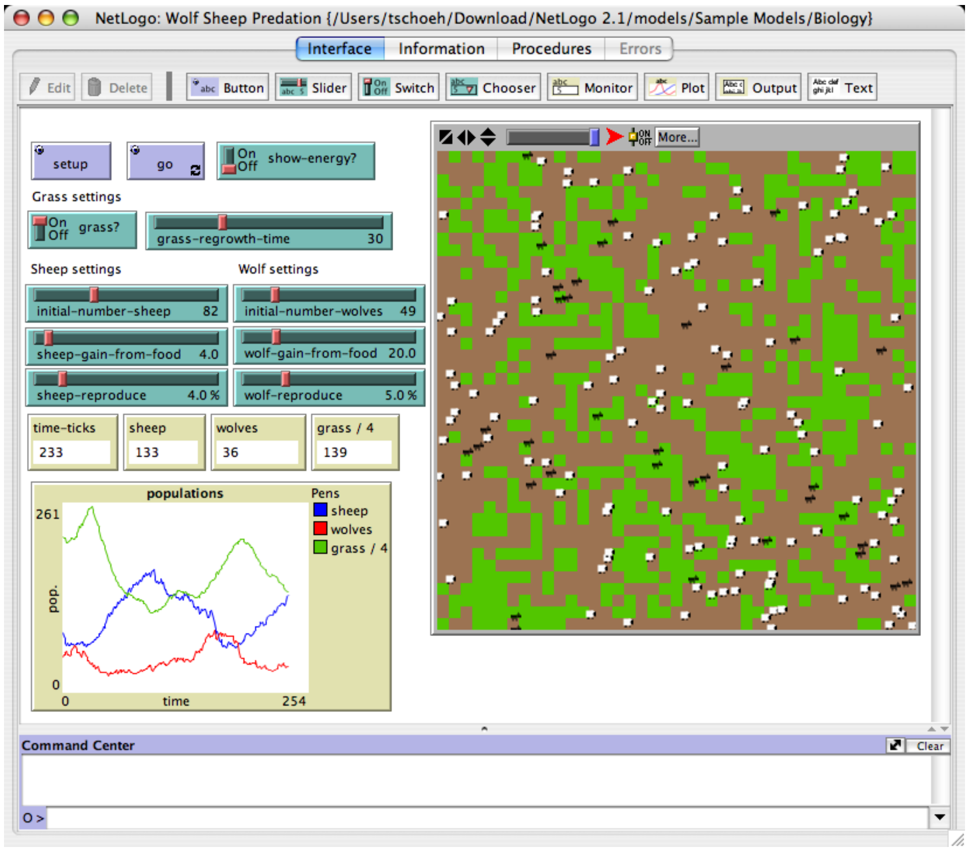
2.4.1 Historical Perspective

Agent-Based Modeling grew out of the Cellular Automata community, growing side by side with the Artificial Life community. The first Agent-Based Models shared much of the look and feel of Cellular Automata simulations. The environment was represented as a grid, and often agents were stationary and bound to a particular location. Simulations were run as a series of integer time steps, which was a severe limitation in terms of modeling real-world processes. The results of these simulations were suggestive models, compelling in terms of their output, but of limited use due to their lack of quantitative results.

The fundamental aspect of Agent-Based Modeling is the use of an agent as the primary entity of interest. An agent in this context is similar to the Artificial Intelligence concept of an agent. An agent is a self-contained entity with some basic actions and decision making that can influence its environment and other agents around it. Unlike Artificial Intelligence agents, ABM agents tend to be less sophisticated in their ability to react and influence the surrounding environment.

Agent-Based Modeling is often contrasted with approaches using systems of differential equations. An example would best illustrate the differences between an Agent-Based approach and an equation-based approach. Consider a simple Predator-Prey model. Current models using systems of differential equations are able to accurately model population fluctuations, but they cannot provide much more information. An Agent-Based Model would model each animal, predator or prey, as an agent in the system. The agents would have a reproductive capability, related to their ability to feed themselves. As the predator agents interacted with the prey agents, the populations would fluctuate, providing a similar output to the equation based model. An example of a predator-prey model is shown in Figure 2.3, a screenshot of the NetLogo model "Wolf Sheep Grass". The power of ABM becomes clear when the model needs to be changed to incorporate a new species into the food web. Adding a new type of agent is relatively easy, a small amount of code, depending on the toolkit. The interaction of the new agent with the ecosystem can be observed during the running of the simulation. In the case of an equation based model, the modeler needs to specify in advance the type of interaction that the new species will have with the system.

A more thorough comparison of Systems Dynamics and Agent Based Modeling is given in [69]. The author presents a case study where the same system was modeled with a Systems Dynamics tool (STELLA) and with an Agent-Based Modeling tool (StarLogo). The study demonstrates that Agent-Based Modeling is a desired approach for traditional researchers, as the modeling approach more closely simulates a traditional (physical) experiment.



NetLogo: Wolf Sheep Predation {/Users/tschoeh/Download/NetLogo 2.1/models/Sample Models/Biology}

On Off show-energy?

Grass settings

On Off grass?

grass-regrowth-time 30

Sheep settings

initial-number-sheep 82

sheep-gain-from-food 4.0

sheep-reproduce 4.0 %

Wolf settings

initial-number-wolves 49

wolf-gain-from-food 20.0

wolf-reproduce 5.0 %

time-ticks	sheep	wolves	grass / 4
233	133	36	139

Figure 2.3. The NetLogo Predator - Prey simulation Wolf-Sheep-Grass

2.4.2 Applicability

One of the earliest examples of an Agent-Based Model for academic research is that of Schelling in his study of housing segregation[57]. Schelling was attempting to explain how segregation in housing can occur as the result of well-meaning individual behaviors and not solely due to systemic prejudice. Schelling set up his simulation in the following manner: The simulation occurs on an 8 by 8 grid, with each space representing a house. An agent can choose to occupy an unoccupied house, but not to take over an occupied house. Once a space (house) is occupied, it receives the value of the agent type that took it over. The agents are given preferences such as "wants to choose a house with at least one neighbor of the same type". With simples rules like this, Schelling was able to demonstrate that the system organized into a state of segregation. Agents of type 1 formed an enclave and agents of type 2 created a separate enclave, and there was very little mixing between them. Schelling's model was not sophisticated enough to prove his point, however it was very valuable in that it showed that segregation could happen as a result of local “innocent” behaviors, instead of solely cause by global (systemic) rules.

Another example of an application of Agent-Based Modeling is in the simulation of the aggregation of slime mold[55]. Biologists had noted an interesting behavior of slime mold. Under normal circumstances colonies of the slime mold *Dictyostelium* do not display any group-level behavior, each individual acts in a local manner, eating, dividing and moving with seeming disregard for its neighbors. However, when the colony was put under stress, such as a lack of nutrients or water, an amazing behavior was noticed. The slime mold cells gathered together and formed a macro-scale structure which was able to move the colony as a whole in search of a more favorable environment. This observation was unprecedented. Some biologists believed that this behavior was only possible if there was a certain type of "pacemaker" or organizer cells that directed the formation of the macro-entity.

However, researchers used an Agent-Based model to dispute this claim[53]. They showed that by modeling the swarm of slime mold cells as homogenous agents and giving them simple chemotactic communication abilities, the colony as a whole could self organize into the macro-entity, without a need for specialized pacemaker cells. The model was comprised of a grid on which the slime mold cells lived. Each grid square contained some nutrients. The slime mold agents were able to communicate in a rudimentary way using chemotaxis, which is the depositing and sensing of chemicals in the environment. Research had previously shown that individual *Dictyostelium* amoeba did indeed communicate through chemotaxis, using the chemical cyclic AMP (cAMP). In the model, Agents released chemical signals, corresponding to cAMP, in the environment as a response to stress. For a few agents under stress, this release of chemicals had little effect on group behavior. However, when an agent detected that the chemical's concentration was above its personal threshold, it changed behavior and attempted to aggregate with other agents, as well as releasing more of the chemical signal into the environment. The releasing of additional cAMP creates a positive feedback loop, encouraging more agents to aggregate, leading to the desired results, the agents congregate and form a macro-entity. The result of this simulation showed that a few pacemaker cells were not necessary to organize the macro-entity.

The examples presented above represented some of the early models that created interest in the techniques of Agent-Based Modeling. More recent examples have built upon these early successes, but also extended them to the point where we can now create models that are capable of generating empirical results. Models such as TRANSIMS [23] and CompuCell [18][38] allow modelers to examine the effects of different starting conditions on the behavior of their systems. In the case of TRANSIMS, researchers can examine the effects of environmental policy changes on automobile pollution [54]. In the case of CompuCell, researchers have developed a complex model of avian limb morphogenesis that accurately models the development of a chicken wing.

Use of Agent-Based Models was initially limited to suggesting potential hypotheses or providing potential explanations of phenomena. Now many Agent-Based Models have reached a level of sophistication that makes it possible to use them in suggesting policy changes[4]. Bankes argues that Agent-Based Models should be viewed as a method of mathematical modeling akin to statistical modeling or systems of differential equations, and as such, subjected to the same standards of rigor, as opposed to being held to the much higher standard of mathematical proof[4]. Bankes goes on to claim that computational science is the "search for credible arguments based on computational experiments" [4] and as such, it provides the ability to suggest policy changes. Bankes cites as an example of this the use of Agent-Based Models of climate change and public policy.

In the context of this research we consider our Agent-Based Model as a way of demonstrating the possibility of local interaction affecting global network topology. The intent is not to prove that the behavior described in the system is in fact that seen in biological neural networks. The model is too simplistic to claim proof, but is clearly sophisticated enough to provide a plausible hypothesis.

2.4.3 Toolkits

There are several important elements to an Agent-Based Modeling toolkit. First is a scheduler. This can be as simple as a priority queue for single processor, discrete event simulations or a more complex method such as the Time Warp algorithm[40] or the Chandy-Misra algorithm [15], both used for the scheduling of events in distributed simulations. Next is an agent, which can be an Interface or Abstract Class which will be made concrete by the user. An agent can be simple and reflexive, with behavior governed by a finite state automaton, or highly sophisticated, using neural networks or other methods to reason about its surroundings. Finally, the last essential element is a space for the agents to occupy. Spaces also range from the simple to the complex. Often a space is represented by a 2D grid, a design constraint that can be traced back to models based on Cellular Automata.

It is within the ability of most programmers to "roll their own" Agent-Based Model, however, there are numerous benefits of using pre-made toolkits.

- Toolkits provide users with all of the basic components.
- Toolkits will often provide sample simulations, which aid immensely in the creation of a new simulation.
- They often provide a richness of features that could not be practically coded by an individual.
- They also often provide data analysis and visualization tools, which aid in rapid development and analysis

Several toolkits of varying maturity are freely available. Table 2.3 lists some of the most popular toolkits and presents a summary of their features. Some of the toolkits allow development with general purpose languages, like Java or C++, while others are limited to the use of toolkit specific languages and tools. Each toolkit was developed for a slightly different area and audience. StarLogo was initially a language for introducing students to computer programming[55]. In its current form, StarLogo allows users to create simulations using a drag and drop GUI. NetLogo is an offshoot of StarLogo, implemented in Java with a new user interface and redesigned modeling language. Swarm is a toolkit written in Objective-C with wrappers that allow models to be developed in Java. Swarm was initially developed at the Santa Fe Institute and is one of the first general-purpose Agent-Based Modeling toolkits. Toolkits are ranked according to Maturity, which is a subjective measure of the robustness of the toolkit and sophistication of its API.

