ABSTRACT

Title of dissertation: ADAPTING SWARM INTELLIGENCE FOR THE SELF-ASSEMBLY AND OPTIMIZATION OF NETWORKS

Charles Eugene Martin, Doctor of Philosophy, 2011

Dissertation directed by: Professor James A. Reggia
Department of Computer Science

While self-assembly is a fairly active area of research in swarm intelligence and robotics, relatively little attention has been paid to the issues surrounding the construction of network structures. Here, methods developed previously for modeling and controlling the collective movements of groups of agents are extended to serve as the basis for self-assembly or "growth" of networks, using neural networks as a concrete application to evaluate this novel approach. One of the central innovations incorporated into the model presented here is having network connections arise as persistent "trails" left behind moving agents, trails that are reminiscent of pheromone deposits made by agents in ant colony optimization models. The resulting network connections are thus essentially a record of agent movements.

The model's effectiveness is demonstrated by using it to produce two large networks that support subsequent learning of topographic and feature maps. Improvements produced by the incorporation of collective movements are also examined through computational experiments. These results indicate that methods for

directing collective movements can be extended to support and facilitate network self-assembly.

Additionally, the traditional self-assembly problem is extended to include the generation of network structures based on optimality criteria, rather than on target structures that are specified *a priori*. It is demonstrated that endowing the network components involved in the self-assembly process with the ability to engage in collective movements can be an effective means of generating computationally optimal network structures. This is confirmed on a number of challenging test problems from the domains of trajectory generation, time-series forecasting, and control. Further, this extension of the model is used to illuminate an important relationship between particle swarm optimization, which usually occurs in high dimensional abstract spaces, and self-assembly, which is normally grounded in real and simulated 2D and 3D physical spaces.

ADAPTING SWARM INTELLIGENCE FOR THE SELF-ASSEMBLY AND OPTIMIZATION OF NETWORKS

by

Charles Eugene Martin

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2011

Advisory Committee:
Professor James A. Reggia, Chair/Advisor
Professor Edward Ott
Assistant Professor Michelle Girvan
Professor Doron Levy
Associate Professor Allen Stairs

Acknowledgments

It's said that we tend to underestimate the role that the people in our lives play in the accomplishments we achieve. In light of this, I would first like to thank my family. My path towards becoming a scientist began at a young age, and I owe you all a substantial debt of gratitude for your loving support and encouragement throughout the years.

Kelly Dickson, I love you, and thank you for sharing your life with me. I know that mine is far richer from having you in it.

I am grateful to my advisor, Professor James Reggia, for his support, advice, and guidance during my graduate career. Jim, you have played a significant role in making my time at the University of Maryland a wonderful experience.

To the members of the Biologically-Inspired Computing Group that I was fortunate enough to get to know: Thank you for the stimulating conversations, assistance, and camaraderie. I wish you all the best in your current and coming endeavors.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Network Growth

Given a set of components, the *self-assembly problem* entails the design of local control mechanisms that enable these components to self-organize into a given target structure, without individually pre-assigned component positions or central control mechanisms. Issues surrounding self-assembly have been a very active research area in swarm intelligence over the last several years, with recent work spanning both computer simulations [2, 49, 50, 70, 79, 128] and physical robotics [5, 45, 80, 93, 129]. A substantial part of this work has taken inspiration from nature, including self-assembly in natural physical systems [131] and closely-related collective construction in which passive components are manipulated by multiple autonomous agents, such as with nest construction by social insects [8].

The research presented in this dissertation is concerned with the self-assembly of network architectures. Like other structures studied in past work on self-assembly, networks have discrete components that need to position themselves in appropriate spatial locations. Examples of real-world network components include resistors and capacitors in electrical circuits, neuronal cell bodies in neural networks, generators and transformers in power grids, and mixing stations in a chemical manufacturing plant. However, unlike with most past work on self-assembly, a major aspect of

constructing networks is establishing the connections that are needed between these discrete components: the wires in circuits, axons (neuron output connections) in neural networks, transmission lines in power grids, pipes in chemical plants, etc. Relatively little consideration has been given to how such connectivity might arise in past swarm intelligence work related to self-assembly, although progress has been made in engineering, such as the self-assembly of electrical circuits using physical processes [44].

For concreteness, the work presented herein focuses on the self-assembly of neural network architectures. While inspiration is taken from neuroscience, the intent is not to create a veridical model of processes in neurodevelopmental biology, nor to model any specific neuroscientific data. Instead, the goal is to extend current methods available for the self-assembly of network structures by examining the extent to which recently-developed swarm intelligence techniques for directing collective movements can be adopted to produce more powerful network self-assembly processes. Neural networks are used as a concrete application to assess this issue. The question of how the principles and methods studied and developed in this work may be useful in future computational studies of neurobiological circuit self-organization during development is deferred to the final chapter.

There has been only very limited past work in engineering and computer science on how to control the growth and development of artificial neural networks, as follows. Some past artificial life models [33, 34, 73] that focus on understanding the principles of neurogenesis rather than on biological fidelity, are similar to the work introduced here in that they explicitly incorporate geometric relations (not just

network topology) in simulating the growth of neural connections through physical space. However, unlike the work presented here they are limited to growth in a two-dimensional space (for an exception, see [111]), typically do not incorporate cell migration and division, do not consider axon-axon interactions during network assembly (for an exception to the latter, see [136]), and are generally applied to relatively small feedforward networks. Comparatively, the method proposed in this dissertation involves network growth in a three-dimensional space, and cell migration and division, along with axon-axon interactions, play an important role in the growth process. Moreover, the largest networks grown by past developmental models that incorporate continuous neural growth consist of about 50 neurons and 100 connections (e.g., [33] and [73]). In contrast, it will be shown that the approach presented here can readily grow recurrent neural networks with over 600 neurons and nearly 50,000 connections. Additionally, past work has focused on feedforward networks rather than recurrent networks which tend to be more difficult to grow. In contrast, the work here is focused on growing recurrent networks. Further, the large recurrent networks grown using this approach are based on target structures with precisely specified topologies. Comparatively, the target structures for most of the networks grown by past developmental models are instead specified in terms of general qualities, such as the existence of connections between two layers, as opposed to specific patterns of connectivity. Relatively small networks based on nonspecific target structures are typical of past related work due in large part to the difficulty of controlling the characteristics of the networks grown by these models. To address these limitations of past models, including the restriction of growth to a

two-dimensional space, the absence of direct interactions among growing recurrent connections, and a lack of controllability, the use of collective movements as a means of producing reliable network self-assembly is examined.

Other related past work in computer science and engineering, where research on artificial neural networks is concerned with application-related performance, has largely ignored the issues of neural network growth, development and self-assembly, with two exceptions. First, a number of computational techniques have been created to optimize neural network architectures by adding/deleting nodes/connections dynamically during learning. Examples are Marchand's algorithm [91], cascade correlation [30], optimal brain damage [85], the upstart algorithm [36] and recursive deterministic perceptrons [26]. Unlike the approach taken here, these past network construction methods do not involve growth or self-assembly in a physical space, and will not be considered further. Second, a technique known as *developmental encoding* has been used by researchers evolving neural network architectures with genetic algorithms/programming. Examples include L-systems [17], matrix rewriting systems [78], cellular encoding [47], and the closely related, descriptive encoding [72]. Some of the models based on these methods have incorporated a significant amount of biological detail [3, 14, 25]. Again, these latter approaches generally do not involve growth or self-assembly in physical space; they typically consider just the topology of networks and not the geometrical relations involved.

To date, the vast majority of research in the field of self-assembly, involving network structures or otherwise, has focused on the self-organization of components into pre-specified target structures. In this dissertation the self-assembly problem

is extended to also include the generation of structures based on optimality criteria that are defined in terms of the quality or performance of the emerging structures. Specifically, consider the task of getting a neural network to self-assemble that is capable of effectively solving some problem. Assuming that little information is given regarding the structure of the optimal network solution, the self-assembly process becomes, in essence, a search process. A process that is driven by the need to find a good solution to a specified computational task differs from past work found in the self-assembly literature, which is primarily focused on the generation of pre-specified target structures. The problem of optimizing neural networks for computational tasks has been tackled in the past using techniques from genetic algorithms and evolutionary programming. However, unlike the work presented here, the network encodings utilized in these past techniques do not undergo any type of growth, and interaction between different networks in a population occurs only indirectly through the processes of selection and crossover. The central question that arises is, by what means is the exploration of solution network structures produced as part of the self-assembly process? Here, it will be demonstrated that endowing the network components involved in the self-assembly process with the ability to exhibit swarm intelligence can be an effective means of generating computationally optimal network structures.

This dissertation presents a novel methodology in which neural network self-assembly arises in a three-dimensional space from direct interactions between components of the developing network. This approach is motivated by numerous potential benefits and applications. Physical realizations of self-assembly require that struc-

tural components move through space during the development process, but how this should occur with networks is poorly understood at present. Since the approach introduced here incorporates growth in a continuous three-dimensional space it could be useful for studying the science of self-assembly, especially as it relates to network structures. Furthermore, the developmental processes implemented in the model makes it suitable for studying phenomena such as self-repair and plasticity, which involve growth and changes in the positions of network components in response to alterations in the network structure. There are also many potential neuroscience applications, in which events that occur during the growth process influence the network that ultimately develops, for example, interactions between growing axons, and between axons and cells such as activity dependent development. Lastly, from the perspective of swarm intelligence based optimization, the continuous nature of the growth process and the local interactions among the network components can be utilized by modifying the model so that input to a growing network results in the network's weights and topology being optimized for a particular problem.

The model developed through the research of this dissertation is intended to grow networks that are more directly inspired by neurobiological processes than most traditional artificial neural networks in computer science and engineering, in the sense that network architectures are defined by their geometry in addition to their topology. Also, the networks grown using the model reproduce the deterministic topological representations of past artificial neural network models in a statistical sense. Agents represent two distinct types of entities: *cells*, which roughly correspond to neuron cell bodies, and *growth cones*, which are named after the special-

ized structures at the leading tips of growing axons in biology. Both types of agents move, divide, and exert local "forces" upon one another, the latter being analogous to influences used to produce flock-like collective movement patterns in past swarm intelligence systems [108, 110]. A central innovation of this approach is that network connections arise from "trails" deposited by moving growth cone agents, something that is reminiscent of pheromone trails produced by ants [8, 20, 21, 35]. Topographic regularity in the developing connections emerges from the collective movements of populations of growth cone agents. These inter-agent influences are integrated during network development with rule-based control mechanisms that govern behaviors such as cell division and axon branching, greatly facilitating one's ability to exert control over resultant network structures. Here, the focus is on the improvements that collective movements bring to the model, as opposed to the importance of the rule set. This was done mostly because rule-based developmental models, such as L-systems, have already been applied to neurogenesis, whereas techniques employing swarm intelligence mediated via local "forces" have not. However, having now used the model to grow many different networks, experience indicates that the rule-based component of the model does play an important role by making it easier to predict and control discontinuous actions, tailor the agents' dynamics, and incorporate only local interactions. Additionally, the model is modified so that given a computational problem it is able to grow networks that are effective solutions. This is accomplished by having multiple networks grow simultaneously and allowing growth cones from different networks to interact in a manner that is similar to the simple, local interactions utilized in particle swarm optimization (PSO), which is a powerful and very

generally applicable optimization method based on swarm intelligence [28]. This modification of the model is used to illuminate an important relationship between PSO, which usually occurs in high dimensional abstract spaces, and self-assembly, which is normally grounded in real and simulated 2D and 3D physical spaces.

The effectiveness of the proposed approach is demonstrated by using it to produce two large networks (large relative to what has been done in past related work) that support the emergence, during subsequent learning, of topographic and feature maps. The development process that these networks undergo involves a significant amount of concurrent axonal growth, which is biologically plausible but complicates the environment in which growth takes place. Also, because the approach presented here accounts for network geometry as well as topology, there is the added challenge of dealing with inhomogeneities in the positions of the neurons. Improvements produced by the incorporation of collective movements are further expounded through computational experiments that examine the robustness of the model. The effectiveness of the adaptation of the model to growing networks that solve specific computational problems is illustrated by growing solution networks to three challenging benchmark problems from the domains of trajectory generation, time-series forecasting, and control. The results suggest that the proposed approach has substantial potential as a methodology for advancing the understanding of network self-assembly and its relationship to the concepts of swarm intelligence, as well as being a useful technique for optimizing neural networks for computational tasks. To my knowledge, this is the first explicit recognition that the use of swarm intelligence methods can serve as the basis for simulating growth and formation of

neural networks, including those with recurrent connectivity.

1.2 Contributions

The five key contributions of the work presented in this dissertation are identified below.

- A readily adaptable framework for the study and application of network self-assembly has been developed. In this framework the components of a growing network utilize only local information and exhibit collective movements in response to simple, local interactions. Two common means of producing growth in developmental models, forces and conditional rules, are the driving mechanisms underlying network growth in this model. A software-based simulator has been written that implements the model. Owing to the flexible framework upon which the simulator is built, it can undoubtedly be adapted for studying a wide range of developmental phenomena in the nervous system as well as the self-assembly of different types of networks.

- The simulator has been critically evaluated by using it to grow two networks that are substantially larger than those generated by past developmental models that feature network growth of a continuous nature. Furthermore, the two grown networks are based on target networks that were specified to a much higher degree of detail than what has normally been tackled with past models of network growth. The ability of both networks to produce computational behavior that is qualitatively similar to that observed in the target networks

serves to illustrate the effectiveness of network self-assembly based on swarm intelligence brought about through collective movements.

- Computational experiments have been performed to test the hypothesis that incorporating collective movements among self-assembling network components improves the robustness of the growth process with respect to variability in the rules. The experimental results support this hypothesis by demonstrating that under perturbations of the rule-set the quality of the grown networks deteriorates far less when the network components can exhibit collective movements. Further, the results provide convincing evidence that incorporating swarm intelligence in the form of collective movements into the self-assembly process can increase the ability to control the characteristics of the structures that emerge.

- The model of network self-assembly presented in this dissertation has been modified so that given a computational problem it can be used to grow networks that are effective solutions. This modification represents an extension of the classic self-assembly problem, in which local control mechanisms are developed for the generation of pre-specified target structures, to growth that is driven by optimality criteria defined in terms of the quality or performance of the emerging structures. Additionally, the modified model illuminates an important connection between PSO, which usually occurs in high dimensional abstract spaces, and self-assembly, which is normally grounded in real and simulated 2D and 3D physical spaces.

- The aforementioned modified model constitutes a novel means of optimizing the weights and topologies of neural networks. The basis of this new approach lies in the incorporation of PSO-like interactions among the components of multiple, simultaneously growing networks. Its effectiveness is demonstrated on a number of challenging benchmark problems from the domains of trajectory generation, time-series forecasting, and control.

1.3 Organization

The rest of this dissertation is organized as follows. Chapter 2 presents background material relating to the fields of swarm intelligence, neural networks and developmental models. Chapter 3 gives a detailed description of the model of network self-assembly explored in this dissertation, along with some examples of the types of growth processes and structures it is capable of generating. Chapter 4 provides an overview of some of the computational experiments that were performed, along with an explanation of the metrics used in these experiments, and the details of how the model is implemented as a simulation environment. It then covers the details of the computational experiments, including the experimental setups, procedures, and collected data. Chapter 5 introduces an adaptation of the original model of swarm intelligent network self-assembly presented in this dissertation. The motivation behind this adaptation is discussed, followed by an explanation of how the adapted model incorporates particle swarm optimization into the network self-assembly process for the purpose of growing neural networks that perform well

on computational problems. Chapter 6 presents an overview of three computational problems used to test the optimization abilities of the adapted model, along with the implementation details of the model when applied to these problems. It then covers the details of a series of experiments involving these computational problems, including the experimental setups, procedures, and collected data. Chapter 7 provides a summary of the research presented in this dissertation, followed by an enumeration of the research's primary contributions. Lastly, it gives some limitations and future directions of the work presented here.

Chapter 2

Background

This chapter is a brief introduction to the concepts and methodologies from the fields of swarm intelligence, neural networks, and developmental models that are of relevance to the work presented in this dissertation. Particular focus is given to topics and previous work that closely relate to the research presented here. First, a qualitative description of swarm intelligence is given, along with some examples from nature and simulations. Collective construction is emphasized. The focus then shifts to the role of swarm intelligence in self-assembly and optimization, with particular emphasis on collective movements. Next, neural networks are introduced along with the concept of an artificial neural network. The three different types of learning that a neural network may undergo are mentioned, followed by explanations of various approaches to network training. The importance of neural network topology is also explained, along with an overview of neural network models and applications. Lastly, developmental models are discussed. The various means by which they incorporate growth are elaborated, with particle systems and L-systems serving as specific examples. This is followed by an explanation of network representation schemes and the reasons for using different types of encodings. The background section concludes with descriptions of two developmental models of neural network growth that are particularly relevant to the research presented in this dissertation.

2.1 Swarm Intelligence

Biological systems such as flocking birds, schooling fish and foraging ants consist of autonomous agents interacting in a simple and local manner. The complex global behavior of these swarms emerges from local interactions among the agents, rather than being imposed by forces external to the system or being the result of any particular leaders in the group. Furthermore, this global behavior often results in coherent patterns across space and time, and in this sense the system as a whole displays self-organizing behavior. In such cases the group or swarm of agents is engaged in displaying a collective intelligence, a *swarm intelligence*. It is not obvious that such complex and coherent group behaviors could arise from agents interacting in an exclusively local manner and with no perception of the dynamics of the group as a whole. Over the past few decades this phenomenon has become the subject of intense investigation, with researchers attempting to understand the underlying computational principles that govern the swarm intelligence and self-organization exhibited by groups of social insects and animals [16, 87]. Examples include, foraging in ant colonies [24, 35], flocking in groups of birds (Fig. 2.1) [60, 108], schooling in groups of fish (Fig. 2.2) [83], marching among locusts [12], herding in groups of mammals [51], and the dynamics of human crowds [58]. Swarm intelligence has also extended into robotics research. The rapidly growing field of swarm robotics includes many examples of swarm intelligence being utilized and explored in systems of both physical and simulated robots [98, 112, 116].

Figure 2.1: Red-billed Quelea (*Quelea quelea*) flocking [105].

Figure 2.2: Fish schooling in a cylindrical pattern [82].

2.1.1 Collective Construction and Self-Assembly

Collective construction, such as the nest building behaviors of social insects, is of particular relevance to this dissertation. Working in an entirely decentralized manner, a spatially distributed group of wasps or termites is able to cooperate to accomplish the complex task of building a nest. A structure that no individual in the group has any concept of. Furthermore, these insects typically exhibit only simple probabilistic stimulus-response behaviors, and there is strong evidence that only local information is needed to guide nest building activities. How is this possible? There are two primary phenomena behind this ability. The first is self-organization. This is the phenomenon in which a system consisting of locally interacting components, in the absence of external ordering influences, is capable of producing organized global dynamics or structure. There are a number of common mechanisms underlying self-organization. Among them are positive feedback (amplification), negative feedback (damping), randomness and fluctuations, and multiple interactions among the components of the system [8]. The second phenomenon is stigmergy [120], which is the process of communicating through persistent, spatially localized modifications of the environment. In order to see how these phenomena give rise to collective construction in a system consisting of agents that interact in a simple and local manner (hence, a system that exhibits swarm intelligence), consider the nest building behavior of termites [7]. Initially worker termites deposit pheromone laden soil pellets (the building material) in a random manner. Inevitably, by chance a number of soil pellets get deposited in close proximity. This increases the strength

of the pheromone signal at this particular location, which makes it more likely that workers will deposit additional soil pellets there. The repeated modifications of the termites' environment through the addition of soil pellets at a particular location is a form of stigmergy, which along with the positive feedback ultimately leads to the construction of a pillar, one of the structural foundations of the termites' nest. Once a certain quantity of soil pellets have been deposited at a particular location negative feedback takes over, which halts the construction of the pillar. At this point, if another pillar is nearby, construction of an arch between the two pillars may commence. Social species of wasps also utilize stigmergy in nest construction [75]. Wasps are capable of recognizing specific configurations of cells during the construction of a comb. These configurations act as stigmergic stimuli by guiding the wasps' placement of new cells (Fig. 2.3).

Figure 2.3: A European paper wasp (*Polistes dominula*) building a nest [39, 40].

Self-assembly is closely related to collective construction. The central difference is that in self-assembly arising from swarm intelligence the components of the structure being assembled are the agents themselves. This is what occurs when Weaver ants form chains out of their own bodies to create "living" bridges across

gaps that are far too wide for an individual ant to cross [61]. Of course, swarm intelligence is not a prerequisite for self-assembly; for example, galaxy formation is a type of self-assembly, and celestial bodies passively acting according to the laws of physics can hardly be thought of as exhibiting intelligent behavior. The work presented in this dissertation is concerned primarily with self-assembly that arises, at least in part, due to the simple and local interactions of autonomous agents. More specifically, the agents' interactions give rise to collective movements that enhance the ability of the system to self-assemble. At first, the swarm intelligence that gives rise to collective movements in natural systems, such as flocking birds, might seem unrelated to self-assembly. After all, flocking birds do not assemble static structures, rather their collective intelligence is manifested in dynamic global behavior. However, it has been demonstrated that the self-assembly process can be made more efficient and effective by tailoring the agents' interactions so that they exhibit collective movements. For example, the work presented in [49] deals with the simulated self-assembly of three-dimensional buildings in which the blocks that constitute the buildings are treated as autonomous agents. In a number of experiments the blocks were endowed with the ability to interact with one another in a simple and local manner in such way that they were capable of exhibiting flocking-like behavior. It was then shown that these collective movements make the self-assembly process more efficient by increasing the speed and consistency with which blocks are able to locate the appropriate construction sites. Furthermore, as will be illustrated in this dissertation, when groups of moving agents are tasked with forming particular spatial patterns, and the specific position of any given agent is not significant, collective

movements can be used to substantially increase the stability of the patterns. This in turn can be harnessed to enhance the robustness of the self-assembly process and the ability to control the grown structures.

2.1.2 Swarm Intelligence as an Optimization Tool

As models were developed to understand the governing dynamics of swarm intelligence in natural systems, it was recognized that the underlying computational principles could be utilized to create novel, very general problem-solving techniques. Freed from the empirical boundaries dictated by nature, researchers have created a wide range of new problem-solving algorithms inspired by nature and based on the governing principles of swarm intelligence. In these algorithms an agent typically represents a solution to the problem, a component of a solution, or it incrementally builds a solution. These methods employ collections of agents that simultaneously explore different potential solutions and communicate the quality of their findings to other agents. An agent directs its behavior so as to exploit the good quality solutions or solution components already discovered by itself or other agents. Additionally, there is almost always a stochastic element to the algorithm, which causes the agent to explore other potential solutions. As a computational tool swarm intelligence based algorithms have been developed based on the foraging and brood sorting behaviors of ants, and the schooling behaviors of fish and swarming behaviors of bees in search of food, among many other natural sources of inspiration. Such algorithms have been successfully applied to a wide range of problems, including

function optimization [32, 74, 76], routing problems such as those that occur in telecommunications and vehicle networks, job-shop scheduling, the traveling salesperson problem, data analysis such as clustering, and graph partitioning [6, 8, 23], to name but a few.

Of particular importance to this dissertation is the swarm intelligence based optimization method known as particle swarm optimization (PSO). This technique, first introduced by Eberhardt and Kennedy [76], typically involves embedding particles (agents) in an abstract space representing the domain of an objective function to be maximized or minimized. The position of a particle is interpreted as input to the objective function, and hence represents a solution, or part of a solution, to the optimization problem. At any given time during the course of a simulation each particle has stored the best position (solution) it is has found so far and is able to view the best positions of some subset of the particles in the system (its neighbors). The particles then use this personal (cognitive) and social information to guide their movements in such a way that they search regions of the space that are likely to contain better solutions. Ultimately the particles converge on the point in the search space that represents the best solution found by the swarm.

Since its inception PSO has undergone a myriad of different variations and enhancements. One of the most basic formulations of PSO, which will be referred to as *canonical* PSO, specifies that the particles velocities are governed by the equation

$$\vec{v}_i\,(t+1) \longleftarrow \chi\left(\vec{v}_i(t) + a_p\vec{u}_1 \otimes (\vec{p}_{best,i} - \vec{r}_i) + a_n\vec{u}_2 \otimes (\vec{n}_{best,i} - \vec{r}_i)\right), \tag{2.1}$$

where $\vec{r}_i(t)$ is the position of the i^{th} particle at time t, $\vec{v}_i(t)$ is its velocity, $\vec{p}_{best,i}$ is the current best position of the i^{th} particle, $\vec{n}_{best,i}$ is the best position among any of its neighbor particles, χ is a scaling factor known as the constriction coefficient, a_p and a_n are positive constants, $\vec{u}_1$ and $\vec{u}_2$ are vectors whose components are drawn from the uniform probability distribution over the unit interval, and the $\otimes$ symbol represents the component-wise vector product (i.e., $[a_1\ a_2] \otimes [b_1\ b_2] = [a_1 b_1\ a_2 b_2]$). It is standard practice to update the positions of the particles using a Forward Euler step with a step-size of 1.0, that is, $\vec{r}_i(t+1) \longleftarrow \vec{r}_i(t) + \vec{v}_i(t+1)$. The appeal of this version of PSO lies in its simplicity, and in its proven effectiveness on a wide range of optimization problems. The diverse formulations of PSO have been successfully applied to many different types of optimization problems, such as the training of neural hardware [9], data clustering in the form of image classification [94], antenna design [106], operational planning [122], economic dispatch in power systems [37], data mining [118], and stock market prediction [125], which is only a very small fraction of the number of applications to date. The application of PSO to the optimization of neural network weights and topologies is addressed in greater detail in Sections 2.2.1 and 2.2.2.

2.2 Neural Networks

Succinctly, a biological neural network can be viewed as a computing device that consists of highly interconnected processing cells called neurons. A neuron is comprised of a central cell body, called the soma, branch like structures called

dendrites that project from the soma, and a longer formation, the axon, which also extends outward from the soma. The axon acts as an output line from the soma by transmitting electrical impulses initiated there to the dendrites and somas of neurons that it synapses on (connects to). In this way dendrites act as input lines by accepting signals sent via the axons of synapsing neurons and then relaying these signals to the soma of the neuron for processing. In reality, biological neural networks are extremely complex, with the human brain containing on the order of 100 billion neurons and an average of 1,000 to 10,000 connections per neuron, a vast array of different neurotransmitters and chemicals, such as hormones, that affect network activity, and abundant exceptions to the rules, for example, axons sometimes connect directly to other axons. However, despite these complexities, from a computational perspective the aforementioned abstract view of biological neural networks is sufficient, and in fact has proven to be very powerful both in terms of modeling biological neural networks and inspiring artificial neural networks that are capable of solving challenging, real-world problems.

Artificial neural networks, (from this point onward referred to as neural networks or NNs), are highly simplified models of biological neural networks. Generally speaking, a neural network is a weighted, directed graph, in which the nodes perform computations on signals that are passed between the nodes via the graph's directed edges. In theory, the computations performed by a node can be virtually anything that suits the problem at hand, but for the purposes of this dissertation, and the majority of neural network models, it suffices to consider a single processing sequence of a node occurring as follows; first, the scalar valued signals from all

input edges to the node are compressed via a linear combination; next, this linear combination is input into a transfer function with a scalar valued range, (the transfer function is sometimes replaced with a more general dynamical system); finally, the scalar output of the transfer function is passed as input to all nodes that the edges of the present node project to, or is treated as part of the network's output. In this way the nodes of the network communicate among one another, input to the network is processed, and network output produced. More succinctly, given a neuron k that receives input from m other neurons in the network, along with its transfer function $f : \mathbb{R} \to \mathbb{R}$, the input signals $x_1, x_2, ..., x_m \in \mathbb{R}$ to the neuron and the weights $w_{k1}, w_{k2}, ...w_{km} \in \mathbb{R}$ on the input connections, neuron $k's$ output $y_k \in \mathbb{R}$ may be expressed as,

$$y_k = f(u_k + b_k), \tag{2.2}$$

where

$$u_k = \sum_{j=1}^{m} w_{kj} x_j, \tag{2.3}$$

and $b_k \in \mathbb{R}$ is a constant referred to as the bias. Often the bias values are treated as node inputs generated by a *bias node* that has a constant output value, usually 1.0. Some common activation functions are the hard limit functions, such as the *step* and *sign* functions, *linear* functions, and sigmoid functions, such as the *logistic* and *hyperbolic tangent* functions.

Figure 2.4 shows a feedforward neural network (FFNN), which is one of the

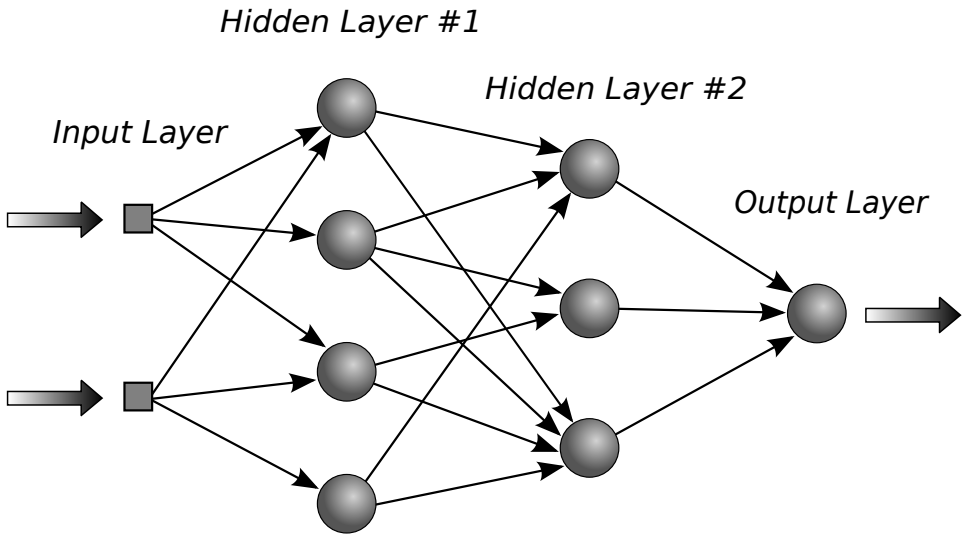


Figure 2.4: The graph representation of a feedforward neural network. Input enters the network via the nodes in the input layer, which typically serve only to pass the input activity to the nodes that they connect to in the first hidden layer. The output of the network is taken to be the output of the node in the output layer. The bias node and its connections are not shown.

most common and well-studied topological classes of neural networks. Feedforward networks do not have connections between nodes in the same layer, nor do they have connections from nodes in a higher level layer to those in a lower level layer. Recurrent neural networks, which are topological generalizations of FFNNs, allow such connections. In applications, it is common for each node in a given layer of an FFNN to connect to each node in the next layer, however this is not a strict requirement. The nodes in the input layer act as an interface to the "outside world" from which the network receives input, and typically do not perform any computations. The output of the nodes in the output layer are treated as the network's output. In this dissertation, as is common in the neural network literature, when no explicit distinction is needed the nodes that are not input, output or bias nodes are simply referred to as *hidden nodes* or as belonging to the *hidden layer*.

2.2.1 Training

One of the most useful properties of neural networks is their proclivity for learning. In general, a neural network learns by receiving input from a training set and then having the weights on its connections modified in response to the activities of its neurons. There are three broad classes of learning that a neural network may undergo. In supervised learning, for each input to the network there is a desired (or target) output that the network must learn to produce when the given input is presented to the network. This type of learning is used to train networks for tasks such as time-series forecasting, function approximation, and categorization. In reinforcement learning, rather than being given specific target outputs during training, a network is only given information about the quality of its outputs or its performance. This type of learning is commonly used when networks are trained to act as controllers, because in such circumstances the desired outputs of the network are not typically known in advance, rather, the network's performance must be judged in terms of its ability to control the system in question. Lastly, during unsupervised learning a neural network learns statistical patterns or features in the input data without the aid of any type of feedback regarding its performance. Unsupervised learning is often utilized in pattern recognition for tasks such as feature extraction and vector quantization.

Depending on the type of learning required, different methods for training a neural network's weights are employed. If the learning is supervised, then an analytical error function can be derived that expresses the relationship between the

network's actual outputs, which depend on the network's weights, and the performance (error) of the network on the training set (inputs and target outputs). A common approach to adapting a network's weights to improve its performance is to minimize the network's error on a training set by performing gradient descent on the error function E in the weight-space. There are many broadly applicable mathematical techniques that make use of gradient information to minimize a function, and many of them have been adapted for use in training neural networks, most notably the back-propagation algorithm [57].

As an illustration of how the back-propagation algorithm works, consider a feedforward neural network with one hidden layer, and the commonly used error function given by

$$E(t) = \frac{1}{2} \sum_k \left(d_k(t) - y_k(t) \right)^2 , \tag{2.4}$$

where t is the current iteration of the algorithm, the summation is taken over the neurons in the output layer, $d_k(t)$ is the target output of the k^{th} neuron in the output layer on the current iteration, and $y_k(t)$ is its actual output. On each iteration of training, the ℓ^{th} weight is updated according to $w_\ell(t+1) = w_\ell(t) + \Delta w_\ell(t)$, where the weight correction term $\Delta w_\ell(t)$ is chosen so as to implement gradient decent on E. Specifically, $\Delta w_\ell(t) \propto -\frac{\partial E(t)}{\partial w_\ell}$. Following this rule, on every iteration, the corrections to the weights w_{jk} on the connections from the *hidden* layer to the *output* layer are computed according to

$$\Delta w_{jk} \propto -\frac{\partial E}{\partial w_{jk}} = y_j \left(d_k - y_k\right) \frac{\partial f_k}{\partial X_k} = y_j \delta_k, \tag{2.5}$$

where y_j is the output of the j^{th} hidden layer neuron, f_k is the transfer function of the k^{th} output layer neuron, and X_k is the net weighted input to the k^{th} output layer neuron. The term $\delta_k = (d_k - y_k) \frac{\partial f_k}{\partial X_k}$ is the *error gradient* at the k^{th} neuron in the output layer. Similarly, on every iteration, the corrections to the weights w_{ij} on the connections from the *input* layer to the *hidden* layer are computed according to

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}} = y_i \frac{\partial f_j}{\partial X_j} \sum_k \left(d_k - y_k\right) \frac{\partial f_k}{\partial X_k} w_{jk} = y_i \delta_j, \tag{2.6}$$

where y_i is the output of the i^{th} input layer neuron, f_j is the transfer function of the j^{th} hidden layer neuron, and X_j is the net weighted input to the j^{th} hidden layer neuron. The term $\delta_j = \frac{\partial f_j}{\partial X_j} \sum_k \delta_k w_{jk}$ is the error gradient at the j^{th} neuron in the hidden layer, and contains the error gradients "back-propagated" from the output layer. The back-propagation of the δ_k's is necessary because the connections from the input layer neurons to the hidden layer neurons do not directly influence the output of neurons in the output layer. Rather, their influence on the output layer neurons is mediated through the outputs of the hidden layer neurons.

In cases where training involves reinforcement learning, or one wishes to avoid certain drawbacks of gradient descent methods, such as their tendency to get trapped in local minima or slow rates of convergence, gradient-free methods are typically used. Evolutionary computation in the form of genetic algorithms, genetic programming, and evolutionary strategies are among the most popular techniques [135].

However, particle swarm optimization has also been applied extensively to train neural networks. Typically, when PSO is used to train a neural network's weights the position of each particle represents a particular solution to the problem by specifying the value of each weight in the network, and sometimes additional network parameters as well. More specifically, each component of a particle's position specifies the value of a unique weight in the network or some other type of network parameter. In [29], it was demonstrated through a number of different function approximation problems that PSO can outperform training methods that rely on local gradient information, such as basic gradient descent and the scaled conjugate gradient method, when the error surface defined by the objective function is very rough. Ge et. al [41] used PSO to train and optimize recurrent neural networks for the challenging task of modeling and controlling the speed of an ultrasonic motor. The PSO algorithm was used to optimize the network weights, along with the initial input to a particular module of the network and a weight scaling factor applied to the connections within this module. In [13], recurrent networks were trained to solve a difficult time-series prediction problem by a hybrid optimization method that combines techniques from PSO with an evolutionary algorithm (EA). The PSO aspect helps to refine the solutions found at each generation, while the EA component helps prevent premature convergence to a sub-optimal solution. Juang [71] developed a hybrid neural network optimization method that combines PSO with genetic algorithms (GA). In this technique the best networks in each generation are improved using PSO, and then these enhanced individuals undergo crossover and mutation according to the specifications of the GA. Particle swarm optimization has also been combined with

training methods that make use of local gradient information. In [138], PSO was combined with the back-propagation algorithm to train feed-forward networks. It was found on a number of different benchmark problems that on average this hybrid method was able to find better solutions, and was able to find them more quickly, than either technique in isolation. The performance of a gradient-based method when used to train a neural network is often highly dependent on the choice of initial weights. In [4], this challenge was addressed by using PSO to find a good set of initial weights prior to applying back-propagation training.

During unsupervised learning a neural network's weights are adjusted according to a training scheme that integrates the effects of competition and reinforcement. Generally speaking, unsupervised learning is a form of self-organization, and as such it incorporates the effects of local interactions among network components, positive and negative feedback, along with randomness and fluctuations. More specifically, the weight on a connection is often modified according to the activities of the neurons in its local vicinity, typically those that it connects to. Positive feedback occurs in the form of reinforcement, where the weights on connections that directly influence highly active neurons tend to increase, which in turn tends to make these neurons even more active. Negative feedback occurs in the form of competition, which results in an increase in the strength of certain connections at the expense of the strength of others. These phenomena, along with the random nature of the initial weight values, and the ensuing fluctuations in the patterns of activity in the network that occur in response to different inputs, allow the network to configure itself so as to produce unique responses to the salient statistical features of the input

data. The self-organizing map (SOM) and the self-organizing feature map (SOFM) are two prominent classes of neural networks that utilize unsupervised learning, and are discussed in Section 2.2.3.

2.2.2 Topology

The weights are not the only aspect of a neural network that affect its performance. Its topology can also have a substantial impact on its ability to learn to solve a problem. Topology or architecture refers to the number and connectedness of a network's nodes. Topology can affect a network's performance in different ways. For example, it can affect its ability to generalize; It has been well established that feedforward networks with too many neurons and connections tend to over-generalize when presented with a limited amount of training data, but networks with too few neurons and connections lack the ability to adequately extract information from training data, and hence perform poorly when presented with data that they have not been trained on. Many network architectures have been derived for dealing specifically with particular types of problems or data, such as Elman networks [27], which were inspired by the need for networks that could handle temporally structured data, like spoken language, and could be trained relatively easily. Sometimes characteristics such as the sparsity of the connectivity strongly affect network performance [66]. In the case of modular neural networks, each module is a distinct portion of the network that may process output from other modules, and learns to solve a portion of the problem, or is trained to solve the problem inde-

pendently of the other modules [72, 114]. In any case, the outputs of the modules must somehow be integrated to produce the collective output of the network. This makes selection of the proper topology, (which includes determining the number of modules, their sizes, and their internal and inter-module connectivity), challenging and highly problem dependent.

As with network weights, methods have been developed to optimize topology as well. Often, the weights and topology are optimized together. Methods from evolutionary computation are frequently employed for this purpose because they are well suited to optimization in discrete domains. In particular, those that employ developmental encoding, which is discussed in Section 2.3, are especially appropriate for this task because they incorporate network growth [17, 47, 78]. Though less prominent than evolutionary computation for this task, PSO has also been used for topology optimization. This typically involves extending the standard formulation of PSO so that both the continuous weights and discrete topology of a network can be optimized. For example, in [77] different network topologies are represented by distinct spaces, where the dimension of a particular space depends on the topology it represents. Particles are able to move within a particular space according to the standard rules of PSO in order to optimize a network's weights, and they are also able to "jump" between the distinct spaces in order to optimize a network's topology. A PSO algorithm has been proposed [15, 137] for the combined optimization of network weights and topology in which there are two different swarms of particles, one for optimizing the topology, and a second for optimizing the weights. The topology optimizing particles (or architecture particles) are embedded in a space

that represents the domain of feasible network topologies. The quality of a given topology, (a topology is specified by the position of an architecture particle), is determined by creating a separate swarm of particles that exist in a space that represents the domain of the connection weights, and then using the PSO algorithm to search for the optimal set of weights for the given topology. Subsequently, once the position (topology) of each architecture particle has been evaluated, one iteration of the PSO algorithm is applied to these particles, and the evaluation process repeats.

Other techniques either incrementally build a network, or prune (remove) connections from a network. Methods such as cascade correlation [30], Marchand's algorithm [91], and the upstart algorithm [36] build feedforward networks in order to improve characteristics like the speed and consistency of training, and network size. These techniques incrementally add neurons to the hidden layers of feedforward networks in order to gradually reduce the error in a network's output on a training set. The procedure for adding neurons, which is specific to each technique, results in each hidden neuron acquiring a particular computational function in the network. Pruning methods typically start with a network that has already been trained and then reduce the network's size and complexity by removing connections. The pruning procedure can be as simple as removing connections with small weights until performance on the training set has decreased to a specified level, or more sophisticated, such as modeling the effects of removing connections and then using the model to optimize the pruning [56], or adding a complexity penalty function to the cost function being minimized by the training process that tends to force the weights on unnecessary connections to zero [127].