RePast was chosen for this project for several reasons. It is a Java API, allowing models to be developed in Java. It is open source, allowing components of the API to be replaced or rewritten if necessary. RePast includes sophisticated display and analysis tools, allowing simulations to be developed quickly. Finally, its fine-grained discrete scheduler allows events to occur at intervals as small as can be represented with a Java double value.

TABLE 2.3: AGENT BASED MODELING TOOLKITS

Toolkit	**Maturity (1-10)**	**Scheduler**	**Language**	**Limitations**
RePast	8	Fine-Grained Discrete	Java	2D Visualization API
Swarm	7	Discrete	Objective-C	TCL/TK Graphics
Ascape	5	Discrete	Java	No Network API
Agent Sheets	6	Discrete	AgenTalk	Language Expressiveness
StarLogo	6	Discrete	Logo	2D Models
Mason	6	Discrete	Java	Newness of API
NetLogo	7	Discrete	Java and Logo	2D Models

CHAPTER 3

THE AGENT BASED MODELING APPROACH TO SIMULATING NEURAL NETWORKS

3.1 Abstract

This chapter describes a modular approach to the modeling and simulation of neural networks using the ABNNSim framework. Building on concepts from Object-Oriented Design and Agent-Based Modeling, the ABNNSim framework provides researchers with a useful tool for exploring and simulating artificial and biological neural networks. The framework was initially developed to explore self organizing structures in biologically inspired neural networks. ABNNSim is built using the RePast Agent-Based Modeling framework. [1]

3.2 Introduction

The tools of neurobiological researchers have increased beyond *in vitro* and *in vivo* experiments. Researchers have a new frontier for conducting research: computational simulation. Simulations, sometimes referred to as *in silico*, have certain limitations regarding their predictive ability, however they have huge benefits over the various *in vitro* and *in vivo* techniques currently available.

[1]Some elements of this work were previously presented in [60] and [58]

Simulations do not require the complex (expensive) lab equipment and constant attention of *in vitro* experiments. Many *in vivo* experiments require specialized equipment, e.g. fMRI, that is expensive and maintenance intensive. In addition, researchers must weigh the cost and ethical considerations involved in conducting experiments on living subjects. Simulations can also be run in parallel, allowing several different experiments to be conducted simultaneously. Most research institutions provide adequate computational resources, researchers can therefore take advantage of these resources at little or no cost.

There are certain drawbacks associated with conducting experiments using simulations. First, researchers must have a certain amount of Computer Science background, not only in terms of programming, but also in Operating Systems, Distributed Systems and System Administration. The performance of the simulations will be related to the programming ability of the researcher. The complexity of the simulation will be limited by the researcher's ability as well. Building a fully developed simulation is beyond the capability of most neurobiological researchers, therefore in order to take advantage of computational simulations, researchers must build upon existing tools. ABNNSim is intended to be such a tool. It it built upon the RePast Agent Based Modeling framework, a robust Java-based API. ABNNSim in turn provides a framework that is directed at neurobiology research, with pre-built Neuron and Axon classes, as well as methods of generating network topology and connecting input and output to the network. The source code for ABNNSim is freely available at the TMANS Sourceforge website[47].

3.3 Background

3.3.1 Similar Work

There exist several simulators and packages that are designed to simulate neurons and neural networks at various levels of detail. These are reviewed here briefly.

3.3.1.1 Artificial Neural Network Toolkits

There exist several simulators for the development of artificial neural networks. Most of these tools focus on the creation of feedforward neural networks for use in classification, optimization and regression. Given this focus, most of these tools are not suitable for biologically inspired simulations. The most applicable packages are reviewed here.

The most sophisticated of the Artificial Neural Network simulators is the Stuttgart Neural Network Simulator [75]. SNNS provides researchers with a powerful tool to create and develop Artificial Neural Networks. It provides a comprehensive set of neuron types and learning rules and allows users to create arbitrary networks. It also includes support for various types of pruning. Although SNNS provides most of the functionality that we would desire, it is limited in its ability to simulate biologically plausible neural networks. Specifically, SNNS does not include functionality to generate large networks with small-world or scale-free topologies and does not support Neuronal Regulation[17], which we believe is important in tempering the effects of Hebbian learning.

The NEURObjects library [67] is a collection of C++ classes that simplify the creation of artificial neural networks. The library is limited to the study of feedforward networks, but is mentioned for its modular approach and object-oriented design. We have used some of the ideas of this approach in the design of ABNNSim, focusing on modularity and making it easy for end users to add functionality.

Several other important Artificial Neural Network toolkits should be mentioned, but the area of focus of these packages makes them unsuitable for the simulation of biologically inspired neural networks. The MatLab Neural Network Toolbox is an addition to the MatLab scientific modeling tool[22]. It is a powerful tool for engineers interested in using artificial neural networks, but it is not well suited to the realm of biologically inspired simulation. JOONE is a neural network package developed entirely in Java[49]. Due to its limited learning rules, lack of support for pruning and arbitrary network topology and the absence of spiking neurons, the JOONE package is not suitable for modeling biologically inspired neural networks.

3.3.1.2 Biological Neural Network Simulators

ABNNSim was designed to complement the existing tools in computational neurobiology. The two top tools for simulating the behavior of neurons are NEURON[32] and Genesis[72]. These simulators provide the ability to precisely model the electrical and chemical behaviors of individual neurons, but are limited in the size of network they can feasibly simulate.

The GENESIS simulator[72][10] simulates biological neurons using a compartment model. Each section of the neuron is simulated using a cable equation with parameters for the capacitance and conductance of the cell and the incoming and outgoing electrical signal.

The NEURON simulator [32][33] is intended to model neurons using the compartment model, just as GENESIS does. NEURON differs from GENESIS in that it is a more recent design and strives to let researchers focus on the designing of models while abstracting away the details of compartment size. NEURON also includes a more advanced GUI for the development of models.

Catacomb2[29][12] is a neural network simulator that uses cortical map data from the CoCoMac database[43]. It models networks at a similar level of complexity as NEURON and GENESIS, but it is significant for its ability to instantiate cortical maps from the CoCoMac database.

The Topographica simulator [6] is a recent project designed to simulate several regions of a mammalian brain in parallel, enabling researchers to study phenonmena such as object segmentation and grouping in the visual system. Topographica is focused on the simulation of the behavior of cortical regions, and is not well suited as a simulator of the development of such regions.

3.3.2 Agent Based Modeling

Agent Based Modeling is an approach to the simulation of complex systems that attempts to model a system from the ground up. Any active component of a model can be considered an agent.Agents have behaviors and can interact with each other and with the simulated environment. The result of the interaction of the agents is a model of the system. Agent Based Modeling is an attractive paradigm for simulating systems where the principal actors are known and the researcher is interested in exploring the overall behavior of the system.

The Agent-Based Modeling approach is intuitive for researchers, as systems are modeled from the individual up. For areas like neurobiology, we have data concerning the functioning of individual neurons, allowing us to build simulated neural networks. The flexibility of the Agent-Based Modelling paradigm allows researchers to extend the behaviors of individual agents to explore new phenomena. In the case of ABNNSim, we extend the behaviors of the neuron beyond the electrical properties to include behaviors that can modify a neuron's physical connections with other neurons. This added behavior enables the model to explore a completely different paradigm, that of network topology change over time.

Agent-Based Modeling is often considered in opposition to the more traditional modeling approach using systems of differential equations. Models based on systems of differential equations are well suited to tasks that attempt to model aggregate behavior, for example predator-prey models. Despite the apparent competition between modeling paradigms, Agent-Based Models are not at odds with traditional equation based approaches. Indeed, Agent-Based Models may incorporate differential equations to approximate behaviors at lower levels, while allowing agent interactions to define high level, aggregate behavior. For example, in ABNNSim, the Neuron's spiking behavior is modeled by an equation and the overall interactions of the Neurons yield the system level behavior that we are studying.

3.4 Design

3.4.1 System Goals

The ABNNSim toolkit is intended to provide computation neuroscientists with a tool to study the growth and development of biologically inspired neural networks. Given the computational demands of simulating large networks of biologically accurate neurons, we argue that it is an acceptable trade off to sacrifice some biological plausibility in order to be able to simulate larger networks. The

Agent-Based Modeling approach allows us to approximate some of the electrical properties of the neuron while synthesizing these with other behaviors, such as pruning, that are of primary interest.

ABNNSim should offer a solid set of basic features and yield good performance, while offering the flexibility to allow researchers to tailor models to their own particular needs. This is a difficult balancing act and we provide evidence that we have done well on all fronts.

3.4.2 System Overview

In this section we give the overview of the ABNNSim framework, including relevant components from the RePast Agent-Based Modeling framework. For a more comprehensive presentation of RePast readers are directed to the RePast website [21]. The basic overview of ABNNSim is shown in the UML diagram in Figure 3.1.

The ABNNSim framework is composed of the following components:

1. ABNNSimModel - The model, this class handles the intialization of the model and the scheduling of various actions.

2. Neuron - The Neuron class is a model of a Neuron. It holds state information on the Neuron relating to its spike level, outgoing and incoming connections. The Neuron can schedule behaviors to occur at later times through ABNNSimModel.

3. Axon - The Axon class acts as a connector between Neurons. It allows spikes to travel in one direction and feedback to travel in the reverse direction.

4. TimeQueue - This is a simple class to help in the calculation of a Neuron's

activity level. Incoming spikes are stored in the TimeQueue in (time, value) pairs. Given a time value, TimeQueue will return the activation level at that time.

5. TimeItem - A simple (time, value) pair stored in a TimeQueue.

6. NetworkConstructor - This class will create networks of a specified distribution. Currently it supports the creation of networks with Scale-Free link distributions in the manner specified by Barabasi et al [1] and random networks. Creation of Small Worlds networks is handled by the RePast API.

The ABNNSimModel class handles not only the set up of the simulation but, as is encapsulates the Schedule, also the actual running of the simulation. ABNNSimModel contains various behaviors that are added to the schedule at different times. These behaviors handle the injection of random noise into the network, pruning and regrowth of connections between Neurons and output of the network and computed statistics at specified intervals.

The Neuron class defines the standard neuron. It encapsulates the functionality commonly found in a spiking neuron, as well as some additional attributes and behaviors that make it more appropriate for simulations exploring neurodevelopment. The standard spiking neuron has a spike level, a mechanism for determining whether its firing threshold has been exceeded and incoming and outgoing connections to other neurons. We implement an interrupt driven mechanism to determining firing times for the neuron. The use of an interrupt driven mechanism allows simulations to run faster and model larger networks. In addition to the standard spiking neuron behaviors, we have added facilities for Hebbian learning and neuronal regulation, two mechanisms that are regarded by researchers as important mechanisms underlying neural development[25][16].

At a more abstract level, the Neuron inherits from the DefaultDrawableNode class in the RePast API. This gives the Neuron fields for x and y coordinates in the display space, a reference to the model's Scheduler as well as various methods controlling the display style of the object. Also inherited are methods that allow the Neuron to be treated as a node in an abstract network.

The Axon class is a very simplified version of a biological Axon. As envisioned in ABNNSim, it acts as a conduit for exchanging various messages between Neurons in the simulation. Spikes travel down the Axon and reach the post-synaptic Neuron. In order to facilitate Hebbian learning, we defined the Axon to allow feedback to travel to the pre-synaptic Neuron. Axons also carry some state that specifies the weight of the connection. This weight can be modified by both the pre-synaptic and post-synaptic Neurons.

As in the Neuron class, the Axon class inherits from DefaultDrawableEdge from the RePast API. This allows Axons to be drawn on the simulation space and includes methods governing the way the Axon is displayed. Methods are also inherited that allow the Axon to be treated as an edge in a network.