2.2.3 Models and Applications

In the sciences and engineering neural networks are primarily used for two purposes; they are used to model the properties and functions of biological neural networks, and as computational problem-solving tools. Neural networks have been applied to an immense variety of different problems. Here, however, the focus is restricted to the specific types of problems and studies related to the work presented in this dissertation.

Neural network models have been used extensively to study a particular class of biological neural networks found in the brains of mammals known as self-organizing maps (SOM's) [81, 109]. SOM's are widely-studied neural networks that have the capacity to learn statistical patterns or features in their input data in an unsupervised manner. There is a great deal of interest in SOM's both for their usefulness as computational tools [19, 57, 86, 103, 123, 124], and because they occur frequently in the brain, in which sensory inputs are mapped in a topographically ordered manner onto different regions such as the cerebral cortex [31, 53, 90, 100, 119]. They typically have two layers of neurons, an input layer, and an output layer in which map formation occurs. Upon receiving input the neurons in the output layer compete for activity until only a small portion of them remain active (the *winning* neurons) while the rest exhibit little to no activity. The Hebbian learning mechanism [10] used is such that for a given input pattern the winning neurons will tend to respond more strongly the next time that pattern or a similar pattern is presented to the network. This learning process results in the formation of a topographic or feature

map, in which neurons in the output layer that are close together tend to respond strongly to similar inputs, and the number of neurons that respond strongly to a particular input typically correlates with the frequency with which that input was presented to the network during training. In Chapter 4, computational experiments are discussed in which the model of network self-assembly presented in this dissertation was used to grow two networks that are based on models of SOMs found in the visual and somatosensory cortical regions.

There is a large body of work concerned with problems that involve the generation and analysis of sequences of data that have both a spatial and a temporal component. The experiments discussed in Chapter 6 incorporate three such problems. One of these problems involves time-series forecasting, which is a very common and important example of a spatiotemporal problem. In time-series forecasting, a neural network must learn a model of the system whose behavior is to be forecasted, and then upon receiving a sample of this behavior as input is able to accurately generate (predict) the future behavior for some period of time. Since the data in a time-series forecasting problem carries essential information in its temporal structure, a neural network solution must be able to learn the temporal qualities that characterize the time-series. In order to make use of this information/knowledge during the computational process some portion of the network must operate directly on past values of the input data, or past network outputs, or past activations of the hidden layer, and use this data to affect the network's output. It is often possible and convenient to conceptualize this portion of the network as a kind of "temporal module", which is a function, and at any given time-step during the network's operation this

function's input is a finite set of one or more of the types of past neuronal activities just mentioned. Examples of temporal modules include tapped delay lines and the context layers of Jordan and Elman networks [22]. In the majority of neural networks designed for processing temporal data the connections that feed into the temporal module, and connections within the temporal module, if any are present, are not trained, and hence the function it defines is predefined and remains fixed throughout the learning process and subsequent operation of the network. A wide variety of neural networks that adhere to this restriction have been developed. Examples include single and multi-layer feedforward networks with separate short-term memory structures, such as tapped delay lines [57], NARX networks [115], Elman networks [22] and their extension to recurrent multilayer perceptrons [104], radial basis function networks [130], fuzzy neural networks [38], and single neuron models [139].

Neural networks that have trainable, recurrent connections within the hidden layer represent a departure from the majority of neural models used for temporal data processing in that the temporal module is the hidden layer itself, and furthermore, the function that it represents, which is the function that operates on past neuronal activities, is not predefined, but rather, it must be learned by training the recurrent connections within the hidden layer. These types of networks tend to be difficult to train for even moderately challenging time-series problems because not only does such a network need to learn how to map its internal (hidden) representations to appropriate outputs, but it must also learn how to process and incorporate information from the past into these representations. Various methods based on

gradient descent have been developed for training these types of networks, such as back-propagation through time, and real-time recurrent learning [18].

The experiments discussed in Chapter 6 involve the growth of a particular class of recurrent neural networks known as *echo state networks*. The echo state network approach to training neural networks was developed largely out of a desire to be able to utilize the powerful temporal processing abilities of recurrent networks with unrestricted topology in the hidden layer, while avoiding the difficulties associated with training recurrent connections. The architecture of an echo state network (ESN) is shown in Figure 2.5. An ESN is a neural network that consists of an input

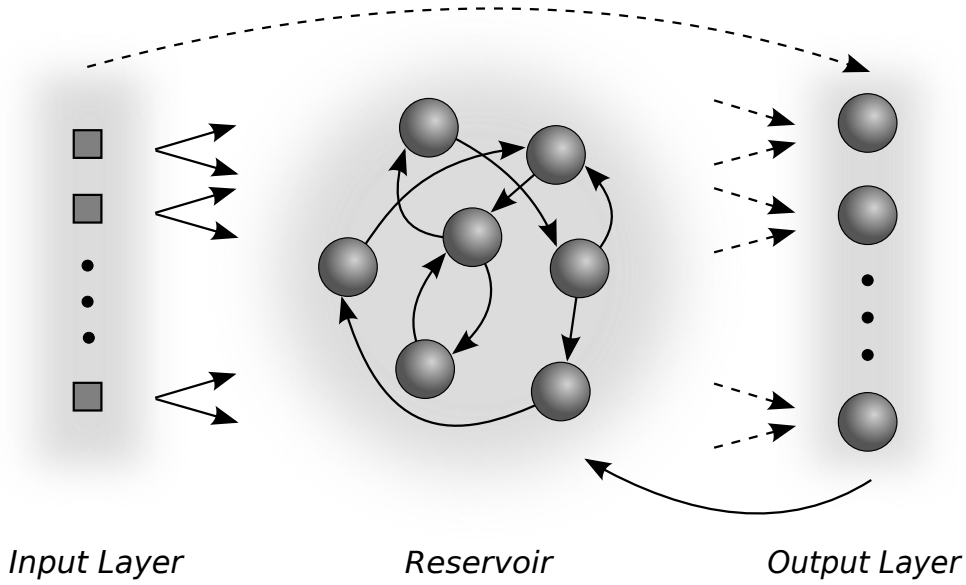


Figure 2.5: Schematic of an echo state network (ESN). The solid arrows represent connections with fixed weights, and the dashed arrows represent connections with trainable weights. Connections from the input layer to the output layer and from the output layer to the reservoir are optional. As usual, input to the network enters through the input layer and the network's output is generated in the output layer. The bias node and its connections with fixed weights to the reservoir are not shown.

layer, a hidden layer or "reservoir", and an output layer. Typically, each neuron in the input layer connects to every neuron in the reservoir, there is randomly-

generated sparse connectivity among reservoir neurons, each neuron in the reservoir connects to each neuron in the output layer, a bias neuron may or may not connect to the neurons in the reservoir, and sometimes connections from the input layer to the output layer and from the output layer to the reservoir are present. Echo state networks have a number of features that are unusual among recurrent neural networks. For one thing, they tend to be relatively large. For a problem in which a more traditional recurrent neural network solution would typically have on the order of tens of neurons in its hidden layer, an echo state network would likely have on the order of hundreds of neurons in its reservoir. Additionally, recurrent neural networks for time-series problems that have recurrent connections within the hidden layer usually exhibit full connectivity, that is each hidden neuron connects to every other hidden neuron. This is in stark contrast to the high degree of sparsity and the random nature of the connectivity within the reservoirs of ESNs.

The central innovation of the echo state network approach is that only the weights on the connections from the reservoir to the output neurons (output weights) are trained, and the activation functions of the output neurons are linear so all that is needed to train them is linear regression. (If they exist, the weights on the connections from the input to the output neurons are trained in conjunction with the output weights, using linear regression). The remaining weights are typically assigned random values. For an echo state network with N_r reservoir neurons and N_o output neurons, the output weights are trained as follows. A sequence of training data of length $L + M$ is chosen, where $M > N_r$. The first L values of the sequence are passed through the network in order to remove the effects of the initial state of

the reservoir. Then, the remaining M values are input into the network, and the resulting reservoir states $\vec{e}_i \in \mathbb{R}^{N_r}$, for $i = 1, ..., M$, are assigned to the rows of the matrix $\mathbf{S} \in \mathbb{R}^{M \times N_r}$. For each network input, and resulting reservoir state $\vec{e}_i$, there is a target network output $\vec{d}_i$. The target network outputs are assigned to the rows of the matrix $\mathbf{D} \in \mathbb{R}^{M \times N_o}$, such that the i^{th} rows of $\mathbf{S}$ and $\mathbf{D}$ are the corresponding reservoir state and target output pair. Let $\mathbf{W} \in \mathbb{R}^{N_r \times N_o}$ be the output weight matrix, where the j^{th} column of $\mathbf{W}$ represents the weights on the connections from the reservoir to the j^{th} output neuron. Training the output weights amounts to finding an approximate solution $\mathbf{W}_a$ to the overdetermined system

$$\mathbf{SW} = \mathbf{D}. \tag{2.7}$$

The output weights $\mathbf{W}_a$ are determined by solving Eq. 2.7 in a "least squares" sense.

It is counterintuitive that such a simple strategy is effective. In essence, the reason that it works is that a large number of randomly, and sparsely interconnected neurons are capable of exhibiting highly varied or "rich" dynamics. The rich set of dynamics produced by different sequences of input to the reservoir allow the network to output highly complex time-series', and to generalize well, despite the fact that the mapping from reservoir activations to network outputs is linear and only the output weights are trained. Furthermore, at any time during operation of the network, the current activity in the reservoir depends on previous reservoir activities and network inputs for only a finite period of time into the past. This

property typically holds when the spectral radius $\rho(\mathbf{W}_r)$ of the reservoir weight matrix $\mathbf{W}_r$ is less than unity. Specifically, the closer $\rho(\mathbf{W}_r)$ is to unity, the longer the effects/"echos" of past inputs and reservoir states last. The spectral radius of the reservoir weight matrix obeys the equation

$$\rho(\alpha_r \mathbf{W}_r) = \alpha_r \rho(\mathbf{W}_r), \tag{2.8}$$

where $\alpha_r \in \mathbb{R}^+$. Thus, the length of time into the past for which reservoir activities depend on past activities and inputs is easily controlled through an appropriate scaling of the weights on the recurrent connections within the reservoir. This is important since this time-window must be tailored to a size that is appropriate for processing the time-series at hand. Echo state networks have been successfully applied to a wide range of problems involving data with a temporal component, including the modeling of chaotic attractors [67], time-series forecasting [65, 66, 88, 134], filtering [67], nonlinear system identification [62], dynamical pattern recognition [97], natural language processing [121], and event detection and localization for mobile robots [1].

Neural networks have been used extensively for control problems, both as the controller itself, and to model the system (plant) one wishes to control (system identification). Most of the neural network-based techniques developed for time-series problems can readily be adapted to control problems. The substantial early success of echo state networks on trajectory generation and time-series forecasting problems has prompted researchers to apply ESNs to control problems. To date, there has

been only very limited work on applying ESNs in this domain. Considering the impetus behind their creation, it is logical to incorporate ESNs into control systems for the purpose of system identification (indirect control) [69]. This is because system identification is generally cast as a supervised learning problem, so only the output weights of an ESN need to be trained, which can easily be accomplished using some form of linear regression. ESNs have also been implemented as the controllers themselves (direct control) without the use of system identification of any kind. When used for this purpose it is sometimes possible to acquire sufficient data for training the ESN controller in a supervised manner by observing the effects of a limited set of control actions on the plant's output. For example, this may be possible when there exists a human "teacher" with adequate knowledge regarding the relevant regions of the plant's state space or output range and the effects of control actions on the plant's state/output. In this case the ESN controller is inevitably trained by applying linear regression to its output weights [52, 55, 96, 113]. However, many control tasks require that a controller learn a nontrivial, long-term control strategy, which generally requires reinforcement learning. This scenario necessitates a departure from the canonical approach to training ESNs in favor of a more complex training strategy. One that may involve training any of an ESNs weights, not just those on the output connections, and that cannot rely solely on linear regression. Few studies have been performed with the aim of examining the suitability of ESNs for this type of control problem, but the preliminary work that exists, including the work presented in Chapter 6 of this dissertation, suggests that ESNs are highly capable of acting as controllers in situations that demand reinforcement learning [11, 54, 68, 133].

2.3 Developmental Models

In this dissertation the term *developmental model* refers to a framework that models a growth process by incorporating a representation scheme for the structures that grow, along with a set of rules that govern the growth of the structures by *iteratively* modifying their individual representations. Developmental models of neural network growth typically serve two purposes. They are used to better understand the principles of and mechanisms behind growth in biological neural networks, and they serve as a means of enhancing the optimization of neural network weights and topologies. A developmental model must specify a representation for the networks it can be used to grow, and it must specify the rules that govern the growth process. The representation can either be explicit, in which case network components (neurons (nodes) and axons (directed edges)) are modeled directly, or implicit, in which case networks are represented by data structures and a function maps the data structures to the corresponding networks. (Of course, the graph representation of a neural network is itself a data structure, albeit one that more directly represents the physical components of biological neural networks). The rules governing the growth process are often expressed either as sets of *if-then* statements or as forces.

2.3.1 Growth Mechanisms

When a developmental model represents network growth in a continuous space it is common to have the model incorporate forces, and to have these forces, coupled with Newton's second law, dictate the movements of network components. Forces

may be used to emulate physical interactions between network components and their environment, such as collisions and viscous drag forces. They may also be used to produce dynamic behaviors in the components, such as gradient following, and attraction to or repulsion from neighboring components. The particle system framework is a useful means of implementing force-based dynamics in developmental models. The particles are simple geometric structures, usually spheres, or point particles (particles without volume), that interact with their environment and each other through forces. At any point in time t the net force $F_{net}(t)$ on a particle determines its current acceleration via Newton's second law. While the form of the function $F_{net}(t)$ is essentially unrestricted, the work presented in this dissertation will demonstrate that particularly interesting and useful dynamics can be achieved when the force function is at least partly determined by simple and local interactions among the particles.

In addition to movements through the environment, developmental models often specify discontinuous, non-movement-based actions for network components, such as cell divisions, axon emissions, and axon branching. Conditional rules are normally used to dictate these types of actions, and if network growth occurs in a discrete space such rules may also dictate the components' movements. Conditional rules typically take the form of *if-then* statements, in which the antecedent is a predicate with conditions that must be satisfied in order for the action specified by the consequent to be executed. Such rules either manipulate network components directly, or they manipulate components of data structures that abstractly represent networks, such as matrices; the modus operandi of the rules depends on whether

an explicit or an implicit network representation is used, respectively. It is common for the rules to be placed in a list that is read sequentially by each component on every time-step of the growth process, however there are exceptions. For example, in cellular encoding [47] rules are placed in a tree structure, and as development progresses different components read and execute rules from different parts of the rule tree.

L-systems provide a good example of how data structures, along with sets of conditional rules for manipulating them, can be used to model the development of biological entities such as neural networks. L-systems were created in 1968 by biologist Aristid Lindenmayer in order to model and study growth patterns in plants and simple multicellular organisms such as algae [102]. Occasionally they have been implemented as an encoding mechanism for growing neural networks [17, 73]. L-systems work by accepting an initial string of allowed symbols and then repeatedly rewriting the string by replacing its symbols with new symbols or substrings defined by rules called productions. More formally an L-system is a triple,

$$L = \{V, w, P\} \tag{2.9}$$

where:

- V is a set of symbols called the *alphabet*;
- $w \in V^*$ is a non-empty starting string (axiom);
- $P \subset VxV^*$ is a set of productions (rules).

Here, V^* is the set of strings consisting of symbols in V. L-systems can also be *context dependent* in which case $P \subset V^*xV^*$. This allows the designer to impose varying degrees of interaction among components in a developing structure. L-systems are very similar to formal grammars except that they update all symbols on every iteration as opposed to just one. This parallel updating convention is particularly important for L-systems based modeling of growth in that it allows different parts of a developing system to emerge simultaneously and independently of each other. In developmental models that use L-systems the symbols in the alphabet are often interpreted as instructions so that reading a string produces some structure or accomplishes a task. For example, L-systems have been used extensively to model the development of plants, and to generate artificial plants by interpreting symbols as movements and changes in orientation of a drawing cursor. The artificial plants shown in Figure 2.6 were grown based on this interpretation. As another example, the neural development model presented in [73] uses an L-system to describe axonal growth. The symbols in the system represent the states of axons growing through a discrete two dimensional space and induce behaviors such as axon branching and movement.

2.3.2 Representation Schemes

One of the most common applications of developmental models of neural networks is as developmental encodings for evolutionary computation. Developmental models used for this purpose typically employ either an implicit representation, or

Figure 2.6: Artificial plants generated by an L-system.

an explicit representation that does not incorporate geometric relationships among network components. The term *developmental encoding* refers to the fact that a network is encoded through the combination of a data structure and a set of rules that develop the network by modifying the data structure. In this application genetic operators modify the growth rules or data structures (in their initial states, before the rules have been used to alter them) to produce new generations of networks (solutions). The growth rules are applied to extract the networks for fitness evaluation. For example, in [17] the topology of each network is encoded by an L-system with a particular initial string (axiom) and productions (rules). A network's topology is determined by interpreting the string of symbols produced by its L-system as a sequence of movements over the elements of the network's connectivity matrix and as instructions for modifying these elements. In [78] a matrix rewriting system is proposed for encoding network topologies. This encoding is very similar to an

L-system except that the data structure is a matrix rather than a string. Cellular encoding is introduced in [47]. It uses an explicit representation (networks are represented directly by graphs), but there are no spatial relationships between network components. A number of encodings have been created that use explicit network representations and incorporate network development in discrete spaces (grids). In [14] neural networks are represented by cell objects and axons that connect the cells, both of which exist on a two-dimensional grid. The developmental model presented in [3] also places network components on a two-dimensional grid, and includes a number of additional biologically-inspired features such as dendrites, and chemical messengers that can diffuse across the grid or within a cell. Similarly, the model presented in [63] incorporates some extra biologically-inspired additions, such as diffusible chemical messengers, but network components exist on a three-dimensional grid.

In cases where the primary interest in a developmental model is its use as a tool to help understand development in biological neural networks, explicit network representations are most common. The amount of biological detail included in these developmental models varies greatly. In addition to modeling neuron cell bodies (somas), and axons, they may include representations of growth cones (objects that occur at the tips of growing axons and guide them to target neurons), dendrites, various diffusible compounds, and any number of different mechanical, electrical, or chemical interactions among the components of a developing neural network, such as collisions. The networks are typically modeled as existing in a two-dimensional space, or occasionally a three-dimensional space, which may be either continuous or

discrete. Two developmental models of neural growth that are especially pertinent to the work presented in this dissertation are described below.

In [73], Kalay, Parnas, and Shamir present a model of neural growth in which an L-system is used to generate the dynamics of axons growing to and from a fixed set of stationary cells in a discrete two dimensional space. The rules of the L-system incorporate signals from the environment in addition to environment-independent actions, (those whose execution does not depend on an axons interaction with the environment). Axon movement is partly controlled by a diffusible chemical emitted by cells. A growth cone, (the tip of a growing axon), may detect the concentration levels of this chemical in its local neighborhood and change its direction of motion or branch in response to it. This mechanism is responsible for guiding axons to target cells over relatively short distances and promoting the appropriate amount of axonal branching. Growth over longer distances, regions in which the diffusible chemical is not available for guidance, is dictated by environment-independent rules. While such rules alone provide enough control over development to, in theory, generate any desired network, doing so requires that an inordinate amount of information be hand-encoded into the L-system. By supplementing this class of rules with those whose execution is dependent on the presence of the aforementioned diffusible chemical, the amount of information that needs to be encoded in the L-system is significantly reduced.

The neural growth model described by Fleischer and Barr in [33] is based primarily on environment-dependent rules. Their model consists of cells that are capable of moving, dividing and emitting axons. The axons are guided by growth

cones and grow through a two dimensional continuous space and make connections with cells to form networks. Certain cells emit chemicals into the environment which diffuse and guide the growth cones by establishing chemical gradients. Cells have states that are determined by the cumulative amounts of various types of proteins that they have acquired. Cells may amass proteins by collecting them as they diffuse through the intercellular environment, receiving them from a contacting cell or by generating them. Rules in this model depend on both the state and local environment. However, each one of the processes by which a cell acquires proteins depends on the cell's interaction with the environment and therefore its state depends on its past interactions with the environment. Thus, behaviors in this model are generated directly and indirectly by interactions with the environment, which is in contrast to the model presented in this dissertation and that presented in [73], which have behavior generating components that ares entirely environment-independent. Because it tends to be difficult to precisely predict and control the local concentrations of diffusible elements, particularly when the dynamics of the sources are uncertain, it is hard to come up with a set of rules that will result in the generation of specific networks and geometric patterns of cells.

Chapter 3

A Developmental Model of Neural Growth

This chapter describes the developmental model that is central to this dissertation, and the simulator that implements it. In essence, the model is one of **S**warm **I**ntelligent **N**etwork **S**elf-**A**ssembly (SINSA). The SINSA model supports the collective growth of individual neurons in a continuous, unbounded, three-dimensional space, with the development of neural networks being governed by two different but integrated mechanisms. The first mechanism, which is inspired by L-systems [102], consists of a set of rules that are repeatedly applied to neurons and which govern discrete decisions such as cell division and axon emission. The second mechanism involves using local forces to govern the interactions between neurons and their constituent components, growth cones and cells (somas). This mechanism is inspired by collective movements in swarm systems and the principles of self-assembly [8, 49, 50]. However, as described below, this approach is somewhat atypical of swarm intelligence systems in that it is deterministic and the agents' local coordinate systems all have the same orientation. The need for probabilistic choices never arose as a design issue, and the rules that were developed worked well without introducing noise. The shared orientation was used because it made writing rules qualitatively easier (such a common reference frame might be implemented in biological systems via multiple chemical gradients). The SINSA model includes a number of features and capabili-

ties not commonly found in past models of network development that are based on growth through a physical space. For one, it represents network growth in a three-dimensional space, as opposed to only two-dimensions. Also, it incorporates cell migration and division, and interactions between growing axons. Lastly, the SINSA model was used to grow relatively large recurrent networks based on target network models that have precisely specified topologies. In contrast, work using past models has typically focused on the growth of relatively small feedforward networks that are based on target structures with topologies that are specified in a very general manner.

3.1 Agents

The SINSA model incorporates a system of agents (particles) that move through a three-dimensional space. There are two classes of agents, both of which are represented as spheres (see Figure 3.1). Cell agents have radius r_c and growth cone agents have radius r_g with $r_c > r_g$. Each agent i has a position $\vec{r}_i$ and a velocity $\vec{v}_i$. In addition, each cell and growth cone has internal state variables. Table 3.1 lists the internal state variables of cells and growth cones. The *cell type* indicates whether or not a cell can divide and prescribes a particular role for it when activated in a neural network. A cell of type "A" is an afferent (input) cell, type "E" an excitatory cell and type "I" an inhibitory cell. Appending "S" to any of the aforementioned cell types means the cell can divide. If the "S" is absent, then it cannot divide. The *life* variable $L \in \mathbb{N}$ influences cell division by restricting the number of generations

of descendent cells. If $L = 0$, then a cell cannot divide. Otherwise, the life variable is often used such that when a cell divides the life variables of its two child cells are $L - 1$, the life variables of their children are $L - 2$, and so forth, until $L = 0$, so the original parent cell (which no longer exists because it divided) will have 2^L Lth generation descendent cells. The *local time* variable indicates how long an agent has existed. The *tag set* partitions agents of the same type into subgroups of agents that have all or some of the same tags. This allows agents with otherwise identical states to execute different rules.

Table 3.1: Internal State Variables Possessed by Cell and Growth Cone Agents

State Variable	Description	Agent Type
Cell Type	one of the following strings: A,E,I,AS,ES,IS	Cell
Life	natural number	Cell
Local Time	nonnegative real number	Cell, Growth Cone
Tag Set	set of alphanumeric strings, e.g., {B0,E54,A3}	Cell, Growth Cone

In the SINSA model, a neural network is specified by its geometry (relative spatial positions of the cells) and its topology (connectedness of the cells). The directed edges between cells are referred to as *axons*. Cells emit growth cones, which occur at the tips of axon branches and guide their growth. An axon branch is implemented as a persistent trail of small, discrete, connected segments deposited behind a moving growth cone. When a growth cone comes in contact with a cell it can establish a connection directed from the cell that emitted the growth cone to the cell that it contacts. In this way the topology of a network is generated by

the collective dynamics of the growth cones and cells. Most past developmental models of neural growth do not incorporate dendritic trees, and neither does the SINSA model. However, the framework presented here is sufficiently general that they could be added.

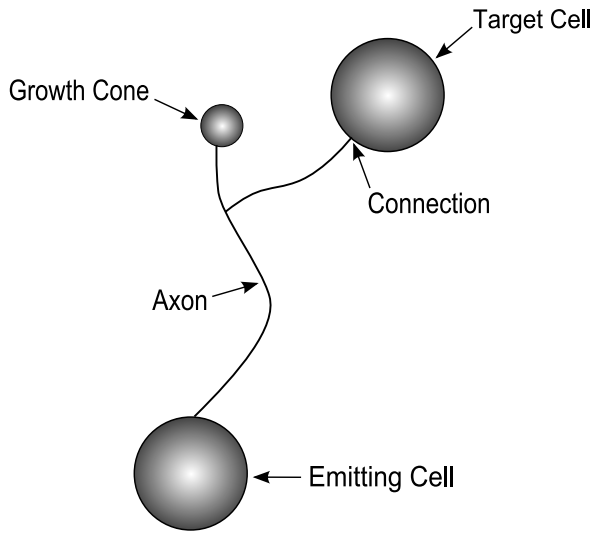


Figure 3.1: The growth of an axon from its emitting cell to target cells. Each branch of a growing axon is guided by a growth cone that occurs at its tip. Under the proper conditions, when a growth cone comes in contact with a cell it will establish a connection with that cell, as has already occurred here with the upper target cell. The growth cone is continuing to move upwards, presumably towards another target cell (not shown).

3.2 Rules

The behavior of the agents (particles) is governed in part by a set of rules that control actions such as cell and growth cone division and axon emission, and which manipulate the values of certain internal state variables such as the tag set. The rules are in the form of *if-then* statements in which the antecedent is a predicate that must be satisfied in order for an agent to execute the consequent, which is an

instruction. An instruction is a command accompanied by the information needed to carry out the command. A command may be thought of as a function in which the information needed to execute the command is passed as arguments to the function. During each time step, every agent looks through its rule set and finds the subset of rules with satisfied antecedents that it may execute. If this subset contains rules that may not be executed on the same time step, then conflict resolution is applied to remove rules from the subset until all conflicts are resolved. In general, each rule takes the form

$$< Agent\ Type; Cell\ Type; Tags; Local\ Times > \Rightarrow < Command; arg\ 1; arg\ 2; ... > .$$

For each agent type, Table 3.2 lists the requirements that may be specified in each predicate (antecedent) field and which must be met by an agent in order for it to execute the corresponding instruction. Table 3.3 describes the meaning of each command function and the arguments that each one accepts. Here *tags* is

Table 3.2: Conditions That May Be Specified in the Predicate Fields

Agent Type	Cell Type	Tags	Local Times
Cell or *Growth Cone*	Must have specified cell type or *null* if Agent Type is *Growth Cone*.	Agent must have all tags in specified set.	Agent's local time must match one time in specified set.
		Agent must have at least one tag in specified set.	Agent may have any local time.
		Agent may have any tag set.	

a set of alphanumeric strings; *force* $\in \mathbb{R}^3$ represents a force acting on the agent, and is specified in spherical coordinates $[\theta, \phi, r]$, where $\theta \in [0°, 360°)$, $\phi \in [0°, 180°]$ and $r \in [0, \infty)$; *duration* $\in \mathbb{R}^+$ is a length of time during which an agent applies or is subject to a force; *cellType* is a string (see Table 3.1); *life* $\in \mathbb{N}$; *direction* is a unit vector in $\mathbb{R}^3$ indicating the orientation of an axon emission or a cell or growth cone division, and is specified in spherical coordinates $[\theta, \phi]$ with $r = 1$; *magnitude* $\in \mathbb{R}^+$ is the maximum magnitude of a force; and *sign* $\in \{\text{“}+\text{”}, \text{“}-\text{”}\}$ indicates whether a force is attractive or repulsive. The *local growth force* (LGF) and the *rule-based force* are defined in Section 3.3. Additionally, there are three special symbols (*, #, and +) that may be specified in certain predicate fields or passed as arguments to certain command functions. The * symbol may be placed in the *Tags* or *Local Times* predicate fields, in which case it indicates that an agent satisfies the predicate field with any tag set or any local time respectively. In GenCell and GenCone commands (see Table 3.3) the *tags* and *life* variables may be set to the # symbol. If *tags* = #, then the newly created agent, a cell in the case of a GenCell command and a growth cone in the case of a GenCone command, will have the same set of tags as the agent executing the rule. If *life* = #, then the life state variable of the newly created cell will be one less than the life of the cell executing the rule or zero. When the + symbol is present in a set of tags specified by the variable *tags* it indicates that when an agent executes the corresponding rule the alphabetically last tag in the executing agent's tag set is to have a value of one added to its numeric portion, and that this new tag is be added to the set *tags* for the current execution of the rule.

Table 3.3: Command Functions Used as Rule Consequents

Command Function	Description
GenCell(*cellType*, *tags*, *life*)	Generates a new cell with the specified cell type, tag set and life.
GenCone(*tags*)	Generates a new growth cone with the specified tag set.
SetTags(*tags*)	Sets the agent's tag set to the one specified.
SetRuleForce(*force*, *duration*)	Constant force applied to the agent for the specified duration or until it executes another SetRuleForce command.
SetLGF(*sign*, *magnitude*, *duration*)	Cell sets its LGF parameter $\pm k$ to the specified magnitude and sign for the indicated duration or until it executes another SetLGF command.
SetCellType(*cellType*)	Sets the cell's type to the specified cell type.
EmitAxon(*direction*, GenCone)	Cell emits an axon in the specified direction.
DivCell(*direction*, GenCell, GenCell)	Cell divides into two new cells along the specified axis of division.
DivCone(*direction*, GenCone)	Child growth cone splits-off from its parent along the specified axis of division, which results in a new axon branch.

For example, the rule

$$(Cell; ES; \{B0, E2\}; *) \;\Rightarrow$$

$$< DivCell; [0, 90]; < GenCell; ES; \{B0\}; \# >; < GenCell; IS; \{D0, F1, +\}; 3 >>$$

instructs a cell to divide. To execute this rule a cell agent must have a cell type of "ES", either a "B0" or an "E2" tag, or both tags, and may have any local time

(indicated by the * symbol). When a cell satisfies these conditions it divides along the x-axis (specified by spherical coordinates $[\theta = 0, \phi = 90]$) into two child cells. The first child cell is of cell type "ES", has "B0" as its only tag, and its life is one less than its parent's life or zero (indicated by the # symbol). The second child cell is of cell type "IS", its tag set consists of "D0", "F1" and "E3", and its life is 3. The plus symbol in the second GenCell command indicates that the new cell is to be given the tag that immediately follows the alphabetically last tag in the parent cell's tag set (hence the "E3"). Upon being created the local time of each child cell is set to 0.0. As a second example, the rule

$$(Cone; AND\{B2, D0\}; \{0.5, 2.0\}) \;\Rightarrow\; < SetRuleForce; [45, 45, 5]; 2.5 >$$

sets a growth cone agent's rule-based force. In order to execute this rule a growth cone agent must have *both* "B2" and "D0" in its tag set, which is indicated by the *AND* preceding the specified set. It must also have a local time of either 0.5 or 2.0. If these conditions are met, then the rule-based force (explained later) of the growth cone agent is set to the spherical coordinate $[\theta, \phi, r] = [45°, 45°, 5]$ for 2.5 time units or until this agent executes another SetRuleForce command. A detailed explanation of how rule sets are designed is given in Appendix A.

3.3 Forces

In addition to rules, agent dynamics are governed by force-based interactions between agents, and between agents and their environment. These "forces" are

not physical forces but are intended to represent influences on an agent including agent-to-agent interactions that are responsible for generating collective movements as in past swarm intelligence systems [76, 108, 110], agents' interactions with their environment (including collisions), and a separate force that is controlled by the rule set. The forces responsible for producing collective movements are $\vec{F}_{cell}$ and $\vec{F}_{cone}$, which produce flocking-like behavior among the cells and growth cones respectively, and $\vec{F}_{lgf}$, which causes growth cones to be attracted to or repelled from cells in their local vicinity. Environmental forces include $\vec{F}_{drag}$, which induces a viscous drag on all agents and $\vec{F}_{collision}$, which approximates collisions between agents. Finally, $\vec{F}_{rule}$ is a rule-dependent force that is specific to each agent.

The force-based dynamics of the agents obey Newton's second law, which states that the time rate of change of the momentum of an agent is equal to the net force upon it. Assume that there are n cells and m growth cones, that the mass of each cell is m_c, and the mass of each growth cone is m_g, with $m_c > m_g$. Then the force-based movements of cells are governed by

$$\frac{d^2\vec{r}_i}{dt^2} = \frac{1}{m_c}\left(\vec{F}_{drag}(\vec{v}_i) + \vec{F}^i_{rule}(t) + \sum_{j\neq i}^{n}\left(\vec{F}_{cell}(\vec{r}_{ji}) + \vec{F}_{collision}(\vec{r}_{ji})\right)\right) \tag{3.1}$$

where $i = 1, 2, 3, ..., n$, $\vec{r}_i$ is the position vector of cell i, $\vec{v}_i$ is its velocity, $\vec{r}_{ji} = \vec{r}_i - \vec{r}_j$ and the summation is taken over the cells. Likewise, the growth cones' movements are governed by

$$\frac{d^2\vec{r}_i}{dt^2} = \frac{1}{m_g}\left(\vec{F}_{drag}(\vec{v}_i) + \vec{F}^i_{rule}(t) + \sum_{j\neq i}^{m}\vec{F}_{cone}(\vec{r}_{ji}) + \sum_{j\neq i}^{n}\vec{F}_{lgf}(\vec{r}_{ji}) + \sum_{j\neq i}^{n+m}\vec{F}_{collision}(\vec{r}_{ji})\right) \tag{3.2}$$

where $i = 1, 2, 3, ...$, $\vec{r}_i$ is the position vector of growth cone i, the first summation is taken over the growth cones, the second summation is taken over the cells, and the third summation is taken over all cells and growth cones. Each of these individual forces is described below.

A rule called the *movement criterion* causes agents that are moving slowly and not accelerating much to come to rest. Let $\vec{F}^i_{net}$ be the net force on the i^{th} agent given by Equation 3.1 or 3.2 excluding the frictional force $\vec{F}_{drag}$. The movement criterion is

$$\text{If } \|\vec{F}^i_{net}\| \leq \epsilon \text{ and } \|\vec{v}_i\| \leq \delta, \text{ then } \vec{F}^i_{net} \leftarrow \vec{0} \text{ and } \vec{v}_i \leftarrow \vec{0}.$$

Here, ϵ and δ are nonnegative constants that are usually small relative to the magnitudes of the typical net forces and velocities in the system and have values that depend on the agent type. This rule implies that a motionless agent will not accelerate until the net force on it exceeds ϵ. The usefulness of this rule is two fold. First, it is desirable that the velocities of agents reach zero in a relatively short amount of time in order to achieve a stable neural network configuration, but the viscous frictional force $\vec{F}_{drag}$ on an agent is directly proportional to its velocity and so the velocity of a slowing agent approaches zero asymptotically. The movement criterion remedies this by forcing a slowing agent's velocity to zero. Second, the

movement criterion provides greater control over the static equilibrium configurations that develop among the cells and does so without the necessity of modifying the intercellular force. These configurations are more stable because the cells are less susceptible to minor disruptive perturbations caused by neighboring cells.

3.3.1 Forces Governing Collective ("Swarm") Movements

The *intercellular force* governs the local interactions between the cells. It encompasses three interactions commonly used to induce "flocking" or "swarming" behavior in particle systems; namely, collision avoidance (separation), flock centering (cohesion) and velocity matching (alignment) [108]. The intercellular force that a cell j exerts on a cell i is given by

$$\vec{F}_{cell}(\vec{r}_{ji}) = \begin{cases} (b - cr_{ji}^{\alpha})e^{(-dr_{ji}^{\alpha})}\hat{r}_{ji}, & \text{if } r_{ji} \leq R_{cc} \\ 0, & \text{if } r_{ji} > R_{cc} \end{cases} \tag{3.3}$$

where R_{cc} is the size of a cell's local neighborhood when interacting with other cells, $\vec{r}_{ji} = \vec{r}_i - \vec{r}_j$ is the vector that points from cell j to cell i, $r_{ji} = \|\vec{r}_{ji}\|$, $\hat{r}_{ji} = \frac{\vec{r}_{ji}}{r_{ji}}$, and $b, c, d, \alpha \in \mathbb{R}^+$ are parameters used to tailor the shape of the function. It has a smooth transition from being repulsive at relatively close distances to attractive at further separations, and then decreasing to zero at still farther separations, as illustrated in Figure 3.2. It thus captures both the collision avoidance and flock centering influences, and although it does not explicitly account for velocity matching, this influence arises as a consequence of the attractive aspect of the force. Note that

the cut-off at $\|\vec{r}_{ji}\| = R_{cc}$ causes the force to adhere to the *local-interactions-only* restriction. Further, the distance at which the transition from repulsion to attraction occurs sets-up a characteristic equilibrium separation among cells in close proximity.

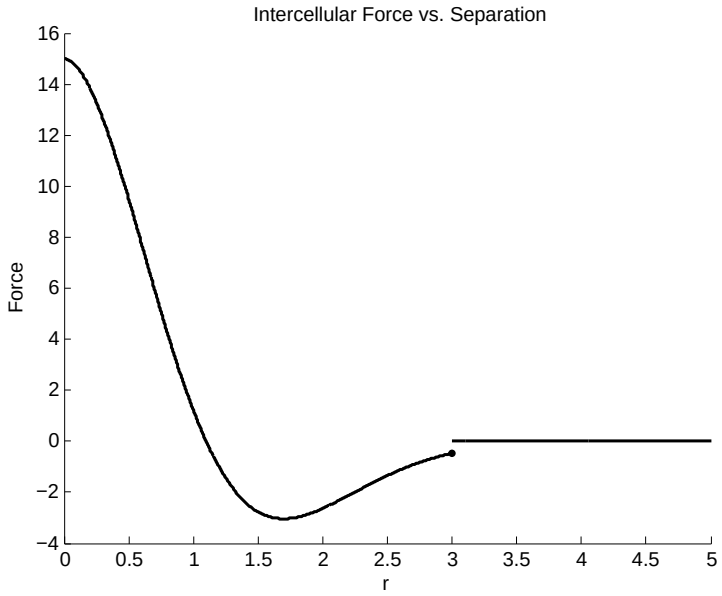


Figure 3.2: The magnitude and sign of the intercellular force as a function of the distance between two cells. A positive value indicates repulsion and a negative value indicates attraction (Eq. 3.3 with $R_{cc} = 3.0$, $b = 15.0$, $c = 12.6$, $d = 0.68$ and $\alpha = 1.84$).

Like cells, growth cones also interact through local forces inspired by flocking behavior. In this case however, implementing the three forces (separation, cohesion and alignment) explicitly produced a more useful collective dynamics. The combined effect of these three forces is represented by a single *intercone force* $\vec{F}_{cone} = \vec{F}_s + \vec{F}_c + \vec{F}_a$. Let $\vec{r}_i$ be the position of the i^{th} growth cone and $\vec{v}_i$ its velocity. Define R_{gg} as its neighborhood radius and $\mathbf{N}_i$ as the set of growth cones within a distance R_{gg}. The position of the i^{th} growth cone relative to its neighboring growth cones in $\mathbf{N}_i$ is given by $\vec{\rho}_i = \vec{r}_i - \frac{1}{|\mathbf{N}_i|}\sum_{j\in\mathbf{N}_i}\vec{r}_j$ and its relative velocity is $\vec{\mu}_i = \vec{v}_i - \frac{1}{|\mathbf{N}_i|}\sum_{j\in\mathbf{N}_i}\vec{v}_j$. The *separation force* $\vec{F}_s$ acts as a repulsive influence between neighboring growth

cones and prevents them from clustering together too closely. It takes the form

$$\vec{F}_s(\vec{\rho}_i) = k_s \left(1 - \frac{\rho_i}{R_{gg}}\right)^2 \hat{\rho}_i. \tag{3.4}$$

The *cohesion force* $\vec{F}_c$ causes neighboring growth cones to be attracted to one another, and is given by

$$\vec{F}_c(\vec{\rho}_i) = -k_c \left(\frac{\rho_i}{R_{gg}}\right)^2 \hat{\rho}_i. \tag{3.5}$$

The *alignment force* $\vec{F}_a$ causes neighboring growth cones to have similar velocities, which increases the uniformity of the collective movements. It takes the form

$$\vec{F}_a(\vec{\rho}_i, \vec{\mu}_i) = -k_a \left(\frac{\rho_i}{R_{gg}}\right)^2 \hat{\mu}_i. \tag{3.6}$$

Here, $k_s, k_c, k_a \in \mathbb{R}^+$, $\hat{\rho}_i$ and $\hat{\mu}_i$ are unit vectors, and $\rho_i = \|\vec{\rho}_i\|$.

A *local growth force* $\vec{F}_{lgf}$ also affects growth cones and is partly responsible for guiding axons from emitting cells to target cells. It represents a local force field that can appear and disappear around specific cells at certain times during the simulation depending upon information encoded in the rule set. As a growth cone passes through these fields its motion is affected, driving it towards or away from nearby cells. $\vec{F}_{lgf}$ is given by

$$\vec{F}_{lgf}(\vec{r}_{ji}) = \begin{cases} \pm k(r_{ji} + 1)^{-\beta} \hat{r}_{ji}, & \text{if } r_{ji} \leq R_{cg} \\ 0, & \text{if } r_{ji} > R_{cg} \end{cases} \tag{3.7}$$

where r_{ji} is the distance between the i^{th} growth cone and j^{th} cell, $\hat{r}_{ji}$ is the unit vector pointing from the cell to the growth cone, $k, \beta \in \mathbb{R}^+$ and R_{cg} is the radius of interaction between cells and growth cones. As specified in Table 3.3, the sign and magnitude of the parameter $\pm k$ are set using the SetLGF command function.

3.3.2 Environment and Rule-Based Forces

All agents are subject to a drag-like friction force that limits their acceleration and prevents the center of mass of the agents from drifting continuously, making this a dissipative system where, under most circumstances, it will come to rest within a reasonable period of time, forming a static network with a particular geometric structure. The force is given by

$$\vec{F}_{drag}(\vec{v}_i) = -c_d \vec{v}_i \tag{3.8}$$

where $\vec{v}_i$ is the velocity of the i^{th} agent and $c_d \in \mathbb{R}^+$.

In order to make the model more realistic in a physical sense, agents collide rather than simply passing through one another. Recall that both cell and growth cone agents are modeled as spheres, not as points. Collisions are modeled using a penalty method (also known as soft collisions), in which a very strong, short range force is engaged between two agents when their boundaries intersect. This force will be on the order of 10 to 1,000 depending on the type of agents colliding and the distance between their centers. Consider an agent i and an agent j and let d_c equal the sum of the two agents' radii. Then the collision force exerted on agent i is given by

$$\vec{F}_{collision}(\vec{r}_{ji}) = \begin{cases} c_f(d_c - \|\vec{r}_{ji}\| + 1)^{\gamma}\hat{r}_{ji}, & \text{if } \|\vec{r}_{ji}\| \leq d_c \\ 0, & \text{if } \|\vec{r}_{ji}\| > d_c \end{cases} \tag{3.9}$$

where $\vec{r}_{ji}$ is the vector that points from the center of agent j to the center of agent i, $\hat{r}_{ji}$ is the corresponding unit vector and $c_f, \gamma \in \mathbb{R}^+$ are constants.

Recall that the command SetRuleForce causes a constant force to be applied to an agent for a finite period of time (Section 2.2, Table 3). An agent may experience a sequence of such forces that depends entirely on its execution of these commands. For the i^{th} agent, the time-dependent function representing this sequence of forces is expressed as $\vec{F}^i_{rule}(t)$ and will be referred to as the *rule-based force*. This force is implemented as a means of tailoring the movements of the agents. It is especially useful for manipulating the trajectories of growing axons and for positioning cells.

3.4 Implementation

The SINSA model is implemented as a simulation environment that is written in Java. The system of ordinary differential equations given by Equations 3.1 and 3.2, which govern the force-based dynamics of the cells and growth cones respectively, have no analytical solution and must therefore be solved numerically. To do so the Forward Euler method is used with a time-step size of 0.02. This numerical scheme allows the simulator to produce dynamics that accurately reflect the growth processes pertinent to the simulations and computational experiments. The two simulations presented in Section 3.5 each ran on a computer with a dual-core 2.27

GHz Intel Core i5 processor, with 4 GB of RAM, and 3 MB of L2 cache. The simulation of axon growth requires about 40 seconds to complete, and the simulation of cell layer growth takes about 3 minutes.

3.5 Simulations of Neural Growth

In this section a number of the basic capabilities of the SINSA model are demonstrated through simulations of neural growth. The the first simulation illustrates the ability of the model to grow layers of cells with different orientations, densities, and spatial patterns. The second simulation demonstrates the model's ability to generate coordinated axonal growth to specific target cells over substantial distances, and in the presence of obstacles and potentially disruptive forces. These two types of behaviors are fundamental to the development of neural networks grown using the model, and are important aspects of many systems that involve network growth, both real and simulated.

Network growth in the SINSA model incorporates the self-assembly of cells (nodes) into physical configurations appropriate to the development of patterns of connectivity among them. To this end, layers of cells are very common structures because they make the development of a wide range of connectivity patterns far less complex through their planar modular structure. This structure arises naturally in many situations because it helps to avoid substantial variations in the spatial positions of a neuron's target cells without creating clutter and it is often the case that many neurons have at least some of the same target cells. Figure 3.3 shows

the results of a simulation in which four distinct layers of cells emerged from a single "seed" cell. The parameters and rule set used in this simulation are listed in Appendix B. Each of the four layers was grown using the same basic pattern of cell divisions, but the cells in different layers executed different SetRuleForce commands, which is why the layers have different densities and geometries. The intercellular force was largely responsible for establishing the appropriate separations among the cells and moderating their cohesion, within each layer, while the movement criterion increased the stability of the layers.

Figure 3.3: (Next page). An example of cell layer growth generated by the SINSA model. The spheres represent cells. *a.* The initial state consists of a single "seed" cell, shown in its initial position. *b* The original seed cell divided producing two child cells each of which divided to produce four new cells. These new cells are seen migrating to their final positions. *c.* Upon reaching their final positions the cells have started to divide, beginning the process of layer formation. *d.* Repeated cell divisions cause the layers to expand. Structural differences between the layers are becoming apparent. *e.* Continued growth has accentuated the discrepancies in layer structure, which are the result of the cells in different layers executing different SetRuleForce commands. *f.* All cell divisions have stopped and the layers are fully formed.

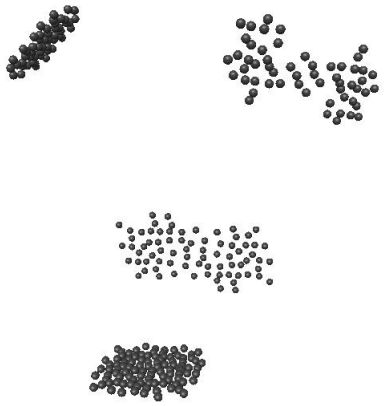
During the process of network self-assembly axons (edges) must grow through a three-dimensional space, possibly over long distances, in order to establish connections with what is often a very specific group of target cells. During the course of establishing a connection from its emitting cell to a target cell, a growth cone (the leading tip of a growing axon) will typically encounter a variety of forces and circumstances that it must be capable of negotiating in order to reach its final destination. Figure 3.4 illustrates the results of a simulation in which growing axons had to navigate around a cluster of cells acting as obstacles in order to reach their targets on the other side. The parameters and rule set used in this simulation are listed in

a.

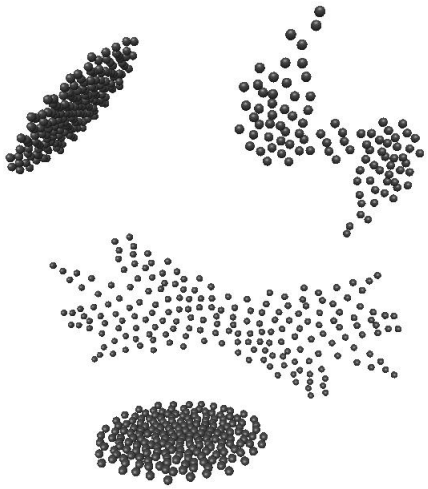
b.

c.

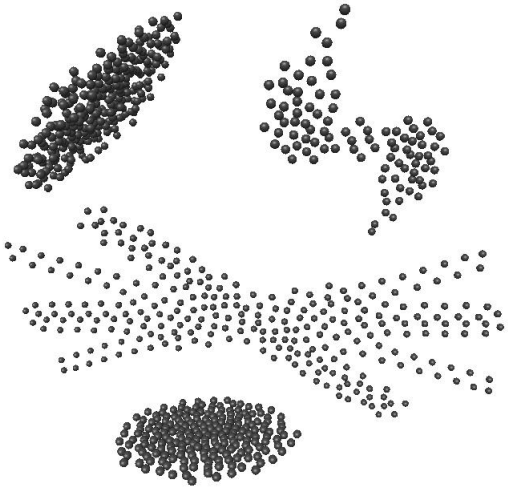
d.



e.



f.



Appendix B. The rule set dictated that the cell depicted on the far left should emit two axons, that these axons should branch, and that each growth cone has a velocity component in the general direction of their target cells (via the SetRuleForce command). The obstacle cells (middle) exhibited a repulsive local growth force, and hence the growth cones were unable to navigate directly through this cluster of non-target cells. Rather, through the influence of the intercone force they were able to quickly and collectively find their way around the central cluster towards their target cells (far right). The target cells exhibited an attractive local growth force, which when combined with the intercone force guided the growth cones to their targets and yielded a uniform distribution of connectivity.

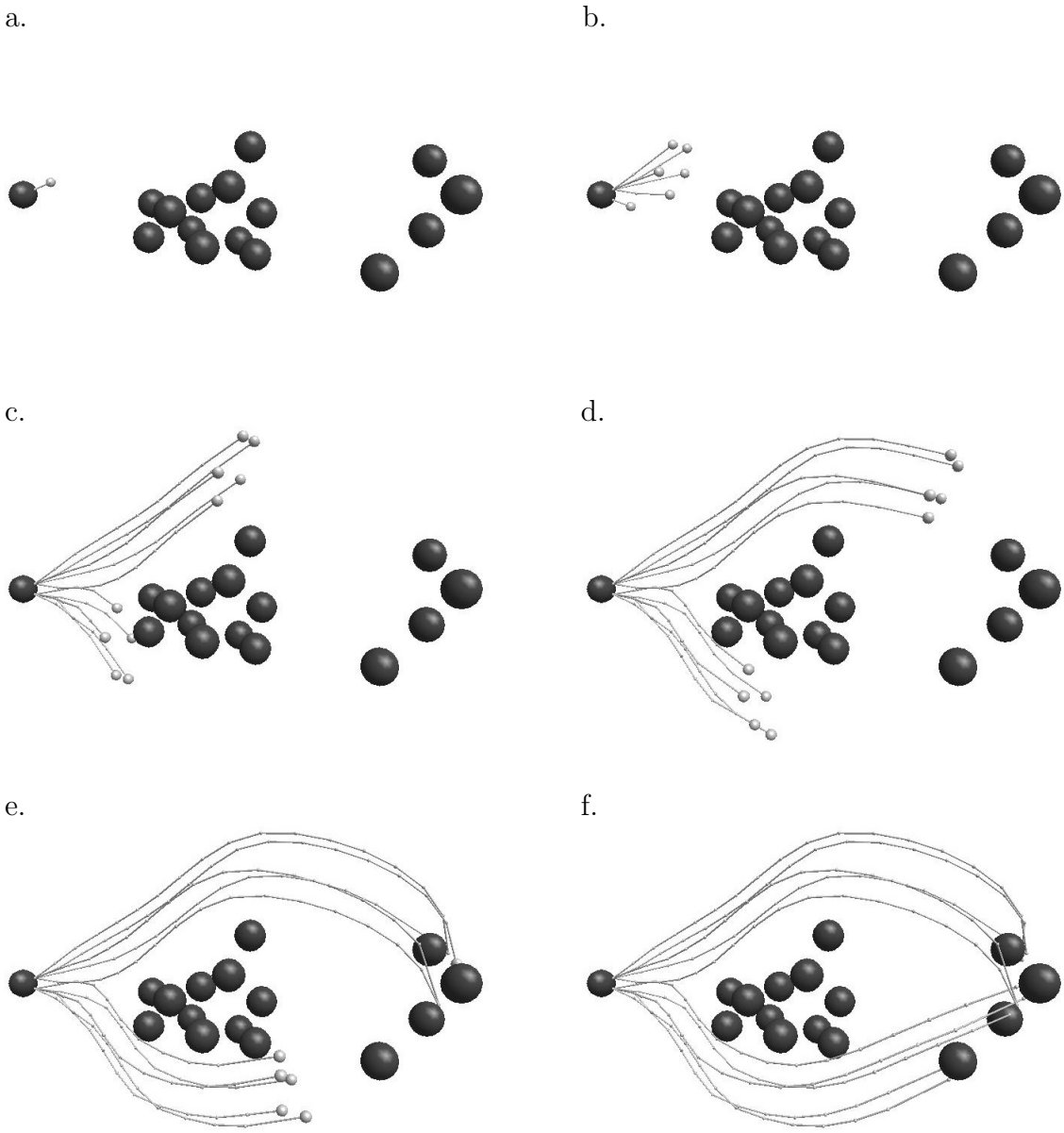


Figure 3.4: An example of axonal growth generated by the SINSA model. The large spheres represent cells, the lines between cells denote axons, and the small spheres at the ends of growing axons represent growth cones. *a.* Early in the growth process the cell on the far left has emitted an axon. *b.* A short time later the same cell has emitted an additional axon (bottom) and the first axon has branched (top). Each axon is being guided by a growth cone towards the four target cells shown on the far right, but must go around the cluster of cells in the middle. *c*, *d.* As the growth cones encounter the central cluster of cells they collectively adjust their motions so as to maneuver around these obstacles. *e.* The axons begin to establish connections with the target cells. *f.* Growth has stopped with two to three connections having been established with each of the four target cells.

Chapter 4

Enhancing Network Self-Assembly through Collective Movements

This chapter describes the experimental methods and results of five computational experiments that were performed with the SINSA model. These experiments were designed to evaluate whether there are any benefits to incorporating swarm intelligence in the form of collective movements into the neural self-assembly process, such as substantial improvements in the controllability of the grown networks' structural characteristics, and/or improvements in the robustness of the growth process.

4.1 Experimental Methods

Here some basic information is given regarding the neural models around which a number of the experiments are based. Additionally, the motivating factors behind the experiments are mentioned, as are some of the unique aspects of the SINSA model. This is followed by an outline of the experimental setups. Lastly, the metrics used in the experiments are explained.

4.1.1 Computational Experiments

In the first two experiments the SINSA model was used to grow self-organizing maps (SOM's) [81, 109], which are widely-studied neural networks that have the ca-

pacity to learn statistical patterns or features in their input data in an unsupervised manner. Two previously published SOM's of idealized cerebral cortex regions were selected as target structures for these experiments. One is a topographic map of a patch of somatosensory cortex involving restricted input-to-cortex connectivity [119], and the other is a more complex feature map of a patch of visual cortex involving full input-to-cortex connectivity as well as locally connected excitatory and inhibitory cortical neurons [90]. These two networks were originally hand-designed and are representative of a broad class of neural network models of self-organizing maps in the cerebral cortex that have generally been manually constructed in the past. The versions of these networks grown with the SINSA model do not necessarily reproduce the patterns of connectivity present in the biological neural networks better than the two original SOM models they are derived from because the intent was to grow networks that approximate the topologies of the models. However, the grown SOM's are more like biological systems in that they do not have periodic boundary conditions and their connectivity reproduces that of the original networks in a statistical sense. Because of this difference, once the networks were fully grown and stable, their abilities to actually form maps during unsupervised Hebbian learning were assessed, comparing their functionality in this respect to that of the original models.

The SOM's that the SINSA model was used to grow are especially challenging for developmental models to generate, particularly those that incorporate a continuous growth process, for several reasons. The networks are large, each consisting of more than 600 neurons and more than 20,000 connections. There is a significant

amount of concurrent axonal growth, which is more like what occurs biologically but complicates the environment in which development takes place. Also, because the SINSA model accounts for network geometry as well as topology, there is the added challenge of dealing with inhomogeneities in the positions of the neurons. Lastly, the growing axons respond to different rules at different times and under different environmental conditions, which means that in the absence of inter-agent forces it is typically very difficult to find an appropriate set of parameters and a parsimonious set of rules that result in the growth of the desired network. Because of the difficulty in specifying rules and parameters to guide network development without inter-agent forces, many of the larger, more complex networks grown by past developmental models have relied upon rules and parameters derived by optimization methods, such as genetic algorithms or genetic programming, as opposed to being hand-written [3, 34], and still these networks are far less complex than the SOM's grown using the approach presented in this disseration, which are based on fixed parameter values and hand-written rule sets, each of which required only about 5 person-days to write. The computational experiments presented herein thus demonstrate the effectiveness of incorporating collective movements into the model and treating the growth process of relatively large and complex networks as one of self-assembly. They also show that the needed rule sets are small enough and intuitive enough that they can be written by hand without the aid of an optimization method. To assess the success of this approach, the resulting network architectures are evaluated by visual inspection, use of the $M1$ and $M2$ similarity measures described below, and by running the resultant networks after self-assembly to verify

that they produce good self-organized maps.

One of the significant drawbacks of many past developmental models of neurogenesis is their lack of robustness. That is, relatively minor changes in the rules or parameters of these models often result in large and unpredictable changes in network growth, which in turn makes it very difficult to control the characteristics of the networks that ultimately emerge. In the third, fourth and fifth experiments the SINSA method was used to grow networks from a variety of randomly modified rule sets, using both a version of the approach that incorporates collective movements and one that does not. Along with these variations the cohesion and velocity alignment forces (Eqs. 3.5 and 3.6) were manipulated by varying the parameters k_c and k_a respectively. These experiments study the effects that swarm intelligence produced by collective movements has on the robustness of the SINSA model.

4.1.2 Implementation Details

In the simulations of cortical network models below, the environment in which networks grow is unbounded and no particular units are assigned to time, distance and mass. Cells and growth cones have radii of $r_c = 0.5$ and $r_g = 0.17$ respectively, and the grown networks have spatial extents of approximately 20 distance units. For each force and the movement criterion Table 4.1 lists the parameter values used in the simulations. A few additional conditions are placed on the agents, as follows. The velocities of all agents are bounded above by $v_{max} = 10.0$. A cell agent of type "AS", "ES" or "IS" dies (disappears) as soon as its *Local Time* exceeds 0.2 time

Table 4.1: Parameter Values Used in the Computational Experiments

Force	Parameter	Value
Equations of Motion (Eqs. 3.1, 3.2)	m_c	1.0
	m_g	0.5
$\vec{F}_{cell}$ (Eq. 3.3)	R_{cc}	3.0
	α	1.84
	b	15.0
	c	12.6
	d	0.68
$\vec{F}_{cone}$ (Eqs. 3.4, 3.5, 3.6)	R_{gg}	2.5
	k_s	10.0
	k_c	10.0
	k_a	10.0
$\vec{F}_{lgf}$ (Eq. 3.7)	R_{cg}	3.0
	$\pm$	$+$
	k	10.0
	β	1.66
$\vec{F}_{drag}$ (Eq. 3.8)	c_d	5.0
$\vec{F}_{collision}$ (Eq. 3.9) Collisions between cells and growth cones.	r_c	0.5
	r_g	0.17
	c_f	75.0
	γ	3.0
$\vec{F}_{collision}$ Collisions between cells.	c_f	100.0
	γ	5.0
$\vec{F}_{collision}$ Collisions between growth cones.	c_f	25.0
	γ	3.0
Movement Criterion for cells.	ϵ	20.0
	δ	10.0
Movement Criterion for growth cones.	ϵ	1.0
	δ	1.0

units. This condition ensures the removal of cells that have the ability to divide but won't because they don't satisfy the appropriate predicates in the rule set. If a growth cone comes in contact with a cell, it establishes a connection with that cell if and only if the cell is of type "E" or "I" and is inducing an attractive local growth force. Whenever new cell or growth cone agents are created, which occurs when a "parent" cell or growth cone executes a rule containing an EmitAxon, DivCell or DivCone command, their initial velocities are the same as the velocity of their parent agent at the time the rule was executed. When a cell divides the centers of the two child cells that replace it are initially a distance of $1.05r_c = 0.525$ from where the parent cell's center was located. When an axon branches the newly formed growth cone is a distance of $2r_g = 0.34$ from the center of its parent growth cone. When a cell emits an axon the center of the newly created growth cone that guides the axon is a distance of $2r_c + r_g + 0.1 = 1.27$ from the center of the emitting cell. All agents share a common reference frame (orientation). This means that given a rule that has a command function that takes a vector as an argument (e.g., variables *direction* or *force*), all agents that execute the rule will interpret the vector with respect to the same fixed coordinate system (basis), even though the agents do not have any explicit reference to a global coordinate system. Lastly, a growth cone agent dies, along with the axon branch connected to it, if its *Local Time* exceeds 3.5 time units. This property is reminiscent of the "pruning" of growing axons that occurs in developing biological nervous systems, and it ensures the removal of the very small fraction of growing axons that originate from cells at layer boundaries and fail to establish connections with cells in neighboring layers.

The five computational experiments each ran on a computer with two dual-core 2.66 GHz Intel Xeon processors, 4 GB of RAM, and 4,096 KB of L2 cache. Two of the experiments consist of growing large recurrent neural networks based on the somatosensory cortex model presented in [119] and the visual cortex model presented in [90]. It takes 1.3 hours of CPU time to grow the somatosensory cortex network (612 cells and 21,188 connections), and 3.8 hours to grow the visual cortex network (631 cells and 46,030 connections). In each of the other three computational experiments a network consisting of 551 cells and approximately 61 connections is grown 1,616 times using different rule sets. Each of these three experiments requires approximately 9 hours of CPU time to complete.

4.1.3 Connectivity Measures

The simulator that implements the SINSA model grows networks in which a neuron may connect with another neuron multiple times and with topologies that are similar, in a statistical sense, to precisely specified templates of connectivity. This means that given a neural network model with a precisely specified topology, such as those often used in neural network applications (e.g., [90, 119]), it is necessary to measure how statistically similar the topology of the network grown by the simulator is to that of the template network. This is accomplished by defining two *similarity measures*, applying them to the connectivity of each neuron individually, and then computing the average of the two measures over all neurons in the network. Given a set of target neurons $\mathbf{T}$ that a neuron n is intended to connect to, the first measure

$M1$ = the percentage of connections from n that connect to a neuron in $\mathbf{T}$

quantifies the degree to which a neuron makes connections to target neurons as opposed to non-target neurons. The second measure

$M2$ = the percentage of neurons in $\mathbf{T}$ that receive at least one connection from n

quantifies the degree to which a neuron makes a connection to all of its target neurons. The averages are taken over all neurons so as to capture the global similarity of the grown network to its template network. The connectivity template is adhered to exactly when both $M1$ and $M2$ are 100%.

Two of the networks grown by the simulator use the networks discussed in [90] and [119] as templates. These papers present a number of experiments that demonstrate some of the computational properties of the template networks. A number of similar experiments were performed using the corresponding networks grown by the simulator, the results of which are used as an additional gauge on the topological similarity between the grown networks and their templates. The geometries of these template networks were general enough (e.g., multiple uniform layers of cells), that it was sufficient to determine the similarity of a grown network's geometry in a qualitative manner by visualizing it using the simulator's 3D graphics component. The desired numbers of cells in the grown networks were obtained by writing rules having appropriate values for the *life* variable. Though important for the growth process, the particular paths taken by growing axons and migrating cells were not rigorously analyzed because the computational properties of the grown

networks depend only on their topologies, and not on qualities such as the lengths of their axons.

4.2 Results

Here the results of the five computational experiments performed with the SINSA model are described. The results of the first two experiments demonstrate the ability of the model to grow large recurrent neural networks based on specific target networks, and that the computational properties of the grown networks are similar to those exhibited by their target models. The results of the other three experiments illustrate the improvements in robustness that can be gained by incorporating swarm intelligence in the form of collective movements into the neural self-assembly process.

4.2.1 Somatosensory Cortex Model

In [119], the authors present a SOM of the primary somatosensory cortex (area S1) in which the network input represents focal stimulation of a hand consisting of four fingers and a palm, and the topographic map depicts different areas of the hand and the frequency with which particular regions have been touched. The first layer of the network represents a region of the thalamus and the second layer represents a region of the primary somatosensory cortex. Both the thalamic and cortical neurons act as excitatory cells, and input to the network from the hand enters through the thalamic layer. Figure 4.1a illustrates the patterns of connectivity in this recurrently connected network. Each neuron in the thalamus connects to its 30 closest cortical

neurons and each cortical neuron connects to its six nearest cortical neighbors. In [119], it was demonstrated through a series of computational experiments that this model is sufficient to reproduce a number of important properties exhibited by cortical maps. For example, it was shown that in response to uniformly distributed input stimuli the SOM organizes into a highly-refined topographic map from an initially coarse topographic representation, and that repeated stimulation of a localized region of the sensory layer results in a substantial increase in the degree of cortical representation of that region.

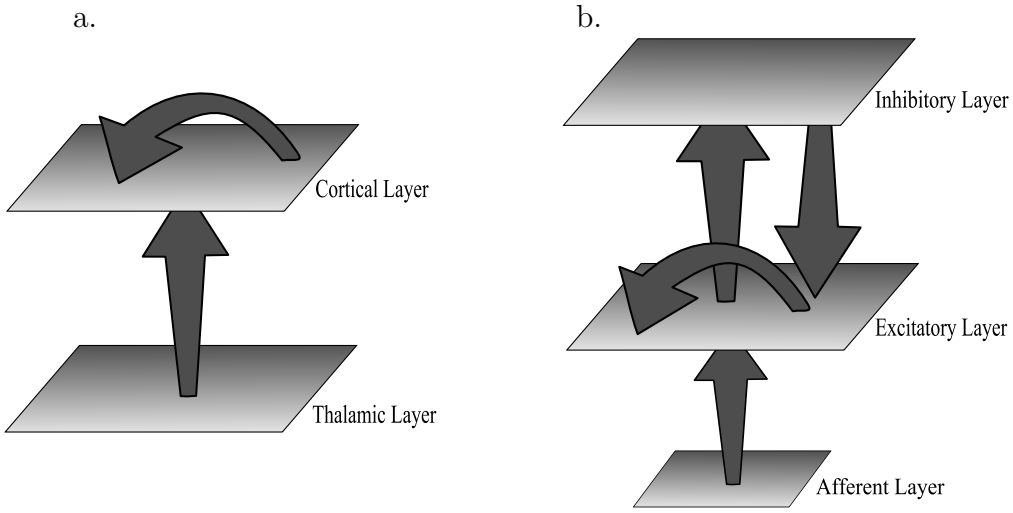


Figure 4.1: *a.* The connectivity of the somatosensory cortex model. The planes represent the layers of cortical and thalamic neurons. *b.* The connectivity of the visual cortex model. The afferent layer corresponds to retina, the other two layers to cortical neurons. The arrows indicate general inter/intra-layer patterns of connectivity.

The SINSA method was applied to grow a network, using the network underlying the self-organizing map described in [119] as a target. The early stages of the growth process are shown in Figure 4.2. This network consists of two layers of neurons, each of which is required to establish roughly topographic connections with

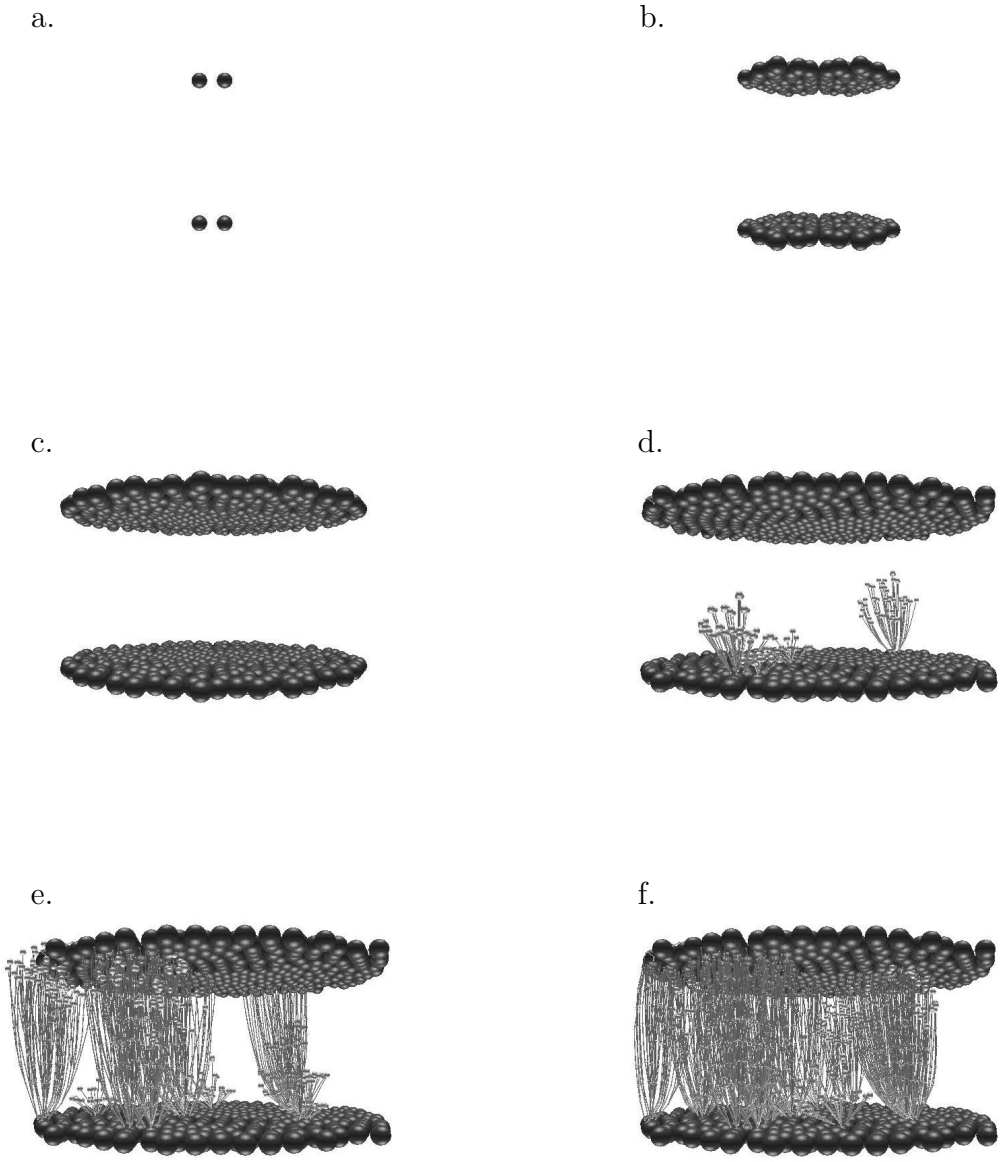


Figure 4.2: An example of network growth generated by the SINSA model. The large spheres represent cells, the lines between cells denote axons, and the small spheres at the ends of growing axons represent growth cones. *a.* The network seen here develops from a set of four "seed" cells, shown in their initial positions. *b*, *c.* Repeated cell divisions cause the thalamic (bottom) and cortical (top) layers to expand. *d.* Cell divisions have stopped and the axons are beginning to grow. *e.* The axons begin to establish connections. The lateral cortical connections are not shown. *f.* Early on in the growth process additional cells have emitted axons. The axons continue to grow and establish connections.

sizable groups of target neurons, and to do so in an environment complicated by a substantial amount of concurrent axonal development. The grown network is quite large, consisting of 612 cells and 21,188 connections, and is a recurrent network, which further increases the difficulty of growing it. The similarity measures of the fully grown network were $M1 = 87\%$ and $M2 = 76\%$. The SINSA model used a rule set consisting of 58 rules to grow the network. Twelve of these rules regulate cell divisions and dynamics, 44 regulate axon emissions, branching and dynamics, and 2 control the local growth force. This rule set and the initial conditions are specified in Appendix C, and a brief explanation of how the rule sets were designed is given in Appendix A. The growth process was considered to be completed once each cell had emitted all of its axons and no axons were still growing.

To assess the degree to which the grown S1 network possesses the computational properties of the original S1 model [119], the ability of the grown network to replicate the main map formation results obtained with the original network was tested. More specifically, to measure the characteristics and quality of the topographic maps formed by the grown network, a comparison was made between measures of the cortical receptive field of each cortical cell before and after training, using the same measures as in [119]. The network grown by the SINSA approach consists of two layers of neurons that are parallel to the x-y plane. Let x_i and y_i be the x and y coordinates of thalamic cell i and a_{ji} the activity level of cortical cell j when an input of 1.0 is applied to thalamic cell i and an input of 0.0 is applied to all other thalamic cells. The *total response* r_j of cortical cell j is defined as

$$r_j = \sum_i a_{ji}, \tag{4.1}$$

where the sum is taken over all thalamic cells. The receptive field of cortical cell j, which measures how a cortical cell responds to the activities of the thalamic cells, is characterized by its *center* and its *moments*. The coordinates of its center are given by

$$\hat{x}_j = \tfrac{1}{r_j} \textstyle\sum_i x_i a_{ji} \quad \text{and} \quad \hat{y}_j = \tfrac{1}{r_j} \textstyle\sum_i y_i a_{ji}.$$

Its x and y moments are defined by

$$wx_j = \sqrt{\left(\tfrac{1}{r_j} \textstyle\sum_i (x_i - \hat{x}_j)^2\right) a_{ji}} \quad \text{and} \quad wy_j = \sqrt{\left(\tfrac{1}{r_j} \textstyle\sum_i (y_i - \hat{y}_j)^2\right) a_{ji}}.$$

The moments measure the spread of a cortical receptive field.

The two primary computational experiments concerning map formation presented in [119] were performed using the same activation dynamics, input-stimuli, Hebbian synaptic changes, and parameter values, but with the thalamocortical network grown using the SINSA method. The first experiment tested the ability of the grown but initially untrained network to self-organize into a well-formed topographic map when trained using a uniformly distributed, random sequence of stimuli. In the second experiment the network trained in the first experiment was subjected to further training, but in this case the random sequence of input stimuli was distributed such that the second finger from the left (finger 2) was seven times more likely to be stimulated than the rest of the hand. This experiment tested the ability

of the network to adapt its topographic map in response to a change in the input distribution.

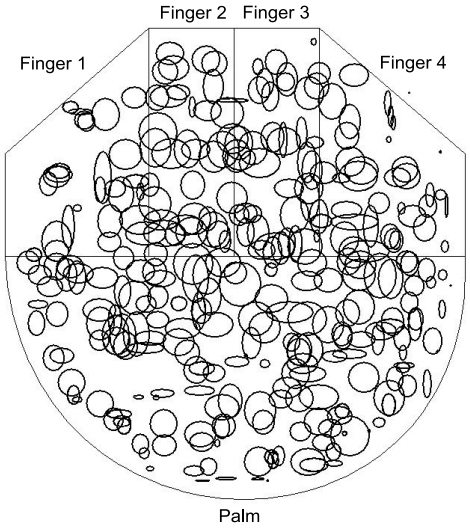
The quality of the resultant topographic maps were assessed visually in terms of the same three characteristics used in the original study: reduction in the size of receptive field moments, development of receptive field centers that reflect the probability distribution with which the thalamic neurons were stimulated, and the extent to which receptive field centers of neighboring cortical neurons become close to one another. Figures 4.3a,c,e illustrate the first two of these characteristics. The ellipses show the centers and moments of the cortical receptive fields plotted in the x-y plane of the input layer (strictly speaking, these are not the full receptive fields but are proportional in size to them). The outline of the hand (four fingers and a palm) is shown. The receptive fields in Figure 4.3a are for the initial untrained network. Their moments are relatively large and their centers have a somewhat irregular distribution across the hand, as in the original study. For the untrained network the average x moment $\overline{wx}$ is 1.008 and the average y moment $\overline{wy}$ is 1.031, with standard deviations $\sigma_{wx} = 0.417$ and $\sigma_{wy} = 0.442$ respectively. In contrast, the receptive fields in Figure 4.3c, which are those of the network trained with the uniformly distributed stimuli, have much smaller moments and their centers are more evenly distributed across the hand. In this case $\overline{wx} = 0.582$ and $\overline{wy} = 0.578$, with standard deviations $\sigma_{wx} = 0.184$ and $\sigma_{wy} = 0.209$ respectively. These are two of the characteristics of high quality topographic maps [119]. Figure 4.3e shows the cortical receptive fields derived from a network in which the second finger from the left was seven times more likely to be stimulated during training than the rest of the

hand. This bias in the stimulations has caused some of the receptive fields from finger 1 and finger 3 and neighboring regions of the palm to shift their centers towards and into the region of the thalamic layer that represents finger 2. Furthermore, the moments of the receptive fields in the finger 2 region have become smaller on average. Specifically, there are 33 receptive field centers in the finger 2 region of the network trained with uniform stimuli, and for these receptive fields $\overline{wx} = 0.572$, $\overline{wy} = 0.593$, $\sigma_{wx} = 0.148$ and $\sigma_{wy} = 0.150$. For the network trained with the biased stimuli the number of receptive fields in the finger 2 region has increased to 68, with $\overline{wx} = 0.432$, $\overline{wy} = 0.456$, $\sigma_{wx} = 0.253$ and $\sigma_{wy} = 0.251$ for these receptive fields. This means that more cortical cells have become selectively tuned to stimulations of finger 2. This adaptability to changes in the input distribution is one of the desirable properties of SOM's, and from a qualitative standpoint, these results match-up well with those presented in [119].

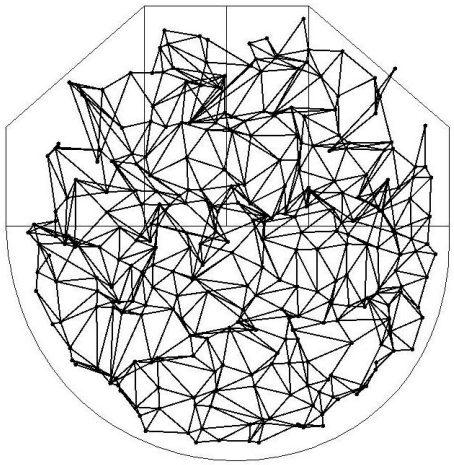
Figure 4.3: (Next page). In *a.-c.* the ellipses indicate the centers and moments of cortical receptive fields plotted in the *x*-*y* plane of the input layer. The lines indicate the boundaries between the four fingers and palm of a hand (bottom region). *a.* Receptive fields of the untrained version of the thalamocortical network grown using the SINSA approach are large and have a somewhat irregular spatial distribution. *b.* Receptive fields from the network trained with uniformly distributed stimuli are much smaller and have a more uniform spatial distribution. *c.* Receptive fields derived from the thalamocortical network trained with finger 2 seven times more likely to be stimulated than the rest of the hand. Some of the receptive fields from surrounding areas have moved towards and into the region of the thalamic layer that represents finger 2 and become even smaller in size. This indicates that the network grown using the SINSA method possesses the topographic adaptability that is characteristic of SOM's. In *d.-f.* the dots indicate the centers of cortical receptive fields plotted in the *x*-*y* plane of the input layer. Dots are connected if they belong to the receptive fields of neighboring cortical neurons. *d.* Receptive field centers of the untrained version of the thalamocortical network grown by the SINSA approach. They indicate a lack of topographic organization in the cortical layer. *e.* Receptive field centers from the network trained with uniformly distributed stimuli. The increased regularity in the grid structure is one of the indicators of a high quality topographic map. *f.* Receptive field centers derived from the thalamocortical network trained with finger 2 seven times more likely to be stimulated than the rest of the hand. A significant number of the receptive field centers have shifted into or towards this finger, illustrating the map's adaptability.

Figures 4.3b,d,f illustrate the third characteristic used to measure the quality of topographic maps. The dots represent the centers of the cortical receptive fields. The edges connect dots that belong to receptive fields of cortical neurons whose centers are within a distance of 1.6 of one another, which for most cortical neurons defines a neighborhood consisting of the six nearest cortical neighbors. Figure 4.3b was derived from the untrained thalamocortical network. The large degree of irregularity in the grid structure (edges cross one another 482 times) is indicative of poor topographic map formation. That is, many of the receptive fields that belong to neighboring cortical neurons are not themselves neighbors in the input space. In contrast, Figure 4.3d, which comes from the network trained with uniformly distributed stimuli, exhibits a far more regular grid (edges cross one another only 53

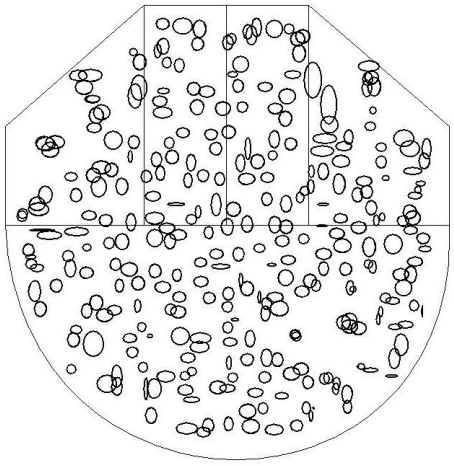
a.