The TimeQueue and TimeItem classes are convenience classes that make it easy to determine when a Neuron has reached its threshold and must fire a spike. A TimeItem holds a time value and a spike value. The time value is set by the Neuron when it fires. The time value is determined by the scheduler, so time values are globally ordered. The spike value is simply the value set by the originating Neuron, modified by the weight in the transmitting Axon. The TimeQueue is an ordered list of TimeItems, sorted by the time value. The TimeQueue maintains a window, and as new TimeItems are added, it checks to see whether all existing TimeItems are within the window. Old TimeItems that are now outside the window are dropped from the TimeQueue.

Each Neuron maintains a seperate TimeQueue to keep track of the spikes that it receives. When a new spike is received, the associated TimeItem is added to the TimeQueue. The Neuron then queries the TimeQueue to determine the activation level of the Neuron at this time step. The TimeQueue calculates the activation level by summing the values of each TimeItem. As the TimeItems represent spikes, the value of the spike degrades over time. The TimeQueue can calculate the current value of a spike simply by using the equations governing the spike and plugging in the elapsed time. Given the nature of the spike, it is only necessary to calculate the activation level of the Neuron when new spikes arrive, a computational convenience that allows large scale simulations to be modeled with good performance.

The NetworkConstructor class builds networks with various topologies. It currently allows users to construct Random networks in the style of Erdős and Rényi [8] and scale-free graphs using the method described by Barabasi et al [1].

3.4.3 Implementation

ABNNSim is written in Java [28] and uses the RePast Agent-Based Modeling Toolkit, version 2.2 [20]. RePast is an ABM API providing a discrete time event scheduler, various GUI and display components and built-in analysis tools. To this we added several Neuron classes, helper classes for managing the timing of Neuron outputs and additional classes for constructing various types of networks. A screen shot of the ABNNSim application is shown in Figure 3.2.

The resulting ABNNSim API is a pure Java framework that runs unmodified on all Java 1.4 compatible systems. Development and testing of the system was conducted on various Sun Solaris, Mac OSX and x86 Linux machines.

3.5 Results

3.5.1 Case Studies

Several case studies were conducted using ABNNSim. These studies were meant to examine the flexibilty of the framework and determine its suitability for use by research groups. Each study began with the ABNNSim framework and developed extensions to explore a particular behavior or mechanism that was considered to be potentially interesting by the research community.

3.5.1.1 Case 1: Rewiring With Distance Bias

In this case study, we extend the ABNNSim model by adding a rewiring process. In the initial model, neurons are selected to add a new connection to another neuron (rewire). The neuron selects another target neuron at random from those in the model. Next, we consider the effects of making the choice of target neurons related to the distance of the target neuron from the source neuron. Neurons that are farther away are less likely to be chosen than those that are nearby.

The distance bias is added to the system by creating a “distance list” for each neuron, basically a list of all other neurons in the system, sorted by distance from the target neuron. The coding time was 3 hours, including debugging. Three new classes were created: DistanceNeuron, DistanceNeuronWrapper and DistanceNeuronTests. The DistanceNeuron is a sub-class of the Neuron class, with an extra 30 lines of code handling the creation and management of a sorted list of the other Neurons in the model. DistanceNeuronWrapper is a wrapper that allows DistanceNeurons to be added to a TreeMap and sorted by distance.

DistanceNeuronTests is a class with unit tests validating the functionality of the DistanceNeuron class. Additional code was added to ABNNSimModel for setting up the distance lists and adding rewiring actions.

A screenshot comparing the effects of random and distance-based rewiring is given in Figure 3.3. As pruning takes place, both random and distance rewired networks suffer a drop in Local Harmonic Mean Distance. The randomly rewired networks show a smooth growth in Local Harmonic Mean Distance, whereas the distance rewired network shows much slower growth. These results are the reverse of what was expected, but can be explained by the way the Local Harmonic Mean Distance is calculated. The Average LHMD is measured without regard to distance on the substrate, only in regards to distance in the network, where each link is considered to be length 1.

3.5.1.2 Case 2: Adding Inhibitory Neurons

In this case study, we add a new Neuron to the ABNNSim model. In the original simulation, all Neurons signal with positive values. Research has shown that there exist Neurons that have an inhibtory effect upon their surrounding neurons. We add a Neuron of this type to the simulation.

The new Neuron type is called InhibitoryNeuron. Instead of activating its surrounding Neurons, this Neuron will suppress them. We start by extending the Neuron class and modifying its default behavior. Then we add some extra setup code to ABNNSimModel to add some InhibitoryNeuron to the simulation.

A visual comparison of the networks resulting from the original simulation and from the simulation including InhibitoryNeurons is given in Figure 3.4. Note that the network formed with 10% InhibitoryNeurons is significantly more dense than the unmodified simulation. The degree distribution histograms confirm this quantitatively.

3.5.1.3 Case 3: Chemical Signaling

In this case study, we add rudimentary chemical signaling to the ABNNSim model. We utilize the RePast API to provide classes for the diffusion of chemicals through the medium. We add behaviors to the standard Neuron class to sense and react to changes in the chemical concentrations in the surrounding medium.

This case study is motivated by research indicating that neuron behavior is greatly affected by the presence of chemicals in the medium. The presence of adrenaline, for example, noticeably lowers the firing threshold of hippocampal neurons. These chemicals can be introduced into the medium by various glands or organs, or by the neurons themselves. Chemicals can be used by the neurons as another means of interacting with their environment.

This modification is made feasible by the Agent-Based Modeling paradigm. When a Neuron agent senses a change in the medium, it adjusts its threshold level accordingly. We extend the Neuron class and include functionality for sensing and releasing chemicals in the new ChemNeuron class.

3.6 Conclusion

We claim that ABNNSim provides a useful framework for simulation, speeds up development time and allows easy modification of the model. The component-based framework allows researchers to develop new models of Neurons and use these in place of current implementations without needing to rewrite their simulation. The approach offered by the Agent-Based Modeling paradigm allows researchers to step away from the traditional equation-based approach, allowing the network to be modeled in an intuitive manner. This leads to thinking about neural networks as a collection of interacting agents, each with individual behaviors, that, when observed over the entire system, yields a similar output to equation-based modeling.

Researchers can use ABNNSim to quickly explore new ideas and provides a level of simulation that is not provided in existing tools. ABNNSim focuses on a level of detail that is at a higher level than some computational neuroscience tools like NEURON and GENESIS, but below the level of behavioral simulations such as Topographica. Although the level of biological plausibility of each Neuron in ABNNSim is well below that found in a computational neuroscience tool like NEURON or GENESIS, the relaxing of these restrictions allows researchers to explore more complex interactions and behaviors. While the accuracy of these results rest upon replication by more accepted tools, they at least can be used to generate possible directions of research.

Users interested in developing simulations with ABNNSim are encouraged to contact the authors or may visit the project website http://tmans.sourceforge.net.

3.7 Future Work

This work provide a solid base on which to build more detailed and biologically plausible neural network simulations. The components currently in ABNNSim already provide researchers with a useful base, however there is room for the framework to grow.

A new, extended version of ABNNSim is currently under development. The next version will offer more neuron types and a more flexible method of sending input to and from the network. In order to make the framework more useful for other researchers, the next version will be as comprehensive as is reasonably possible. Several biological structures with relevance to neural modeling, such as glial cells and various neuron components may be added to the framework. Refined input to and output from the networks will allow researchers to use constructed networks for computation and validation against biological neural networks. Currently there are plans to add support for restarting the simulation given a snapshot of a prior simulation. This could be important when exploring dynamic networks. Another important addition to ABNNSim is the inclusion of support for writing simulation output data to a database. Such work would simplify the task of categorizing and analyzing simulation results.

Users of the current framework would like to be able to better calibrate the model with data on the functioning of biological neurons. Should a researcher want to create a network of, for example, frog neurons, it would be beneficial to be able to easily calibrate the system's neurons with data from a frog neuron. Making it easier to calibrate and validate the model will ensure that ABNNSim is a relevant and useful tool for researchers.

There is ongoing work being conducted that will allow a model or series of models to be run across a cluster of machines in parallel. Currently there exist methods for running simulations in parallel, but each simulation must run independently. Having simulations running in a distributed manner would allow researchers to simulate very large scale neural networks. The ability to have simulations span multiple machines will allow researchers the ability to study networks of a unprecedented size, allowing the modeling of not only the entire brain of an animal, but potentially a colony of interacting animals.

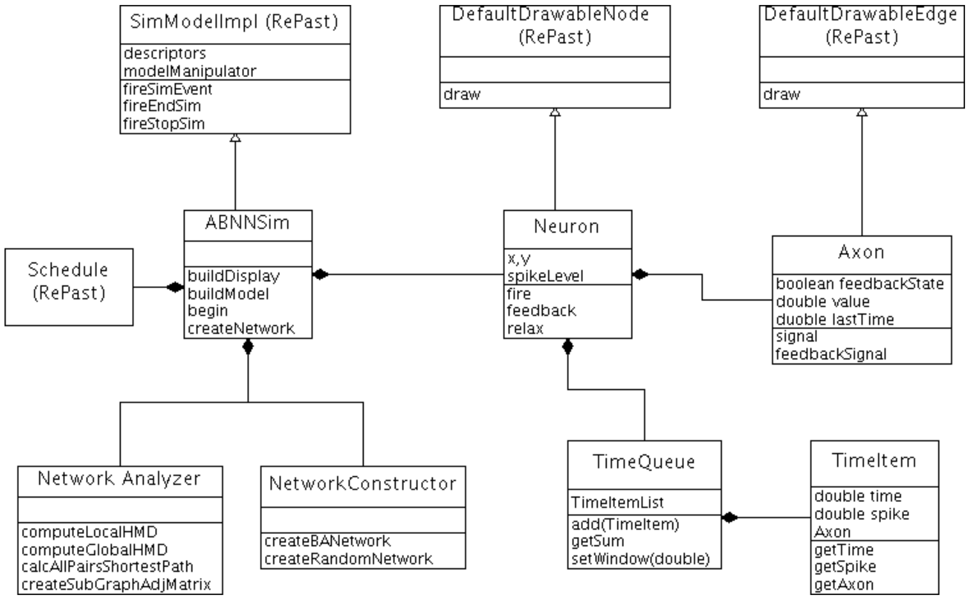
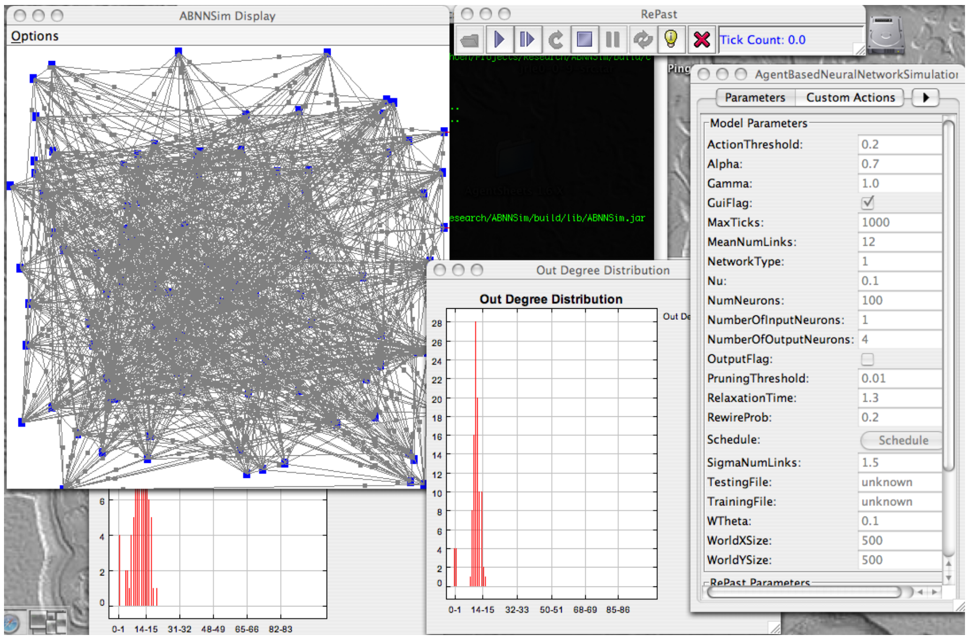


Figure 3.1. UML of ABNNSim.