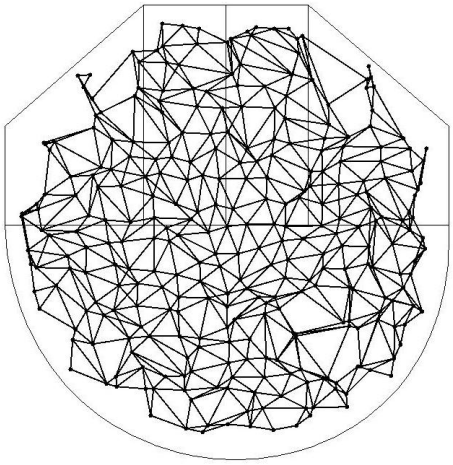
b.



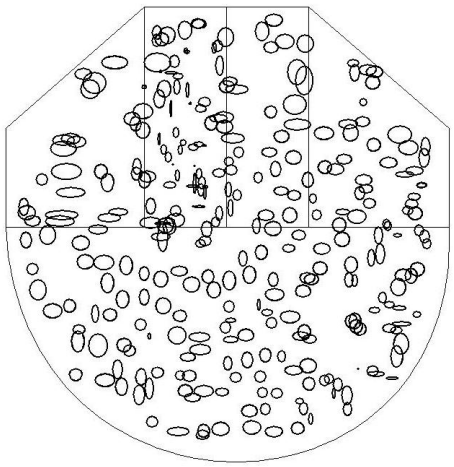
c.



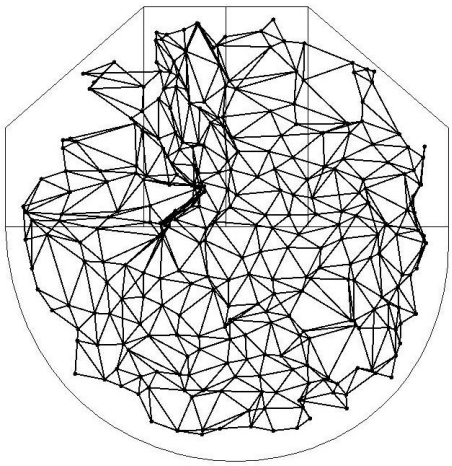
d.



e.



f.



times), and thus indicates the formation of a better topographic map. In this case, receptive fields that are topological neighbors tend to be geometric neighbors in the input space as well. Figure 4.3f shows the reorganization of the topographic map in response to an increase in the frequency with which finger 2 was stimulated relative to the rest of the hand. The map can be seen to have adapted, in that a significant number of the receptive field centers in regions surrounding finger 2 have moved towards and into the region of the thalamus representing this finger while largely preserving the topological/geometrical neighbors duality (edges cross one another 198 times). This is further evidence that the network grown via the SINSA approach possesses the desirable characteristics of SOM's. These results are all qualitatively very similar to those observed in [119], demonstrating that the grown network is as equally suited for SOM formation as the original model.

4.2.2 Visual Cortex Model

In [90], the author presented the first and one of the most influential models of self-organizing maps. It was proposed as an explanation for how organization of visual cortex (area V1) might arise through self-organization (learning) rather than being entirely genetically predetermined. Unlike the topographic map of somatosensory cortex described in the preceding section, this model is a *feature map*: a map of the sensitivity of cortical excitatory neurons to the orientation of lines that pass through their receptive fields. Further, this visual cortex model differs from the somatosensory model in that it has a more complex architecture, and one

that involves highly constrained recurrent connectivity. Past developmental models of neural network growth have most often focused on the easier task of producing feedforward networks without recurrent connections. Figure 4.1b shows the architecture of the visual cortex model, which consists of an afferent layer (retina), and two hexagonally-tessellated cortical layers of excitatory and inhibitory neurons. Recurrent connectivity arises because each excitatory neuron must connect to its six immediately-adjacent excitatory neurons and to a corresponding local patch of inhibitory neurons, the latter of which send connections back to a larger annulus of excitatory neurons. This recurrent connectivity among the excitatory and inhibitory neurons results in short range excitation and longer range inhibition among the excitatory cells, which constitute the map forming layer. Given "bars" (lines) of activity as input stimuli, it was shown in [90] that this model is able to self-organize so as to qualitatively reproduce the type of topographic feature map of orientation sensitive neurons observed in the visual cortex.

Figure 4.4 shows the early stages of the developmental process of a SOM modeled after the one developed by von der Malsburg and grown using the SINSA methodology. This network has a complex topology and there is a large amount of simultaneous axonal development. The network is also very large, consisting of 631 cells and 46,030 connections by the time the growth process is complete, and was grown with a rule set that contains 177 rules. Fifteen of those rules regulate cell divisions and dynamics, 160 regulate axon emissions, branching, and dynamics, and the remaining 2 govern the local growth force. The network consists of the same three layers as the original: an afferent (input) layer at the bottom, which is

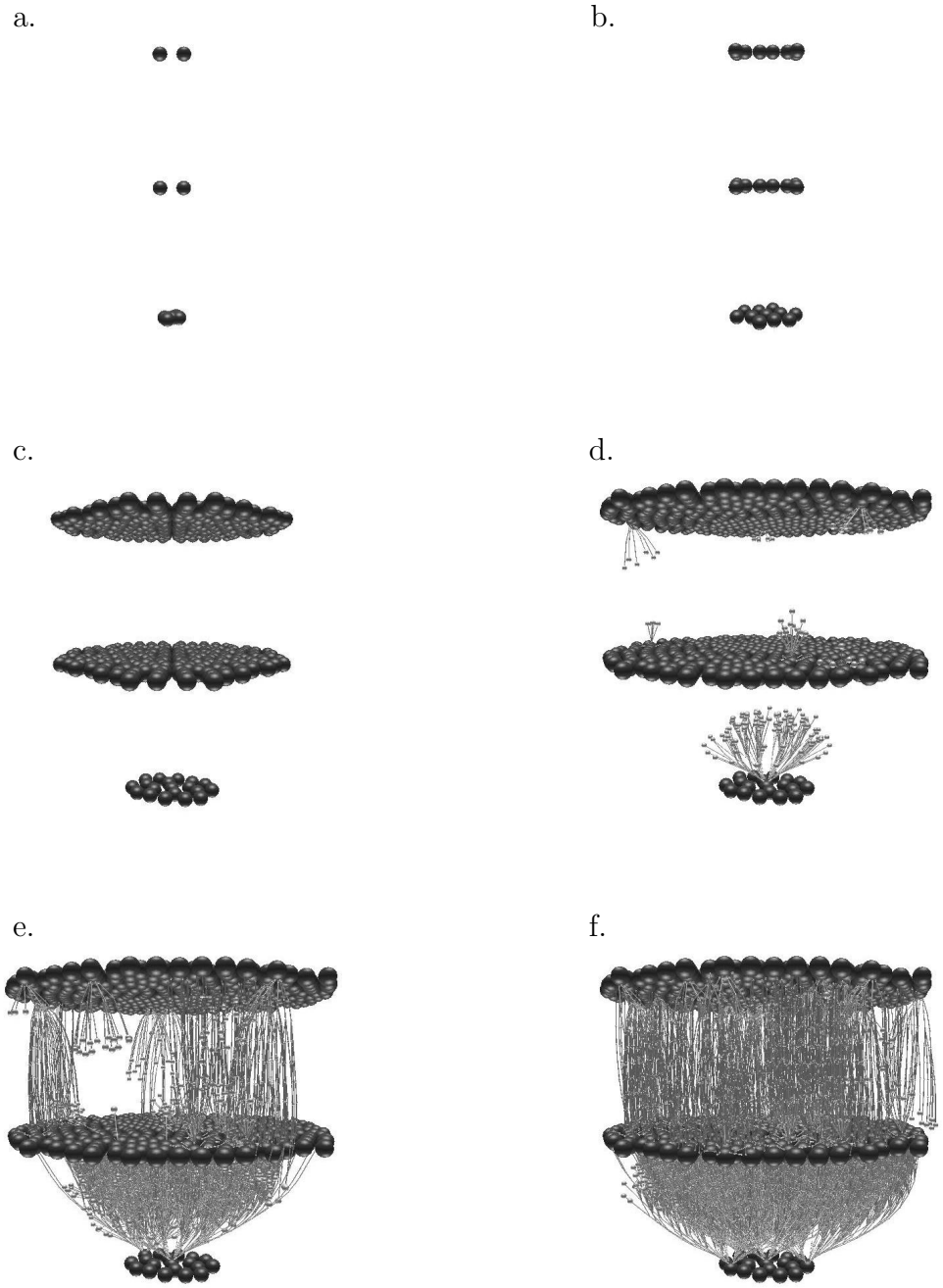


Figure 4.4: The V1 (visual cortex) network grown using the SINSA model. *a.* The network develops from a set of nine "seed" cells, shown here in their initial positions. *b.* Repeated cell divisions cause the layers to expand. *c.* The fully formed afferent (bottom) layer, and partially formed excitatory (middle) and inhibitory (top) layers. *d.* Cell divisions have stopped and the axons have begun to grow and establish connections. *e.* Recurrent connectivity similar to that of the original target model [90] is forming between the excitatory and inhibitory layers. *f.* Early on in the growth process additional cells have emitted axons. The axons continue to grow and establish connections.

intended to model a retina; a layer of excitatory cells in the middle, which is where map formation occurs; and a layer of inhibitory cells at the top. The excitatory and inhibitory layers jointly represent a patch of the striate cortex. The recurrent network grown via the SINSA method based on the template presented by von der Malsburg had $M1 = 73\%$ and $M2 = 77\%$. The growth process was considered to be completed once each cell had emitted all of its axons and no axons were still growing. The rule set and an animation of the neural growth process depicted in Figure 4.4 may be viewed on the World Wide Web [107].

To further assess the extent to which the grown V1 network corresponds to the original von der Malsburg model [90], the ability of the grown network to replicate the main map formation results obtained with the original network were evaluated. More specifically, once fully assembled, the neural network shown in Figure 4.4 was trained with the same nine orientations of line-like input patterns, activation dynamics, Hebbian learning rule, and parameter values as in von der Malsburg's original study described in [90]. Figure 4.5 illustrates each excitatory neuron's preferred orientation for input stimuli (if it has a preference) before and after training, respectively. A neuron's preferred orientation was defined to be the orientation of input stimuli to which the neuron responded most strongly with an activity level of greater than 1.44 standard deviations from its mean response over all nine input orientations. If a neuron's activity level did not satisfy this condition for any of the input orientations, then that neuron was considered to have no preferred orientation. The untrained network exhibits poor topographic organization of the input stimuli. Many of the neurons are not tuned to a particular input orientation and those that

are frequently have neighbors with input orientation preferences very different from their own. Furthermore, a disproportionately large number of neurons respond most strongly to bars with orientations of 90, 70 and 350 degrees, and relatively few respond to orientations of 30, 330 and 310 degrees. In contrast, the trained network is seen to have self-organized into a high quality topographic feature map. Almost all of the neurons have become tuned to a particular input orientation and transitions between regions of the map containing neurons with different orientation tunings are much smoother. Moreover, for each input orientation the number of neurons tuned to it is roughly the same. These results are qualitatively similar to those exhibited by the original network [90].

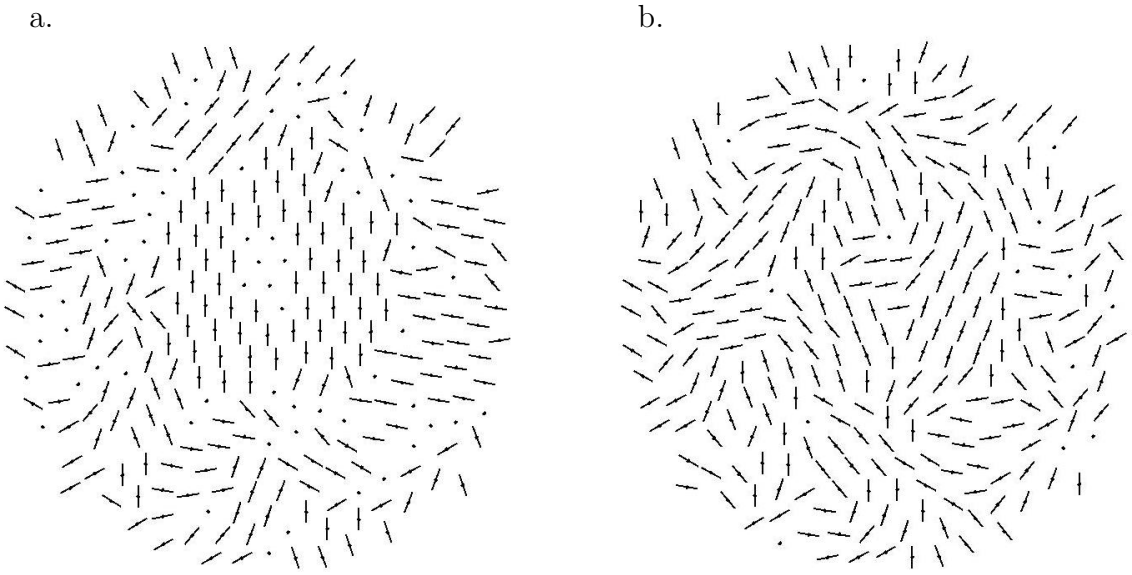


Figure 4.5: Each dot represents the center of a neuron in the excitatory layer. The line passing through a dot represents the orientation of the input stimulus that the corresponding neuron responds to most strongly and is absent (only a dot appears) if a neuron does not exhibit any responses that are greater than 1.44 standard deviations from its mean response over all nine input orientations. *a.* The orientation tuning of the excitatory neurons prior to training. *b.* After training. The latter is a well formed topographic feature map similar to that observed in von der Malsburg's original model.

4.2.3 Robustness Experiments

Three experiments were performed to determine what, if any, improvements in robustness are brought to the SINSA model by incorporating swarm intelligence in the form of locally coordinated collective movements. In this context, robustness is a measure of the degree to which the rule set can be varied without substantially reducing the quality of the networks being grown. This is an important characteristic for any approach to network self-assembly/growth/development to have because developmental models with degrees of complexity on par with the SINSA model often exhibit a considerable degree of sensitivity to their parameters, making it difficult to grow predefined complex networks [33, 73]. The robustness of two different versions of the SINSA methodology were compared. In the *rules-only* version growth was dictated by only the rule set and did not incorporate any of the forces that generate collective movements of the growth cones (Eqs. 3.4-3.7). Such an approach is comparable to producing growth with an L-system. In contrast, in the *rules-and-swarm* version networks grew by incorporating both the rule set and the collective movements of "swarms" of growth cones mediated through the intercone force (Eqs. 3.4-3.6), and the local growth force (Eq. 3.7). The intercellular force (Eq. 3.3) was not used in either version.

An example network grown in these experiments is shown in Figure 4.6. It was chosen because it captures many of the characteristics that make growing neural networks with a developmental model difficult. Specifically, it has a small region of target cells, coupled with axonal growth over a substantial distance, and inter-

mediary cells acting as obstacles. Ideal networks were defined as those adhering to a connectivity template in which all connections are made to target cells and each target cell receives at least one connection. That is, for the highest quality networks the two similarity measures defined in Section 4.1.3 are both equal to 100%.

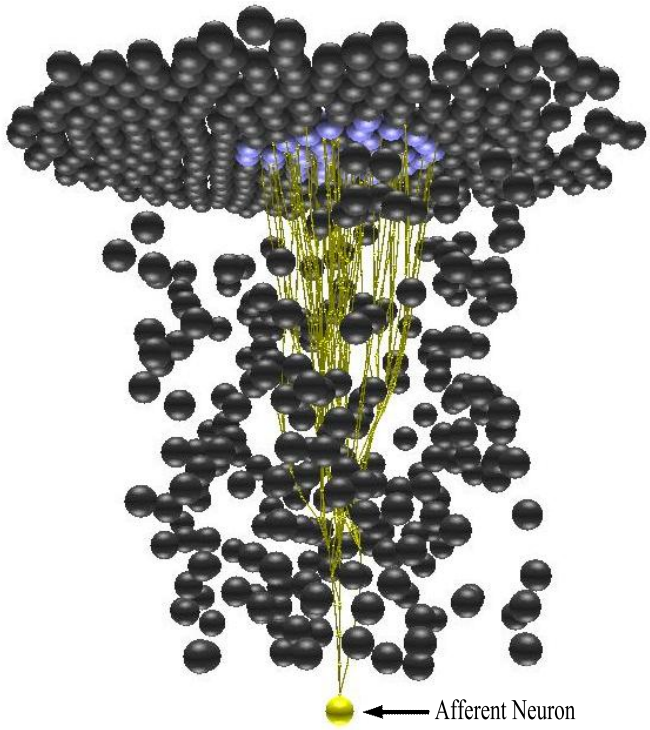


Figure 4.6: An example of the networks being grown by the simulator during the robustness experiments described in Section 4.2.3. A single afferent neuron emits axons that must then grow through a dense region of obstacle cells to reach their target neurons shown in blue.

An initial rule set was developed by hand (henceforth referred to as the base rule set), such that the *rules-only* model would grow a good quality network. This base rule set was also separately combined with a set of parameter values governing the force-based interactions of the agents such that the *rules-and-swarm* model would grow a good quality network. For the intercone force $k_s = 10$ in each experiment below, $R_{gg} = 1.5$ in the first experiment, and $R_{gg} = 2.5$ in the second and third

experiments. In the first experiment $k_a = 14$ and $k_c = 14$; in the second experiment $k_a = 10$ and k_c was a variable; in the third experiment both k_a and k_c were variables. In each experiment the local growth force was attractive with $k = 30$, $R_{cg} = 6$ and β set so that the strength of the force at its boundary ($r_{ji} = R_{cg}$) was 10 percent of its maximum value.

The experiments consisted of varying the base rule set in a random manner and then comparing the changes in the quality of the networks grown by the two versions of the SINSA methodology. It was hypothesized that collective movements improve robustness and that one of the primary ways that they do so is by making network growth less sensitive to the rule set. This hypothesis would be supported if the quality of the networks grown by the *rules-only* version deteriorate significantly more than those generated by the *rules-and-swarm* version as the random perturbations to the base rule set increase in size.

The base rule set was elected to be perturbed by making random changes to two variables named *Local Times* and *force*. In the SINSA approach, axonal growth is partly determined by the rule set, and there are two types of rules that are very frequently used and which have a particularly strong influence over the growth process. These are rules that contain either a SetRuleForce command or a DivCone command (see Table 3.3). More specifically, it is the *Local Times* variable, which is present in the predicate of rules that contain a DivCone command, and the *force* variable, which is an argument to a SetRuleForce command, that are largely responsible for the widely varying effects that these types of rules often have. It is the values of these variables that are among the most challenging to derive when

writing a rule set to grow a particular network.

All of the rule sets used in the following experiments consisted of 47 rules, which are listed in Appendix D. Each of the 20 rules with a SetRuleForce command that were varied in the base rule set had a *force* variable with the same magnitude, but the directions were different. The directions were represented in spherical coordinates $[\theta, \phi]$, where $\theta \in [0°, 360°)$ and $\phi \in [0°, 180°]$. For a particular SetRuleForce command in the base rule set, let θ_0 be the specified azimuth and ϕ_0 the polar angle. Likewise, for one of the 24 rules in the base rule set that contains a DivCone command, let T_0 be the specified value of $T \in R^+$, which denotes the *Local Times* variable. In the experiments, the rules in the base rule set were varied by drawing θ, ϕ and T from the uniform probability distributions $\theta \in U[\theta_0 - \Delta\theta_d, \theta_0 + \Delta\theta_d]$, $\phi \in U[\phi_0, \phi_0 + \Delta\phi_d]$ and $T \in U[(1 - p_d)T_0, (1 + p_d)T_0]$. Here, $\Delta\theta_d = 1.7d$, $\Delta\phi_d = d$ and $p_d = 0.017d$, where $d \in \{1, 2, 3, ..., 15\}$ is the *degree of variability* of the rule sets. Larger values of d correspond to greater perturbations, and thus to a user having more freedom in choosing the rule set.

A comparison was made between the effects of increasing the variability of the rule sets on the networks grown by the version of the SINSA model without collective movements (*rules-only*) and the version with collective movements (*rules-and-swarm*) using the similarity measures (Sect. 4.1). For each value of d, and for both versions of the model, 101 trials were performed. Each trial consisted of growing a network using a rule set that was derived from the base rule set by subjecting the θ, ϕ and T components of its constituent rules to random perturbations according to the aforementioned probability distributions. In all three experiments each of

the parameters of the forces responsible for inducing collective movements (k_s, k_c, k_a, R_{gg}, k, and R_{cg} in Eqs. 3.4-3.7), were randomly varied by up to ± 20 percent at the beginning of each trial, as were the positions of all cells, except the afferent cell. For each value of d these trials were used to determine the average values of the similarity measures $M1$ and $M2$. The base rule set and initial parameter values were chosen so that the corresponding similarity measures of the networks grown by the two versions of the SINSA model were as close in value as could be feasibly attained when there was no rule set variability ($d = 0$). In these experiments the interest is in the changes in the similarity measures *relative* to their values when $d = 0$.

In the first experiment the parameters of the intercone force were held constant ($R_{gg} = 1.5$, $k_s = 10$, $k_a = 14$ and $k_c = 14$), as d was varied. The results of this experiment are shown in Figure 4.7. They illustrate the ability of the *rules-and-swarm* model to persistently grow good quality networks despite progressively larger values of rule set variability, and the comparatively rapid deterioration of network quality with increasing rule set variability for the *rules-only* model. This indicates that incorporating collective movements into the network growth process improves the robustness of the SINSA model. It was found through trial and error that the improvement occurs as long as the neighborhood radius R_{gg} is made small enough, and the coefficients k_a and k_c are made large enough relative to k_s. When this is the case $M1$ will decrease with increasing d at a significantly slower rate than it does for the *rules-only* version, and $M2$ will increase up to about $d = 8$. However, enforcing these constraints on the parameters in the intercone force tends to cause

$M2$ to have a lower value when d is small (seen on the left in Figure 4.7b). It was hypothesized that both $M1$ and $M2$ could be made to take on higher values when d is small while retaining the greater robustness of the *rules-and-swarm* version of the model by allowing one or more of the coefficients in the intercone force to vary as functions of d.

Consequently, in the second experiment the parameter k_c in the cohesion force was varied according to the function $k_c(d) = 1.36d - 0.36$. That is, the cohesion among the growth cones was increased linearly with respect to increasing rule set variability. The parameters $k_a = 10$ and $R_{gg} = 2.5$ were held constant. This experiment was performed multiple times, each time using a different linear function for k_c. Specifically, $k_c(d) = md + (1 - m)$ and the experiment was repeated using different values for $m \in \mathbb{R}^+$. For each of the values $m \in \{1.36, 1.71, 1.86, 1.93\}$ the results were very similar. Figure 4.8 illustrates that if the amount of cohesion among the swarming growth cones is allowed to increase as a linear function of the degree of rule set variability d, then the *rules-and-swarm* version of our model is capable of growing networks with relatively high values for $M1$ and $M2$ when d is small and that the values of both similarity measures decrease at significantly slower rates with respect to increasing d than they do for the *rules-only* version.

In conducting the second experiment it was discovered that imposing the constraint $m \geq \frac{15}{7}$ in the equation $k_c(15) = m(15) + (1 - m)$ results in $M1$ remaining very close to its base value of 90.7%, but that $M2$ then decreases further from its base value of 77.2%. In response to this the following question was asked: can further improvements be gained by allowing more than one of the parameters in

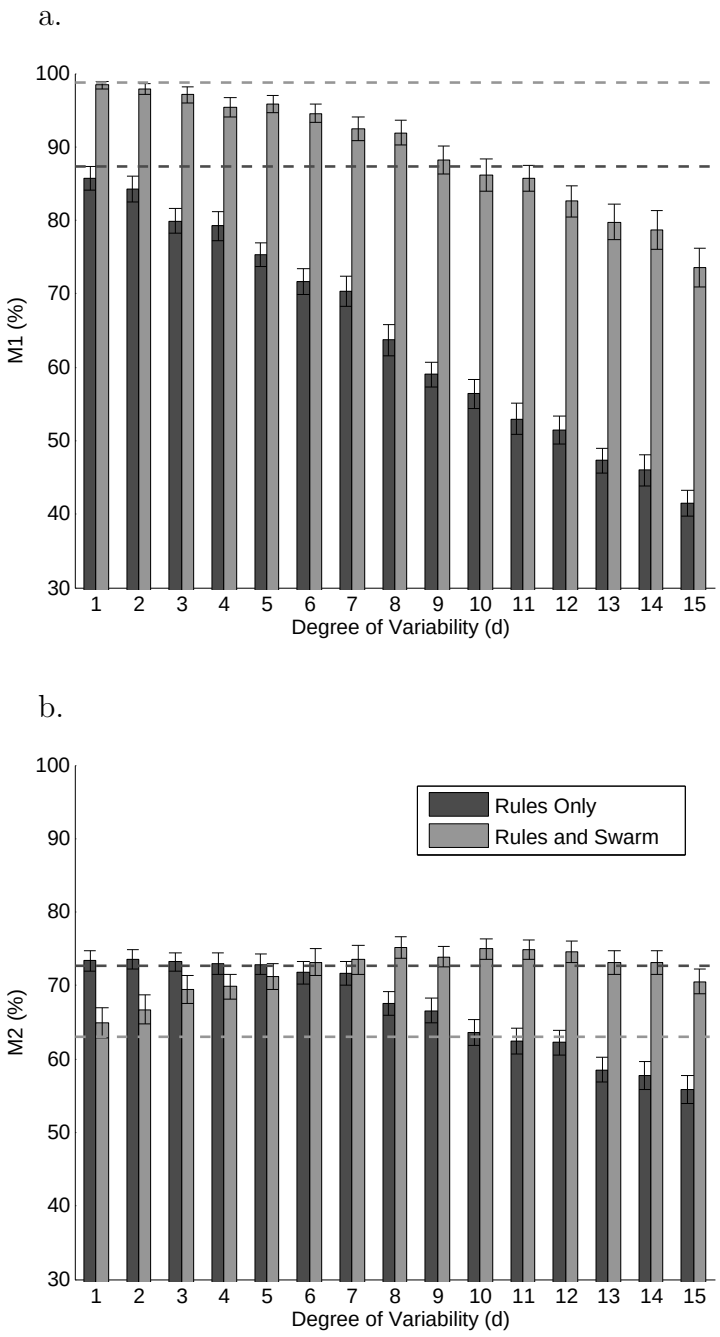


Figure 4.7: A comparison of the change in neural network quality measures *a.* $M1$ and *b.* $M2$ as rule set variability d increases, between a version of the SINSA model that incorporates only rule-based growth and a version that utilizes both rules and collective movements generated via local forces. The parameters of the intercone force were held constant ($R_{gg} = 1.5$, $k_s = 10$, $k_a = 14$ and $k_c = 14$). The dashed dark-gray lines indicate the average correctness of the networks grown by the *rules-only* model when the rule sets have zero variability ($d = 0$). The dashed light-gray lines indicate the same thing but for the *rules-and-swarm* model. The error bars are 95% confidence intervals.

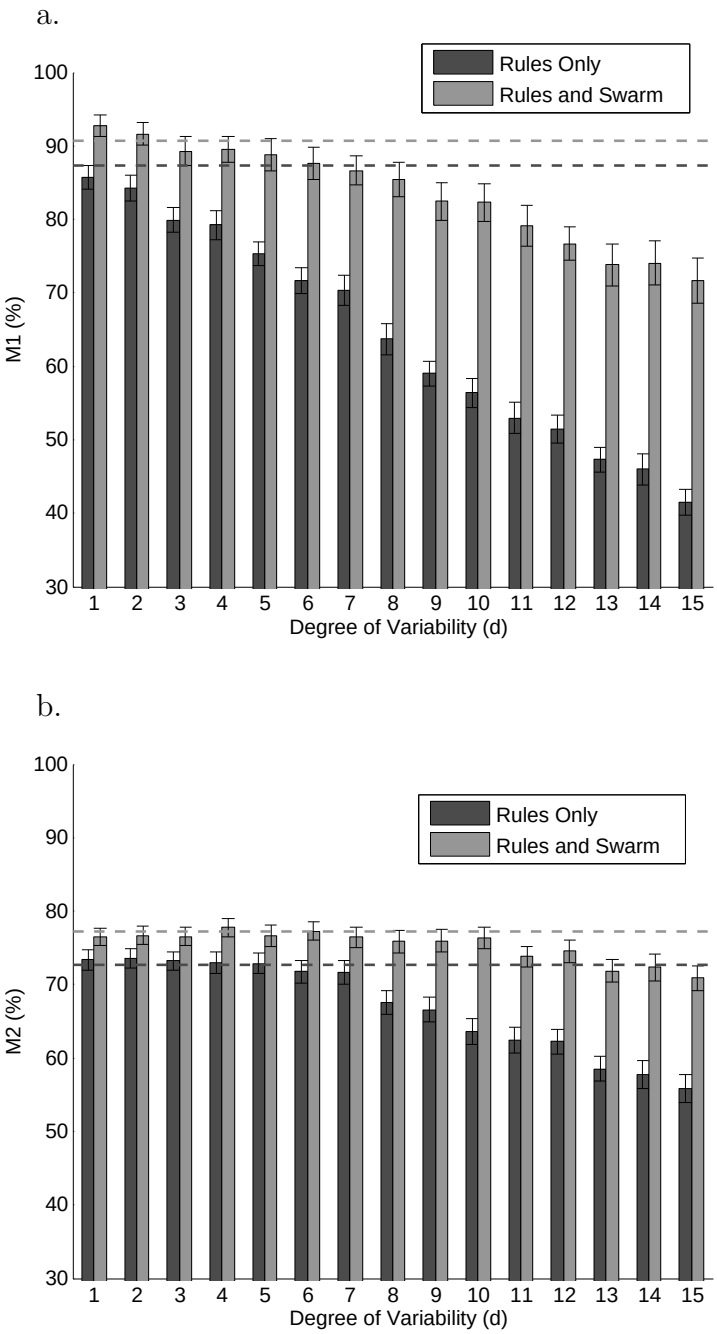


Figure 4.8: Comparison of the dependence of neural network quality on the variability of the rule sets, between a version of the SINSA model that incorporates only rule-based growth and a version that utilizes both rules and swarm intelligence generated via local forces. The degree of cohesion k_c was an increasing linear function of the degree of variability (d). Same notation as in Figure 4.7.

the intercone force to vary simultaneously? It was hypothesized that the decrease in similarity measure $M1$ could be minimized, while keeping the reduction in $M2$ small, by allowing both the cohesion factor k_c and the velocity alignment factor k_a from Equation 3.6 to vary together as functions of d. To test this hypothesis a third experiment was performed that was identical to the previous experiments except that $k_c(d) = d$ and $k_a(d) = 1.07d + 8.93$ were both variables. In this experiment $R_{gg} = 2.5$. The results of this third experiment are shown in Figure 4.9. For the *rules-and-swarm* version of the SINSA model similarity measure $M1$ remains very close to its base value, and $M2$ decreases by only about six percentage points. These results further indicate the usefulness of incorporating collective movements into the network growth process. This experiment was performed five more times using various linear functions for k_c and k_a. Specifically, $k_c(d) = m_c d + (1 - m_c)$ and $k_a(d) = m_a d + (10 - m_a)$, where $m_a, m_c \in \mathbb{R}^+$. For each ordered pair $(m_c, m_a) \in \{(0.64, 1.07), (0.64, 1.43), (1.0, 0.71), (1.0, 1.07), (1.0, 1.43)\}$ the results of the experiment were similar.

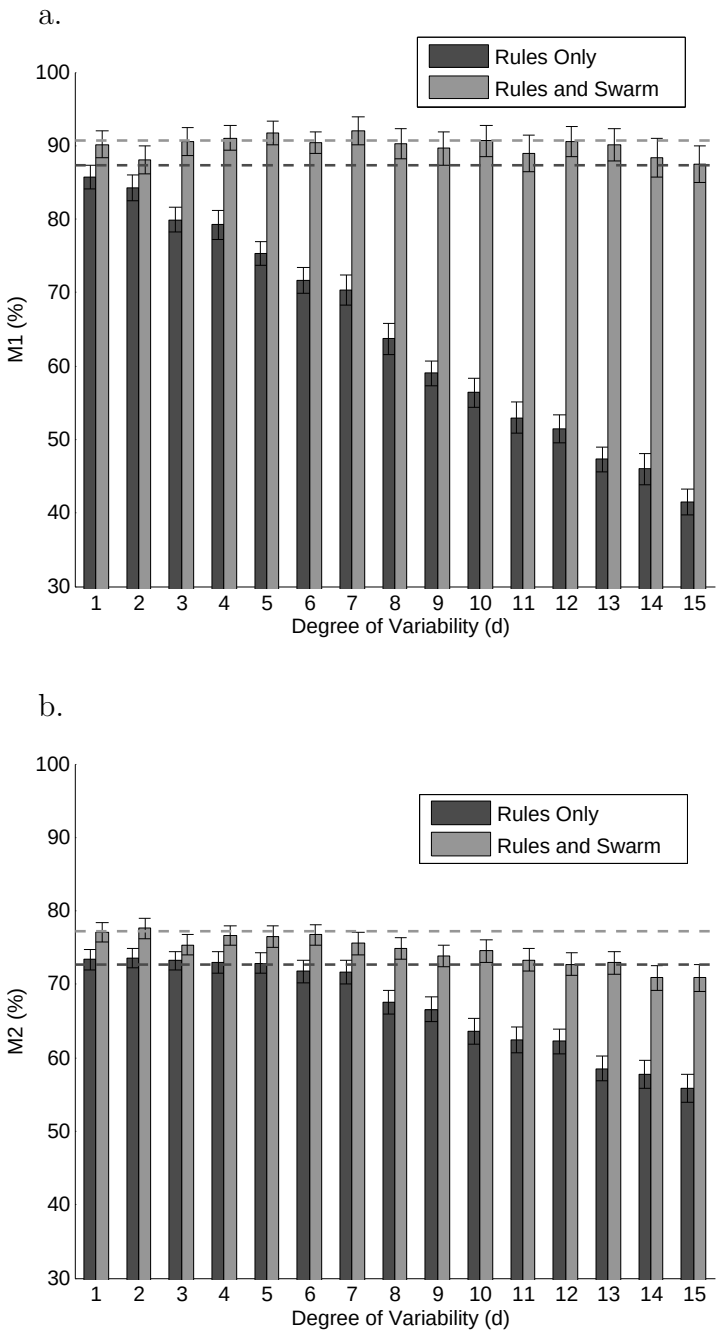


Figure 4.9: Comparison of the dependence of neural network quality on the variability of the rule sets, between a version of the SINSA model that incorporates only rule-based growth and a version that utilizes both rules and swarm intelligence generated via local forces. The degree of cohesion k_c and velocity alignment k_a were now both increasing linear functions of the degree of variability (d). Same notation as that used in Figure 4.7.

Chapter 5

Swarm Intelligence and the Self-Assembly of Optimal Neural Networks

This chapter introduces a means of adapting the SINSA model so that it is capable of growing neural networks that perform well on specified computational tasks. The traditional self-assembly problem is extended to include the generation of structures based on optimality criteria, rather than on target structures that are specified *a priori*. First, the motivation behind this adaptation is discussed, followed by an explanation of how techniques inspired by particle swarm optimization can be harnessed to turn the network self-assembly process into a search for an optimal topology and set of weights. The chapter concludes with a discussion of the implementation details surrounding the adapted SINSA model.

5.1 Motivation

Both the weights and topology affect a neural network's performance. To date, substantially more focus has been placed on techniques for optimizing a neural network's weights as opposed to its topology (the number and connectedness of its nodes). One of the primary reasons for this discrepancy is that the space searched by an optimization method for a good set of weights (the "weight space") is continuous. Thus a good set of weights can be found using one of a wide variety of powerful

and well-studied optimization techniques based on local gradient information. Additionally global optimization techniques such as evolutionary computation (EC) and particle swarm optimization (PSO) have proven to be very effective at optimization in continuous search domains. The "topology space" on the other hand is a discrete space. While EC has been applied fairly extensively to network topology optimization, PSO has found only limited application in this problem domain, and in regards to neural networks such applications have been restricted to feedforward networks. This is unfortunate because PSO has been immensely successful at tackling many different types of problems, however, the bulk of these problems involve searching for solutions in a continuous space.

The challenge faced in extending PSO to effectively search discrete topology spaces is evidenced by the creation of complicated representations of the topology space [77, 84], and fundamental modifications to the way the canonical PSO algorithm works [126]. In the case of neural networks specifically, the problem is made all the more daunting by the fact that if both the weights and topology are to be optimized, they cannot be considered in isolation. The result is a substantial increase in the number of variables that need to be searched and algorithm parameters that need to be fine tuned, all of which are interdependent. These complications can be reduced somewhat, but doing so comes at the cost of severely limiting the domain of feasible network topologies [15, 137], which is a restriction that may be very undesirable. This leads one to wonder: Is there an intuitive means of representing the weight space and the topology space so that a more elegant formulation of the PSO algorithm can be used to simultaneously optimize both the weights and the

topology of a neural network? The term "elegant" is intended to imply that there should be only one continuous space that the particles move and interact in, and that the simplest (canonical) form of the PSO algorithm can be used effectively (Eq. 2.1). It is demonstrated below how network self-assembly can provide a novel and useful answer to this question.

5.2 The Issue of Network Representation

Particle swarm optimization was originally devised for use in continuous domains, specifically multidimensional Euclidean space $\mathbb{R}^n$. It operates under the basic heuristic (referred to here as the PSO-heuristic) that, given two points in a search space, each of which represents a good solution, it is likely that a better solution exists somewhere between or around these points. Following this rule-of-thumb, a particle in the PSO algorithm tends to move in such a way that its position over time obeys a unimodal distribution centered somewhere between the particle's personal best position and the best position among all of its neighbor particles. The location of the distribution's mean depends on the values of certain algorithm parameters. This heuristic has been found to be generally useful in optimizing objective functions defined on $\mathbb{R}^n$ for a couple of reasons. First, a large fraction of the objective functions of practical interest are continuous. This implies that given a solution (point) and any positive number, the magnitude of the difference in the fitness values between this solution and any other solution will be bounded above by the chosen positive number so long as the solutions are similar enough (the points are close

enough together). This means that the PSO algorithm will be capable of effectively searching in localized regions of the space, and given enough time the swarm of particles will likely converge to a good solution. Second, the similarity/dissimilarity of two solutions is well represented by the Euclidean distance between them. This means that when a particle searches the region of space between two different good solutions, it is guaranteed to be exploring new solutions that are combinations of the two known good solutions, which is generally an effective technique for finding even better solutions.

Because the canonical form of PSO was developed for use in continuous spaces, its application to the optimization of neural networks has been limited almost exclusively to the training of network weights. It is tempting to try and apply this elegant form of PSO to optimization in a network's discrete topology space. In the most straight-forward scenario a network's weights and topology could be optimized by creating two separate swarms of particles. One swarm would exist in the network's continuous weight space, and the other would exist in the network's discrete topology space. A solution to the optimization problem would consist of the set of weights and the topology represented by a weight-particle/topology-particle pair. In this representation the weight space is a subset of $\mathbb{R}^n$, where $n \in \mathbb{N}$ is the number of trainable weights in the network, and the topology space is an m-dimensional binary space, where m is the total number of non-fixed connections. For example, if a network is to consist of N neurons, every neuron is allowed to make a directed connection to any neuron in the network, all of the network's weights are trainable, and no fixed (permanent) connections are specified, then $n = m$ and $m = N^2$.