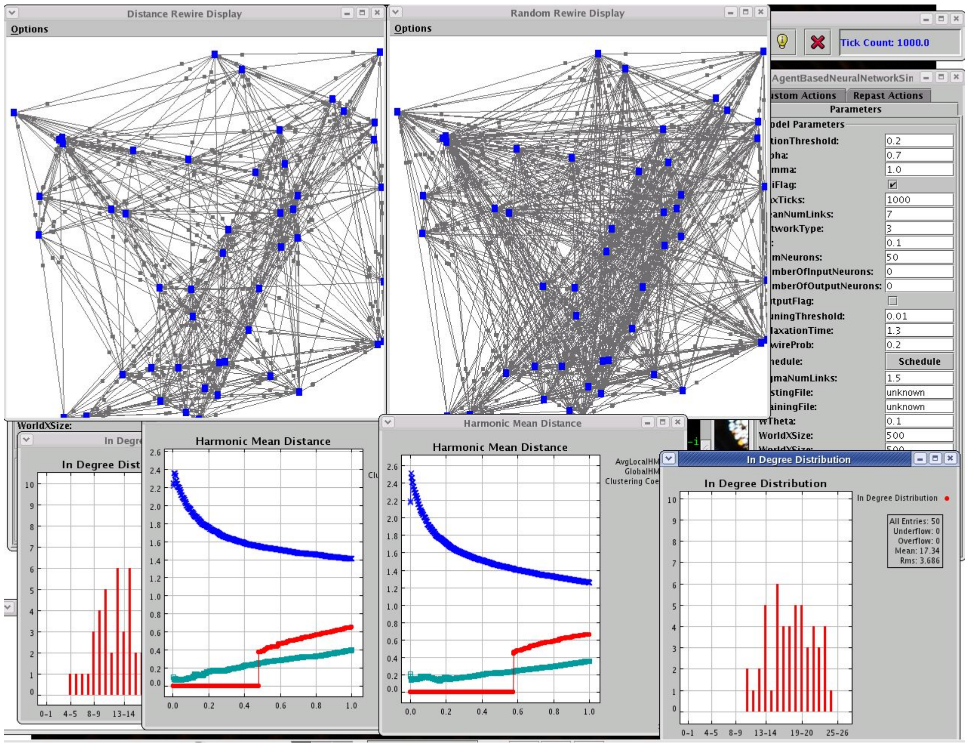


Tick Count: 0.0

```
esearch/ABNNSim/build/lib/ABNNSim.jar
```

ActionThreshold:	0.2
Alpha:	0.7
Gamma:	1.0
GuiFlag:	✓
MaxTicks:	1000
MeanNumLinks:	12
NetworkType:	1
Nu:	0.1
NumNeurons:	100
NumberOfInputNeurons:	1
NumberOfOutputNeurons:	4
OutputFlag:	☐
PruningThreshold:	0.01
RelaxationTime:	1.3
RewireProb:	0.2
Schedule:	Schedule
SigmaNumLinks:	1.5
TestingFile:	unknown
TrainingFile:	unknown
WTheta:	0.1
WorldXSize:	500
WorldYSize:	500

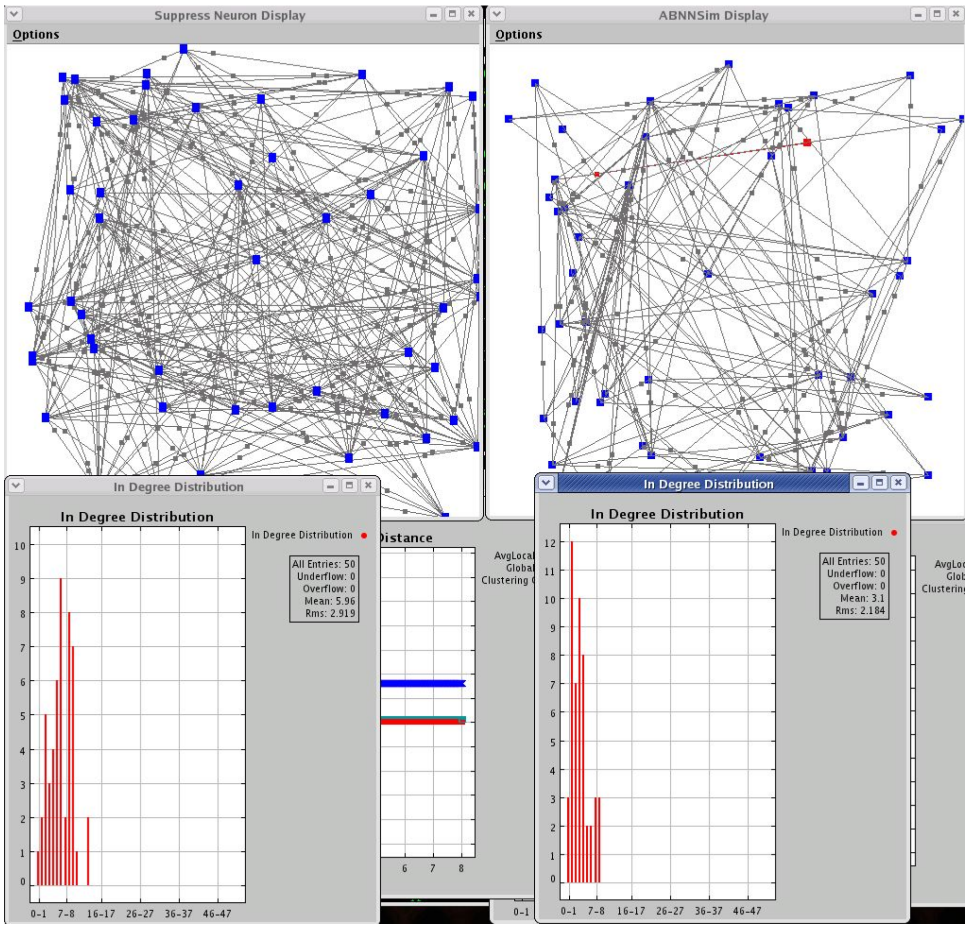
Figure 3.2. A screen shot of the ABNNSim application. The RePast framework provides the simulation control panel and model parameter window. ABNNSim users can view the network as the simulation progresses. Also shown are two dynamic histograms that show the link distributions of the network at the current time step.



```
odel Parameters
tionThreshold:          0.2
pha:                    0.7
mma:                    1.0
iFlag:                  ✓
xTicks:                 1000
anNumLinks:             7
tworkType:              3
:                       0.1
mNeurons:               50
mberOfInputNeurons:     0
mberOfOutputNeurons:    0
tputFlag:               ☐
uningThreshold:         0.01
laxationTime:           1.3
wireProb:               0.2
hedule:                 Schedule
gmaNumLinks:            1.5
stingFile:              unknown
ainingFile:             unknown
wTheta:                 0.1
WorldXSize:             500
```

```
All Entries: 50
Underflow: 0
Overflow: 0
Mean: 17.34
Rms: 3.686
```

Figure 3.3. Side by side comparison of distance vs. random rewired networks



```
All Entries: 50
Underflow: 0
Overflow: 0
Mean: 5.96
Rms: 2.919
```

```
All Entries: 50
Underflow: 0
Overflow: 0
Mean: 3.1
Rms: 2.184
```

Figure 3.4. Side by side comparison of original simulation and one with InhibitoryNeuron. The InhibitoryNeuron simulation is on the left, the unmodified ABNNSim is on the right.

CHAPTER 4

A TOPOLOGICAL EXPLORATION OF SELF-ORGANIZING NEURAL NETWORK STRUCTURES

4.1 Abstract

In this chapter we present an analysis of some of the global topologies that can be induced in a neural network using only local rules. Given a network of neurons we utilize the principles of overgrowth and selective pruning to induce topological changes in the network. This general process is in line with accepted research into biological neural network development. Pruning is guided via a process incorporating Hebbian learning and a similar result from neurobiology research, Neuronal Regulation. Results and discussion are presented. [1]

4.2 Introduction

The model presented is of a biologically-inspired neural network. Effort has been made to bring the model in line with current thinking on biological neural networks, however, the authors acknowledge the limitations of the presented model and its distance from accepted theory. This work is intended to demonstrate the possible topological structures that can be derived from local rules in biologically inspired neural networks.

[1]Elements of this work were previously presented in [59] and [61]

Understanding the development of the human brain is one of the foremost challenges facing scientists today. The brain is a complex network of the highest order, with a number of elements that rivals the size of the Internet. However, exposing the mechanisms at work in the organization of the brain is significantly more difficult that charting the course of a man-made network like the Internet. Numerous mechanisms at different levels are responsible for this organization. In this work, we examine the effects of overgrowth and pruning on network development.

4.3 Background

This work draws upon recent research in biological and artificial neural networks, complex network analysis and the techniques of Agent-Based Modeling. Each of these diverse areas is given a basic introduction here.

4.3.1 Complex Networks

One method of evaluating the results of our simulation is to use accepted network measurements and compare our results with those of known networks. We use the Harmonic Mean Distance metric of Latora and Marchiori [48] to examine whether the networks form Small World graphs and we examine the link structure using histograms as in the work of Barabasi et al [2].

The Harmonic Mean Distance measurement is a metric related to both the clustering coefficient, C, and characteristic path length, L, measurements introduced by Watts and Strogatz[71]. It is a measurement of the efficiency of a network, in terms of information propagation[48]. Harmonic Mean Distance can be calculated as a local value, with Equation 4.2, in which case it is related to $\frac{1}{C}$. When Harmonic Mean Distance is calculated as a global value, as in Equation 4.1, it is similar to the characteristic path length L. In the calculation of these metrics, G is a given graph with N nodes and $d_{i,j}$ is the distance (in number of edges traversed) between vertices i and j. In the calculation of the Global Harmonic Mean distance, the distance is calculated between each pair of vertices. For the Local Harmonic Mean Distance calculation, we pick a vertex i, then calculate the distance between every pair of vertices in the subgraph (without including paths through i).

$$D_{global}(G) = \frac{N(N-1)}{\sum_{i,j \in G} \frac{1}{d_{i,j}}} \tag{4.1}$$

$$D_{local}(G) = \frac{1}{N} \sum_{i=1}^{N} \frac{N(N-1)}{\sum_{j,k \in G_i} \frac{1}{d_{j,k}}} \tag{4.2}$$

The Harmonic Mean Distance has some advantages over the clustering coefficient and characteristic path length. First, Harmonic Mean Distance can be calculated as a local or global value, giving a unified description of the network from local and global perspectives. Next, the global Harmonic Mean Distance value is meaningful for graphs that are not connected. Given that we have no guarantees of connectivity, this is relevant to our work. Finally, Harmonic Mean Distance provides meaningful insight on graphs that are embedded in a metric space [48], such as graphs existing in Euclidean space.

We use the Harmonic Mean Distance measurement to determine whether resulting graphs are Small Worlds graphs. Given that the networks resulting from the simulations are of differing sizes, we normalize the graphs using the Harmonic Mean Distance values associated with a regular lattice of the same size, as described in [48].