The topology space is constructed by first assigning an arbitrary mapping from the set of non-fixed connections to the natural numbers $0, 1, 2, ...m$. A particular topology is then represented as a binary vector of length m (i.e., a value of 0 means that the connection is not present, and a value of 1 means that it is). Unfortunately, the topology space induced by this representation is very difficult to search from an optimization perspective. First, it is common for many of the nodes in a neural network to be identical from a computational perspective, such as the nodes in the hidden layer. This means that many pairs of points in the topology space that are far apart will represent identical or very similar topologies because the indistinguishability of nodes is not accounted for by the arbitrary mapping from the set of non-fixed connections to the natural numbers. Second, the optimality of a neural network is usually entirely or largely defined in terms of its ability to perform well on a specified computational task. Depending on a network's topology, the learning algorithm used, and the computational problem a network is tasked with solving, certain connections may influence performance much more than others. For example, if a neural network is used as a controller, then a direct connection from an input neuron to an output neuron would likely have a far greater impact on network performance than any particular connection within the hidden layer. This means that in the topology space under consideration here, a network that has an input-to-output connection may have a nearly identical binary vector representation to one that does not have such a connection, despite the fact that these two networks would likely have vastly different fitness values. Third, the quality of a particular topology is dependent on the set of weights associated with it, and vice versa. This interde-

pendence means that, rather than having a fixed fitness value, a point (topology) in the topology space has a distribution of fitness values generated by associating different sets of weights with its connections. This fact, along with the stochastic nature of the search process in the weight space, increases the "roughness" of the fitness landscape defined over the topology space. The first of the above characteristics implies that the distance between two points in the topology space often does not accurately reflect the similarity/dissimilarity of the topologies represented by the points. Viewed another way, the topology space lacks a "smooth" transition from one solution (topology) to another as the distance between them is traversed. The second and third characteristics, coupled with the discrete nature of the topology space, imply that nearby points often represent topologies with very different fitness values, making for a very rough fitness landscape. The implications of these characteristics makes it clear that the canonical version of the PSO algorithm would have a difficult time performing optimization in this topology space because the properties that make the PSO-heuristic useful are largely absent from this space.

The work presented in the following portion of this dissertation arose from the hypothesis that *canonical PSO can be made useful for the simultaneous optimization of weights and topology by finding a much more integrated means of representing these network features.* This integration is largely the result of two ideas. The first idea is about viewing the particles in PSO as being part of a larger structure. Almost all implementations of PSO consider the *particle* to be the fundamental type of object capable of movement and interaction during the optimization process. In the research presented here, the growing *network* plays the role of the fundamental

type of object involved in the optimization process. That is, instead of a population of moving particles, there is a population of growing networks. The transition from particles to networks is achieved by having *growth cones* play the role that *particles* do in traditional PSO. The growth cones' movements are dictated by the canonical PSO equation (Eq. 2.1), and because growth cones occur at the leading tips of growing axons, their movements generate network growth. Unlike in traditional PSO, the position of a growth cone (particle), however it is interpreted, is only meaningful when the axon/neuron that it is a part of is taken into account.

The second idea is concerned with the nature of the relationship between the concepts of connection weight and topology in regards to neural networks. To motivate this idea consider the two weighted, directed graphs shown in Figure 5.1. Strictly speaking these graphs have different topologies because the connection from the lower left node to the upper left node is present in the graph shown in Figure 5.1*a*, but not in the graph shown in Figure 5.1*b*. However, if these graphs represent neural networks, then from a computational perspective it can be argued that they have roughly the same "effective" topology. This is because the connection from the lower left node to the upper left node has a weight that is relatively small in magnitude, which means that signals transmitted via this connection will be highly attenuated, and thus tend to have very little influence on network dynamics. It's as if the connection isn't actually there. This is illustrated in Figure 5.1*c*, which shows that the dynamics of the two neural networks are nearly identical. The first of the aforementioned ideas suggests producing network self-assembly by having the movements of network components obey the dynamical equations of the PSO

algorithm. The second idea leads to a way of turning this self-assembly process into an optimization process.

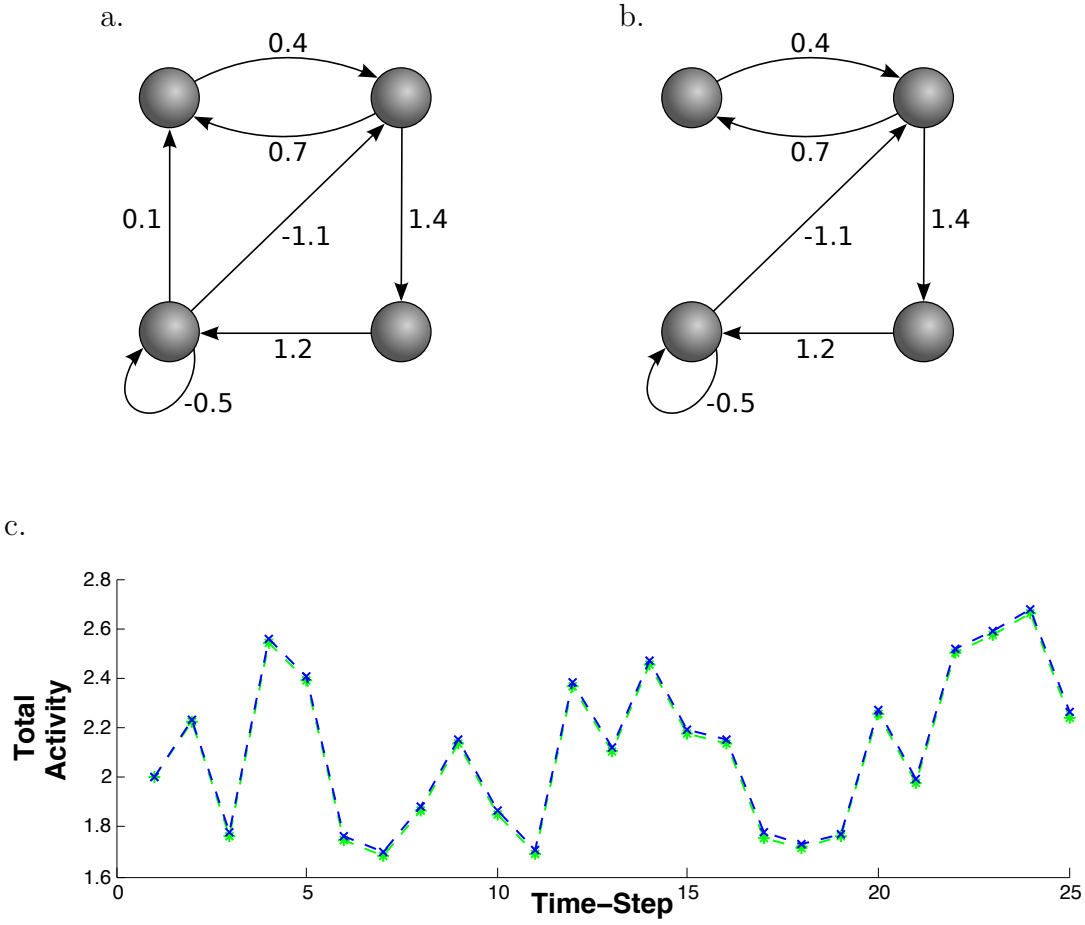


Figure 5.1: Two neural networks and the dynamics of their activity. *a*, *b*. The networks have different topologies because the connection from the lower left node to the upper left node is present in one network but not the other. However, the weight on this connection is small in magnitude so it has little impact on the activity of the network. Thus, from the perspective of activity dynamics these two networks effectively have the same topology. *c*. The total activity of network *a* (shown in blue) and network *b* (shown in green) is plotted for a duration of twenty-five time-steps. This plot illustrates that the activity of the two networks is nearly identical when they are started in the same initial state and given the same sequence of inputs. Each neuron has a logistic transfer function, and receives randomly generated input from the uniform probability distribution over the interval $[-0.5, 0.5]$. The total activity is the sum of the activation of each neuron in a given network on particular time-step.

5.3 Integrating Self-Assembly and Particle Swarm Optimization

This section presents the details of a new model of **S**warm **I**ntelligent **N**etwork **O**ptimization through **S**elf-**A**ssembly (SINOSA). In this model groups of growth cones that belong to different networks simultaneously grow through the same three-dimensional space. During the growth process the growth cones from different networks interact with one another through a mechanism inspired by particle swarm optimization. Concurrently, the networks receive input derived from a computational problem that they must learn to solve. The combination of this interaction, and the activity run through the networks during the development process, leads to the self-assembly of neural networks with weights and topologies that are optimized for solving the problem at hand.

A detailed mathematical description of the SINOSA model is given below. Throughout this description the concrete example of the model illustrated in Figure 5.2 is referenced for clarification. The SINOSA model consists of a set of cells $\mathbf{C}$ with fixed positions assigned *a priori*. The cells represent neuron cell bodies. Each cell $c_i \in \mathbf{C}$ has a set $\mathbf{N}_{c_i} \subseteq \mathbf{C}$, which may be empty, of "neighbor" cells that it can connect to, where $i = 1, 2, ..., |\mathbf{C}|$. In Figure 5.2 the three large grey spheres represent cells, and each cell is allowed to connect to any other cell, including itself. Thus, $\mathbf{N}_{c_i} = \mathbf{C}$, for $i = 1, 2, 3$.

Each growing network consists of the same set of cells $\mathbf{C}$, and a unique set of growth cones that guide the network's axons through the three-dimensional space. Given n simultaneously growing networks, each cell c_i has n sets of growth cones $\mathbf{G}_{ij}$,

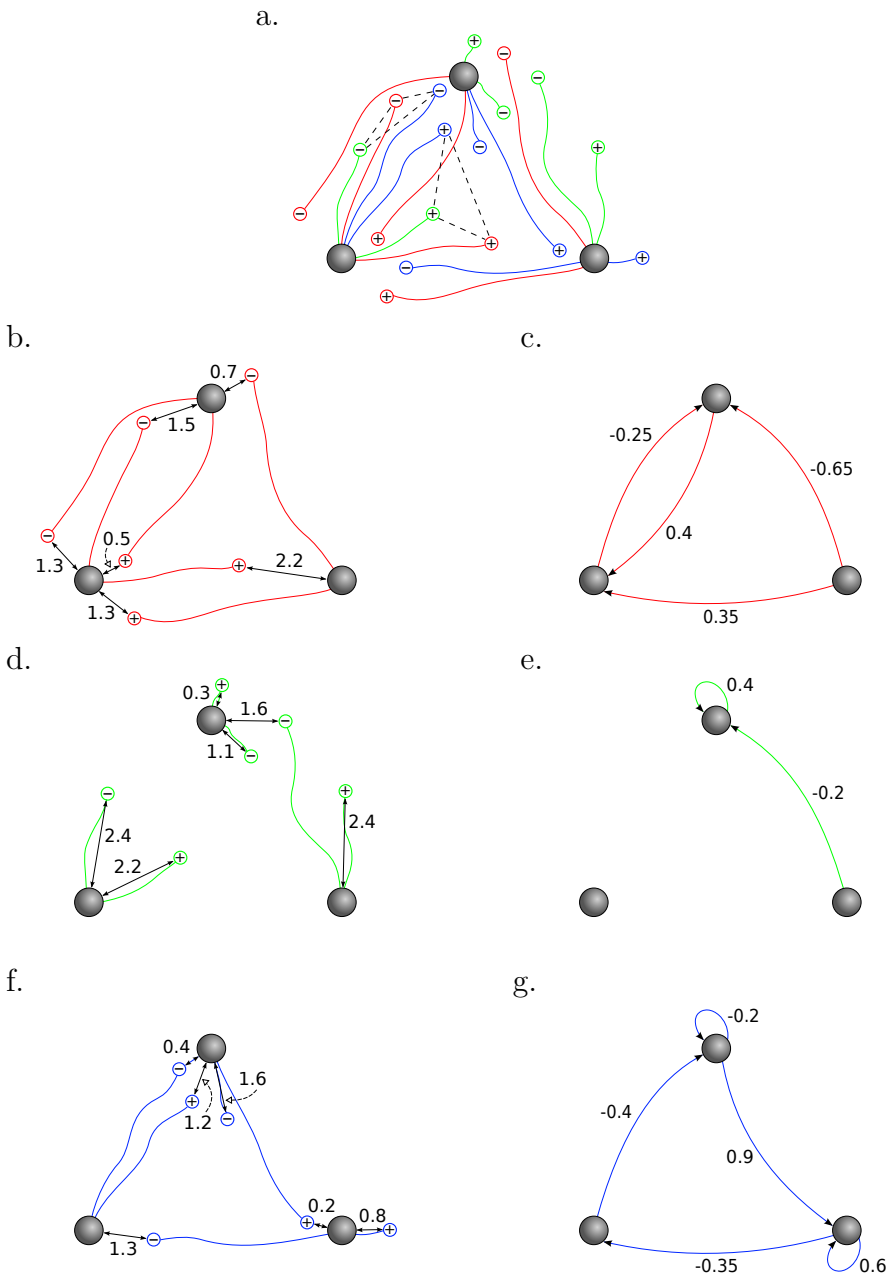


Figure 5.2: Three growing neural networks and their interpretations as static neural networks based on the SINOSA model. The three large spheres represent cells, the smaller colored circles represent growth cones, and the solid lines between cells and growth cones denote axons. The growth cones that are drawn with a "+" symbol have a positive weight field, and those that are drawn with a "−" symbol have a negative weight field. *a.* The growth cones (and axons) that belong to a particular growing network are all shown in the same color. The dashed lines indicate two of the six growth cone neighborhoods. Growth cones within a neighborhood interact with one another according to the canonical PSO algorithm. All three growing networks (red, green, and blue) share the same three cells. *b*, *d*, *f*. Each growing network is shown without the other two. *c*, *e*, *g*. The corresponding static networks to which the growing networks are mapped based on the proximity of the growth cones to their target cells.

where $j = 1, 2, ..., n$. Any given cell c_i contributes the same number of growth cones to each growing network. That is, for all j and ℓ, $|\mathbf{G}_{ij}| = |\mathbf{G}_{i\ell}|$. This restriction is implemented to ensure that all of the growth-cone-neighborhoods (explained below) among the growth cones in $\mathbf{G}_i = \bigcup_j \mathbf{G}_{ij}$ are of the same size. If $\mathbf{N}_{c_i}$ is empty, then so is $\mathbf{G}_{ij}$, for all j. The j^{th} growing network $gnet_j$ consists of the set of cells $\mathbf{C}$, and the set of growth cones $\mathbf{G}_j = \bigcup_i \mathbf{G}_{ij}$ that produce the network's growth. That is, $gnet_j$ is defined by the ordered pair $\langle \mathbf{C}, \mathbf{G}_j \rangle$. Because each growing network consists of the same set of cells $\mathbf{C}$, they all have exactly the same number of growth cones ($|\mathbf{G}_j| = |\mathbf{G}_\ell|$, where $j, \ell = 1, 2, ..., n$). In Figure 5.2 the small colored circles represent growth cones, and the solid, colored lines that connect the cells and growth cones are axons. In this case there are $n = 3$ growing networks. The growing axons of any particular network are shown in a unique color (red, green, or blue). For example, Figure 5.2b shows only the red growing network. Figure 5.2a illustrates how all three networks simultaneously grow through the same space, and share the same three cells. It can be seen that each cell c_i contributes two growth cones to each network (i.e., $|\mathbf{G}_{ij}| = |\mathbf{G}_{i\ell}| = 2$ for $j, \ell = 1, 2, 3$), for a total of six growth cones per network.

The k^{th} growth cone $g_{ijk} \in \mathbf{G}_{ij}$, which belongs to the j^{th} set of growth cones from cell c_i, has a set of target cells $\mathbf{T}_{g_{ik}} \subseteq \mathbf{N}_{c_i}$ to which it may establish connections. The subscript j is not included in the notation for the set $\mathbf{T}_{g_{ik}}$, because for each cell c_i, the k^{th} growth cone in each set $\mathbf{G}_{ij}$ has exactly the same set of target cells. If a cell c_i has growth cones that strictly represent positively-weighted connections, and growth cones that strictly represent negatively-weighted connections, then for each

growing network $gnet_j$ the set of growth cones the cell contributes to the network can be expressed as $\mathbf{G}_{ij} = \mathbf{G}_{ij}^{+} \cup \mathbf{G}_{ij}^{-}$, where $\mathbf{G}_{ij}^{+}$ consists of the *positive growth cones* and $\mathbf{G}_{ij}^{-}$ consists of the *negative growth cones*. The cell c_i contributes the same number of positive and negative growth cones to each network. Specifically, $|\mathbf{G}_{ij}^{+}| = |\mathbf{G}_{i\ell}^{-}|$ for all j and ℓ. For any two networks $gnet_j$ and $gnet_\ell$, the number of positive growth cones contributed by cell c_i to $gnet_j$ is equal to the number of negative growth cones contributed by c_i to $gnet_\ell$. Specifically, $|\mathbf{G}_{ij}^{+}| = |\mathbf{G}_{i\ell}^{-}|$ for all j and ℓ. Furthermore, for any j and k, the growth cones $g_{ijk}^{+} \in \mathbf{G}_{ij}^{+}$ and $g_{ijk}^{-} \in \mathbf{G}_{ij}^{-}$ have the same set of target neurons. It can be seen that each of the aforementioned conditions is satisfied by the growing networks shown in Figure 5.2. The positive growth cones are drawn with a "+" symbol, and the negative growth cones are drawn with a "−" symbol. Each cell c_i contributes one positive growth cone and one negative growth cone to each network, so $|\mathbf{G}_{ij}^{+}| = |\mathbf{G}_{i\ell}^{-}| = 1$ for j,ℓ=1,2,3. It is assumed that each growth cone is allowed to connect to any of the three cells, so $\mathbf{T}_{g_{ik}} = \mathbf{N}_{c_i} = \mathbf{C}$, for $i = 1, 2, 3$ and $k = 1, 2$. It follows that g_{ij1}^{+} and g_{ij1}^{-} have the same set of target neurons, namely $\mathbf{C}$, for $i = 1, 2, 3$, and $j = 1, 2, 3$.

The SINOSA model grows neural networks that, in their completed form, have fixed connections. Thus, it is necessary to interpret the positions of a growing network's growth cones relative to their target cells so as to map a *growing network* $gnet_j$, to a *static network* $snet_j$. In particular, if multiple growth cones from cell c_i and growing network $gnet_j$ are able to establish a connection to cell c_ℓ, then the weight on the connection from c_i to c_ℓ in the static network $snet_j$ is the sum of the individual weights contributed by the growth cones involved in creating the connection.

Two different interpretations have been implemented in the SINOSA model to construct the mapping from growing network to static network. Both of them reflect the need to create a neural network representation that more closely integrates the concepts of topology and connection weight so that the canonical PSO algorithm can be used to optimize these network characteristics effectively. In the first interpretation, each growth cone has a fixed weight value that is small in magnitude. A growth cone establishes a connection with a target cell if the distance between the center of the growth cone and the closest point on the surface of the cell is less than a specified distance (the growth cone's *connection radius*), otherwise no connection is created. The weight on the connection is the fixed value carried by the growth cone.

In the second interpretation, each growth cone is considered to be at the center of its own spherically symmetric "weight field" that is finite in extent, and its corresponding weight has a magnitude that decreases to zero as the distance from the growth cone increases. A growth cone establishes a connection with a target cell if the cell is within the boundary of its weight field, otherwise no connection is created. The weight on the connection is the value of the field at the target cell's center. Formally, a *weight field* is a function from $\mathbb{R}$ to $\mathbb{R}$ with the form

$$w(r) = \begin{cases} ar^{\alpha} + b, & \text{if } r < r_0 \\ \\ 0, & \text{if } r \geq r_0, \end{cases} \tag{5.1}$$

where $a, b \in \mathbb{R}$, $r \geq 0$ is the distance of the target cell from the center of the growth

cone, $r_0 > 0$ is the extent of the weight field, and $\alpha > 0$. In the SINOSA model it is assumed that $w(r) \longrightarrow 0$ as $r \longrightarrow r_0$. Thus $w(r_0) = ar_0^\alpha + b = 0$, which implies that $a = \frac{-b}{r_0^\alpha} = \frac{-w(0)}{r_0^\alpha}$. Figures 5.2b-g illustrate how the three interacting, growing networks, shown together in Figure 5.2a, are mapped to static networks based on the weight field interpretation of the growth cones' positions relative to their target cells. The growth cones that are drawn with a "+" symbol have a positive weight field represented by the function

$$w_+(r) = \begin{cases} -\frac{1}{2}r + 1, & \text{if } r < 2 \\ 0, & \text{if } r \geq 2, \end{cases} \tag{5.2}$$

where r is the distance between a growth cone and one of its target cells. The growth cones that are drawn with a "−" symbol have a negative weight field expressed by the function

$$w_-(r) = \begin{cases} \frac{1}{2}r - 1, & \text{if } r < 2 \\ 0, & \text{if } r \geq 2. \end{cases} \tag{5.3}$$

Figure 5.2b shows the red network's growing axons, along with the distance between each growth cone and its nearest target cell. Figure 5.2c shows the static network derived from the red growing network. The numbers are the connection weights. This mapping occurs as follows. The cell shown in the lower lefthand corner of Figures 5.2b and 5.2c establishes a connection with weight $w_-(1.5) = -0.25$ to the upper cell, but does not establish a connection with the cell in the lower righthand

corner because $w_+(2.2) = 0$. The upper cell makes a connection to the lower left-hand cell with weight $w_+(0.5) + w_-(1.3) = 0.4$. The lower righthand cell makes a connection to the lower lefthand cell with weight $w_+(1.3) = 0.35$, and it makes a connection to the upper cell with weight $w_-(0.7) = -0.65$. Figures 5.2d and 5.2e show the growing-to-static mapping for the green network, with the connections and their weights being derived according to the calculations $w_+(2.2) + w_-(2.4) = 0$, $w_+(0.3) + w_-(1.1) = 0.4$, $w_+(2.4) = 0$, and $w_-(1.6) = -0.2$. The derivation of the connections and weights for the blue network is illustrated in Figures 5.2f and 5.2g, and is based on the calculations $w_+(1.2) + w_-(0.4) = -0.4$, $w_+(0.2) = 0.9$, $w_-(1.6) = -0.2$, $w_+(0.8) = 0.6$, and $w_-(1.3) = -0.35$.

Both of the aforementioned interpretations for determining the presence of a connection and its weight are formulated so that a small change in the position of a growth cone produces a small change in the weight on a connection, or if the change in position results in the addition or removal of a connection, then the added or removed connection has a weight that is small in magnitude. In other words, a small change in the physical configuration of a growing network will produce a small change in the weights and topology of the static network to which it is mapped. This means that given a short interval of time Δt during the growth process, and a growing network $gnet(t)$ that is mapped to the static network $snet(t)$ at time t, the static network $snet(t + \Delta t)$ will have a performance value that is very similar to that of $snet(t)$, even though they may have different topologies. So, in the SINOSA model network optimization occurs in a single, continuous space, and the integrated representation of network weights and topology results in a much smoother objective

function. This allows the use of the simple but powerful canonical form of PSO to drive the optimization process.

The growth cones from different growing networks interact with one another according to the canonical PSO algorithm. This means that during the self-assembly process each growth cone must be assigned a fitness value that indicates the usefulness of the best solution component (connection) the growth cone has found, and it must remember its personal best position, which represents the best connection found by the growth cone up to the current point in the growth process. Specifically, at each discrete time-step $t \in \mathbb{N}$ the performance of each static network $snet_j(t)$ on some set of training data is determined, where $j = 1, 2, .., n$. For each growing network $gnet_j(t)$, if the performance of $snet_j(t)$ is better than the performance of $snet_j(t - \tau)$ for all $\tau \in \mathbb{N}$ such that $0 < \tau \leq t$, then the fitness value of $gnet_j(t)$, or more specifically its growth cones $g_{ijk} \in \mathbf{G}_j$, is set to the performance value of $snet_j(t)$, and the personal best position of each growth cone g_{ijk} is set to its current position. Additionally, any growth cone g_{ijk} must have a set of neighbor growth cones $\mathbf{N}_{g_{ijk}}$ that influence its movements. As in most implementations of PSO, the research presented herein adheres to the condition that the neighbor relation is symmetric. That is, if $g_{ij\ell}$ is a neighbor of g_{ijk}, then g_{ijk} is a neighbor of $g_{ij\ell}$.

There is a wide variety of different ways that a growth cone's neighbors could be selected. However, certain characteristics of the self-assembly/optimization process limit the number of useful choices. It is an underlying assumption of the PSO algorithm, that the closer two neighbor particles get to one another, the more similar are the solutions or solution components, that their positions represent. It is

essential for the effectiveness of the PSO algorithm that if two growth cones g_{ijk} and $g_{i\ell k}$ are neighbors, and they occupy the same position, then they represent exactly the same weighted connection(s) in their respective static networks $snet_j$ and $snet_\ell$.

In the SINOSA model, if two growth cones occupy the same position, but are guiding axons from different cells, then they represent two completely different connections (solution components). Likewise, if two growth cones occupy the same position, but do not have exactly the same set of target cells, then they may represent different connections. These two scenarios, and the need for growth cones that are neighbors to avoid circumstances were they occupy the same position and yet represent different weighted connections, leads to three necessary growth-cone-neighborhood properties. First, if a pair of growth cones are neighbors, then they must be guiding axons from the same cell. Second, if a pair of growth cones are neighbors, then they must have exactly the same set of target cells. Third, if a pair of growth cones are neighbors, then their weight fields must be expressed by the same function, or they must carry the same fixed weight value and connection radius. The following is a simple and effective way of choosing a growth cone's neighbors such that these properties are satisfied. For any cell c_i and growing network $gnet_j$, the neighbor growth cones of the k^{th} growth cone $g_{ijk} \in \mathbf{G}_{ij}$ are members of the set $\mathbf{N}_{g_{ijk}} \subset \{g_{i\ell k} \in \mathbf{G}_{i\ell} \mid \ell = 1, 2, ..., n\}$. In Figure 5.2a the dashed lines explicitly show two of the six growth-cone-neighborhoods. Because each growth-cone-neighborhood consists of three growth cones connected in a ring topology, $\mathbf{N}_{g_{ijk}} = \{g_{i\ell k} \in \mathbf{G}_{i\ell} \mid \ell = 1, 2, 3 \wedge \ell \neq j\}$. The growth cones within a neighborhood interact with one another according to the canonical PSO algorithm.

When the SINOSA model is used to grow a network that is optimized for a computational problem, on every time-step of the growth process, the performance of each static network is evaluated and used to update the fitness values of the growth cones. The positions of the growth cones are then updated according to the canonical PSO algorithm. Then, on the next time-step, the new physical configurations of the three growing networks are mapped to their corresponding static networks, and the evaluation process repeats. The growth process terminates, and the best performing static network found during the growth process is returned, after a predefined number of time-steps, or once one of the static networks satisfies a pre-specified performance criterion.

5.4 Discussion

This chapter introduced the SINOSA model of network self-assembly and optimization. It is an adaptation of the SINSA model, with the purpose of growing networks that perform optimally on pre-specified computational tasks. The primary difference between the SINOSA and SINSA models is the manner in which the growth cones are organized and interact during the growth process. In the SINSA model all of the growth cones belong to the same, single growing network, and they interact with one another in a manner inspired by the flocking behavior of birds. On the other hand, in the SINOSA model the growth cones belong to multiple, simultaneously growing networks, and the growth cones that belong to different networks interact with one another in the manner specified by the canonical PSO algorithm.

The PSO technique has proven to be very useful for optimizing network weights, but much less so for optimizing network topologies. It was recognized that the self-assembly of network structures provides a means of representing a network's weights and topology in a highly integrated manner. In turn, this integrated representation allows the use of the simple, but powerful canonical PSO algorithm to drive the network growth/optimization process in the SINOSA model. Two techniques, one relying on growth cones carrying fixed weights and the other relying on them carrying weight fields, were introduced as methods for interpreting the physical configuration of a growing network, as a network with fixed connections and weights.

Chapter 6

Applying the SINOSA Model

6.1 Experimental Methods

This section begins by discussing the details of the computational problems that were used to test the ability of the SINOSA model to grow neural networks as solutions. Next, the details of the two different versions of the SINOSA model are introduced, and their implementations are covered. This includes their relationship to PSO, how the motions of the objects represented by these models are generated, the objects' means of interaction, and a description of the models' parameters and their values in the experiments. The section concludes with an explanation of how the networks grown as solutions to the computational problems operate.

6.1.1 Computational Test Problems

Three different types of computational problems were used to test the ability of the SINOSA model to grow optimal neural networks. The problems are from the domains of trajectory generation, time-series forecasting, and control. These problems were chosen because they are challenging, or at least non-trivial, they are frequently used to test neural network optimization methods, and together they represent three of the most common application domains for recurrent neural networks.

The figure-eight trajectory generation problem is a relatively simple but non-

trivial benchmark problem for recurrent neural networks [99]. The task is to train a neural network to generate the trajectory expressed by the system of equations

$$y_1(n) = \sin\left(\frac{4\pi n}{100}\right), \tag{6.1}$$

$$y_2(n) = \sin\left(\frac{2\pi n}{100} + \frac{\pi}{2}\right), \tag{6.2}$$

where $n \in \mathbb{N}$. Equations 6.1 and 6.2 represent a discrete approximation to the continuous trajectory shown in Figure 6.1. Because the trajectory crosses itself at the origin, a neural network must be able to remember and make use of its most recent outputs or hidden states in order to generate the trajectory. The SINOSA model was used to grow relatively small echo state networks that are capable of accurately generating the figure-eight trajectory. Echo state networks were reviewed briefly in Section 2.2.3.

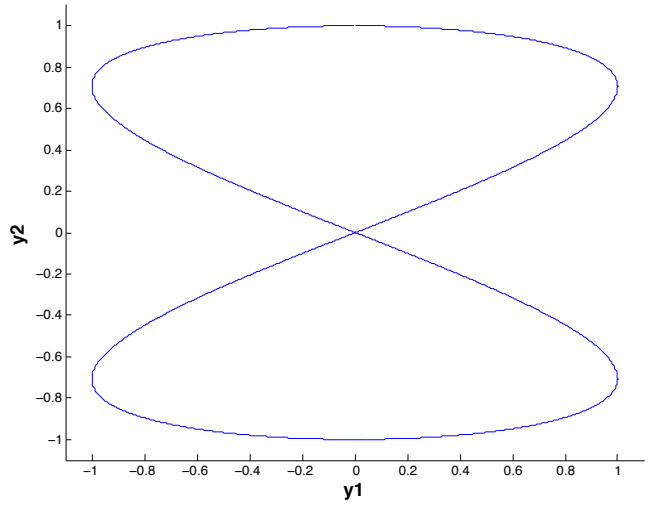


Figure 6.1: The continuous version of the figure-eight trajectory.

The second problem tackled by the SINOSA model is forecasting the chaotic Mackey-Glass time-series, which is a challenging benchmark problem in time-series forecasting [67, 130, 139]. The time-series is generated by the delay differential equation

$$\frac{dy}{dt} = \frac{\alpha y(t-\tau)}{1+y(t-\tau)^{\beta}} - \gamma y(t), \tag{6.3}$$

where $\alpha, \beta, \gamma, \tau \in \mathbb{R}^{+}$. When $\tau > 16.8$ the time-series is chaotic. Figure 6.2 shows a sample of the time-series produced when these parameters are set to the commonly used values $\alpha = 0.2$, $\beta = 10.0$, $\gamma = 0.1$, and $\tau = 17$. The SINOSA model was used to grow larger echo state networks with the ability to forecast this time-series.

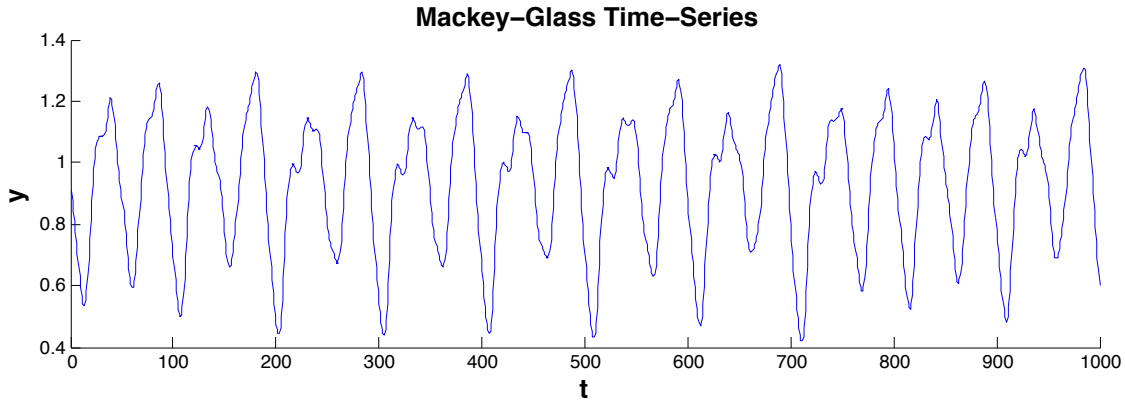


Figure 6.2: An example of the time-series generated by Equation 6.3 with parameters $\alpha = 0.2$, $\beta = 10.0$, $\gamma = 0.1$, and $\tau = 17$.

Third, the SINOSA model was used to grow solution networks for the double pole balancing problem, which is a classic benchmark control problem, particularly for neural network controllers (neural controllers) [48, 68, 132]. The double pole balancing problem consists of using a controller to balance two poles with different lengths that are hinged to the top of a cart that moves along a track of finite length.

The controller attempts to keep the poles up-right by applying a force to either side of the cart in a direction parallel to the track. To be successful, the controller must keep the cart within a specified distance x_{limit} from the center of the track, and it must keep each pole within a specified angular limit θ_{limit} from the vertical. Figure 6.3 shows a schematic of the cart-pole system. The equations governing the dynamics of a cart with N poles are

$$\ddot{x} = \frac{F_c - \mu_c \operatorname{sgn}(\dot{x}) + \sum_{i=1}^{N} \tilde{F}_i}{m_c + \sum_{i=1}^{N} \tilde{m}_i}, \tag{6.4}$$

$$\ddot{\theta} = -\frac{3}{4l_i}\left(\ddot{x}\cos\theta_i + g\sin\theta_i + \frac{\mu_i \dot{\theta}_i}{m_i l_i}\right), \tag{6.5}$$

$$\tilde{F}_i = m_i l_i \dot{\theta}_i^2 \sin\theta_i + \frac{3}{4} m_i \cos\theta_i \left(\frac{\mu_i \dot{\theta}_i}{m_i l_i} + g\sin\theta_i\right),$$

$$\tilde{m}_i = m_i\left(1 - \frac{3}{4}\cos^2\theta_i\right),$$

for $i = 1, 2, ..., N$, (see [132]). Here, F_c is the control force applied to the cart, and $g = 9.8\text{m/s}^2$ is the acceleration due to gravity. The variable x represents the position of the cart relative to the center of the track, $\dot{x}$ is its velocity, and $\ddot{x}$ is its acceleration. The mass of the cart is m_c, and the coefficient of friction between the cart and the track is μ_c. The angular position of the i^{th} pole relative to the vertical is θ_i, and its angular velocity and angular acceleration are $\dot{\theta}_i$ and $\ddot{\theta}_i$, respectively. The mass of the i^{th} pole is m_i, its length is $2l_i$, and the coefficient of friction between

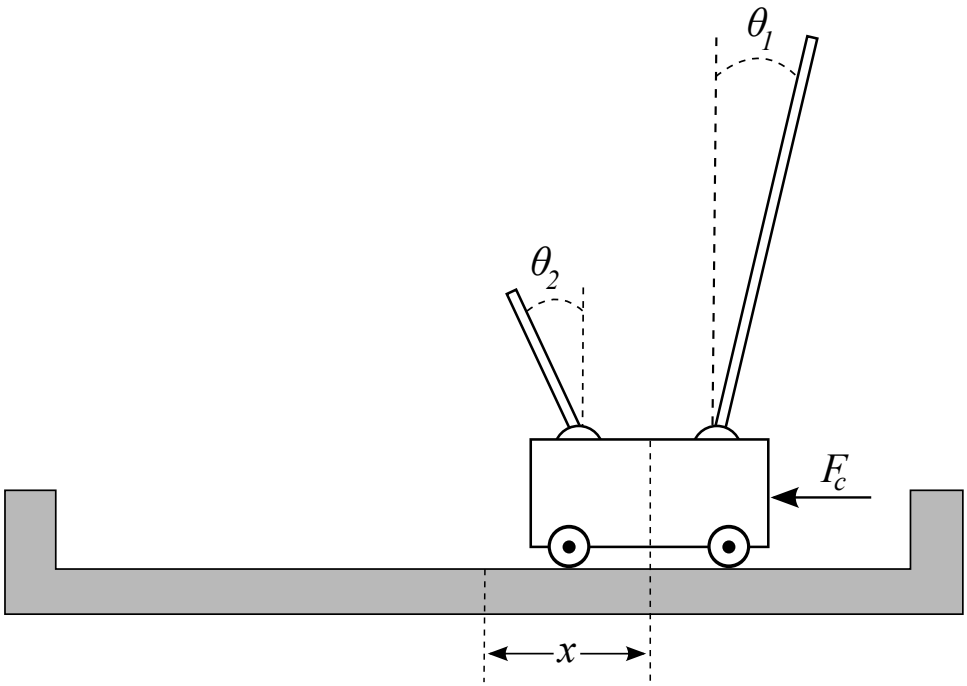


Figure 6.3: The cart-pole system used in the double pole balancing problem. The state of the system is defined by the position x of the cart relative to the center of the track, and the angular positions θ_1 and θ_2 of the large and small poles relative to the vertical. The control force F_c is applied to the side of the cart, in a direction parallel to the track.

the pole and its hinge is μ_i. The variable $\tilde{F}_i$ represents the effective force from the i^{th} pole on the cart, and $\tilde{m}_i$ is its effective mass. The "sgn" symbol represents the signum function. The SINOSA model was used to grow echo state networks, and smaller, non-echo state networks that are able to control the cart-pole system.

6.1.2 Implementation Details

The computational experiments described in Section 6.2 involve two different versions of the SINOSA model. In the first version, which will be denoted SINOSA$_a$, the growth cones have weight fields (Eq. 5.1), the components of growing networks (cells, axons, and growth cones) are not able to collide with one another, and forces

are not present. In the second version, which will be denoted SINOSA$_b$, the growth cones carry fixed weight values, growth cones are able to collide with cells and with each other, and forces are present. In both of these models the cells' positions remain fixed throughout the growth process.

Both the SINOSA$_a$ and SINOSA$_b$ models are implemented as simulation environments that are written in Java. In the experiments described below, the environment in which networks grew was unbounded and no particular units were assigned to time, distance, and mass. Unless stated otherwise, in every experiment involving either version of the SINOSA model, the positions of the cells were fixed on a 2D centered rectangular lattice with 8.0 distance-units between adjacent lattice points, there were 16 growing networks, and the growth cone neighborhoods adhered to a von Neumann topology.

The SINOSA$_a$ model does not incorporate collisions or forces. The dynamics of the growth cones are governed by the canonical PSO equation (Eq. 2.1), where $a_p = a_n = 2.0$, $\chi = 0.75$ for the figure-eight experiments, $\chi = 0.65$ for the Mackey-Glass experiments, and $\chi = 0.725$ for the double pole balancing experiments. For all of the experiments, each growth cone's weight field was linear ($\alpha = 1$) and had a radius $r_0 = 2.0$. By convention, one time-step in the SINOSA$_a$ model is equivalent to 1.0 unit of time.

The SINOSA$_b$ model does incorporate collisions and forces. The dynamics of the growth cones are partly governed by a force inspired by PSO, which will be referred to as the PSO force. This force has the same form as Equation 2.1, but rather than directly representing the change in velocity of a growth cone (particle),

as is typically the case in PSO, it instead expresses a component of a growth cone's instantaneous acceleration. Let $\vec{r}_i$ be the current position of the i^{th} growth cone, then the PSO force is given by

$$\vec{F}_{pso}(\vec{r}_i) = a_p\vec{u}_1 \otimes (\vec{p}_{best,i} - \vec{r}_i) + a_n\vec{u}_2 \otimes (\vec{n}_{best,i} - \vec{r}_i), \tag{6.6}$$

where $\vec{p}_{best,i}$ is the current best position of the i^{th} growth cone, $\vec{n}_{best,i}$ is the best position among any of its neighbor growth cones, a_p and a_n are positive constants, $\vec{u}_1$ and $\vec{u}_2$ are vectors whose components are drawn from the uniform probability distribution over the unit interval, and the $\otimes$ symbol represents the component-wise vector product (i.e., $[a_1\ a_2] \otimes [b_1\ b_2] = [a_1b_1\ a_2b_2]$). In every experiment involving the PSO force $a_p = a_n = 2.0$. In the SINOSA$_b$ model the growth cones are subject to a viscous drag force, which has the same form as the drag force present in the SINSA model (Eq. 3.8), and has $c_d = 1.0$. This force restricts the velocities of the growth cones, and is present in lieu of the constriction coefficient χ, which serves the same purpose in the canonical PSO equation. The equation that governs collisions between growth cones, and between growth cones and cells is the same as that used in the SINSA model, namely Equation 3.9, with $c_f = 25.0$ and $\gamma = 3.0$ for both types of collisions. In all of the experiments involving the SINOSA$_b$ model, it was required that a growth cone be less than 1 cell radius away from the surface of a target cell in order to establish a connection with that cell. Cells had a radius of $r_c = 2.0$, and growth cones had a radius of $r_g = 0.05$, unless stated otherwise. Cells and growth cones had masses of $m_c = 1.0$ and $m_g = 0.5$, respectively.