We also examine the link structure of resulting graphs in a manner similar to that of Barabasi [2]. We analyze the graphs by creating a histogram of the number of incoming links per node for all nodes in the network. We then determine whether the graphs are indicative of an exponential or scale-free distribution.

4.3.2 Biological Neural Networks

The development of mammalian brains is characterized by a rapid overgrowth of axons and neurons before birth, a rapid loss of axons and neurons after birth and an associated rise of synapses after birth, peaking at adolescence[14]. This overgrowth and associated pruning coincides with a critical time period of network development and is suggestive of the importance that pruning plays in network formation.

Given the size of the human neural network (10^{11} neurons and 10^{15} connections), Segev and Ben Jacob have argued that the wiring diagram of the brain exceeds the ability of DNA to encode that information[63]. Thus the brain must self organize. As a potential method of self-organization, Segev and Ben Jacob present a mechanism using chemical signaling.

In this chapter we describe a complementary approach, based on network overgrowth and selective pruning. Here pruning is guided by two local processes: Hebbian learning and neuronal regulation. Hebbian learning is the accepted theory that neurons that fire in sequence have a strengthened connection. Neuronal regulation, as presented in [17], is a mechanism that regulates the neuron's output multiplicatively with an inverse relationship to its activity level. Hebbian learning adjusts the strength of individual synapses between neurons and neuronal regulation maintains the overall output level of each individual neuron, preventing a positive feedback loop.

The equations for neuronal regulation are presented in Equation 4.3 and Equation 4.4. The first step is the degradation, where each synapse's weight is reduced according the Equation 4.3. Next, Hebbian learning occurs. Then, the neuron uses Equation 4.4 to restore its input field to the previous level. In Equation 4.3, the initial weight of a synapse at time $t + 1$, W'^{t+1} is determined by subtracting the weight at time t, W^t, by W^t raised to the degradation dimension α, where $0 \leq \alpha \leq 1$, and multiplied by a Gaussian distributed noise term η^t. The input field is restored by multiplying the intermediate weight ${W'}^{t+1}$ by the ratio of the input field at time 0, f_i^0, to the input field at time t, f_i^t.

$$W'^{t+1} = W^t - (W^t)^\alpha \eta^t \tag{4.3}$$

$$W^{t+1} = W'^{t+1} \cdot \frac{f_i^0}{f_i^t} \tag{4.4}$$

4.3.3 Related Work

This work is closely related to the HebbNets project[65]. HebbNets is designed to examine the types of network topologies that can form in a network of Perceptrons using Hebbian learning and random noise. The results of the work showed that many different network topologies could be formed with just Hebbian learning. The authors of the HebbNets study use the Harmonic Mean Distance measurement to analyze their work. For comparative purposes we display the results of our work in terms of Harmonic Mean Distance as well.

4.4 Design

This work is built on the ABNNSim framework, a tool for simulating neural networks that uses simulation techniques from Agent-Based Modeling. ABNNSim is in turn built upon the RePast 2.2 API, a toolkit for building Agent-Based Models. ABNNSim models each neuron as an agent, endowing it with a rich set of behaviors. Thus in addition to the standard electrical properties, we can represent some of the physical processes that affect neural network development.

4.4.1 Neuron Model

This work examines pruning in networks of Perceptrons. Perceptrons are artificial neuron models that were chosen for their computational simplicity. Although Perceptrons do not accurately represent biological neurons, in this context they allow us to explore behaviors in large networks.

4.4.2 Pruning

Pruning seems a likely candidate for being a local mechanism that directs the development of the mammalian brain. We look at pruning directed by Hebbian learning and neuronal regulation and examine its effect on network topology. The simulation is setup as follows. First, the network is created in an "overgrown" state. We examine three initial topologies, Random, Scale-Free and Small World. Next, we activate the network with random noise. The random noise stimulates the network and Hebbian learning and neuronal regulation occur, leading to the strengthening of some synapses and the weakening of others. Finally, connections that have been weakened below a certain threshold are removed from the system.

4.4.3 Networks

We chose to examine several initial network configurations, to see if initial conditions played a role in the outcome of the network self-organization. We initially used only Random networks, then added Scale-Free networks and Small World networks. Small World networks are constructed using the methods in the RePast framework. These involve constructing a network as a ring lattice and doing probabilistic rewiring. The Scale-Free network constructor was written to create networks using the algorithm described by Barabasi et al [1]. The Random network constructor uses the Erdos-Renyi method, as described in [8].

4.4.4 Measurements

In order to determine our progress towards self-organization, it is important to be able to rate the outcome of the experiment. We analyze the networks using a link distribution histogram, as done by Barabasi[2] and others. With this measurement we are interested in determining whether the resulting pruned network displays a discernible link structure. We also examine the networks using the Harmonic Mean Distance measurement, as described above. This measurement allows us to determine whether the graph is a Small Worlds graph or not.

We chose these measurements out of the belief that biological neural networks would display a Small Worlds or Scale-Free distribution. This belief is grounded upon the large body of work that suggests that many large, complex networks found in nature display one or both of these characteristics. The selection of the Harmonic Mean Distance measurement also allows this work to be more closely compared with the HebbNets project[65].

4.5 Results

4.5.1 Sensitivity Analysis

The simulation, as presented, has an 11-dimension parameter space. The parameters are the number of neurons, average number of connections per neuron, the two neuronal regulation parameters α and η, the activation threshold, the pruning threshold, $w\theta$, a Hebbian learning parameter, the relaxation time, the mean amount of gaussian distributed noise to inject into each neuron and the type of network to use initially. The parameters and their values are given in Table 4.1.

The selection of appropriate values for these parameters was a nontrivial task. Appropriate values were inferred from the associated literature for Hebbian learning and neuronal regulation, as well as a parameter sweep.

4.5.2 Pruning

In this section, we present an analysis of data from an ensemble of simulation runs. The simulations were run with 1000 nodes and 10 links per node. All neuronal regulation and Hebbian learning parameters were kept constant. We ran the simulations for 200 time steps. These time steps correspond to the firing rate of the neurons. The simulations began with initial graphs that were constructed with Small Worlds, random and scale-free topologies, respectively.

First we observe the link distribution before and after the simulation. Figure 4.1 shows a comparison of the in degree distribution measured at the beginning and end of the simulation. Before running the simulation, the in degree is distributed according to a Gaussian normal distribution with mean 10, variance 1.5. The graph construction method ensures that the links are distributed in this fash-

TABLE 4.1

DEFAULT PARAMETERS OF SIMULATION

Parameter	**Value**
Number of Neurons	1000
Mean Connections	10
Sigma Connections	1.5
α	.7
η	.1
Activation Threshold	.2
Pruning Threshold	.01
W_θ	.1
Relaxation Time	1.3
Noise Mean	.2
Network Type	Small Worlds

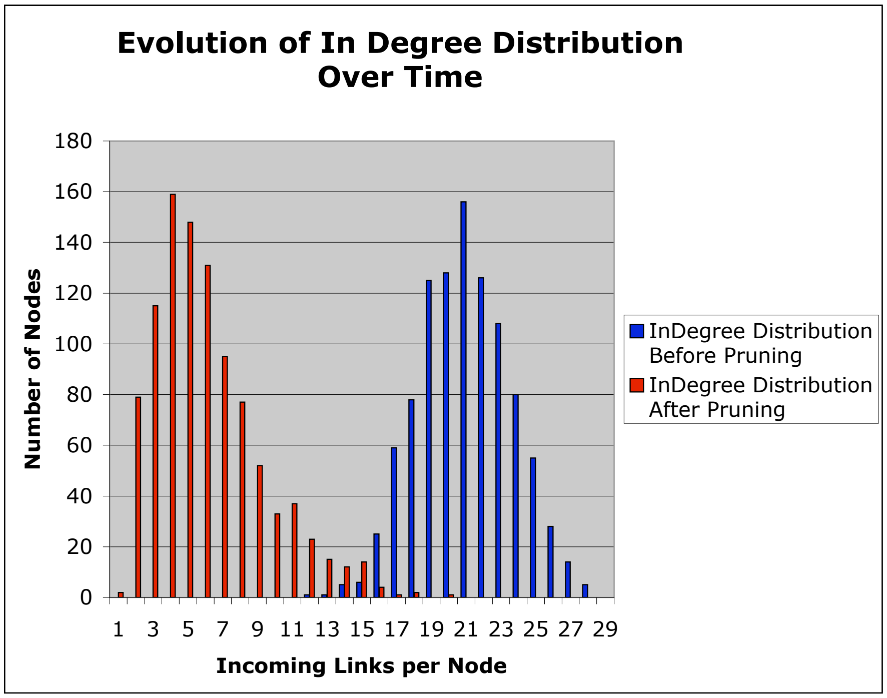


Figure 4.1. A Comparison of the In Degree Distribution Before and After the Simulation, Starting with a Watts-Strogatz Small Worlds Graph

ion. After the simulation, the links display an exponential distribution. This transformation indicates that the resulting link structure resembles what would be expected of a random graph of equivalent size.

We start with a scale-free link distribution and repeat the above simulation. The results, plotted on a log-log scale are given in Figure 4.2. The simulation starts with a Barabasi-Albert network, displaying the characteristic scale-free link distribution, evidenced by the linear distribution of points and heavy tail. After pruning, the simulation displays a somewhat more sparse distribution, but it is still consistent with a scale-free distribution.

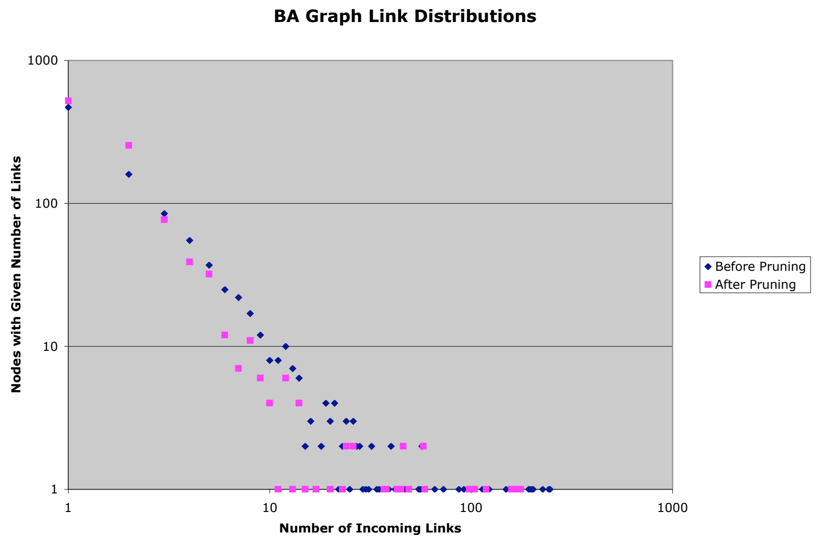


Figure 4.2. A Log-Log Plot of the Link Distribution of the Network Before and After Pruning

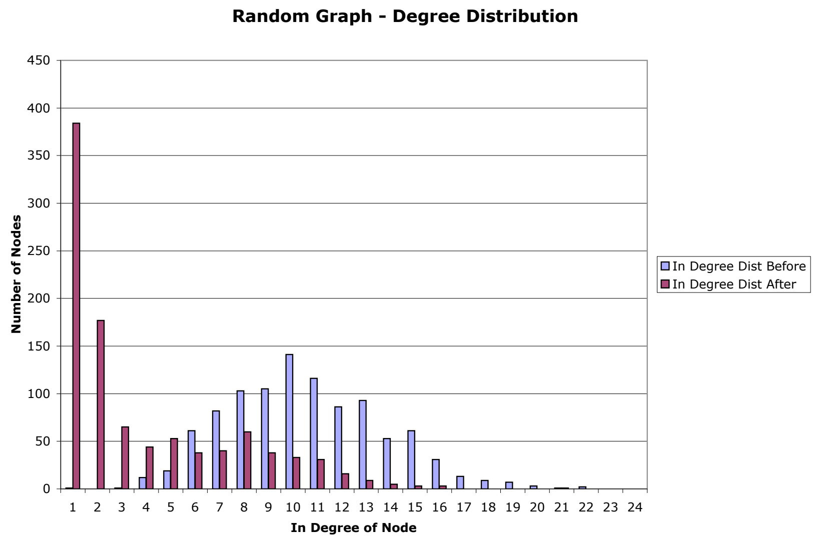


Figure 4.3. A Comparison of the In Degree Distribution Before and After the Simulation, Starting with a Random Graph

Next, we repeat the above experiments with a random network. The results of this simulation are shown in Figure 4.3. The random graph displays a much greater degree of perturbation than either the Watts-Strogatz or Barabasi-Albert graph. However, like the Watts-Strogatz graph, the resulting link distribution is also exponentially distributed.