Many of the experimental results presented in Section 6.2 are compared to control cases that incorporated random growth cone movements, as opposed to movements driven by the canonical PSO equation or the PSO force. From an optimization perspective, the network growth produced by the random movements is equivalent to random search. These random movements were generated in two different ways depending on which version of the SINOSA model was used. In the SINOSA$_a$ model, at every time-step of the growth process and for each growth cone, a target neuron is randomly selected and the growth cone is placed at a randomly selected position that is less than a distance r_0 from the center of the chosen target cell, where r_0 is the extent of the growth cone's weight field. In this way, both the cell that a growth cone establishes a connection with, and the weight on that connection, are randomly generated. In the SINOSA$_b$ model, every 1.0 time-units and for each growth cone, a target neuron is randomly selected and a point in space $\vec{r}^{\,\prime}$ is randomly chosen that is less than one cell radius away from the surface of the selected target neuron, but not within its physical boundary. If $\vec{r}$ is the position of the growth cone, then the *centering force* that is applied to the growth cone is expressed as

$$F_{center}\left(\vec{r}\right) = a_c\vec{u} \otimes \left(\vec{r}^{\,\prime} - \vec{r}\right), \tag{6.7}$$

where a_c is a positive constant, $\vec{u}$ is a vector whose components are drawn from the uniform probability distribution over the unit interval, and the $\otimes$ symbol represents the component-wise vector product. In every experiment involving the centering

force $a_c = 2.0$. The centering force was used, as opposed to a force with purely random magnitude and direction, because it causes the growth cones to remain relatively close to their target neurons, which is a property exhibited by growth processes that incorporate the PSO force, and is essential for the growth of good performing networks.

As in the SINSA model, the force-based dynamics of growth cones in the SINOSA$_b$ model are governed by Newton's second law. Assume that there are n cells and m growth cones, and that the mass of each growth cone is m_g. Then the force-based movements of growth cones are governed by

$$\frac{d^2\vec{r}_i}{dt^2} = \frac{1}{m_g}\left(\vec{F}_{drag}(\vec{v}_i) + \vec{F}_{pso}(\vec{r}_i) + \sum_{j\neq i}^{n+m} \vec{F}_{collision}(\vec{r}_{ji})\right), \tag{6.8}$$

where $i = 1, 2, 3, ..., m$, $j = 1, 2, 3, ..., n+m$, $\vec{r}_i$ is the position vector of the i^{th} growth cone, $\vec{v}_i$ is its velocity, $\vec{r}_{ji} = \vec{r}_i - \vec{r}_j$ is the position of the i^{th} growth cone relative to the j^{th} cell or growth cone, and the summation is taken over all cells and growth cones. If the centering force (Eq. 6.7) is implemented, it replaces the PSO force in Equation 6.8. In all experiments, Equation 6.8 was integrated using the Forward Euler method. A time-step of 0.01 was used in experiments that incorporated a cell radius of 0.5, and a time-step of 0.004 was used when the cell radius was 2.0. The different temporal resolutions were necessary in order to ensure that collisions were accurately computed for the different system scales. For each growth cone, the PSO force or the centering force was computed every 1.0 time units, whereas all other forces were computed on every time-step. This was necessary in order to

give the growth cones adequate time to be influenced by these forces since they are stochastic. Additionally, applying the PSO force at a constant value for 1.0 time-unit intervals makes it the continuous-time analogy of Equation 2.1, which represents an instantaneous change in a growth cone's velocity as opposed to an acceleration.

During the training process for the figure-eight trajectory generation problem or the Mackey-Glass time-series forecasting problem, an echo state network does not have connections from the output layer to the reservoir (output feedback). The assumption is that before and during the early parts of training the network's outputs will tend to have large errors, and hence it is better for the reservoir to only "see" the actual trajectory or time-series, which is fed as input to the network during training. Once the network has been adequately trained it is able to generate the trajectory or forecast the time-series without receiving input. After an initial sequence of data (lead-in) from the trajectory/time-series is input into the network, output feedback is added so that the network can make use of the information carried in its own output. In a trajectory generation or time-series forecasting problem the dimension of the output layer will equal that of the input layer. The feedback connections from an output neuron are constructed by giving that neuron the connections (and weights) that connect from the corresponding input neuron to the reservoir.

The activities of neurons in networks grown as controllers for the double pole balancing problem are updated in a particular and asynchronous manner. First, input to the network is passed to the reservoir, the reservoir neurons pass their outputs to other reservoir neurons, and the state of the reservoir is updated. Second,

the reservoir neurons pass their new outputs to the output neuron, and the output of the network is computed. This is done so that the reservoir is able to integrate information about the current configuration of the cart-pole system into its state before the network produces a control signal.

The computational experiments presented in Section 6.2 each ran on a computer with two quad-core 2.33 GHz Intel Xeon processors, 8 GB of shared RAM, and 12 MB of L2 cache per processor. The computational requirements listed here are for the growth of echo state networks using collective movements, unless stated otherwise. The trials run in the figure-eight experiments require approximately 0.4 minutes of CPU time for the SINOSA$_a$ model, and 28 minutes of CPU time for the SINOSA$_b$ model. In the Mackey-Glass experiments, on average, one epoch of growth (explained in Sec. 6.2.2) of an ESN with 50 neurons in its reservoir requires 2.0 hours of CPU time for the SINOSA$_a$ model, and four epochs of growth requires 4.3 hours of CPU time for the SINOSA$_b$ model. One epoch of growth of an ESN with 100 neurons in its reservoir requires 5.0 hours of CPU time for the SINOSA$_a$ model. One epoch of growth of an ESN with 400 neurons in its reservoir requires 3.0 days of CPU time for the SINOSA$_a$ model. In the double pole balancing experiments, for the SINOSA$_a$ model, 200 time-steps of growth requires approximately 0.3 hours of CPU time, 400 time-steps requires 1.4 hours, and 600 time-steps requires 3.3 hours. For the SINOSA$_b$ model, 200 time-steps of growth requires approximately 3.0 hours of CPU time, 400 time-steps requires 6.2 hours, and 600 time-steps requires 9.2 hours. When the SINOSA$_a$ model is used to grow smaller networks, with 7 hidden layer neurons, for the double pole balancing problem, 200 time-steps of growth re-

quires approximately 1.1 hours of CPU time, 400 time-steps requires 3.5 hours, and 600 time-steps requires 5.9 hours.

6.2 Results

For each of the computational problems discussed in Section 6.1.1, the SINOSA$_a$ and SINOSA$_b$ models were used to grow echo state networks as solutions. The SINOSA$_a$ model was also used to grow smaller, non-echo state networks as solutions to the double pole balancing problem. These networks had 7 hidden layer neurons, and unlike the echo state networks grown for the double pole balancing problem, they were permitted to have connections from the input neurons to the output neuron, from the output neuron back to itself, and from the output neuron to the neurons in the hidden layer. Four groups of computational experiments were performed. These experiments were designed to test the optimization capabilities of the SINOSA model, and the impacts of various parameters. The first group of computational experiments compares the quality of the networks grown when collective movements among the growth cones is incorporated, to the quality of those grown when the growth cones do not engage in collective movements, but instead exhibit movements of a random nature (explained in Sect. 6.1.2). These experiments were performed for each of the computational problems, and with both versions of the SINOSA model. The second group of experiments studies the effects of using different growth cone neighborhood sizes and topologies on the performance of the echo state networks grown by the SINOSA$_a$ model for the Mackey-Glass and double pole

balancing problems. The third group of experiments examines the impact of using different physical dimensions for the cells, growth cones, and inter-cell spacing when the SINOSA$_b$ model is used to grow echo state networks for the Mackey-Glass and double pole balancing problems. The fourth group of experiments test the ability of the SINOSA$_a$ model to grow small, non-echo state networks that act as controllers for the double pole balancing problem.

6.2.1 Figure-Eight Trajectory

The SINOSA$_a$ and SINOSA$_b$ models were used to grow echo state networks for the purpose of generating the figure-eight trajectory (Eqs. 6.1 and 6.2). These networks consisted of two input neurons, two output neurons, and had 10 neurons in the reservoir. No bias neuron was used due to the symmetry of the figure-eight trajectory. Neurons in the reservoir were integrator neurons (explained below) with $c = 0.44$ and $r = 0.9$ (Eq. 6.11). Reservoir neurons used the hyperbolic-tangent function as their transfer function

$$tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}. \tag{6.9}$$

The output neuron used the linear transfer function $f(x) = x$. The growth cones were permitted to establish connections from the input neuron to the reservoir neurons, and from the reservoir back to the reservoir. Additionally, each reservoir neuron had a permanent connection to each output neuron. The weights on the reservoir-to-output connections were derived using linear regression once a growing

network was mapped to its static network representation. For echo state networks grown using the SINOSA$_a$ model, each input neuron's set of neighbor neurons (neurons it can make connections to) was the entire reservoir. Each reservoir neuron's set of neighbor neurons consisted of 3 randomly selected neurons in the reservoir. The output neurons did not have any neighbor neurons, because they did not have any growing axons. For each one of the simultaneously growing networks (sets of growth cones), each neuron contributed one positively-weighted growth cone ($b = 1$ in Eq. 5.1), and one negatively-weighted growth cone ($b = -1$), per neighbor neuron. Each of these positive-negative growth cone pairs had the same, single target neuron. For echo state networks grown using the SINOSA$_b$ model, the neighbor neuron set of both input neurons consisted of all of the neurons in the reservoir. The neighbor neuron set of a neuron in the reservoir consisted of a randomly selected group of 3 reservoir neurons that were geometric neighbors in the three-dimensional "physical" space in which network growth took place. Specifically, a neighbor neuron set was chosen by first selecting a reservoir neuron at random, and then selecting the 2 neurons that were closest in physical proximity to that neuron. If more than 2 neurons satisfied this criterion, then those neurons that were the same distance from the initially chosen neuron were selected at random for membership in the neighbor neuron set. The restriction that these neurons be geometric neighbors was implemented so that the target neurons of any growth cone that was guiding an axon from a reservoir neuron would be in the same general region of space. This in turn ensured that the growth cones would tend to stay relatively close to their targets. The output neuron did not have any neighbor neurons. For a reservoir of

N_{res} neurons, both input neurons contributed N_{res} growth cones with a fixed weight of 0.1, and N_{res} growth cones with a fixed weight of -0.1, to each of the simultaneously growing networks. All of these growth cones had the same set of target neurons, which consisted of all of the neurons in the reservoir. Each neuron in the reservoir contributed 5 growth cones with a fixed weight of 0.1 and 5 growth cones with a fixed weight of -0.1 to each of the growing networks. Each growth cone that belonged to a reservoir neuron had a set of target neurons that consisted of the 3 neighbor neurons of the growth cone's parent neuron.

During the growth process the performance of each static network was computed on every time-step. For the echo state networks grown to generate the figure-eight trajectory, the performance measure was the root mean square error (RMSE) computed over a sequence of N target data points from the figure-eight trajectory. The RMSE is given by

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{N} (\vec{e}_i)^\intercal \, \vec{e}_i}{N}}, \tag{6.10}$$

where the i^{th} error term $\vec{e}_i = \vec{y}_i - \hat{\vec{y}}_i$ is the difference between the target data point and the output from the network. On every time-step, each growing network was mapped to the static network represented by its current physical configuration. Before applying input to a network, the internal state and/or output of each neuron was set to zero. Prior to computing the RMSE, a network always received 100 data points (1 figure-eight cycle) as input with the purpose of removing the effects of the initial state of the reservoir. For each static network, the weights on the

connections from the reservoir neurons to the two output neurons were trained by passing a sequence of 200 data points from the figure-eight trajectory through the network, and then performing linear regression in the form of the least squares method between the last 100 reservoir states (activities of the reservoir neurons) and the desired network outputs. The topology and weights of the connections from the input neurons to the reservoir, and from the reservoir neurons back to the reservoir, were determined by the growth process. On every time-step, after training the output weights, the performance (RMSE) of each static network was computed using a sequence of 200 data points (2 figure-eight cycles). At the end of the growth process, which lasted 1000 time-units, the performance of each growing network's best static network was validated by computing the RMSE for each network on a sequence of 1000 data points (10 figure-eight cycles). The best performing network on this validation data was taken as the solution.

For the SINOSA$_a$ model, 33 trials (growth processes) were run with collective movements, and 20 trials were run with random movements. When the growth cones exhibited collective movements the median RMSE on the validation data for the grown networks was $1.47 \cdot 10^{-11}$ with a 95% confidence interval of $[7.54 \cdot 10^{-12}, 2.19 \cdot 10^{-11}]$, and when they exhibited random movements the median RMSE was 0.0737 with a 95% confidence interval of $[0.0415, 0.106]$. For the SINOSA$_b$ model, 45 trials were run with collective movements, and 45 trials were run with random movements. When the growth cones exhibited collective movements the median RMSE on the validation data for the grown networks was $2.95 \cdot 10^{-6}$ with a 95% confidence interval of $[1.32 \cdot 10^{-6}, 4.59 \cdot 10^{-6}]$, and when they exhibited random

movements via the centering force (Eq. 6.7) the median RMSE was $2.63 \cdot 10^{-4}$ with a 95% confidence interval of $[1.23 \cdot 10^{-4}, 4.03 \cdot 10^{-4}]$. The networks grown using collective movements outperform those grown with random movements by multiple orders of magnitude for both versions of the SINOSA model.

6.2.2 Mackey-Glass Time-Series

In all of the experiments that involve the Mackey-Glass time-series the parameters of Equation 6.3 were set to the following commonly used values: $\alpha = 0.2$, $\beta = 10$, $\gamma = 0.1$, $\tau = 17$. These values yield a "mildly" chaotic time-series. The time-series was generated by solving Equation 6.3 using the Matlab delay differential equation solver *dde23* with a maximum step-size of 1.0, a relative error tolerance of 10^{-4}, and an absolute error tolerance of 10^{-16}. For every time-series generated from Equation 6.3, an initial sequence of data points was randomly generated from the uniform probability distribution over the interval $[0, 1]$, and Equation 6.3 was integrated for 1000 time-steps before collection of the time-series data began. This initial run-off period was necessary to remove the effects of the randomly generated initial condition. Consecutive data points in the sequences generated by the Mackey-Glass system were separated by 1.0 units of time. Data from the Mackey-Glass system was made more appropriate for processing by neural networks by mapping it into the interval $[-1, 1]$ using the hyperbolic tangent function (Eq. 6.9). Network output was mapped back to the original range using the inverse of Equation 6.9 for testing, validation, and analysis. Reservoir neurons used Equation 6.9 as their transfer

function. The output neuron used the linear transfer function $f(x) = x$.

The SINOSA$_a$ and SINOSA$_b$ models were used to grow echo state networks for the purpose of forecasting the Mackey-Glass time-series. These networks consisted of a single input neuron, a bias neuron, a single output neuron, and 50 neurons in the reservoir. The growth cones were permitted to establish connections from the input neuron to the reservoir neurons, from the bias neuron to the reservoir, and from the reservoir back to the reservoir. Additionally, each reservoir neuron had a permanent connection to the output neuron. The weights on the reservoir-to-output connections were derived using linear regression.

For echo state networks grown using the SINOSA$_a$ model, the input neuron's set of neighbor neurons was the entire reservoir, as was the case for the bias neuron. Each reservoir neuron's set of neighbor neurons consisted of 5 randomly selected neurons in the reservoir. The output neuron did not have any neighbor neurons, because it did not have any growing axons. For each one of the simultaneously growing networks, each neuron contributed one positively-weighted growth cone ($b = 1$ in Eq. 5.1), and one negatively-weighted growth cone ($b = -1$), per neighbor neuron. Each of these positive-negative growth cone pairs had the same, single target neuron.

For echo state networks grown using the SINOSA$_b$ model, the neighbor neuron set of both the input neuron and the bias neuron consisted of all of the neurons in the reservoir. The neighbor neuron set of a neuron in the reservoir consisted of a randomly selected group of 5 reservoir neurons that were geometric neighbors. (The process for constructing a set of geometric neighbors is explained in Section 6.2.1).

The output neuron did not have any neighbor neurons. For a reservoir of N_{res} neurons, both the input neuron and the bias neuron contributed N_{res} growth cones with a fixed weight of 0.1, and N_{res} growth cones with a fixed weight of -0.1, to each of the simultaneously growing networks. For both the input neuron and the bias neuron, and for each growing network, the k^{th} positively-weighted growth cone and the k^{th} negatively-weighted growth cone had the same set of target neurons, which consisted of a randomly selected group of 10 reservoir neurons that were geometric neighbors. The geometric neighbor restriction was implemented so that the volume of space occupied by each set of target neurons for input and bias growth cones was not too much larger than the volume of space occupied by each set of target neurons for the reservoir growth cones. This ensured that for each type of growth cone the PSO force could be used with the same parameter values. Each neuron in the reservoir contributed 5 growth cones with a fixed weight of 0.1, and 5 growth cones with a fixed weight of -0.1, to each of the growing networks. Each growth cone that belonged to a reservoir neuron had a set of target neurons that consisted of the 5 neighbor neurons of the growth cone's parent neuron.

The Mackey-Glass time-series is continuous, and smooth over short intervals of time. This means that it is typically suitable for the output of any neuron in the reservoir to change by only a small amount from one time-step to the next. Smooth dynamics are often generated by using neurons that incorporate an internal state. The rate of change of an internal state usually depends on the current input to the neuron and on the current or past states. In this way, an internal state acts as a form of memory, where at any given time the output of the neuron is the

image of its current state under its transfer function, as opposed to the image of its current input. The reservoirs of the echo state networks grown using the SINOSA models consisted of neurons with internal states. These neurons will be referred to as *integrator* neurons. The internal state $s(t)$ of an integrator neuron is governed by the ordinary differential equation

$$\frac{ds}{dt} = c(-rs(t) + u(t)), \tag{6.11}$$

where $u(t)$ is the neuron's input at time t, $c \in \mathbb{R}^+$ is a time constant, and $r \in \mathbb{R}^+$ is the decay rate. When used in neural networks this equation is normally discretized using a time-step of 1.0. Thus, at time $t+1$ the output of an integrator neuron is determined by first updating its state according to $s(t+1) = s(t) + c(-rs(t) + u(t+1))$, and then computing its output $y(t+1) = f(s(t+1))$, where f is the neuron's transfer function. In all of the experiments presented herein $c = 0.44$ and $r = 0.9$, which are the empirically determined values suggested in [66] for the Mackey-Glass time-series. Each reservoir neuron had an internal state, and the output neuron did not have an internal state.

During the growth process the performance of each static network was computed on every time-step. For the echo state networks grown to forecast the Mackey-Glass time-series, the performance measure was the normalized root mean square error computed over a set of 84-step predictions (NRMSE_{84}). Eighty-four time-steps is a commonly used prediction horizon, and was used here to facilitate comparison with other studies. To compute the NRMSE_{84}, N sequences ($i = 1, 2, ..., N$) of

length $n+84$ are generated from the Mackey-Glass system using randomly selected initial states, where $n \in \mathbb{N}$. The first n data points of the i^{th} sequence are input to the network, and then the network is made to predict the next 84 data points in the sequence. Let $\hat{y}_i(n+84)$ be the 84^{th} data point predicted by the network, and $y_i(n+84)$ be the actual data point in the i^{th} sequence. Then the NRMSE_{84} is given by

$$\text{NRMSE}_{84} = \left(\frac{\sum_{i=1}^{N} \left(y_i(n+84) - \hat{y}_i(n+84) \right)^2}{N\sigma^2} \right)^{1/2}, \tag{6.12}$$

where σ^2 is the variance of the time-series generated by the Mackey-Glass system.

On every time-step each growing network was mapped to the static network represented by its current physical configuration. Before applying input to a network, the internal state and/or output of each neuron was set to zero. For each static network, the weights on the connections from the reservoir neurons to the output neuron were trained using a sequence generated from the Mackey-Glass system that had 2100 data points. The first 100 data points of this sequence were fed into a network prior to any training with the purpose of removing the effects of the initial state of the reservoir. The next 2000 data points were then fed into the network, and linear regression in the form of the least squares method was performed between the resulting reservoir states (activities of the reservoir neurons) and the desired network outputs. The topology and weights of the connections from the input neuron to the reservoir, from the bias neuron to the reservoir, and from the reservoir neurons back to the reservoir, were determined by the growth process.

The performance of an ESN on the data used to train the output weights is typically not a good measure of the network's ability to generalize to new data [101]. Thus, on every time-step, after training the output weights, the NRMSE_{84} was computed for each static network on a group of 20 randomly selected sequences from the Mackey-Glass system. Each of these sequences consisted of 184 data points ($n = 100$). Twenty different sequences were used, as opposed to just one, so that the growth process would produce networks with the ability to generalize well. In order to prevent overgeneralization, every 10 time-steps a new set of 20 sequences was randomly pulled from a pool of 200 different sequences. Whenever this occurred, the NRMSE_{84} was computed, using this new data, for the best performing static network discovered by each of the growing networks up to the current time-step. The resulting NRMSE_{84} values were then assigned to be the fitness values of the corresponding growing networks. This was done to reward(punish) networks with good(bad) generalization abilities.

Twenty sequences consisting of 184 data points each is a fairly small sample of the Mackey-Glass time-series. A larger sample consisting of 100 sequences, each of length 2084, was used for testing every 200 time-steps once the 2000 time-step of the growth process had been reached. Each growing network's best performing static network was instantiated, and these 100 sequences were used to compute the NRMSE_{84} of each static network. The resulting performances were used to maintain a list of the top 10 best performing networks over the course of the growth process. At the end of the growth process, which lasted 3600 time-units for the SINOSA_a model and 200 time-units for the SINOSA_b model, the performances of these top

10 networks were validated by computing the NRMSE_{84} of each network using 100 new sequences, each of length 2084. The best performing network on this validation data was taken as the solution.

For the SINOSA_a model, 37 trials were run with collective movements, and 38 trials were run with random movements. When the growth cones exhibited collective movements the mean NRMSE_{84} on the validation data for the grown networks was $5.89 \cdot 10^{-3} \pm 3.3 \cdot 10^{-4}$, and when they exhibited random movements the mean NRMSE_{84} was $1.84 \cdot 10^{-2} \pm 8 \cdot 10^{-4}$. These values are shown with their 95% confidence intervals. For the SINOSA_b model, 37 trials (growth processes) were run with collective movements, and 38 trials were run with random movements. When the growth cones exhibited collective movements the mean NRMSE_{84} on the validation data for the grown networks was $2.80 \cdot 10^{-2} \pm 1.5 \cdot 10^{-3}$, and when they exhibited random movements via the centering force the mean NRMSE_{84} was $3.69 \cdot 10^{-2} \pm 3.7 \cdot 10^{-3}$. The networks grown using collective movements have a mean NRMSE_{84} that is 68% smaller than those grown with random movements for the SINOSA_a model, and a mean NRMSE_{84} that is 24% smaller for the SINOSA_b model.

Once the growth process had finished (after 3600 time-units for the SINOSA_a model and 200 time-units for the SINOSA_b model) the grown networks were further optimized by refining the search process. This refinement was implemented by continuing the growth (search) process with new growth cones that had weight fields that were smaller in maximum magnitude, or that had fixed weights that were smaller in magnitude. Specifically, once the first epoch of growth was completed, a

second epoch of growth was begun that lasted for the same number of time-units. This second period of growth was initiated by first destroying all of the growth cones that were present in the first epoch. Next, for each connection in the best performing static network found during the first epoch, except the connections from the reservoir to the output neuron, a fixed connection with the same weight was created between the corresponding cells in the set **C**. This had the effect of fixing a point in the dual topology-weight space around which the new growing networks would perform a local search.

New sets of growth cones (growing networks) were then created, as follows. For a newly created fixed connection from cell c_i to cell c_ℓ with weight $w_{i\ell}$, cell c_i generated $2n$ new growing axons (growth cones), where the number of growing networks n was the same as in the first epoch. Based on this fixed connection, cell c_i contributed one positive growth cone g_{ij}^+, and one negative growth cone g_{ij}^-, to each set of growth cones $j = 1, 2, ..., n$. In the SINOSA$_a$ model, the maximum magnitude of the weight field carried by these growth cones was $|c_w w_{i\ell}|$, where the constant $0 < c_w < 1$ controlled how localized the search process was during the second epoch. The single target neuron of g_{ij}^+ and g_{ij}^- was c_ℓ. The initial position of each growth cone was a randomly selected position within 2.0 distance-units of the center of cell c_ℓ. In the SINOSA$_b$ model, the fixed weight carried by g_{ij}^+ was $|c_w w_{i\ell}|$, and the fixed weight carried by g_{ij}^- was $-|c_w w_{i\ell}|$. If c_i was a neuron in the reservoir, then the target neurons of g_{ij}^+ and g_{ij}^- were all of the neighbor neurons of c_i. If c_i was the input or bias neuron, then the target neurons of g_{ij}^+ and g_{ij}^- were a set of 10 reservoir neurons, including c_ℓ, that were geometric neighbors centered around c_ℓ. The initial

position of each growth cone was a randomly selected position that was less than 1 cell radius away from the surface of a randomly chosen target neuron.

In both versions of the SINOSA model, the growth cones in one of the n newly created growing networks were initially positioned such that their weights would cancel when they were used to instantiate a static network for the first time, thus producing the best network found during the previous epoch. The second epoch of growth began once all of the growth cones had been generated based on the newly created fixed connections. When a static network was instantiated from a growing network during the second epoch, the weights on the connections in the static network were the sum of the weight values contributed by the growth cones and the fixed connections. Epochs beyond a second were initialized in the same manner as described above, except that both the growth cones and the fixed connections that were created at the beginning of the previous epoch were destroyed, and new growth cones and fixed connections were created based on the best performing static network found during all of the previous epochs.

The network growth process generated by the SINOSA_a model was extended by one epoch, for a total of two epochs of growth, and for the SINOSA_b model it was extended by three epochs, for a total of four epochs of growth. Tables 6.1 and 6.2 compare the results obtained using collective movements, with those obtained using random movements, when the growth process was extended. Each numeric value represents the mean NRMSE_{84} and the 95% confidence interval for the corresponding epoch of growth and class of movements. It can be seen that for both versions of the SINOSA model, and for both collective and random movements,

Table 6.1: Mean $NRMSE_{84}$ Values on the Mackey-Glass time-series for Networks Grown with the $SINOSA_a$ Model

Epoch	Collective Movements	Random Movements
1	$5.89 \cdot 10^{-3} \pm 3.3 \cdot 10^{-4}$	$1.84 \cdot 10^{-2} \pm 8 \cdot 10^{-4}$
2	$4.98 \cdot 10^{-3} \pm 3.2 \cdot 10^{-4}$	$1.48 \cdot 10^{-2} \pm 7 \cdot 10^{-4}$

Table 6.2: Mean $NRMSE_{84}$ Values on the Mackey-Glass time-series for Networks Grown with the $SINOSA_b$ Model

Epoch	Collective Movements	Random Movements
1	$2.80 \cdot 10^{-2} \pm 1.5 \cdot 10^{-3}$	$3.69 \cdot 10^{-2} \pm 3.7 \cdot 10^{-3}$
2	$2.11 \cdot 10^{-2} \pm 1.0 \cdot 10^{-3}$	$2.72 \cdot 10^{-2} \pm 9 \cdot 10^{-4}$
3	$1.45 \cdot 10^{-2} \pm 9 \cdot 10^{-4}$	$2.11 \cdot 10^{-2} \pm 1.0 \cdot 10^{-3}$
4	$1.12 \cdot 10^{-2} \pm 6 \cdot 10^{-4}$	$1.59 \cdot 10^{-2} \pm 1.0 \cdot 10^{-3}$

there is a small but statistically significant reduction in the mean $NRMSE_{84}$ with each epoch of growth. Furthermore, for each epoch, the mean $NRMSE_{84}$ of the networks grown using collective movements is smaller than that of the networks grown using randomly generated movements.

Using collective movements, and the same parameters and methods described above, the $SINOSA_a$ model was used to grow echo state networks with 100 and 400 neuron reservoirs. After two epochs of growth, the average $NRMSE_{84}$ over 63 trials for the ESNs with 100 neurons was $9.28 \cdot 10^{-4} \pm 3.4 \cdot 10^{-5}$, and the average over 16 trials for the ESNs with 400 neurons was $3.86 \cdot 10^{-5} \pm 3.1 \cdot 10^{-6}$. On average, the 400 neuron echo state networks grown using the $SINOSA_a$ model perform nearly an order of magnitude better than the best performing 400 neuron ESN presented in [66], which an expert designed by hand. Furthermore, on average, the 400 neuron

grown ESNs perform better than the 1000 neuron ESN presented in [67], which was also hand-designed by an expert.

6.2.3 Double Pole Balancing Problem

In all of the experiments that dealt with the double pole balancing problem, the parameters of Equations 6.4 and 6.5 were set to the most commonly used values [68], as follows: $m_c = 1\text{kg}$, $m_1 = 0.1\text{kg}$, $m_2 = 0.01\text{kg}$, $\mu_c = 5 \cdot 10^{-4}\text{Ns/m}$, $\mu_1 = \mu_2 = 2 \cdot 10^{-6}\text{Nms}$, $l_1 = 0.5\text{m}$, $l_2 = 0.05\text{m}$. The control force was restricted to the interval $F_c \in [-10\text{N}, 10\text{N}]$. The parameters defining the domain of successful control were set to $x_{limit} = 2.4\text{m}$, and $\theta_{limit} = 36°$. As is the case in most past work, Equations 6.4 and 6.5 were solved numerically using a fourth-order Runge-Kutta method with a step-size of 0.01s. During a simulation, a portion of the state of the cart-pole system was given to a neural controller every 0.02s, at which point the control force was updated. In the experiments presented herein, a neural controller was not given velocity information as input, rather, it only received the current positions of the cart and two poles (x, θ_1, and θ_2). These values were scaled to be in the interval $[-1, 1]$ prior to being input into a neural controller. This was done so that the values were in a range that was more appropriate for processing by neurons with hyperbolic-tangent transfer functions. The network output (control signal), which was in the interval $[-1, 1]$, was multiplied by 10.0N in order to produce the control force. Reservoir neurons and the output neuron used the hyperbolic-tangent function as their transfer function (Eq. 6.9). None of the neurons had an internal state.

The SINOSA$_a$ and SINOSA$_b$ models were used to grow echo state networks as controllers for the double pole balancing problem. These networks had three input neurons, one for each type of information the network was given regarding the state of the cart-pole system (cart position, position of pole #1, and position of pole #2). The reservoir always consisted of 20 neurons. One output neuron was present, which produced the control signal. No bias neuron was used due to the symmetry of the cart-pole system. The growth cones were permitted to establish connections from the input neurons to the reservoir, and from the reservoir neurons back to the reservoir. Additionally, each reservoir neuron had a permanent connection to the output neuron. The weights on the reservoir-to-output connections were fixed, and drawn randomly with uniform probability from the interval $[-30, 30]$.

For echo state networks grown using the SINOSA$_a$ model, each input neuron's set of neighbor neurons was the entire reservoir. Each reservoir neuron's set of neighbor neurons consisted of 4 randomly selected neurons in the reservoir. The output neuron did not have any neighbor neurons, because it did not have any growing axons. For each one of the simultaneously growing networks, each neuron contributed one positively-weighted growth cone ($b = 1$ in Eq. 5.1), and one negatively-weighted growth cone ($b = -1$), per neighbor neuron. Each of these positive-negative growth cone pairs had the same, single target neuron.

For echo state networks grown using the SINOSA$_b$ model, the neighbor neuron set of each input neuron consisted of all of the neurons in the reservoir. The neighbor neuron set of a neuron in the reservoir consisted of a randomly selected group of 4 reservoir neurons that were geometric neighbors. The output neuron did not have

any neighbor neurons. Each input neuron contributed 100 growth cones with a fixed weight of 0.1, and 100 growth cones with a fixed weight of -0.1, to each of the simultaneously growing networks. For each input neuron, and for each growing network, the k^{th} positively-weighted growth cone and the k^{th} negatively-weighted growth cone had the same set of target neurons, which consisted of a randomly selected group of 10 reservoir neurons that were geometric neighbors. Each neuron in the reservoir contributed 20 growth cones with a fixed weight of 0.1 and 20 growth cones with a fixed weight of -0.1 to each of the growing networks. Each growth cone that belonged to a reservoir neuron had a set of target neurons that consisted of the 4 neighbor neurons of the growth cone's parent neuron.

During the growth process the performance of each static network was computed on every time-step. The function f_{pole} was evaluated to determine the performance of the echo state networks grown as controllers for the double pole balancing problem, and is given by

$$f_{pole} = 10^{-4} n_I + 0.9 f_{stable} + 10^{-5} n_{II} + 30 \frac{n_S}{625}. \tag{6.13}$$

Equation 6.13 was introduced in [68], and is based on performance (fitness) functions presented in past works on the double pole balancing problem. To compute the first term in Equation 6.13 the cart-pole system is set to the initial state $\Big(x(0), \dot{x}(0), \theta_1(0), \dot{\theta}_1(0), \theta_2(0), \dot{\theta}_2(0)\Big) = (0, 0, 4.5^\circ, 0, 0, 0)$. The network is then allowed to control the system for up to 1,000 time-steps. The number of time-steps n_I that the controller keeps the cart and poles in the success domain ($x \in [-2.4\text{m}, 2.4\text{m}]$

and $\theta_1, \theta_2 \in [-36^\circ, 36^\circ]$) is counted. If the system leaves the success domain at any time prior to time-step 1,000, then the simulation stops. The second term is a measure of the stability of the system during the last 100 time-steps while under neural network control, and is expressed by the function

$$f_{stable} = \begin{cases} 0, & \text{if } n_I < 100 \\ \frac{0.75}{\sum_{i=n_I-100}^{n_I} \left(|x(i)|+|\dot{x}(i)|+|\theta_1(i)|+\left|\dot{\theta}_1(i)\right|\right)}, & \text{otherwise} \end{cases} \tag{6.14}$$

The third and fourth terms are measures of a neural controller's ability to generalize. If $n_I = 1000$ after computing the first term, then the neural controller is allowed to control the system for up to an additional 100,000 time-steps. The number of additional time-steps n_{II} that the controller keeps the cart and poles in the success domain is counted, and the simulation stops if the system leaves the success domain or $n_{II} = 100,000$. The fourth term is computed by putting the cart-pole system in 625 different initial conditions, and allowing the network to control it for up to 1,000 time-steps from each starting configuration. The variable n_S represents the number of different initial conditions from which the neural controller was able to keep the system in the success domain for 1,000 consecutive time-steps. The 625 unique initial conditions are defined by the set

$$\left(x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2\right) \in \Big\{(k_1 \cdot 4.32\text{m} - 2.16\text{m}, k_2 \cdot 2.70\text{m/s} - 1.35\text{m/s}, k_3 \cdot 7.2^\circ - 3.6^\circ, \\ k_4 \cdot 17.2^\circ\text{s}^{-1} - 8.6^\circ\text{s}^{-1}, 0^\circ, 0^\circ\text{s}^{-1}) \mid k_1, k_2, k_3, k_4 \in \{0.05, 0.25, 0.5, 0.75, 0.95\}\Big\}.$$

On every time-step of the growth process each growing network was mapped to the static network represented by its current physical configuration so that its performance could be computed by evaluating Equation 6.13. Before applying input to a network the output of each neuron was always set to zero. Before a network was permitted to control the cart and poles the dynamics of the cart-pole system were evolved for 0.2s, and the resulting sequence of 10 system states were input into the network. For both the SINOSA$_a$ and SINOSA$_b$ models, the neural network growth process lasted 600 time-units, after which the static network with the best performance (largest value of f_{pole}) was taken as the solution.

For the SINOSA$_a$ model, 51 trials were run starting from different, randomly generated initial conditions. For the SINOSA$_b$ model, 38 trials were run. For the SINOSA$_a$ model, Table 6.3 compares the performance of networks grown using collective movements to the performance of networks grown using random movements. Table 6.4 provides the same comparison for the SINOSA$_b$ model. The comparison of performance is made every 200 time-steps during the growth process. Each of the numeric values in the tables is shown with its 95% confidence interval. The values in Tables 6.3 and 6.4 were computed as follows. For each trial, and at each of the three predefined time-steps (200,400,600), two measures of the best performing network at that point in the growth process were recorded. The first measure was whether or not the network succeeded in achieving $n_{II} = 100,000$ when computing Eq. 6.13. The second measure was the value of n_S. In Tables 6.3 and 6.4 the term $Measure_{II}$ refers to the fraction of best performing networks that achieved $n_{II} = 100,000$.

Table 6.3: Performance Values on the Double Pole Balancing Problem for Networks Grown with the SINOSA$_a$ Model

Time-Step	Collective, Measure$_{II}$	Random, Measure$_{II}$	Collective, Measure$_S$	Random, Measure$_S$
200	$0.667, [0.530, 0.780]$	$0.026, [0.005, 0.135]$	372 ± 29	10 ± 5
400	$0.961, [0.868, 0.989]$	$0.053, [0.015, 0.173]$	462 ± 10	28 ± 15
600	$1.0, [0.930, 1.0]$	$0.053, [0.015, 0.173]$	478 ± 7	41 ± 17

Table 6.4: Performance Values on the Double Pole Balancing Problem for Networks Grown with the SINOSA$_b$ Model

Time-Step	Collective, Measure$_{II}$	Random, Measure$_{II}$	Collective, Measure$_S$	Random, Measure$_S$
200	$0.156, [0.077, 0.288]$	$0.0, [0.0, 0.099]$	209 ± 42	9 ± 8
400	$0.4, [0.270, 0.545]$	$0.0, [0.0, 0.099]$	322 ± 36	14 ± 8
600	$0.6, [0.455, 0.730]$	$0.0, [0.0, 0.099]$	375 ± 29	24 ± 12

The term *Measure*$_S$ refers to the average value of n_S taken over all of the best performing networks. From these results it is clear, that for both versions of the SINOSA model, the networks grown with collective movements vastly outperform those grown with randomly generated movements on both performances measures. Additionally, the networks grown with collective movements using the SINOSA$_a$ model were less computationally expensive to generate and outperformed the echo state networks presented in [68], which were optimized using a state-of-the-art form of evolutionary strategies that uses covariance matrix adaptation (CMA-ES).

6.2.4 Effects of Neighborhood Size and Topology on Performance

Four experiments were performed in order to asses the impact of the size and topology of the growth cone neighborhoods on the quality of the networks grown by the SINOSA$_a$ model with collective movements. In the first two experiments networks were grown to forecast the Mackey-Glass time-series, and the growth cone neighborhoods were changed from a von Neumann topology to a ring topology. There were 16 growth cones per neighborhood in the first experiment, and 9 growth cones per neighborhood in the second experiment. The third and fourth experiments were analogous, and involved the growth of neural controllers for the double pole balancing problem. The first, second, third, and fourth experiments consisted of 43, 49, 56, and 51 trials, respectively. The results of the first and second experiments are shown in Table 6.5. For both epochs, the networks with the lowest

Table 6.5: Mean NRMSE$_{84}$ Values on the Mackey-Glass Time-Series for Networks Grown with the SINOSA$_a$ Model Using Different Growth Cone Neighborhood Characteristics

Epoch	Ring Topology, 9 Networks	Ring Topology, 16 Networks	von Neumann Topology, 16 Networks
1	$8.56 \cdot 10^{-3} \pm 5.2 \cdot 10^{-4}$	$7.37 \cdot 10^{-3} \pm 5.1 \cdot 10^{-4}$	$5.89 \cdot 10^{-3} \pm 3.3 \cdot 10^{-4}$
2	$6.69 \cdot 10^{-3} \pm 3.6 \cdot 10^{-4}$	$5.70 \cdot 10^{-3} \pm 3.2 \cdot 10^{-4}$	$4.98 \cdot 10^{-3} \pm 3.2 \cdot 10^{-4}$

mean NRMSE$_{84}$ are those generated using a von Neumann topology and 16 growing networks. However, the mean NRMSE$_{84}$ values of those generated using a ring topology are still quite good, and using 9 growing networks instead of 16 results in a substantial reduction in the computation time of a simulation. The results of the

third and fourth experiments are shown in Tables 6.6 and 6.7. The performance values of the grown networks are relatively close regardless of the neighborhood size or topology. For the double pole balancing problem, it is certainly feasible to use only 9 growing networks in order to reduce the computation time of the growth process. Based on these results, the impact of neighborhood size and topology on performance appears to be problem dependent. However, for some problems the discrepancy may be small enough to justify using smaller neighborhoods that employ a ring topology in order to benefit from the reduction in computational expense.

Table 6.6: Values of Measure$_{II}$ on the Double Pole Balancing Problem for Networks Grown with the SINOSA$_a$ Model Using Different Growth Cone Neighborhood Characteristics

Time-Step	Ring Topology, 9 Networks	Ring Topology, 16 Networks	von Neumann Topology, 16 Networks
200	0.490, [0.359, 0.623]	0.719, [0.592, 0.819]	0.667, [0.530, 0.780]
400	0.863, [0.743, 0.932]	1.0, [0.937, 1.0]	0.961, [0.868, 0.989]
600	0.902, [0.790, 0.957]	1.0, [0.937, 1.0]	1.0, [0.930, 1.0]

Table 6.7: Values of Measure$_S$ on the Double Pole Balancing Problem for Networks Grown with the SINOSA$_a$ Model Using Different Growth Cone Neighborhood Characteristics

Time-Step	Ring Topology, 9 Networks	Ring Topology, 16 Networks	von Neumann Topology, 16 Networks
200	324 ± 40	382 ± 22	372 ± 29
400	439 ± 16	463 ± 9	462 ± 10
600	462 ± 11	478 ± 5	478 ± 7

6.2.5 Effects of Growth Cone Size and System Scale on Performance

Three experiments were performed in order to asses the impact of the size of the growth cones, and the physical scale of the system, on the quality of the networks grown by the SINOSA$_b$ model with collective movements. Each experiment involved growing neural controllers for the double pole balancing problem. In the first experiment the radius of the growth cones was increased by a factor of 4.0, from 0.05 to 0.2 distance-units, while the radius of the cells and the distance between adjacent cells remained the same as in the experiments discussed in Section 6.2.3. In the second and third experiments the overall system scale was reduced by shrinking the cell radius and spacing between adjacent cells by a factor of 0.25, to 0.5 and 2.0 distance-units, respectively. Collisions between growth cones were present in the second experiment, and absent in the third experiment. This was done in order to study the degree to which growth cones colliding with one another hinders the growth of effective neural controllers. In both the second and third experiments collisions between growth cones and cells were present. The first, second, and third experiments consisted of 52, 44, and 53 trials, respectively.

The results of the first experiment are shown in Table 6.8. Increasing the size of the growth cones has a substantial impact on the ability of the growth process to generate good quality neural controllers. On time-step 600, the size increase resulted in the success rate dropping by 68%, and the mean number of successes being reduced by 47%. The results of the second and third experiments are shown in Table 6.9. Decreasing the physical dimensions of the system caused an increase in

Table 6.8: Performance Values on the Double Pole Balancing Problem for Networks Grown with the SINOSA$_b$ Model Using Different Growth Cone Sizes

Time-Step	Radius = 0.2, Measure$_{II}$	Radius = 0.05, Measure$_{II}$	Rad. = 0.2, Measure$_S$	Rad. = 0.05, Measure$_S$
200	$0.019, [0.003, 0.101]$	$0.16, [0.077, 0.288]$	80 ± 25	209 ± 42
400	$0.058, [0.020, 0.156]$	$0.4, [0.270, 0.545]$	147 ± 30	322 ± 36
600	$0.192, [0.108, 0.319]$	$0.6, [0.455, 0.730]$	199 ± 31	375 ± 29

Table 6.9: Performance Values on the Double Pole Balancing Problem for Networks Grown with the SINOSA$_b$ Model Using a Smaller System Scale

Time-Step	Collisions, Measure$_{II}$	No Collisions, Measure$_{II}$	Collisions, Measure$_S$	No Coll., Measure$_S$
200	$0.0, [0.0, 0.080]$	$0.189, [0.106, 0.314]$	27 ± 13	237 ± 38
400	$0.0, [0.0, 0.080]$	$0.566, [0.433, 0.691]$	61 ± 21	340 ± 29
600	$0.023, [0.004, 0.118]$	$0.830, [0.708, 0.908]$	111 ± 29	394 ± 24

the frequency of collisions between growth cones. When collisions between growth cones were present, there was a dramatic reduction in the average performance of the grown networks. However, when these collisions were absent, the average performance improved substantially. The results indicate that an increase in growth cone size or a reduction in system scale, which increases interference of the growth cones' movements, leads to a decrease in the average performance of the grown networks.

6.2.6 Self-Assembly of Small Optimal Neural Networks

All of the previously mentioned experiments in Section 6.2 involve the growth of echo state networks. The SINOSA$_a$ model was also used to grow smaller, non-echo

state networks as controllers for the double pole balancing problem. These networks consisted of 3 input neurons, 1 output neuron, and 7 neurons in the hidden layer. The growth cones were permitted to establish connections from the input neurons to the hidden layer, from the input neurons to the output neuron, from the hidden layer back to the hidden layer, from the hidden layer to the output neuron, and from the output neuron back to itself. The primary difference between these smaller networks, and echo state networks used as controllers, is that the smaller networks have only 7 neurons in the hidden layer, which would be a very small number for an ESN on a challenging problem, and their topology is completely unrestricted, whereas an ESN used as a neural controller would not have output feedback connections to the reservoir or to the output neuron, and typically would not have direct connections between the input and output neurons. Each neuron in the network had a neighbor neuron set that consisted of the entire hidden layer and the output neuron. For each one of the simultaneously growing networks, each neuron contributed one positively-weighted growth cone ($b = 30.0$ in Eq. 5.1), and one negatively-weighted growth cone ($b = -30.0$), per neighbor neuron. Each of these positive-negative growth cone pairs had the same, single target neuron. All of the other parameters and methods for growing and testing the networks were the same as in Section 6.2.3.

The results of growing smaller, non-echo state networks for the double pole balancing problem are shown in Table 6.10 in the columns labelled *General Search*. When the growth process was governed by random movements, none of the 40 trials run produced a neural controller that had performance values greater than zero. When the growth process was governed by collective movements, the results were

much better. However, the mean performance values computed over the 54 trials run, are much lower than those of the echo state networks grown under the same conditions (Sect. 6.2.3).

Table 6.10: Performance Values on the Double Pole Balancing Problem for Networks Grown with the SINOSA$_a$ Model Using General and Local Search Processes

Time-Step	General Search, Measure$_{II}$	Local Search, Measure$_{II}$	General Search, Measure$_S$	Local S., Measure$_S$
200	$0.037, [0.010, 0.125]$	$1.0, [0.929, 1.0]$	25 ± 20	463 ± 17
400	$0.222, [0.132, 0.349]$	$1.0, [0.929, 1.0]$	126 ± 47	498 ± 9
600	$0.296, [0.191, 0.428]$	$1.0, [0.929, 1.0]$	190 ± 50	505 ± 9

Of the 54 smaller networks grown using collective movements, about 30% of them exhibited performances that were significantly better than the mean performance level. There were a number of easily discernible commonalities among these higher performing smaller networks. Most prominent was the absence of connections from the hidden layer to the output neuron. This topological characteristic amounts to removing the hidden layer from the network, since without connections to the output neuron it has no impact on the network's output. In other words, these particular networks were equivalent to single layer networks. It was hypothesized that removing the hidden layer was an effective strategy employed by the growth process because the activities of the hidden layer neurons were too large in magnitude. In response to this, the maximum magnitude of the weight fields carried by the growth cones responsible for establishing connections to neurons in the hidden layer was set to 1.0 ($b = \pm 1.0$). Additionally, it was recognized that in good performing networks connections to the output neuron tended to have weights

that fell within particular ranges. These ranges depended on the emitting neuron, and prompted the following modifications to the maximum magnitudes of the weight fields carried by the growth cones. Let the ordered pair $(b_+, b_-)_{neuron}$ represent the most positive and most negative values of the weight fields carried by the growth cones from the specified neuron. The extreme values of the weight fields were changed to: $(0.0, -10.0)_{input1}$, $(0.0, -30.0)_{input2}$, $(20.0, 0.0)_{input3}$, $(10.0, -10.0)_{hidden}$, and $(0.0, -5.0)_{output}$. Here, *input1* is the input neuron that transmits the position of the cart on the track, *input2* is the input neuron that transmits the angular position of the large pole, *input3* is the input neuron that transmits the angular position of the small pole, *hidden* is any neuron in the hidden layer, and *output* is the output neuron. Making these changes to the weight fields is equivalent to restricting the growth process to search a smaller, more specific region of the weight space.

The results of growing smaller, non-echo state networks with more restricted weight values are shown in Table 6.10 in the columns labelled *Local Search*. The changes made to the growth cones' weight fields resulted in the growth of much better performing networks. After 600 time-steps these networks even have a slightly better mean performance than the echo state networks grown using collective movements. At least part of this substantial improvement in the average performance of the grown networks was due to their increased ability to utilize the hidden layer.

6.3 Discussion

Two versions of the SINOSA model were introduced in this chapter. The SINOSA$_a$ model does not incorporate collisions or forces, and the collective movements of the growth cones are governed by the canonical PSO equation. In the SINOSA$_b$ model, growth cones and cells are able to collide with one another, and the collective movements of the growth cones are generated by interpreting the canonical PSO equation as a force and coupling it with a viscous drag force.

Three different computational problems were used to test the ability of the SINOSA model to grow optimal neural networks. On all three problems, the echo state networks grown using the SINOSA$_a$ model with collective growth cone movements substantially outperformed those generated in the control cases, which involved random growth/search. The echo state networks grown using the SINOSA$_a$ model for the Mackey-Glass time-series forecasting problem outperformed ESNs of a similar size that were hand-designed by an expert. Likewise, the ESNs grown for the double pole balancing problem outperformed comparable ESNs that were optimized using a state-of-the-art form of evolutionary strategies. The smaller, non-echo state networks grown by the SINOSA$_a$ model for the double pole balancing problem performed equally well. These results indicate that the methodology incorporated by the SINOSA$_a$ model is an effective means of optimizing neural network weights and topologies compared to other, more well-established techniques. The results of experiments involving different growth cone neighborhood sizes and topologies suggest that, for many computational problems, it may be possible to use the SINOSA$_a$

model to grow high-quality solutions using a relatively small number of simultaneously growing, interacting networks.

On all three computational problems, the echo state networks grown using the SINOSA$_b$ model with collective growth cone movements outperformed those generated by random growth cone movements. This suggests that a real-world, physical implementation of the SINOSA$_b$ model, or some of its methods, may be useful for the growth and optimization of tangible networks. However, the results of additional experiments involving the SINOSA$_b$ model indicate that the size of the growth cones and system scale have a significant impact on the performance of the grown networks. These characteristics would undoubtedly require attention in a material implementation of the model.

Chapter 7

Discussion

This chapter concludes this dissertation by first giving a summary of the research that was done, including the motivation behind and development of the SINSA and SINOSA models, and an overview of the experiments performed using these models along with the results of the experiments. Next, the practical and theoretical contributions of the work presented herein are discussed. The chapter concludes with an examination of the current limitations of the models, and how these limitations might be addressed, and the models enhanced, in future work.

7.1 Summary

During any non-trivial neural network growth process the many "agents" involved are subjected to different influences that arise in their local environments. A local environment is highly dynamic, especially if the agent is moving, making it difficult to predict an agent's behavior. Furthermore, large numbers of agents are involved in the growth process and these agents must exhibit different behaviors in order to generate the desired neural network. This tends to make it very difficult to derive a parsimonious set of rules and parameters that will result in the growth of a particular target network. However, it was hypothesized that incorporating swarm intelligence in the form of collective movements into the network self-assembly pro-

cess would allow agents to act collectively in a way that overcomes many of the challenges of a dynamic and inhomogeneous environment. In particular, agents could utilize information from their neighbors to guide their own dynamics.

The swarm intelligent network self-assembly model (SINSA) was inspired by this hypothesis. It is a developmental model that incorporates the collective growth and interactions of discrete neurons in a continuous three-dimensional space. Unlike most artificial neural network models, the networks grown using the SINSA model are characterized by a geometric relationship among their neurons in addition to a topological one. Furthermore, the grown networks are more like natural networks in that they have non-uniform densities of cells, less regular topologies, redundant connections between neurons, patterns of connectivity that are statistical in nature rather than being exact, and non-periodic boundary conditions. For example, on average a thalamic cell in the grown network based on the somatosensory cortex model makes 1.9 connections with any target cortical cell, and on average an afferent cell in the grown network based on the visual cortex model makes 5.6 connections with any target excitatory cell. Such characteristics could be of interest to those studying the growth and properties of real-world networks such as biological neural networks or other types of networks that self-assemble from physical components and exhibit similar features.

Increasing the amount of detail in a developmental model of neural growth tends to make it more difficult to control the dynamics of growth and the characteristics of the networks. This lack of controllability is a significant limitation that is common among past models that incorporate continuous neural network growth.

The SINSA model exhibits greatly improved controllability due largely to the incorporation of local forces among the components of growing neurons. By creating appropriate rule sets for two relatively large target network structures, it was demonstrated that these local interactions can produce collective movements among the components that ultimately result in effective self-assembly of fairly complex neural networks. These two networks would, at best, be very difficult to generate using most past models that incorporate a continuous neural growth process (e.g., those that primarily use gradient following). The self-organizing maps that emerged on the grown networks during subsequent learning were shown to exhibit computational properties similar to their archetypes, indicating the SINSA model's ability to grow three-dimensional networks with topologies that are statistical realizations of more abstract neural templates.

A series of computational experiments were performed in which the network growing capabilities of a version of the SINSA that incorporates collective movements were compared to those of a version that does not. The results indicate that incorporating collective movements into the SINSA model increases its robustness. Based on these experiments it is evident that swarming growth cones are able to counterbalance significant increases in the degree of rule set variability. The resulting gain in robustness is one of the primary reasons the SINSA model offers increased controllability with respect to previous models. This is evident in the first of the robustness experiments, where the model's parameters were independent of the degree of variability, and yet good networks continued to be grown even for large degrees of rule set variability. In two additional experiments the amount of

cohesion and velocity alignment among the swarming growth cones were allowed to vary as functions of the degree of rule set variability, yielding substantial additional improvements in the model's ability to grow high quality networks over a range of rule set variabilities. These improvements stemmed from the growth cones' ability to collectively guide one another's trajectories; for example, they were able to establish the proper density among the growing axons and correct abnormal courses of growth. A local growth force also played a role by helping to guide growing axons towards their target cells without obfuscating the self-assembly process.

The results of the robustness experiments demonstrate how incorporating collective movements into the SINSA model significantly improves its robustness by making network growth less sensitive to the rule set. Furthermore, they show that the improvements in robustness are not highly dependent on the values of the forces' parameters. The resulting increase in robustness, combined with the intuitiveness that the collective movements give to the agents' dynamics, makes it easier to find a set of rules and parameters that cause the SINSA model to grow networks with desired characteristics. In other words, collective movements improve the controllability of the model. It is very likely that this is one of the central reasons why the SINSA model has been successful at growing pre-specified networks that are much larger than those grown by past models. It was also found that collective movements tend to reduce the size of the rule sets, although this has not been quantified in any way. Experience suggests that this phenomenon is a consequence of the fact that in writing a rule set to grow a particular network it is easy to take advantage of the collective dynamics of the agents to guide the growth process, which reduces the

need to explicitly encode trajectory information in the rule set.

Discontinuous actions, such as cell division, axon emission and branching, and changes in a cell's local growth force, have a significant impact on network development. However, when using a developmental model to grow complex networks it is often difficult to predict and control when and where an agent will execute such actions. This is especially prevalent when their execution occurs in response to an agent's environment-dependent interactions (the agent's gradual acquisition of a substance diffusing through the environment, its contact with other agents, etc.) because such interactions are typically very hard to predict and control. The incorporation of the rule set, and its control over discontinuous actions, helps the SINSA model to overcome this challenge by making an agent's "decision" to execute such an action relatively environment-independent. More specifically, none of an agent's state variables (see Table 3.1) depend on its interactions with the environment, and hence neither do the truth values of the rule predicates. Another way in which the rule set enhances controllability is via the rule-based force which allows growth dynamics to be tailored to the specific network being grown by granting each agent a certain degree of autonomy from the other agents and its environment. Furthermore, this force eliminates the need for any long range guidance forces, thus allowing the model to adhere to the local-interactions-only criterion. Experience indicates that this is important because the use of nonlocal forces in the computational experiments described here tends to create a more convoluted environment due to the increase in the number of forces to which an agent is simultaneously subjected. This in turn reduces the model's controllability and the intuitiveness of the agents' dynamics.

Particle swarm optimization has proven to be very successful at optimization in continuous domains, and in particular, at optimizing neural network weights. However, a neural network's performance typically depends on both its weights, and on its topology, and hence it is often desirable to optimize a network's topology in addition to its weights. Unfortunately, the discrete nature of the topology space has made it challenging to successfully adapt PSO to this optimization task, and thus there are very few versions of PSO that optimize both network weights and topologies. Rather than adapting the PSO method itself, the SINOSA model incorporates a more integrated representation of a network's weights and topology. The objects in this representation are cells (neurons), axons, and growth cones. The the cells have fixed positions, but the growth cones are able to guide the cells' axons through a continuous, three-dimensional space. A function is defined that maps the positions of the growth cones with respect to their target neurons, to a corresponding static network with fixed connections and weights. Network growth is produced by using this function to interpret the positions of the growth cones as they guide their axons through the 3D space. As a result of this integrated representation, it is possible to incorporate the simplest, canonical form of PSO into the model for the purpose of simultaneously optimizing network weights and topologies. In effect, the SINOSA model treats the network self-assembly process as an optimization or search process, in which the simultaneous growth of multiple neural networks is driven by their interactions with one another and with problem related network input.

The ability of the SINOSA model to optimize neural networks for computational tasks was tested using three different problems. These computational prob-

lems were from the domains of trajectory generation, time-series forecasting, and control. Two different versions of the SINOSA model were used to generate solution networks for each problem. The version denoted SINOSA$_a$ does not incorporate forces or collisions, and the growth cones carry weight fields. On the other hand, the SINOSA$_b$ model does incorporate forces and collisions, and the growth cones carry fixed weight values. For each of the computational problems, and both versions of the SINOSA model, the echo state networks grown using collective movements generated via PSO outperformed those grown using randomly generated movements, and in most circumstances the performance gap was very large. Specifically, compared to the networks grown with random movements, those grown using the SINOSA$_a$ model with collective movements performed 10 orders of magnitude better on the figure-eight trajectory generation problem, 3 times better on the Mickey-Glass time-series forecasting problem, and 19 times better and 12 times better on two generalization measures of the double pole balancing problem. For the SINOSA$_b$ model, the networks grown with collective movements performed 2 orders of magnitude better on the figure-eight trajectory problem, 1.3 times better on the Mackey-Glass time-series problem, and 5 to 16 times better on the double pole balancing problem. Because the canonical PSO algorithm is a computationally inexpensive means of generating collective movements, and the evaluation of network performance is often a bottleneck, the large improvements in network performance gained over random search comes at very little additional computational cost. For example, on the figure-eight and Mackey-Glass problems, when collective movements are used with the SINOSA$_a$ model the self-assembly process takes less than 1% longer to complete than when

random movements are used.

Three additional experiments were performed. The first experiment indicated that the average performance of networks grown to forecast the Mackey-Glass time-series is best when growth cone neighborhoods consist of 16 growth cones connected in a von Neumann topology. However, growth processes that incorporate neighborhoods that consist of 9 or 16 growth cones connected in a ring topology also produce good performing networks. The average performance of networks grown for the double pole balancing problem does not depend significantly on whether 9 or 16 growth cones constitute a neighborhood, or whether a ring or von Neumann topology is used. The second experiment demonstrated that when the components of growing networks are able to collide, this can interfere with the self-assembly process, resulting in poorer performing networks. The size of the growth cones with respect to the overall scale of the system is particularly important in determining the degree of interference caused by collisions. The third experiment suggested that on control problems, such as the double pole balancing problem, where a neural network can be trained using reinforcement learning to act as a direct controller, the SINOSA$_a$ model can be used to grow small neural controllers with performances that are at least as good as those of larger echo state networks trained using much more well established optimization techniques, such as evolutionary strategies.

Comparison with the control cases that involve random search provides a base level of support for the optimization capabilities of the SINOSA model. Evidence of the effectiveness of the model at optimizing networks, beyond that obtained by comparison with random search, can be found by comparing the results of Chapter 6

with studies that involve different methods of optimizing networks for the Mackey-Glass time-series forecasting problem and the double pole balancing problem. In [66], echo state networks with 400 neuron reservoirs were optimized to forecast for Mackey-Glass time-series ($\tau = 17.0$) using a sequence of 2000 data points to train the output weights. The best performing of these networks, which was hand-designed by an expert, had an NRMSE_{84} of $2.8 \cdot 10^{-4}$. In [67], using the same parameter values for the Mackey-Glass time-series and for training, an echo state network with a 1000 neuron reservoir was hand-designed by an expert that had an NRMSE_{84} of $6.3 \cdot 10^{-5}$. Adhering to the methods described in Section 6.2.2, the SINOSA_a model was used to grow echo state networks with 400 neurons in their reservoirs to forecast the Mackey-Glass time-series. The grown networks produced an average NRMSE_{84} of $3.86 \cdot 10^{-5}$, and the best of these networks had an NRMSE_{84} of $2.73 \cdot 10^{-5}$. On average, the grown networks outperformed the best hand-designed 400 neuron ESN by about an order of magnitude, and they also performed better than the 1000 neuron ESN. These results provide strong evidence of the effectiveness of using the SINOSA model to grow echo state networks, as opposed to the standard approach of optimizing them through trial and error.

In addition to smaller networks and improved performance, there is another benefit of using the SINOSA model to optimize echo state networks. When an echo state network is designed by hand it is often necessary to run a large amount of problem specific data through the network in order to make it generate accurate output. This initial sequence of lead-in data allows the network to "tune-in" to the time- series. In [66] and [67], the amount of lead-in data needed to get good results

was on the order of 1000 data points. In contrast, the grown networks were able to produce accurate output after a lead-in of only 100 data points. This property could be very beneficial in circumstances where problem specific data is limited.

Another study that lends itself to comparison is presented in [68]. In this case echo state networks were optimized as controllers for the double pole balancing problem via a state-of-the-art form of evolutionary strategies that uses covariance matrix adaptation (CMA-ES). In this study CMA-ES was used to optimize the output weights, and the spectral radius of the reservoir $\rho(\mathbf{W}_r)$ (Eq. 2.8). The experiments discussed in Section 6.2.3, in which the SINOSA model was used to grow ESNs as controllers for the double pole balancing problem, adhered to the same experimental setup and methods used in [68], except that the grown neural controllers received only 10 inputs from the cart-pole system prior to beginning control instead of 20. Because evaluating the fitness/performance of the networks during the optimization process is the computational bottleneck, the number of such evaluations during an optimization run is a good measure of the overall computational cost of the process. On average it required 19,796 evaluations for the CMA-ES approach to find a neural controller capable of successfully controlling the cart for at least 200 out of the 625 initial configurations (the average was 224), and of these networks 91.4% of them were able to successfully control the cart for the additional 100,000 time-steps when it was started in the standard initial configuration. These results are very good with respect to past work on the double pole balancing problem. The SINOSA$_a$ model was able to grow much better performing neural controllers, and at much less computational expense. After only 9600 evaluations, on average the

best performing grown networks were able to successfully control the cart for 478 of the initial configurations, and of these networks 100% of them were able to successfully control the cart for the additional 100,000 time-steps. These results are also interesting in that they represent one of the few instances where echo state networks have been successfully trained as neural controllers using reinforcement learning.

The SINOSA$_b$ model incorporates forces and collisions among the cells and growth cones, and the growth cones carry fixed weight values, which would most likely be easier to implement in an actual physical system than weight fields. These more realistic components were implemented with the intent of moving the self-assembly process generated by the model closer to physical reality. The SINOSA$_b$ model was used to grow networks that perform substantially better than those generated in the control cases involving random growth cone movements. These results suggest that it might be useful to incorporate the methods of this model into a real-world system that implements the self-assembly of optimal network structures.

7.2 Contributions

In this section I state, and elaborate upon, the main contributions of the work presented in this dissertation.

- A novel framework has been developed that supports and facilitates swarm intelligent network self-assembly. In this framework, network growth occurs in a continuous, three-dimensional space, and is generated by the movements of distinct network components. Swarm intelligence is incorporated into the

model by having the components of growing networks exhibit collective movements. The collective movements arise from simple, local interactions between the components. Network growth is governed by forces and conditional rules, which are two of the most common mechanisms for generating growth in developmental models. The forces are largely responsible for generating the movements of the network components, while the conditional rules dictate other types of "discrete" actions that the components can take. The framework incorporates a variety of features and capabilities not commonly found in past developmental models of network growth in a physical space. These include the incorporation of collective movements among network components, network growth in a three-dimensional space, instead of a two-dimensional space, cell migration and division, interactions between growing axons, and the ability to grow relatively large recurrent networks based on target network models that have precisely specified topologies. The framework can readily be extended to include new types of forces, actions, and components relevant to different types of network growth. This flexibility would be very useful in adapting the model to study phenomena related to development in the nervous system, or of other classes of network structures. The model is implemented in a software-based simulator that includes a 3D graphics component for visualization, and allows users to specify rule sets in plain text files, which are compiled by the simulator at runtime.

- The simulator is capable of growing networks that are substantially larger,

and have much more specific patterns of connectivity, than networks grown using past models that incorporate continuous network growth. This capacity was illustrated by using the simulator to grow two neural networks based on previous models of networks in the somatosensory cortex and visual cortex. The grown networks contained far more neurons and connections than most of the networks grown with past models. Also, the target cortical networks have patterns of connectivity that are specified to much greater levels of detail than the topologies of most target networks tackled in past work on network development. The accuracy of the topologies of the grown networks was confirmed by quantifying the statistical properties of the connectivity, and by demonstrating that the grown networks are capable of learning topographic and feature maps that are qualitatively similar to those exhibited by the target networks. These results support the conclusion that swarm intelligence in the form of collective movements can be used to facilitate network self-assembly.

- A series of computational experiments were performed to test the hypothesis that *incorporating swarm intelligence in the form of collective movements into the network self-assembly process increases its robustness with respect to variability in the rule set.* The results of the experiments support this hypothesis by demonstrating that the quality of grown networks deteriorates far less under perturbations of the rule set when network components are capable of exhibiting collective movements. This improvement stems from the growth cones' ability to collectively guide one another's trajectories using local com-

munication. Incorporating collective movements among the components of growing networks improves robustness, and does so in a way that is not highly dependent on the specific parameter values of the forces. It also enhances the intuitiveness of the components' dynamics. These are two of the primary reasons why collective movements increases ones ability to control the geometric and topological characteristics of the network that emerges from a growth process.

- The model of network self-assembly presented in this dissertation has been modified so that, given a computational problem, it can be used to grow networks that are effective solutions. This modification represents an extension of the classic self-assembly problem, which entails the design of local control mechanisms that enable a set of components to self-organize into a *given* target structure. Specifically, this problem formulation is extended to include the self-assembly of network structures with growth driven by optimality criteria defined in terms of the quality or performance of the emerging structures, as opposed to growth directed towards assembling a pre-specified target structure. To accomplish this, the modified model incorporates an elegant form of particle swarm optimization (PSO) to govern the growth process. In the vast majority of past work on PSO, the particles are embedded in a high dimensional abstract space, such as the domain of a function, they are the fundamental class of "objects" in the space, and the position of a particle represents a solution or solution component to the problem being solved. In

contrast, in the modified model, growth cones (particles) are embedded in a continuous, three-dimensional space that is intended to model physical space, *growing networks* are the fundamental class of objects in the space, and the position of a growth cone is only meaningful as a solution component when interpreted in the context of the growing network to which the growth cone belongs.

- The aforementioned modified model constitutes a new, and effective means of optimizing the weights and topologies of neural networks. At the heart of this unique approach lies a novel application of particle swarm optimization, in which the components of multiple, simultaneously growing networks exhibit PSO-like interactions. The effectiveness of this approach for optimizing neural networks was demonstrated on a number of challenging benchmark problems from the domains of trajectory generation, time-series forecasting, and control. The modified model was used to grow echo state networks, and smaller, non-echo state networks, both of which performed substantially better on these problems than networks optimized via random search. The grown networks also outperformed the echo state networks presented in two different past studies, one in which the networks were hand-designed by an expert, and the other in which they were optimized using a state-of-the-art form of evolutionary strategies (CMA-ES).

7.3 Limitations and Future Directions

The work described in this dissertation is limited in that it does not try to model specific biological data. Thus, an important issue for future investigation is whether the SINSA approach has any implications for neuroscience. In neuroscience, there is currently an intense experimental effort underway to better understand how complex interactions between genetic and activity-dependent factors determine the wiring of neural circuitry during an organism's developmental period (Grove and Fukuchi-Shimogori 2003; Lopez-Bendito and Molnar 2003; Spitzer 2006). While the vast majority of models in computational neuroscience do not involve network self-assembly or connection growth, there has been substantial recent interest in modeling neural development. Much of this work has focused on the formation of topographically-structured connections in specific brain regions, and is based upon axon growth that is guided by growth cones that are sensitive to local biomolecular gradients (Goodhill et al. 2004; Goodhill and Xu 2005; Hentschel and van Ooyen 1999; Honda 2003). Growth cones "steer" the direction in which axons grow to their target termination locations.

These past models of neurobiological development are like the work presented here in that they explicitly incorporate geometric relations (not just network topology) and they simulate the growth of axons through physical space. However, unlike the work described here they are often but not always limited to two-dimensional space, typically do not incorporate cell migration and division, are generally applied to relatively small networks, are usually concerned with feedforward networks,

and do not consider axon-axon interactions during network assembly (for exceptions to the latter, see Goodhill et al. 2004; Yates et al. 2004). To my knowledge, no past models of neurogenesis done in computational neuroscience have explicitly recognized the relationship of work in that area to concepts that have emerged from swarm intelligence research on collective movements and self-assembly over the last several years (Grushin and Reggia 2008; Reynolds 1987; Rodriguez and Reggia 2004).

The SINSA model thus has a great deal of potential as a tool for future neuroscience studies of biological network growth, in part because it grows networks that incorporate geometry as well as topology. The flexibility of the model allows for straightforward incorporation of additional biological detail such as dendritic trees, more biologically realistic parameter values (e.g., making viscous drag negligible), and the diffusion of chemical messengers like neural growth factor. Moreover, the continuous nature of the growth process and the incorporation of neural activity dynamics also makes it well suited for studying the role of network activity during development (van Ooyen 1994). Finally, it would also be very valuable to construct a precise mapping between neuro-chemical mechanisms and model rules/equations to stimulate further theoretical advances.

Another important area for future research is making the rule sets for the SINSA model more parsimonious and easier to generate. One way to accomplish this is through the automatic generation of the needed control rules when given a target network, rather than their manual creation. Although this has been achieved for self-assembly of some structures (e.g., Grushin and Reggia 2006; Grushin and Reggia

2008), to my knowledge it has not previously been studied for network structures like those considered here. In this scenario a high-level description of the desired network would be automatically translated into a set of rules that the model would implement to grow the specified network. For example, one might specify the size and number of neural layers to grow and the patterns of connectivity to be established between them. A different approach is to enhance the language in which the rules are written. For example, incorporating variables and constructs such loops into the language would substantially reduce the amount of code necessary to express a given rule set. Also, allowing more general logic formulas in the *Tags* predicate field of rules, such as those in conjunctive normal form, would allow the same information to be expressed with far fewer rules.

At present, the SINOSA model also has a number of limitations. For one, it only permits a fixed number of neurons to be placed in the three-dimensional physical space at the beginning of a growth process, and that number does not change while the networks are growing. In a sense, the model does allow the number of neurons in a grown static network to be less than the number of neurons in the physical space, but this requires that the connections to and from the "removed" neurons be such that these neurons have no effect on the output of the static network. It would be advantageous for the number of neurons in the physical space to be able to increase or decrease depending on the computational requirements of the problem being solved. While it is not entirely clear what mechanism would dictate the addition or removal of neurons, or exactly how this mechanism would be integrated into the self-assembly process, inspiration could likely be drawn from the fairly large

number of past studies that involve dynamically modifying the number of nodes in a neural network.

The parameters of the SINOSA model were selected based on the computational problems addressed in this dissertation. Some of these decisions were based on past work on particle swarm optimization. While other parameters were selected based on informal experimentation, such as the use of 9 or 16 simultaneously growing networks, which is a much smaller number than one would presume to be effective based on the PSO literature. The SINOSA model has also introduced parameters that have not been addressed in past work on PSO or self-assembly. A few examples are: the degree of linearity and physical extent of the weight fields; the relationship between the number of neighbor neurons that a particular neuron has, and the number of growth cones that the neuron contributes to each growing network; the appropriate number of target neurons, particularly for growth cones that carry fixed weights; and, for growth cones that carry fixed weights, the appropriateness of the growth cones being large in number and carrying weight values that are small in magnitude, or being small in number and carrying weight values that are large in magnitude. Further studies are needed to determine to what extent these and other parameters are problem dependent, and what values work well on a wide variety of different problems.

By incorporating components such as forces and collisions, the SINOSA$_b$ model is intended to explore the feasibility of implementing swarm intelligent network optimization through self-assembly in an actual physical environment. Randomness and uncertainty are ever-present aspects of real-world systems. Thus, in the interest of

increasing the fidelity of the SINOSA$_b$ model, it would be useful to incorporate such things as a random component in the PSO force, or uncertainty in the best positions of the growth cones. There are additional practical considerations. For example, in a real-world system, how would a growth cone remember its best position, and how would it keep track of its neighbors and access their best positions? With a particular physical implementation in mind, one could relatively easily incorporate these types of details into the framework of the model.

Since its inception, the canonical form of particle swarm optimization has undergone a vast array of enhancements. These include methods for dynamically adapting the constriction coefficient χ or the acceleration coefficients a_p and a_n, growing and hierarchical particle neighborhoods, advanced information sharing strategies among particles, and hybrid algorithms that combine PSO and evolutionary computation, among others. The SINOSA model implements the canonical form of PSO to drive the optimization process. This means that many of these enhancements can be incorporated into the SINOSA model without having to alter its fundamental constructs. Future renditions of the model could be tailored to specific problems by including these types of modifications.

One of the benefits of using the canonical PSO algorithm is that it specifies a relatively simple, but effective form of communication for use among the growth cones. Because this communication is not very computationally expensive, computing the performance of the networks on each time-step of the growth process tends to be the bottleneck. This means that a substantial reduction in the computation time of a growth process could be achieved by having the performances of any given

growing network computed by a distinct processor or CPU core. Future implementations of the SINOSA model could take advantage of parallel processing in this manner to reduce computation time by about an order of magnitude.

The SINOSA model implements a novel technique for growing optimal networks. In essence, the model turns the network self-assembly process into an optimization process by having multiple network structures grow simultaneously in the same space, which allows the networks to interact with one another in a manner based on the canonical PSO equation. Future research could explore the applicability of this methodology to the optimization of other types of structures, in addition to networks. In general, this extension would entail representing the solutions to a problem as structures in a single Euclidean space, and then defining a mapping between the current configuration of a structure and its quality as a solution to the problem. Once this representation and mapping were defined, the canonical PSO algorithm could be employed to generate the self-assembly/search process.

The research presented in this dissertation demonstrates some of the benefits that swarm intelligence in the form of collective movements can bring to the self-assembly of networks, and how the SINSA and SINOSA models incorporate these benefits to improve the modeling of neural network growth and the optimization of networks for computational problems. Using parsimonious sets of rules the SINSA model can be used to grow large, three-dimensional, recurrent networks with fairly complex patterns of connectivity that are statistical realizations of rigorously specified topologies. The SINOSA model is capable of optimizing both neural network weights and topologies to solve challenging computational problems. Future research

will focus on applying these models and extending their methodologies to practical modeling, engineering, and optimization tasks.

Appendix A

Designing Rule Sets

The general strategy followed in writing rule sets is to first grow the layers of cells through specified cell divisions, and then to grow the connections between/within these layers. This approach is illustrated by giving a top-level overview of the rules used to grow the different parts of the somatosensory cortex network. These rules are listed in Appendix C. The first and second subsets of rules (rules 1 through 6 and 7 through 12) are used to grow the thalamic and cortical layers of cells, respectively. These two groups of rules work in the same way. They each specify that the initial "seed" cells (cells #3 and #4 for the thalamic layer and cells #1 and #2 for the cortical layer) should each divide with the specified orientations. In turn, the resulting child cells divide, as do their children, and so on. The total number of cells in each layer is dictated by the values that these rules specify for the *life* variables of the child cells (e.g., "8" in rules 3 and 4), as well as the value of each "seed" cell's *life* variable. The particular orientations of the cell divisions were chosen so that the grown layers have fairly uniform densities and are roughly symmetrical. The remaining rules specify axon growth, both to establish thalamocortical connections (rules 13-43, 57, and 58) and intracortical connections (rules 44-58). To establish thalamocortical connections, once the layers of cells are finished growing, thalamic cells are selected at random on regular intervals and given

an “E50” tag. When a thalamic cell receives this tag rules 14 through 35 cause it to emit five axons. A short period of time after being emitted the DivCone commands within this group of rules cause four of these axons to branch twice and the fifth axon to branch four times. The growth cones guiding these axon branches begin to move towards the cortical layer of cells based on their execution of the SetRuleForce commands contained within the aforementioned group of rules and within rules 38 and 41. Rule 13 ensures that each thalamic cell emits axons only once. Rules 36, 37, 39 and 40 induce additional axonal branching, the amount of which is controlled by rules 42 and 43. Specifically, the numerical portion of the tag “T2” in rule 43 may be increased or decreased to provide more or less branching respectively. These rules provide an appropriate amount of branching for each thalamic neuron to connect to its closest 30 cortical neurons. Similarly, to establish intracortical connections, a cortical cell is given an “F50” tag and rules 45 through 56 cause it to emit six axons. The SetRuleForce commands contained within this group of rules cause the growth cones to guide their axons towards the emitting cell’s six nearest cortical neighbors. Rule 44 ensures that each cortical cell emits axons only once. Rules 57 and 58 cause both the cortical and thalamic cells to induce an attractive neural growth force, which helps guide the growth cones towards their target cells.

Appendix B

Parameters and Rule Sets for Figures 3.3 and 3.4

Table B.1 summarizes the values of the parameters used in the simulations of cell layer growth and axon growth shown in Figures 3.3 and 3.4, respectively. The rules used in these simulations are listed below. The rules are shown as they appear in the ASCII text files that are read, parsed, and then compiled by the simulator at runtime. All characters on a line following a % symbol are comments. Each rule is terminated by a colon.

The initial state of the simulation of cell layer growth consisted of a single cell positioned at $[x, y, z] = [0, 0, 0]$ with $CellType$ = ES, $Life = 1$, $LocalTime = 0.0$, and $TagSet = \{\text{S0}\}$. The initial state of the simulation of axon growth consisted of an emitting cell (far left) positioned at $[-7, 0, 0]$, 11 randomly positioned obstacle cells (middle), and 4 target cells (far right) positioned at $[7, 1, -1]$, $[6, -2, 2]$, $[8, 0, 2]$, and $[7, -1, 0]$. The emitting cell, obstacle cells, and target cells had tag sets {G1}, {G2}, and {G3}, respectively. For each of the cells, $CellType$ = E, $Life = 0$, and $LocalTime = 0.0$.