Harmonic Mean Distance data from the simulation runs was averaged and plotted in Figure 4.4. At the given parameters, the graph quickly diverges from its initial values. The Global Harmonic Mean Distance value increases within the first two time steps, then remains unchanged for the rest of the simulation. Similarly, the Average Local Harmonic Mean Distance value (plotted here as the inverse

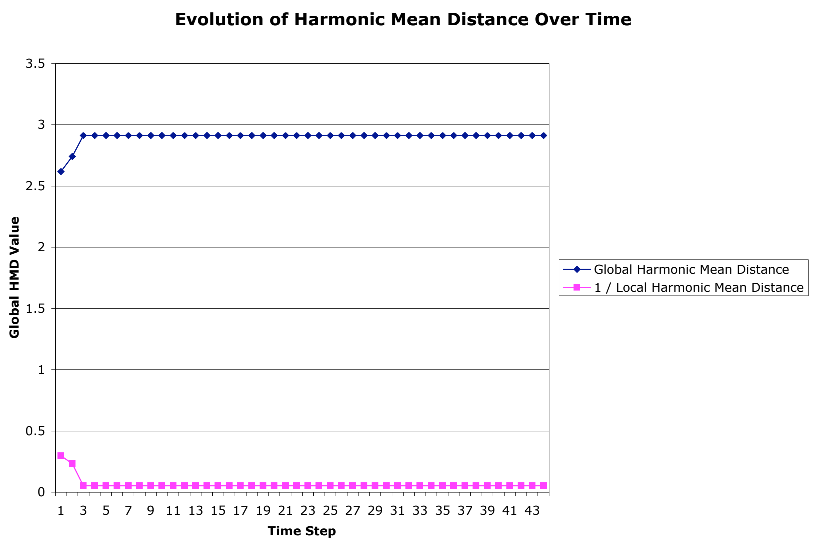


Figure 4.4. The Evolution of Harmonic Mean Distance Over Time

of the recorded value in order to correspond with C the Clustering Coefficient) degrades within the first two steps of the simulation, then holds for the rest of the simulation. For these parameters, the (inverse) Average LHMD stabilizes at 0, putting the resulting graph outside the regime of the Small World.

The results from several runs are presented in Figure 4.5. We present a graph of the number of links over time, in order to compare the results of the simulation to the known degradation of synapses in mammalian brains as presented in [14]. Although this result does not completely correlate with the results from [14], we believe that some tuning of the model parameters should allow us to achieve a better fit.

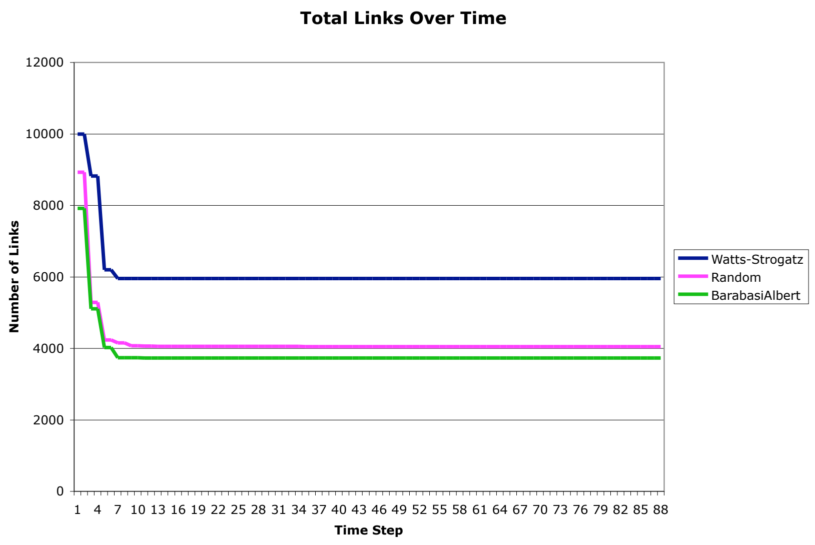


Figure 4.5. The Number of Connections (Synapses) Over Time

4.6 Conclusion

4.6.1 Discussion of Results

We have shown the power of locally-directed pruning as a tool for topology formation in neural networks. Our results demonstrate that for some starting topologies, neuronal regulation and Hebbian learning guided pruning will maintain link distribution. This result indicates the importance of developing a resilient initial topology.

The presented results demonstrate that Barabasi-Albert networks are very resilient to change, whereas Watts-Strogatz networks rapidly degenerate to random networks.

4.6.2 Implications

This work suggests that pruning is indeed a potential method of self-organization in biological networks. We have demonstrated that under certain conditions, various network topologies can form as a result of only local mechanisms.

Additionally this work highlights the importance of the value of Agent-Based Modeling in the construction of models of complex systems. By modeling neurons as agents, we can construct intuitive models using the behaviors observed in biological neurons. These include electrical and physical behaviors and mechanisms, yielding a rich, complex model. The Agent-Based Modeling approach is well suited to our approach of focusing on local mechanisms like neuronal regulation and Hebbian learning.

4.7 Future Work

This work would be more biologically plausible with the addition of integrate and fire (spiking) neurons. That work is currently underway. The difficulty of the adjustment lies in the translation of the Hebbian and neuronal regulation mechanisms to a rate-based paradigm.

It would be interesting to compare this work to a more accurately modeled simulation using a computational neuroscience tool such as NEURON or GENESIS. Modeling several thousand neurons using compartments and cable equations would be computationally challenging. It would be interesting to note to what degree this added realism affects the results of the model.

CHAPTER 5

RESULTS AND CONCLUSION

5.1 Summary of Results

In this thesis, we have shown that Agent-Based Modeling is a useful and natural approach to the simulation of biologically inspired neural networks. This approach yields a simulation environment that is flexible and extensible. The three presented variations on the project demonstrate the ease with which researchers can adapt the model to their own needs. It is therefore reasonable to conclude that the model is sufficiently extensible. One in depth experiment conducted with the ABNNSim framework is presented: an exploration of the self-organizing topologies that can be generated with local rules.

5.1.1 ABNNSim: A Framework for Building Neural Network Simulations

We presented the ABNNSim framework for the simulation of neural networks. ABNNSim provides researchers with a flexible yet powerful tool for the simulation of neural networks. Included in the framework is a simple threshold gate (Perceptron) neuron model, support for discrete and continuous timing, the ability to build networks with Scale-Free, Small World or Random distributions and the ability to record various information about the simulation at user-defined intervals. Data is output as flat ASCII text files, suitable for processing with standard

Unix text editing tools. Researchers can specify the amount of information they want to store, from computed statistics (average clustering coefficient, link distribution) to a snapshot of the entire system, including a connectivity matrix of the network as well as the state of each entity in the system.

We demonstrated three small projects that extended the ABNNSim structure. These projects demonstrate the extensibility of the ABNNSim framework. In each project, we begin with the ABNNSim framework and extend it. The projects were intended to be examples of the type of extensions that real researchers would add to the system.

The first project looked at the effects of distance-based rewiring on global topology, an extension of the work done in Chapter 4. In the initial version of the simulation, when a neuron was selected to rewire, it did so in a random manner. This project modified the rewiring routine to add a distance-based bias. Thus when a neuron is selected to rewire, it is more likely to pick a nearby neuron.

The second project documented the addition of a new Neuron implementation to the framework and the work required to do so. Here we add a spiking neuron to the simulation. The spiking neuron is a more accurate model of the neurons in biological neural networks. This work is demonstrative of one of the typical extensions that a researcher would add to the framework.

The third project added chemical signaling in the medium. The existence and quantity of certain compounds (adrenalin, etc) in the medium surrounding biological neurons affects a neuron's behavior. This project shows that it is possible to add support for the release, diffusion and sensing of various compounds in the medium to the model.

5.1.2 An Exploration of Topological Structures Generated by Local Rules

We presented a study of topological structures generated by local rules in a biologically-inspired neural network. We constructed a network of Perceptrons, initialized the network to a given topology (we examined random networks, Small World networks and Scale-Free networks), stimulated the network with random noise and observed the changes in topology. The topological changes were a result of Hebbian learning and Neuronal Regulation, two local mechanisms based on established neurobiological research.

Part of our work replicated that done by Szirtes, Palotai and Lorincz in [65]. We demonstrated that the ABNNSim framework can produce similar results. The replication also served to validate the results of the HebbNets study, that various global network structures can be derived from the application of global rules to neurons in a neural network. All structures are derived using random noise as input. The resulting changes in network topology are a function of pruning. The pruning of connections in the network is guided by Hebbian learning and neuronal regulation, which hints at the value of these mechanisms in the development of topology in biological neural networks.

We evaluated the topological changes that were observed as a result of various initial conditions. We explored the parameter space over a large range of values in order to determine the robustness of our results. We evaluated the effects of different numbers of connections, starting topologies and link distributions. We also conducted a sensitivity analysis on the different parameters in the equations for Hebbian learning and Neuronal Regulation, in order to find suitable values.

5.2 Future Work

This work provide a solid base on which to build more detailed and biologically plausible neural network simulations. The components currently in ABNNSim already provide researchers with a useful base, however there is room for the framework to grow. There are several areas that I would like to explore further, both in the development of the framework and in the exploration of biologically inspired neural networks.

First, we would like to develop a new version of ABNNSim, with different neuron types and a more flexible method of sending input to and from the network. In order to make the framework more useful for other researchers, we would like to make it as comprehensive as is reasonably possible. Several biological structures with relevance to neural modeling, such as Glial cells and various Neuron components could be added to the framework. Refined input to and output from the networks will allow researchers to use constructed networks for computation and validation against biological neural networks. We would like to add support for restarting the simulation given a snapshot of a prior simulation. This could be important when exploring dynamic networks. Another important addition to ABNNSim would be to include support for writing simulation output data to a database. Such work would simplify the task of categorizing and analyzing simulation results.

Second, we would like to be able to better calibrate the model with data on the functioning of biological neurons. Should a researcher want to create a network of, for example, frog neurons, it would be beneficial to have a pre-calibrated model of a frog neuron. Making it easier to calibrate and validate the model will ensure that ABNNSim is a relevant and useful tool for researchers.

Third, we would like to develop extensions to ABNNSim that allow a model or series of models to be run across a cluster of machines in parallel. Currently there exist methods for running simulations in parallel, but each simulation must run independently. This would allow researchers to simulate very large scale neural networks. The ability to have simulations span multiple machines will allow researchers the ability to study networks of a unprecedented size, allowing the modeling of not only the entire brain of an animal, but potentially a colony of interacting animals.

APPENDIX A

ANT BUILD FILE

A.1 Overview

This section contains the ANT build file for the construction and distribution of the project.

A.2 Build File

```
 <?xml version="1.0"?>


<!-- build.xml - a simple Ant buildfile -->
<!-- Author: Tim Schoenharl, based on sample from OReilly "Ant: The
Definitive Guide" -->


<project name="Agent Based Neural Network Sim" default="all" basedir=".">


  <!-- The Repast lib directory -->
  <property name="repast.lib.dir" Value="/Applications/repast-2.0.2/lib"/>


  <!-- Libraries needed to execute RePast sims -->
  <property name="class.path" value="${repast.lib.dir}/repast.jar
 ${repast.lib.dir}/junit.jar ${repast.lib.dir}/colt.jar "/>

<!-- Name of executable jar file -->
<property name="jar.file" value="ABNNSim.jar"/>

<!-- Name of executable jar file -->
<property name="jar.nogui.file" value="ABNNSimNoGUI.jar"/>

<!-- Name of cluster targeted executable jar file -->
<property name="export.jar.file" value="ClusterABNNSim.jar"/>

<!-- The directory containing source code -->
<property name="src.dir" value="src/abnnsim"/>

<!-- Temporary build directories -->
<property name="build.dir" value="build"/>
<property name="build.classes" value="${build.dir}/classes"/>
<property name="build.lib" value="${build.dir}/lib"/>
<property name="build.tests" value="${build.dir}/tests"/>
<property name="build.docs" value="${build.dir}/docs"/>

<!-- Target to create the build directories prior to the -->
<!-- compile target. -->
<target name="prepare">
  <mkdir dir="${build.dir}"/>
  <mkdir dir="${build.classes}"/>
  <mkdir dir="${build.lib}"/>
  <mkdir dir="${build.tests}"/>
  <mkdir dir="${build.docs}"/>
  <copy file="glass.cod" todir="${build.classes}"/>
</target>

<target name="clean" description="Remove all generated files.">
  <delete dir="${build.dir}"/>
</target>

<target name="compile" depends="prepare"
        description="Compiles all source code except tests.">
  <javac srcdir="${src.dir}" destdir="${build.classes}"
         excludes="tests" />
</target>

<target name="jar" depends="compile"
        description="Generates ABNNSim.jar in the 'dist' directory.">
  <!-- Exclude unit tests from the final JAR file -->
  <jar jarfile="${build.lib}/${jar.file}" Basedir="${build.classes}">
    <manifest>
      <attribute name="Main-Class" value="abnnsim.ABNNSimModel"/>
      <attribute name="Class-Path" value="${class.path}"/>
    </manifest>
  </jar>
</target>

<target name="jarnogui" depends="compile"
```

```
        description="Generates jar file in the ’dist’ directory.">
  <!-- Exclude unit tests from the final JAR file -->
  <jar jarfile="${build.lib}/${jar.nogui.file}"
   Basedir="${build.classes}">
    <manifest>
      <attribute name="Main-Class" value="abnnsim.ABNNModelNoGUI"/>
      <!-- attribute name="Class-Path" value="${class.path}"/-->
      <attribute name="Class-Path"
        value="/usr/local/repast-2.0/lib/repast.jar :
        /usr/local/repast-2.0/lib/colt.jar :
        /home/tschoenh/Research/classes12.jar : ./"/>
    </manifest>
  </jar>
</target>

<target name="export" depends="compile"
        description="Generates ABNNSim.jar in the ’dist’ directory and
        uses scp to copy the file to apocalypse.cse.nd.edu">
  <!-- Exclude unit tests from the final JAR file -->
  <jar jarfile="${build.lib}/${export.jar.file}"
   Basedir="${build.classes}">
    <manifest>
      <attribute name="Main-Class" value="abnnsim.ABNNSimModel"/>
      <attribute name="Class-Path"
        value="/usr/local/repast-2.0/lib/repast.jar :
        /usr/local/repast-2.0/lib/colt.jar :
        /home/tschoenh/Research/classes12.jar : ./"/>
    </manifest>
  </jar>
  <exec executable="scp" dir="${build.lib}" timeout="6000">
    <arg line="${export.jar.file} tschoenh@apocalypse.cse.nd.edu:" />
  </exec>
</target>

<target name="test" depends="compile" description="Runs junit tests">
  <java fork="true" classname="junit.textui.TestRunner"
        dir="${build.classes}"
        classpath="${class.path}" failonerror="true">
    <arg value="abnnsim.tests.AllTests"/>
    <!-- arg value="junit.samples.AllTests"/ -->
  </java>
</target>

<target name="testgui" depends="compile" description="Runs junit tests">
  <java fork="true" classname="junit.swingui.TestRunner"
        dir="${build.classes}"
        classpath="${class.path}" failonerror="true">
    <arg value="abnnsim.tests.AllTests"/>
    <!-- arg value="junit.samples.AllTests"/ -->
  </java>
</target>

<target name="exec" depends="jar" description="Runs the executable jar">
  <echo message="Running  ${build.lib}/${jar.file}"/>
  <java fork="true" jar="${build.lib}/${jar.file}"/>
</target>


<target name="nogui" depends="jarnogui"
  description="Runs the executable jar">
  <echo message="Running ${build.lib}/${jar.nogui.file}"/>
  <java fork="true" jar="${build.lib}/${jar.nogui.file}"/>
</target>


<target name="batch" depends="jar"
 description="Runs the sim in batch mode">
  <tstamp>
    <format property="now" pattern="hh:mm:ss:SSS aa" />
  </tstamp>
  <echo> now = ${now} </echo>
  <java fork="true" classpath ="${build.lib}/${jar.file};${class.path}"
    classname="uchicago.src.sim.engine.SimInit">
      <arg line="-b"/>
      <arg value="abnnsim.SFNNModel"/>
      <arg value="ABNNSimParams.pf"/>
  </java>
  <tstamp>
    <format property="then" pattern="hh:mm:ss:SSS aa" />
  </tstamp>
  <echo> now = ${then} </echo>
</target>
```

```
  <target name="doc" depends="prepare"
   description="Creates javadoc of project">
    <javadoc excludepackagenames="tests.*" destdir="${build.docs}">
       <package name="abnnsim.*"/>
       <sourcepath location="src"/>
       <classpath location="${class.path}"/>
    </javadoc>
  </target>



  <target name="all" depends="clean,test,jar"
        description="Cleans, compiles, tests then builds the JAR file."/>

</project>
```

APPENDIX B

SAMPLE SIMULATION OUTPUT

B.1 Overview

This Appendix contains an example of the type of output file generated by ABNNSim. The file given here is reduced in size from a standard output file. This file contains only information from the beginning and end of the simulation, but output files usually contain information from every time step.

B.2 Naming Convention

In order to ensure that subsequent runs of the program do not overwrite existing output files, we have created a naming convention that uniquely identifies each output file. The names are generated with the following Java code:

```
String fileName = "ABNNSimOutput" + getRngSeed() + "RngSeed" + \\
      getMeanNumLinks() + "Links" + getNetworkType() + "NetType" + \\
      ".dat";
```

This convention not only creates unique file names, but allows users to search and sort the output files based on the simulation parameters. With standard Unix tools like sed and awk, these flat ASCII files can be used in a manner similar to a database system, but in a significantly more portable way.

B.3 Sample Output File

```
Clock is 2


Network Type 1

Clustering Coefficient is: 0.0885275557775558

Global Harmonic Mean Distance is: 2.140771975348663

Avg Local Harmonic Mean Distance is: 9.730779541632263


In Degree Hist:

Tue Mar 08 15:37:43 EST 2005

RngSeed: 1110314257229

ActionThreshold: 0.2

Alpha: 0.7

Gamma: 1.0

GuiFlag: false

MaxTicks: 100

MeanNumLinks: 10

NetworkType: 1

Nu: 0.1

NumNeurons: 100

NumberOfInputNeurons: 1

NumberOfOutputNeurons: 4

OutputFlag: true

PruningThreshold: 0.01

RelaxationTime: 1.3
```

```
RewireProb: 0.2

Schedule: uchicago.src.sim.engine.Schedule@16e46f5

SigmaNumLinks: 1.5

TestingFile: unknown

TrainingFile: unknown

WTheta: 0.1

WorldXSize: 500

WorldYSize: 500


In Degree Distribution:

   Entries=100, ExtraEntries=0

   Mean=9.57, Rms=3.00418

   MinBinHeight=0, MaxBinHeight=14

   Axis: Bins=101, Min=0, Max=101


Heights:

0 1 2 3 4 5 6  7 8  9  10 11 12 13 14 15 16 17 18 19 20 21 22

----------------------------------------------------------------

0 0 0 0 2 6 10 8 12 14 13 10 7  7  5  3  1  1  1  0  0  0  0


Out Degree Hist:

Tue Mar 08 15:37:43 EST 2005

RngSeed: 1110314257229

ActionThreshold: 0.2

Alpha: 0.7
```

```
Gamma: 1.0

GuiFlag: false

MaxTicks: 100

MeanNumLinks: 10

NetworkType: 1

Nu: 0.1

NumNeurons: 100

NumberOfInputNeurons: 1

NumberOfOutputNeurons: 4

OutputFlag: true

PruningThreshold: 0.01

RelaxationTime: 1.3

RewireProb: 0.2

Schedule: uchicago.src.sim.engine.Schedule@16e46f5

SigmaNumLinks: 1.5

TestingFile: unknown

TrainingFile: unknown

WTheta: 0.1

WorldXSize: 500

WorldYSize: 500


Out Degree Distribution:

   Entries=100, ExtraEntries=0

   Mean=9.18, Rms=2.174304

   MinBinHeight=0, MaxBinHeight=30
   Axis: Bins=101, Min=0, Max=101


Heights:
0 1 2 3 4 5 6 7 8  9  10 11 12 13 14 15 16 17 18 19 20 21 22
----------------------------------------------------------------
0 4 0 0 0 0 1 6 15 30 17 19 7  1  0  0  0  0  0  0  0  0  0



Clock is 100


Network Type 1
Clustering Coefficient is: 0.0
Global Harmonic Mean Distance is: 44.27064107141005
Avg Local Harmonic Mean Distance is: 0.0
In Degree Hist:
Tue Mar 08 15:38:32 EST 2005
RngSeed: 1110314257229
ActionThreshold: 0.2
Alpha: 0.7
Gamma: 1.0
GuiFlag: false
MaxTicks: 100
MeanNumLinks: 10
```

```
NetworkType: 1

Nu: 0.1

NumNeurons: 100

NumberOfInputNeurons: 1

NumberOfOutputNeurons: 4

OutputFlag: true

PruningThreshold: 0.01

RelaxationTime: 1.3

RewireProb: 0.2

Schedule: uchicago.src.sim.engine.Schedule@16e46f5

SigmaNumLinks: 1.5

TestingFile: unknown

TrainingFile: unknown

WTheta: 0.1

WorldXSize: 500

WorldYSize: 500


In Degree Distribution:

   Entries=100, ExtraEntries=0

   Mean=1.36, Rms=1.228983

   MinBinHeight=0, MaxBinHeight=55

   Axis: Bins=101, Min=0, Max=101

Heights:

0  1  2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

-----------------------------------------------------------------
```

```
18 55 13 6 4 3 1 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0


Out Degree Hist:


Tue Mar 08 15:38:32 EST 2005

RngSeed: 1110314257229

ActionThreshold: 0.2

Alpha: 0.7

Gamma: 1.0

GuiFlag: false

MaxTicks: 100

MeanNumLinks: 10

NetworkType: 1

Nu: 0.1

NumNeurons: 100

NumberOfInputNeurons: 1

NumberOfOutputNeurons: 4

OutputFlag: true

PruningThreshold: 0.01

RelaxationTime: 1.3

RewireProb: 0.2

Schedule: uchicago.src.sim.engine.Schedule@16e46f5

SigmaNumLinks: 1.5

TestingFile: unknown

TrainingFile: unknown
```

```
WTheta: 0.1

WorldXSize: 500

WorldYSize: 500


Out Degree Distribution:

   Entries=100, ExtraEntries=0

   Mean=0.93, Rms=0.255147

   MinBinHeight=0, MaxBinHeight=93

   Axis: Bins=101, Min=0, Max=101

Heights:

0 1  2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22

---------------------------------------------------------------

7 93 0 0 0 0 0 0 0 0 0  0  0  0  0  0  0  0  0  0  0  0  0
```

BIBLIOGRAPHY

1. Reka Albert and Albert-Laszlo Barabasi. Topology of evolving networks: Local events and universality. *Physical Review Letters*, 85(24), 2000.

2. Reka Albert, Hawoong Jeong, and Albert-Laszlo Barabasi. Diameter of the world wide web. *Nature*, 401:130–131, 1999.

3. J. Anderson. *An Introduction to Neural Networks.* MIT Press, Cambridge, MA, 1996.

4. Steven C. Bankes. Agent-based modeling: A revolution? *Proceedings of the National Academy of Sciences*, 99:7199–7200, May 2002.

5. A. Barabasi. *Linked: The New Science of Networks.* Perseus Publishing, Cambridge, MA, 2002.

6. James A. Bednar, Yoonsuck Choe, Judah De Paula, Risto Miikkulainen, Jefferson Provost, and Tal Tversky. Modeling cortical maps with topographica. *Neurocomputing*, in press, 2004.

7. Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., 1995.

8. Bela Bollobas. *Random Graphs Second Edition.* Cambridge University Press, United Kingdom, 2001.

9. Eric Bonabeau. Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Science*, 99:7280 – 7287, May 2002.

10. James M. Bower, David Beeman, and Michael Hucka. The genesis simulation system. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks.* The MIT Press, 2002.

11. R. Cancho and R. Sole. Optimization in complex networks, 2001.

12. R. C. Cannon, M. E. Hasselmo, and R. A. Koene. From biophysics to behavior catacomb2 and the design of biologically-plausible models for spatial navigation. *Neuroinformatics*, 1(1):3–42, February 2003.

13. Gail A. Carpenter and Stephen Grossberg. *Adaptive resonance theory (ART)*, pages 79–82. MIT Press, 1998.

14. S. Chalup. Issues of neurodevelopment in biological and artificial neural networks. In *Proceedings of the Fifth Biannual Conference on Artificial Neural Networks and Expert Systems (ANNES 2001)*, pages 40–45, 2001.

15. K. Mani Chandy and Jayadev Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Commun. ACM*, 24(4):198–206, 1981.

16. D. Horn Chechik, G. and E. Ruppin. Neuronal regulation and hebbian learning. In M. Arbib, editor, *The handbook of brain theory and neural networks*, Cambridge, MA, 2000. MIT Press.

17. G. Chechik, I. Meilijson, and E. Ruppin. Neuronal regulation: A mechanism for synaptic pruning during brain maturation. *Neural Computation*, 11(8), 1999.

18. T. Cickovski, C. Huang, R. Chaturvedi, T. Glimm, H.G.E. Hentschel, M. Alber, J. A. Glazier, S. A. Newman, and J. A. Izaguirre. A framework for three-dimensional simulation of morphogenesis. *IEEE Transactions on Computational Biology and Bioinformatics*, 2004. Submitted, preprint at.

19. A. Cochocki and Rolf Unbehauen. *Neural Networks for Optimization and Signal Processing.* John Wiley & Sons, Inc., 1993.

20. N. Collier. Repast: An extensible framework for agent simulation. Technical report, University of Chicago, 2003.

21. Nick Collier. Repast homepage. http://repast.sourceforge.net.

22. Howard Demuth and Mark Beale. *Neural Network Toolbox Users Guide.* The Math Works, 2004.

23. C. L. Barrett et al. Transportation analysis simulation system. Technical report, Los Alamos National Laboratory, 2004.

24. Yongqin Gao. Topology and evolution of the open source software community. Master's thesis, University of Notre Dame, Notre Dame IN, 2003.

25. W. Gerstner and W.M. Kistler. Mathematical formulations of hebbian learning. *Biological Cybernetics*, 87:404–415, 2002.

26. Andrew Gillies and David Sterratt. A neuron programming tutorial. Technical report, 2004.

27. Nigel H. Goddard and Greg Hood. Large-scale simulation using parallel genesis. In James M. Bower and David Beeman, editors, *The Book of GENESIS*. Springer-Verlag, 1998.

28. James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

29. Mike Hasselmo. Catacomb2: Components and tools for accessible computer modeling in biology. http://www.compneuro.org/catacomb/index.shtml, 2001.

30. Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, 1998.

31. D.O. Hebb. *The Organization of Behavior*. John Wiley and Sons, NY, 1949.

32. M. L. Hines and N. T. Carnevale. The neuron simulation environment. *Neural Computation*, 9:1179–1209, 1997.

33. M. L. Hines and N. T. Carnevale. The neuron simulation environment. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. The MIT Press, 2002.

34. J.J. Hopfield and A.V.M. Herz. Rapid local synchronizations of action potentials: Towards computation with coupled integrate-and-fire neurons. In *Proceedings of the National Academy of Science*, volume 92, pages 6655–6662, 1995.

35. F. W. Howell, Jonas Dyhrfjeld-Johnsen, Reinoud Maex, Nigel H. Goddard, and Erik De Schutter. A large-scale model of the cerebellar cortex using pgenesis. *Neurocomputing*, 32-33:1041–1046, 2000.

36. Alfred Hubler. Lecture notes for *Nonlinear Dynamics: An Introduction*, 2004.

37. S. L. Hung and Hojjat Adeli. Object-oriented backpropagation and its application to structural design. *Neurocomputing*, 6(1):45–55, 1994.

38. J. A. Izaguirre, R. Chaturvedi, C. Huang, T. Cickovski, J. Coffland, G. Thomas, G. Forgacs, M. Alber, G. Hentschel, S. A. Newman, and J. A. Glazier. CompuCell, a multimodel framework for simulation of morphogenesis. *Bioinformatics*, 20(7):1129–1137, 2004.

39. William James. *Principles of Psychology*. Henry Holt, 1890.

40. David R. Jefferson, Brian Beckman, Frederick Wieland, Leo Blume, Mike Di Loreto, Phil Hontalas, Pierre Laroche, Kathy Sturdevant, Jack Tupman, Van Warren, John J. Wedel, Herb Younger, and Steve Bellenot. Distributed simulation and the time warp operating system. In *SOSP*, pages 77–93, 1987.

41. H. Jeong, B. Tombor, R. Albert, Z. Oltvai, and A. Barabasi. The large-scale organization of metabolic networks. *Nature*, 407:651, 2000.

42. Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

43. Rolf Kotter. Online retrieval, processing, and visualization of primate connectivity data from the cocomac database. *Neuroinformatics*, 2(2):127–144, 2004.

44. Marcel Kunze and Johannes Steffens. The neural network objects. *Proceedings of the 5th Annual AIHENP Workshop*, 1996.

45. Y. LeCun, J. Denker, S. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems II*, San Mateo, CA, 1990. Morgan Kauffman.

46. W. Maass. Networks of spiking neurons: The third generation of neural networks. In *Australian Conference on Neural Networks*, 1996.

47. Greg Madey, Matthias Scheutz, Sunny Boyd, Tim Schoenharl, and John Korecki. Tmans home page. http://tmans.sourceforge.net, 2004.

48. Massimo Marchiori and Vito Latora. Harmony in the small-world. *Physica A*, 285:539–546, 2000.

49. Paolo Marrone. *JOONE: The Complete Guide*. 2005.

50. W. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

51. Stanley Milgram. The small world problem. *Psychology Today*, pages 60 – 67, May 1967.

52. Michael C. Mozer and Paul Smolensky. *Skeletonization: a technique for trimming the fat from a network via relevance assessment*, pages 107–115. Morgan Kaufmann Publishers Inc., 1989.

53. Athanasius F. M.Marée and PaulienHogeweg. How amoeboids self-organize into a fruiting body: Multicellular coordination in dictyostelium discoideum. *Proceedings of the National Academy of Science*, 98(7):3879–3883, 2001.

54. K. Nagel, R. Beckman, and C. Barrett. Transims for urban planning. Technical Report LA-UR 984389, Los Alamos National Laboratory, Los Alamos, NM., 1999.

55. Mitchel Resnick. *Turtles, Termites and Traffic Jams.* MIT Press, 1994.

56. D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

57. Thomas C. Schelling. Dynamic models of segregation. *Journal of Mathematical Sociology*, 1:143–186, 1973.

58. Timothy Schoenharl. Agent based modeling aproach to self-organizing neural networks. In *Institute for Math and its Applications: Workshop on Agent-Based Modeling*, November 2003.

59. Timothy Schoenharl. Exploring alternate topologies in neural networks. In *Proceedings of Swarmfest 2003*, March 2003.

60. Timothy Schoenharl. Using agent based modeling in the exploration of self-organizing neural networks. In *Proceedings of the Workshop on Agent/Swarm Programming*, October 2003.

61. Timothy Schoenharl. Agent based exploration of self-organizing neural networks. In *Understanding Complex Systems Symposium 2004*, May 2004.

62. John Scott. *Social Network Analysis.* SAGE Publications, 1991.

63. R. Segev and E. Ben-Jacob. From neurons to brain: Adaptive self-wiring of neurons. *Advances in Complex Systems*, 1:67–78, 1998.

64. Jimmy Shadbolt and John G. Taylor. *Neural networks and the financial markets: predicting, combining and portfolio optimisation.* Springer-Verlag, 2002.

65. G. Szirtes, Zs. Palotai, and A. Lorincz. Hebbnets: Dynamic network with hebbian learning rule. http://arxiv.org/pdf/nlin.AO/0212010, 2002.

66. G. Szirtes, Zs. Palotai, and A. Lorincz. Emergence of scale-free properties in hebbian networks. http://arxiv.org/pdf/nlin.AO/0308013, 2003.

67. Giorgio Valentini and Francesco Masulli. Neurobjects: An object-oriented library for neural network development. *TBA*, 2001.

68. http://www.wormbase.org/.

69. Wayne W. Wakeland, Edward Gallaher, Louis Macovsky, and C. Athena Aktipis. A comparison of system dynamics and agent-based simulation applied to the study of cellular receptor dynamics. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, 2004.

70. Duncan Watts. *Small Worlds: The Dynamics of Networks Between Order and Randomness.* Princeton University Press, Princeton, NJ, 1999.

71. Duncan Watts and Steven Strogatz. Collective dynamics of 'small world' networks. *Nature*, 393:440–442, 1998.

72. Matthew A. Wilson, Upinder S. Bhalla, John D. Uhley, and James M. Bower. Genesis: A system for simulating neural networks. In *NIPS*, pages 485–492, 1988.

73. Robin J Wilson. *Introduction to Graph Theory.* John Wiley & Sons, Inc., 1986.

74. J. Xu and G. Madey. Exploration of the open source software community. In *Proceedings of the NAACSOS Conference 2004*, June 2004.

75. Andreas Zell, Niels Mache, Ralf Huebner, Michael Schmalzl, Tilman Sommer, and Thomas Korb. SNNS: Stuttgart neural network simulator. Technical report, University of Stuttgart, Stuttgart, 1992.

This document was prepared & typeset with pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v1.0[2004/06/15]) provided by Sameer Vijay.*