Table B.1: Parameter Values From Simulations of Cell Layer and Axon Growth

Force	Parameter	Value
Equations of Motion (Eqs. 3.1, 3.2)	m_c	1.0
	m_g	0.5
$\vec{F}_{cell}$ (Eq. 3.3)	R_{cc}	3.5
	α	2.79
	b	15.0
	c	1.16
	d	0.078
$\vec{F}_{cone}$ (Eqs. 3.4, 3.5, 3.6)	R_{gg}	2.5
	k_s	10.0
	k_c	10.0
	k_a	10.0
$\vec{F}_{lgf}$ (Eq. 3.7)	R_{cg}	3.0
	$\pm$	$+, -$
	k	$15.0, 10.0$
	β	1.66
$\vec{F}_{drag}$ (Eq. 3.8)	c_d	5.0
$\vec{F}_{collision}$ (Eq. 3.9) Collisions between cells and growth cones.	r_c	0.5
	r_g	0.17
	c_f	75.0
	γ	3.0
$\vec{F}_{collision}$ Collisions between cells.	c_f	100.0
	γ	5.0
$\vec{F}_{collision}$ Collisions between growth cones.	c_f	25.0
	γ	3.0
Movement Criterion for cells.	ϵ	8.0
	δ	10.0
Movement Criterion for growth cones.	ϵ	1.0
	δ	1.0

Rule Set for Cell Layer Growth (Fig. 3.3)

```
% The rules used to grow a group of 4 cell layers
% with different shapes, sizes, and orientations.

%============================================================
% BEGIN: Rules to position seed cells.
%============================================================

(Cell;ES;{S0};{1}) |= <DivCell;[0,90];
<GenCell;ES;{S20};1>; <GenCell;ES;{S10};1>>:

(Cell;ES;{S10};{1}) |= <DivCell;[90,90];
<GenCell;ES;{S4};1>; <GenCell;ES;{S1};1>>:

(Cell;ES;{S20};{1}) |= <DivCell;[90,90];
<GenCell;ES;{S3};1>; <GenCell;ES;{S2};1>>:

(Cell;ES;{S1};{0.1}) |= <SetRuleForce;[270,170,25];4>:  %Layer 1, Bottom
(Cell;ES;{S2};{0.5}) |= <SetRuleForce;[180,35,28];4.5>: %Layer 2, Top Left
(Cell;ES;{S3};{0.1}) |= <SetRuleForce;[90,125,25];4>:   %Layer 3, Top Center
(Cell;ES;{S4};{0.5}) |= <SetRuleForce;[350,50,28];4.5>: %Layer 4, Top Right

%============================================================
% END: Rules to position seed cells.
%============================================================


%============================================================
% BEGIN: Layer 1, Bottom
%============================================================

% Generation #1
(Cell;ES;{S1};{4.2}) |= <DivCell;[0,90];
<GenCell;ES;{A2,P1,Z1};1>; <GenCell;ES;{A1,P1,Z1}1>>:

% Generation #2
(Cell;ES;AND{A1,P1,Z1};{1}) |= <DivCell;[90,90];
<GenCell;ES;{A1,P1,Z2};1>; <GenCell;ES;{A3,P1,Z2};1>>:

(Cell;ES;AND{A2,P1,Z1};{1}) |= <DivCell;[90,90];
<GenCell;ES;{A2,P1,Z2};1>; <GenCell;ES;{A4,P1,Z2};1>>:

% Generation #3
(Cell;ES;AND{A1,P1,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A2,P1,Z3};1>; <GenCell;ES;{A1,P1,Z3};1>>:

(Cell;ES;AND{A2,P1,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A4,P1,Z3};1>; <GenCell;ES;{A3,P1,Z3};1>>:

(Cell;ES;AND{A3,P1,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A6,P1,Z3};1>; <GenCell;ES;{A5,P1,Z3};1>>:
```

```
(Cell;ES;AND{A4,P1,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A8,P1,Z3};1>; <GenCell;ES;{A7,P1,Z3};1>>:

% Generation #4
(Cell;ES;AND{A1,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{V1,P1,Z4};1>; <GenCell;ES;{L1,P1,Z4};1>>:

(Cell;ES;AND{A2,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{T1,P1,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A3,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{T1,P1,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A4,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{V2,P1,Z4};1>; <GenCell;ES;{R1,P1,Z4};1>>:

(Cell;ES;AND{A5,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{L1,P1,Z4};1>; <GenCell;ES;{V4,P1,Z4};1>>:

(Cell;ES;AND{A6,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P1,Z4};1>>:

(Cell;ES;AND{A7,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P1,Z4};1>>:

(Cell;ES;AND{A8,P1,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{R1,P1,Z4};1>; <GenCell;ES;{V3,P1,Z4};1>>:

% "Top" cell divisions.
(Cell;ES;AND{T1,P1};{1}) |= <DivCell;[90,90];
<GenCell;ES;{T1,P1,+};1>; <GenCell;ES;{T2,P1};1>>:

(Cell;ES;AND{T2,P1};{1}) |= <DivCell;[180,90];
<GenCell;E;{C1,P1,T01};0>; <GenCell;E;{C1,P1,T02};0>>:

% "Right" cell divisions.
(Cell;ES;AND{R1,P1};{1}) |= <DivCell;[0,90];
<GenCell;ES;{R1,P1,+};1>; <GenCell;ES;{R2,P1};1>>:

(Cell;ES;AND{R2,P1};{1}) |= <DivCell;[90,90];
<GenCell;E;{C1,P1,R01};0>; <GenCell;E;{C1,P1,R02};0>>:

% "Bottom" cell divisions.
(Cell;ES;AND{B1,P1};{1}) |= <DivCell;[270,90];
<GenCell;ES;{B1,P1,+};1>; <GenCell;ES;{B2,P1};1>>:

(Cell;ES;AND{B2,P1};{1}) |= <DivCell;[0,90];
<GenCell;E;{C1,P1,B01};0>; <GenCell;E;{C1,P1,B02};0>>:

% "Left" cell divisions.
(Cell;ES;AND{L1,P1};{1}) |= <DivCell;[180,90];
<GenCell;ES;{L1,P1,+};1>; <GenCell;ES;{L2,P1};1>>:

(Cell;ES;AND{L2,P1};{1}) |= <DivCell;[270,90];
```

```
<GenCell;E;{C1,P1,L01};0>; <GenCell;E;{C1,P1,L02};0>>:

% "Top Left Corner" cell divisions.
(Cell;ES;AND{V1,P1};{1}) |= <DivCell;[135,90];
<GenCell;ES;{V1,P1,+};1>; <GenCell;ES;{V12,P1};1>>:

(Cell;ES;AND{V12,P1};{1}) |= <DivCell;[225,90];
<GenCell;E;{V13,P1,V101};0>; <GenCell;E;{V13,P1,V102};0>>:

% "Top Right Corner" cell divisions.
(Cell;ES;AND{V2,P1};{1}) |= <DivCell;[45,90];
<GenCell;ES;{V2,P1,+};1>; <GenCell;ES;{V22,P1};1>>:

(Cell;ES;AND{V22,P1};{1}) |= <DivCell;[135,90];
<GenCell;E;{V23,P1,V201};0>; <GenCell;E;{V23,P1,V202};0>>:

% "Bottom Right Corner" cell divisions.
(Cell;ES;AND{V3,P1};{1}) |= <DivCell;[315,90];
<GenCell;ES;{V3,P1,+};1>; <GenCell;ES;{V32,P1};1>>:

(Cell;ES;AND{V32,P1};{1}) |= <DivCell;[45,90];
<GenCell;E;{V33,P1,V301};0>; <GenCell;E;{V33,P1,V302};0>>:

% "Bottom Left Corner" cell divisions.
(Cell;ES;AND{V4,P1};{1}) |= <DivCell;[225,90];
<GenCell;ES;{V4,P1,+};1>; <GenCell;ES;{V42,P1};1>>:

(Cell;ES;AND{V42,P1};{1}) |= <DivCell;[315,90];
<GenCell;E;{V43,P1,V401};0>; <GenCell;E;{V43,P1,V402};0>>:

% Straight-Back movements.
(Cell;ES;AND{T1,P1};{0.1}) |= <SetRuleForce;[270,90,15];0.6>:
(Cell;ES;AND{T2,P1};{0.1}) |= <SetRuleForce;[270,90,15];0.6>:
(Cell;ES;AND{R1,P1};{0.1}) |= <SetRuleForce;[180,90,15];0.6>:
(Cell;ES;AND{R2,P1};{0.1}) |= <SetRuleForce;[180,90,15];0.6>:
(Cell;ES;AND{B1,P1};{0.1}) |= <SetRuleForce;[90,90,15];0.6>:
(Cell;ES;AND{B2,P1};{0.1}) |= <SetRuleForce;[90,90,15];0.6>:
(Cell;ES;AND{L1,P1};{0.1}) |= <SetRuleForce;[0,90,15];0.6>:
(Cell;ES;AND{L2,P1};{0.1}) |= <SetRuleForce;[0,90,15];0.6>:
(Cell;ES;AND{V1,P1};{0.1}) |= <SetRuleForce;[315,90,15];0.6>:
(Cell;ES;AND{V12,P1};{0.1}) |= <SetRuleForce;[315,90,15];0.6>:
(Cell;ES;AND{V2,P1};{0.1}) |= <SetRuleForce;[225,90,15];0.6>:
(Cell;ES;AND{V22,P1};{0.1}) |= <SetRuleForce;[225,90,15];0.6>:
(Cell;ES;AND{V3,P1};{0.1}) |= <SetRuleForce;[135,90,15];0.6>:
(Cell;ES;AND{V32,P1};{0.1}) |= <SetRuleForce;[135,90,15];0.6>:
(Cell;ES;AND{V4,P1};{0.1}) |= <SetRuleForce;[45,90,15];0.6>:
(Cell;ES;AND{V42,P1};{0.1}) |= <SetRuleForce;[45,90,15];0.6>:


% Lateral-Only movements.
(Cell;E;AND{T01,P1};{0.1}) |= <SetRuleForce;[180,90,15];0.3>:
(Cell;E;AND{T02,P1};{0.1}) |= <SetRuleForce;[0,90,15];0.3>:
(Cell;E;AND{R01,P1};{0.1}) |= <SetRuleForce;[90,90,15];0.3>:
(Cell;E;AND{R02,P1};{0.1}) |= <SetRuleForce;[0,90,15];0.3>:
```

```
(Cell;E;AND{B01,P1};{0.1}) |= <SetRuleForce;[0,90,15];0.3>:
(Cell;E;AND{B02,P1};{0.1}) |= <SetRuleForce;[180,90,15];0.3>:
(Cell;E;AND{L01,P1};{0.1}) |= <SetRuleForce;[270,90,15];0.3>:
(Cell;E;AND{L02,P1};{0.1}) |= <SetRuleForce;[90,90,15];0.3>:
(Cell;E;AND{V101,P1};{0.1}) |= <SetRuleForce;[225,90,15];0.3>:
(Cell;E;AND{V102,P1};{0.1}) |= <SetRuleForce;[45,90,15];0.3>:
(Cell;E;AND{V201,P1};{0.1}) |= <SetRuleForce;[135,90,15];0.3>:
(Cell;E;AND{V202,P1};{0.1}) |= <SetRuleForce;[315,90,15];0.3>:
(Cell;E;AND{V301,P1};{0.1}) |= <SetRuleForce;[45,90,15];0.3>:
(Cell;E;AND{V302,P1};{0.1}) |= <SetRuleForce;[225,90,15];0.3>:
(Cell;E;AND{V401,P1};{0.1}) |= <SetRuleForce;[315,90,15];0.3>:
(Cell;E;AND{V402,P1};{0.1}) |= <SetRuleForce;[135,90,15];0.3>:

% Stop growth in layer 1 (P1) upon reaching generation # (Z#).
(Cell;ES;AND{T1,P1,Z12};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{R1,P1,Z12};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{B1,P1,Z12};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{L1,P1,Z12};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V1,P1,Z12};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V2,P1,Z12};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V3,P1,Z12};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V4,P1,Z12};{0.9}) |= <SetTags;{C1}>:

%=================================================================
% END: Layer 1, Bottom
%=================================================================


%=================================================================
% BEGIN: Layer 2, Top Left
%=================================================================

% Generation #1
(Cell;ES;{S2};{5.3}) |= <DivCell;[0,45];
<GenCell;ES;{A2,P2,Z1};1>; <GenCell;ES;{A1,P2,Z1};1>>:

% Generation #2
(Cell;ES;AND{A1,P2,Z1};{1}) |= <DivCell;[90,90];
<GenCell;ES;{A1,P2,Z2};1>; <GenCell;ES;{A3,P2,Z2};1>>:

(Cell;ES;AND{A2,P2,Z1};{1}) |= <DivCell;[90,90];
<GenCell;ES;{A2,P2,Z2};1>; <GenCell;ES;{A4,P2,Z2};1>>:

% Generation #3
(Cell;ES;AND{A1,P2,Z2};{1}) |= <DivCell;[0,45];
<GenCell;ES;{A2,P2,Z3};1>; <GenCell;ES;{A1,P2,Z3};1>>:

(Cell;ES;AND{A2,P2,Z2};{1}) |= <DivCell;[0,45];
<GenCell;ES;{A4,P2,Z3};1>; <GenCell;ES;{A3,P2,Z3};1>>:

(Cell;ES;AND{A3,P2,Z2};{1}) |= <DivCell;[0,45];
<GenCell;ES;{A6,P2,Z3};1>; <GenCell;ES;{A5,P2,Z3};1>>:

(Cell;ES;AND{A4,P2,Z2};{1}) |= <DivCell;[0,45];
```

```
<GenCell;ES;{A8,P2,Z3};1>; <GenCell;ES;{A7,P2,Z3};1>>:

% Generation #4
(Cell;ES;AND{A1,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{V1,P2,Z4};1>; <GenCell;ES;{L1,P2,Z4};1>>:

(Cell;ES;AND{A2,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{T1,P2,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A3,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{T1,P2,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A4,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{V2,P2,Z4};1>; <GenCell;ES;{R1,P2,Z4};1>>:

(Cell;ES;AND{A5,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{L1,P2,Z4};1>; <GenCell;ES;{V4,P2,Z4};1>>:

(Cell;ES;AND{A6,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P2,Z4};1>>:

(Cell;ES;AND{A7,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P2,Z4};1>>:

(Cell;ES;AND{A8,P2,Z3};{1}) |= <DivCell;[90,90];
<GenCell;ES;{R1,P2,Z4};1>; <GenCell;ES;{V3,P2,Z4};1>>:

% "Top" cell divisions.
(Cell;ES;AND{T1,P2};{1}) |= <DivCell;[90,90];
<GenCell;ES;{T1,P2,+};1>; <GenCell;ES;{T2,P2};1>>:

(Cell;ES;AND{T2,P2};{1}) |= <DivCell;[180,135];
<GenCell;E;{C1,P2,T01};0>; <GenCell;E;{C1,P2,T02};0>>:

% "Right" cell divisions.
(Cell;ES;AND{R1,P2};{1}) |= <DivCell;[0,45];
<GenCell;ES;{R1,P2,+};1>; <GenCell;ES;{R2,P2};1>>:

(Cell;ES;AND{R2,P2};{1}) |= <DivCell;[90,90];
<GenCell;E;{C1,P2,R01};0>; <GenCell;E;{C1,P2,R02};0>>:

% "Bottom" cell divisions.
(Cell;ES;AND{B1,P2};{1}) |= <DivCell;[270,90];
<GenCell;ES;{B1,P2,+};1>; <GenCell;ES;{B2,P2};1>>:

(Cell;ES;AND{B2,P2};{1}) |= <DivCell;[0,45];
<GenCell;E;{C1,P2,B01};0>; <GenCell;E;{C1,P2,B02};0>>:

% "Left" cell divisions.
(Cell;ES;AND{L1,P2};{1}) |= <DivCell;[180,135];
<GenCell;ES;{L1,P2,+};1>; <GenCell;ES;{L2,P2};1>>:

(Cell;ES;AND{L2,P2};{1}) |= <DivCell;[270,90];
<GenCell;E;{C1,P2,L01};0>; <GenCell;E;{C1,P2,L02};0>>:
```

```
% "Top Left Corner" cell divisions.
(Cell;ES;AND{V1,P2};{1}) |= <DivCell;[125.2643897,120];
<GenCell;ES;{V1,P2,+};1>; <GenCell;ES;{V12,P2};1>>:

(Cell;ES;AND{V12,P2};{1}) |= <DivCell;[234.7356103,120];
<GenCell;E;{V13,P2,V101};0>; <GenCell;E;{V13,P2,V102};0>>:

% "Top Right Corner" cell divisions.
(Cell;ES;AND{V2,P2};{1}) |= <DivCell;[54.73561030,60];
<GenCell;ES;{V2,P2,+};1>; <GenCell;ES;{V22,P2};1>>:

(Cell;ES;AND{V22,P2};{1}) |= <DivCell;[125.2643897,120];
<GenCell;E;{V23,P2,V201};0>; <GenCell;E;{V23,P2,V202};0>>:

% "Bottom Right Corner" cell divisions.
(Cell;ES;AND{V3,P2};{1}) |= <DivCell;[305.2643897,60];
<GenCell;ES;{V3,P2,+};1>; <GenCell;ES;{V32,P2};1>>:

(Cell;ES;AND{V32,P2};{1}) |= <DivCell;[54.73561030,60];
<GenCell;E;{V33,P2,V301};0>; <GenCell;E;{V33,P2,V302};0>>:

% "Bottom Left Corner" cell divisions.
(Cell;ES;AND{V4,P2};{1}) |= <DivCell;[234.7356103,120];
<GenCell;ES;{V4,P2,+};1>; <GenCell;ES;{V42,P2};1>>:

(Cell;ES;AND{V42,P2};{1}) |= <DivCell;[305.2643897,60];
<GenCell;E;{V43,P2,V401};0>; <GenCell;E;{V43,P2,V402};0>>:

% Straight-Back movements.
(Cell;ES;AND{T1,P2};{0.1}) |= <SetRuleForce;[270,90,15];0.4>:
(Cell;ES;AND{T2,P2};{0.1}) |= <SetRuleForce;[270,90,15];0.4>:
(Cell;ES;AND{R1,P2};{0.1}) |= <SetRuleForce;[180,135,15];0.4>:
(Cell;ES;AND{R2,P2};{0.1}) |= <SetRuleForce;[180,135,15];0.4>:
(Cell;ES;AND{B1,P2};{0.1}) |= <SetRuleForce;[90,90,15];0.4>:
(Cell;ES;AND{B2,P2};{0.1}) |= <SetRuleForce;[90,90,15];0.4>:
(Cell;ES;AND{L1,P2};{0.1}) |= <SetRuleForce;[0,45,15];0.4>:
(Cell;ES;AND{L2,P2};{0.1}) |= <SetRuleForce;[0,45,15];0.4>:
(Cell;ES;AND{V1,P2};{0.1}) |= <SetRuleForce;[305.2643897,60,15];0.4>:
(Cell;ES;AND{V12,P2};{0.1}) |= <SetRuleForce;[305.2643897,60,15];0.4>:
(Cell;ES;AND{V2,P2};{0.1}) |= <SetRuleForce;[234.7356103,120,15];0.4>:
(Cell;ES;AND{V22,P2};{0.1}) |= <SetRuleForce;[234.7356103,120,15];0.4>:
(Cell;ES;AND{V3,P2};{0.1}) |= <SetRuleForce;[125.2643897,120,15];0.4>:
(Cell;ES;AND{V32,P2};{0.1}) |= <SetRuleForce;[125.2643897,120,15];0.4>:
(Cell;ES;AND{V4,P2};{0.1}) |= <SetRuleForce;[54.73561030,60,15];0.4>:
(Cell;ES;AND{V42,P2};{0.1}) |= <SetRuleForce;[54.73561030,60,15];0.4>:

% Lateral-Only movements.
(Cell;E;AND{T01,P2};{0.1}) |= <SetRuleForce;[180,135,15];0.3>:
(Cell;E;AND{T02,P2};{0.1}) |= <SetRuleForce;[0,45,15];0.3>:
(Cell;E;AND{R01,P2};{0.1}) |= <SetRuleForce;[90,90,15];0.3>:
(Cell;E;AND{R02,P2};{0.1}) |= <SetRuleForce;[0,45,15];0.3>:
(Cell;E;AND{B01,P2};{0.1}) |= <SetRuleForce;[0,45,15];0.3>:
(Cell;E;AND{B02,P2};{0.1}) |= <SetRuleForce;[180,135,15];0.3>:
```

```
(Cell;E;AND{L01,P2};{0.1}) |= <SetRuleForce;[270,90,15];0.3>:
(Cell;E;AND{L02,P2};{0.1}) |= <SetRuleForce;[90,90,15];0.3>:
(Cell;E;AND{V101,P2};{0.1}) |= <SetRuleForce;[234.7356103,120,15];0.3>:
(Cell;E;AND{V102,P2};{0.1}) |= <SetRuleForce;[54.73561030,60,15];0.3>:
(Cell;E;AND{V201,P2};{0.1}) |= <SetRuleForce;[125.2643897,120,15];0.3>:
(Cell;E;AND{V202,P2};{0.1}) |= <SetRuleForce;[305.2643897,60,15];0.3>:
(Cell;E;AND{V301,P2};{0.1}) |= <SetRuleForce;[54.73561030,60,15];0.3>:
(Cell;E;AND{V302,P2};{0.1}) |= <SetRuleForce;[234.7356103,120,15];0.3>:
(Cell;E;AND{V401,P2};{0.1}) |= <SetRuleForce;[305.2643897,60,15];0.3>:
(Cell;E;AND{V402,P2};{0.1}) |= <SetRuleForce;[125.2643897,120,15];0.3>:

% Stop growth in layer 2 (P2) upon reaching generation # (Z#).
(Cell;ES;AND{T1,P2,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{R1,P2,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{B1,P2,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{L1,P2,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V1,P2,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V2,P2,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V3,P2,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V4,P2,Z15};{0.9}) |= <SetTags;{C1}>:

%=================================================================
% END: Layer 2, Top Left
%=================================================================


%=================================================================
% BEGIN: Layer 3, Top Center
%=================================================================

% Generation #1
(Cell;ES;{S3};{4.7}) |= <DivCell;[0,90];
<GenCell;ES;{A2,P3,Z1};1>; <GenCell;ES;{A1,P3,Z1};1>>:

% Generation #2
(Cell;ES;AND{A1,P3,Z1};{1}) |= <DivCell;[90,160];
<GenCell;ES;{A1,P3,Z2};1>; <GenCell;ES;{A3,P3,Z2};1>>:

(Cell;ES;AND{A2,P3,Z1};{1}) |= <DivCell;[90,160];
<GenCell;ES;{A2,P3,Z2};1>; <GenCell;ES;{A4,P3,Z2};1>>:

% Generation #3
(Cell;ES;AND{A1,P3,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A2,P3,Z3};1>; <GenCell;ES;{A1,P3,Z3};1>>:

(Cell;ES;AND{A2,P3,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A4,P3,Z3};1>; <GenCell;ES;{A3,P3,Z3};1>>:

(Cell;ES;AND{A3,P3,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A6,P3,Z3};1>; <GenCell;ES;{A5,P3,Z3};1>>:

(Cell;ES;AND{A4,P3,Z2};{1}) |= <DivCell;[0,90];
<GenCell;ES;{A8,P3,Z3};1>; <GenCell;ES;{A7,P3,Z3};1>>:
```

```
% Generation #4
(Cell;ES;AND{A1,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;ES;{V1,P3,Z4};1>; <GenCell;ES;{L1,P3,Z4};1>>:

(Cell;ES;AND{A2,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;ES;{T1,P3,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A3,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;ES;{T1,P3,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A4,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;ES;{V2,P3,Z4};1>; <GenCell;ES;{R1,P3,Z4};1>>:

(Cell;ES;AND{A5,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;ES;{L1,P3,Z4};1>; <GenCell;ES;{V4,P3,Z4};1>>:

(Cell;ES;AND{A6,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P3,Z4};1>>:

(Cell;ES;AND{A7,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P3,Z4};1>>:

(Cell;ES;AND{A8,P3,Z3};{1}) |= <DivCell;[90,160];
<GenCell;ES;{R1,P3,Z4};1>; <GenCell;ES;{V3,P3,Z4};1>>:

% "Top" cell divisions.
(Cell;ES;AND{T1,P3};{1}) |= <DivCell;[90,160];
<GenCell;ES;{T1,P3,+};1>; <GenCell;ES;{T2,P3};1>>:

(Cell;ES;AND{T2,P3};{1}) |= <DivCell;[180,90];
<GenCell;E;{C1,P3,T01};0>; <GenCell;E;{C1,P3,T02};0>>:

% "Right" cell divisions.
(Cell;ES;AND{R1,P3};{1}) |= <DivCell;[0,90];
<GenCell;ES;{R1,P3,+};1>; <GenCell;ES;{R2,P3};1>>:

(Cell;ES;AND{R2,P3};{1}) |= <DivCell;[90,160];
<GenCell;E;{C1,P3,R01};0>; <GenCell;E;{C1,P3,R02};0>>:

% "Bottom" cell divisions.
(Cell;ES;AND{B1,P3};{1}) |= <DivCell;[270,20];
<GenCell;ES;{B1,P3,+};1>; <GenCell;ES;{B2,P3};1>>:

(Cell;ES;AND{B2,P3};{1}) |= <DivCell;[0,90];
<GenCell;E;{C1,P3,B01};0>; <GenCell;E;{C1,P3,B02};0>>:

% "Left" cell divisions.
(Cell;ES;AND{L1,P3};{1}) |= <DivCell;[180,90];
<GenCell;ES;{L1,P3,+};1>; <GenCell;ES;{L2,P3};1>>:

(Cell;ES;AND{L2,P3};{1}) |= <DivCell;[270,20];
<GenCell;E;{C1,P3,L01};0>; <GenCell;E;{C1,P3,L02};0>>:

% "Top Left Corner" cell divisions.
```

```
(Cell;ES;AND{V1,P3};{1}) |= <DivCell;[161.1182788,131.6411433];
<GenCell;ES;{V1,P3,+};1>; <GenCell;ES;{V12,P3};1>>:

(Cell;ES;AND{V12,P3};{1}) |= <DivCell;[198.8817212,48.35885672];
<GenCell;E;{V13,P3,V101};0>; <GenCell;E;{V13,P3,V102};0>>:

% "Top Right Corner" cell divisions.
(Cell;ES;AND{V2,P3};{1}) |= <DivCell;[18.88172120,131.6411433];
<GenCell;ES;{V2,P3,+};1>; <GenCell;ES;{V22,P3};1>>:

(Cell;ES;AND{V22,P3};{1}) |= <DivCell;[161.1182788,131.6411433];
<GenCell;E;{V23,P3,V201};0>; <GenCell;E;{V23,P3,V202};0>>:

% "Bottom Right Corner" cell divisions.
(Cell;ES;AND{V3,P3};{1}) |= <DivCell;[341.1182788,48.35885672];
<GenCell;ES;{V3,P3,+};1>; <GenCell;ES;{V32,P3};1>>:

(Cell;ES;AND{V32,P3};{1}) |= <DivCell;[18.88172120,131.6411433];
<GenCell;E;{V33,P3,V301};0>; <GenCell;E;{V33,P3,V302};0>>:

% "Bottom Left Corner" cell divisions.
(Cell;ES;AND{V4,P3};{1}) |= <DivCell;[198.8817212,48.35885672];
<GenCell;ES;{V4,P3,+};1>; <GenCell;ES;{V42,P3};1>>:

(Cell;ES;AND{V42,P3};{1}) |= <DivCell;[341.1182788,48.35885672];
<GenCell;E;{V43,P3,V401};0>; <GenCell;E;{V43,P3,V402};0>>:

% Left and Right (-x and +x) movements.
(Cell;E;AND{T01,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;E;AND{T02,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{R01,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{R02,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{B01,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{B02,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;E;AND{L01,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;E;AND{L02,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;E;AND{V101,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;E;AND{V102,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;E;AND{V201,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{V202,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{V301,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{V302,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;E;AND{V401,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;E;AND{V402,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:

(Cell;ES;AND{V1,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;ES;AND{V12,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;ES;AND{V2,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{V22,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{V3,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{V32,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{V4,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;ES;AND{V42,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
```

```
(Cell;ES;AND{T1,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{T2,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{B1,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;ES;AND{B2,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:

(Cell;ES;AND{R1,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{R2,P3};{0.1}) |= <SetRuleForce;[0,90,10];1>:
(Cell;ES;AND{L1,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:
(Cell;ES;AND{L2,P3};{0.1}) |= <SetRuleForce;[180,90,10];1>:

% Stop growth in layer 3 (P3) upon reaching generation # (Z#).
(Cell;ES;AND{T1,P3,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{R1,P3,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{B1,P3,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{L1,P3,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V1,P3,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V2,P3,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V3,P3,Z15};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V4,P3,Z15};{0.9}) |= <SetTags;{C1}>:

%================================================================
% END: Layer 3, Top Center
%================================================================


%================================================================
% BEGIN: Layer 4, Top Right
%================================================================

% Generation #1
(Cell;ES;{S4};{5.1}) |= <DivCell;[0,60];
<GenCell;ES;{A2,P4,Z1};1>; <GenCell;ES;{A1,P4,Z1};1>>:

% Generation #2
(Cell;ES;AND{A1,P4,Z1};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{A1,P4,Z2};1>; <GenCell;ES;{A3,P4,Z2};1>>:

(Cell;ES;AND{A2,P4,Z1};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{A2,P4,Z2};1>; <GenCell;ES;{A4,P4,Z2};1>>:

% Generation #3
(Cell;ES;AND{A1,P4,Z2};{1}) |= <DivCell;[0,60];
<GenCell;ES;{A2,P4,Z3};1>; <GenCell;ES;{A1,P4,Z3};1>>:

(Cell;ES;AND{A2,P4,Z2};{1}) |= <DivCell;[0,60];
<GenCell;ES;{A4,P4,Z3};1>; <GenCell;ES;{A3,P4,Z3};1>>:

(Cell;ES;AND{A3,P4,Z2};{1}) |= <DivCell;[0,60];
<GenCell;ES;{A6,P4,Z3};1>; <GenCell;ES;{A5,P4,Z3};1>>:

(Cell;ES;AND{A4,P4,Z2};{1}) |= <DivCell;[0,60];
<GenCell;ES;{A8,P4,Z3};1>; <GenCell;ES;{A7,P4,Z3};1>>:

% Generation #4
```

```
(Cell;ES;AND{A1,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{V1,P4,Z4};1>; <GenCell;ES;{L1,P4,Z4};1>>:

(Cell;ES;AND{A2,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{T1,P4,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A3,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{T1,P4,Z4};1>; <GenCell;E;{C1};0>>:

(Cell;ES;AND{A4,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{V2,P4,Z4};1>; <GenCell;ES;{R1,P4,Z4};1>>:

(Cell;ES;AND{A5,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{L1,P4,Z4};1>; <GenCell;ES;{V4,P4,Z4};1>>:

(Cell;ES;AND{A6,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P4,Z4};1>>:

(Cell;ES;AND{A7,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;E;{C1};0>; <GenCell;ES;{B1,P4,Z4};1>>:

(Cell;ES;AND{A8,P4,Z3};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{R1,P4,Z4};1>; <GenCell;ES;{V3,P4,Z4};1>>:

% "Top" cell divisions.
(Cell;ES;AND{T1,P4};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;ES;{T1,P4,+};1>; <GenCell;ES;{T2,P4,+};1>>:

(Cell;ES;AND{T2,P4};{1}) |= <DivCell;[180,120];
<GenCell;E;{C1,P4,T01};0>; <GenCell;E;{C1,P4,T02};0>>:

% "Right" cell divisions.
(Cell;ES;AND{R1,P4};{1}) |= <DivCell;[0,60];
<GenCell;ES;{R1,P4,+};1>; <GenCell;ES;{R2,P4,+};1>>:

(Cell;ES;AND{R2,P4};{1}) |= <DivCell;[63.43494883,127.7612439];
<GenCell;E;{C1,P4,R01};0>; <GenCell;E;{C1,P4,R02};0>>:

% "Bottom" cell divisions.
(Cell;ES;AND{B1,P4};{1}) |= <DivCell;[243.4349488,52.23875607];
<GenCell;ES;{B1,P4,+};1>; <GenCell;ES;{B2,P4,+};1>>:

(Cell;ES;AND{B2,P4};{1}) |= <DivCell;[0,60];
<GenCell;E;{C1,P4,B01};0>; <GenCell;E;{C1,P4,B02};0>>:

% "Left" cell divisions.
(Cell;ES;AND{L1,P4};{1}) |= <DivCell;[180,120];
<GenCell;ES;{L1,P4,+};1>; <GenCell;ES;{L2,P4,+};1>>:

(Cell;ES;AND{L2,P4};{1}) |= <DivCell;[243.4349488,52.23875607];
<GenCell;E;{C1,P4,L01};0>; <GenCell;E;{C1,P4,L02};0>>:

% "Top Left Corner" cell divisions.
(Cell;ES;AND{V1,P4};{1}) |= <DivCell;[125.9325292,141.8657555];
```

```
<GenCell;ES;{V1,P4,+};1>; <GenCell;ES;{V12,P4,+};1>>:

(Cell;ES;AND{V12,P4};{1}) |= <DivCell;[210.1049818,85.44251236];
<GenCell;E;{V13,P4,V101};0>; <GenCell;E;{V13,P4,V102};0>>:

% "Top Right Corner" cell divisions.
(Cell;ES;AND{V2,P4};{1}) |= <DivCell;[30.10498182,94.55748764];
<GenCell;ES;{V2,P4,+};1>; <GenCell;ES;{V22,P4,+};1>>:

(Cell;ES;AND{V22,P4};{1}) |= <DivCell;[125.9325292,141.8657555];
<GenCell;E;{V23,P4,V201};0>; <GenCell;E;{V23,P4,V202};0>>:

% "Bottom Right Corner" cell divisions.
(Cell;ES;AND{V3,P4};{1}) |= <DivCell;[305.9325292,38.13424445];
<GenCell;ES;{V3,P4,+};1>; <GenCell;ES;{V32,P4,+};1>>:

(Cell;ES;AND{V32,P4};{1}) |= <DivCell;[30.10498182,94.55748764];
<GenCell;E;{V33,P4,V301};0>; <GenCell;E;{V33,P4,V302};0>>:

% "Bottom Left Corner" cell divisions.
(Cell;ES;AND{V4,P4};{1}) |= <DivCell;[210.1049818,85.44251236];
<GenCell;ES;{V4,P4,+};1>; <GenCell;ES;{V42,P4,+};1>>:

(Cell;ES;AND{V42,P4};{1}) |= <DivCell;[305.9325292,38.13424445];
<GenCell;E;{V43,P4,V401};0>; <GenCell;E;{V43,P4,V402};0>>:

% Right arm rotation.

% ("Top Right" <-- 90 deg., "Right" <-- 70 deg., "Bottom Right" <-- 50 deg.)
% Above angles expressed relative to 2D coordinate system that is
% parallel to plane that contains 4th cell layer.
(Cell;ES;AND{V2,P4,Z4};{0.1}) |= <SetRuleForce;[63.43494883,127.7612439,15];1>:
(Cell;ES;AND{V22,P4,Z4};{0.1}) |= <SetRuleForce;[63.43494883,127.7612439,15];1>:
(Cell;ES;AND{R1,P4,Z4};{0.1}) |= <SetRuleForce;[46.59644057,113.8555249,15];1>:
(Cell;ES;AND{R2,P4,Z4};{0.1}) |= <SetRuleForce;[46.59644057,113.8555249,15];1>:
(Cell;ES;AND{V3,P4,Z4};{0.1}) |= <SetRuleForce;[33.20819964,98.49428129,15];1>:
(Cell;ES;AND{V32,P4,Z4};{0.1}) |= <SetRuleForce;[33.20819964,98.49428129,15];1>:

% (TR cells <-- 110 deg., R cells <-- 90 deg., BR cells <-- 70 deg.)
(Cell;ES;AND{V2,P4,Z5};{0.1}) |= <SetRuleForce;[86.89592912,138.2839608,15];1>:
(Cell;ES;AND{V22,P4,Z5};{0.1}) |= <SetRuleForce;[86.89592912,138.2839608,15];1>:
(Cell;ES;AND{R1,P4,Z5};{0.1}) |= <SetRuleForce;[63.43494883,127.7612439,15];1>:
(Cell;ES;AND{R2,P4,Z5};{0.1}) |= <SetRuleForce;[63.43494883,127.7612439,15];1>:
(Cell;ES;AND{V3,P4,Z5};{0.1}) |= <SetRuleForce;[46.59644057,113.8555249,15];1>:
(Cell;ES;AND{V32,P4,Z5};{0.1}) |= <SetRuleForce;[46.59644057,113.8555249,15];1>:

% (130,110,90)
(Cell;ES;AND{V2,P4,Z6};{0.1}) |= <SetRuleForce;[117.8198476,142.2321034,15];1>:
(Cell;ES;AND{V22,P4,Z6};{0.1}) |= <SetRuleForce;[117.8198476,142.2321034,15];1>:
(Cell;ES;AND{R1,P4,Z6};{0.1}) |= <SetRuleForce;[86.89592912,138.2839608,15];1>:
(Cell;ES;AND{R2,P4,Z6};{0.1}) |= <SetRuleForce;[86.89592912,138.2839608,15];1>:
(Cell;ES;AND{V3,P4,Z6};{0.1}) |= <SetRuleForce;[63.43494883,127.7612439,15];1>:
(Cell;ES;AND{V32,P4,Z6};{0.1}) |= <SetRuleForce;[63.43494883,127.7612439,15];1>:
```

```
% (150,130,110)
(Cell;ES;AND{V2,P4,Z7};{0.1}) |= <SetRuleForce;[148.3344925,137.6632204,15];1>:
(Cell;ES;AND{V22,P4,Z7};{0.1}) |= <SetRuleForce;[148.3344925,137.6632204,15];1>:
(Cell;ES;AND{R1,P4,Z7};{0.1}) |= <SetRuleForce;[117.8198476,142.2321034,15];1>:
(Cell;ES;AND{R2,P4,Z7};{0.1}) |= <SetRuleForce;[117.8198476,142.2321034,15];1>:
(Cell;ES;AND{V3,P4,Z7};{0.1}) |= <SetRuleForce;[86.89592912,138.2839608,15];1>:
(Cell;ES;AND{V32,P4,Z7};{0.1}) |= <SetRuleForce;[86.89592912,138.2839608,15];1>:

% (170,150,130)
(Cell;ES;AND{V2,P4,Z8};{0.1}) |= <SetRuleForce;[171.1815450,126.7797983,15];1>:
(Cell;ES;AND{V22,P4,Z8};{0.1}) |= <SetRuleForce;[171.1815450,126.7797983,15];1>:
(Cell;ES;AND{R1,P4,Z8};{0.1}) |= <SetRuleForce;[148.3344925,137.6632204,15];1>:
(Cell;ES;AND{R2,P4,Z8};{0.1}) |= <SetRuleForce;[148.3344925,137.6632204,15];1>:
(Cell;ES;AND{V3,P4,Z8};{0.1}) |= <SetRuleForce;[117.8198476,142.2321034,15];1>:
(Cell;ES;AND{V32,P4,Z8};{0.1}) |= <SetRuleForce;[117.8198476,142.2321034,15];1>:

% (190,170,150)
(Cell;ES;AND{V2,P4,Z9};{0.1}) |= <SetRuleForce;[187.6492002,112.7099670,15];1>:
(Cell;ES;AND{V22,P4,Z9};{0.1}) |= <SetRuleForce;[187.6492002,112.7099670,15];1>:
(Cell;ES;AND{R1,P4,Z9};{0.1}) |= <SetRuleForce;[171.1815450,126.7797983,15];1>:
(Cell;ES;AND{R2,P4,Z9};{0.1}) |= <SetRuleForce;[171.1815450,126.7797983,15];1>:
(Cell;ES;AND{V3,P4,Z9};{0.1}) |= <SetRuleForce;[148.3344925,137.6632204,15];1>:
(Cell;ES;AND{V32,P4,Z9};{0.1}) |= <SetRuleForce;[148.3344925,137.6632204,15];1>:

% (210,190,170)
(Cell;ES;AND{V2,P4,Z10};{0.1}) |= <SetRuleForce;[200.8812097,97.28624520,15];1>:
(Cell;ES;AND{V22,P4,Z10};{0.1}) |= <SetRuleForce;[200.8812097,97.28624520,15];1>:
(Cell;ES;AND{R1,P4,Z10};{0.1}) |= <SetRuleForce;[187.6492002,112.7099670,15];1>:
(Cell;ES;AND{R2,P4,Z10};{0.1}) |= <SetRuleForce;[187.6492002,112.7099670,15];1>:
(Cell;ES;AND{V3,P4,Z10};{0.1}) |= <SetRuleForce;[171.1815450,126.7797983,15];1>:
(Cell;ES;AND{V32,P4,Z10};{0.1}) |= <SetRuleForce;[171.1815450,126.7797983,15];1>:

% Left arm rotation.

% ("Bottom Left" <-- 270 deg., "Left" <-- 250 deg., "Top Left" <-- 230 deg.)
% Above angles expressed relative to 2D coordinate system that is
% parallel to plane that contains 4th cell layer.
(Cell;ES;AND{V4,P4,Z4};{0.1}) |= <SetRuleForce;[243.4349488,52.23875607,15];1>:
(Cell;ES;AND{V42,P4,Z4};{0.1}) |= <SetRuleForce;[243.4349488,52.23875607,15];1>:
(Cell;ES;AND{L1,P4,Z4};{0.1}) |= <SetRuleForce;[226.5964406,66.14447506,15];1>:
(Cell;ES;AND{L2,P4,Z4};{0.1}) |= <SetRuleForce;[226.5964406,66.14447506,15];1>:
(Cell;ES;AND{V1,P4,Z4};{0.1}) |= <SetRuleForce;[213.2081996,81.50571866,15];1>:
(Cell;ES;AND{V12,P4,Z4};{0.1}) |= <SetRuleForce;[213.2081996,81.50571866,15];1>:

% (BL cells <-- 290 deg., L cells <-- 270 deg., TL cells <-- 250 deg.)
(Cell;ES;AND{V4,P4,Z5};{0.1}) |= <SetRuleForce;[266.8959291,41.71603919,15];1>:
(Cell;ES;AND{V42,P4,Z5};{0.1}) |= <SetRuleForce;[266.8959291,41.71603919,15];1>:
(Cell;ES;AND{L1,P4,Z5};{0.1}) |= <SetRuleForce;[243.4349488,52.23875607,15];1>:
(Cell;ES;AND{L2,P4,Z5};{0.1}) |= <SetRuleForce;[243.4349488,52.23875607,15];1>:
(Cell;ES;AND{V1,P4,Z5};{0.1}) |= <SetRuleForce;[226.5964406,66.14447506,15];1>:
(Cell;ES;AND{V12,P4,Z5};{0.1}) |= <SetRuleForce;[226.5964406,66.14447506,15];1>:

% (310,290,270)
(Cell;ES;AND{V4,P4,Z6};{0.1}) |= <SetRuleForce;[297.8198476,37.76789659,15];1>:
```

```
(Cell;ES;AND{V42,P4,Z6};{0.1}) |= <SetRuleForce;[297.8198476,37.76789659,15];1>:
(Cell;ES;AND{L1,P4,Z6};{0.1}) |= <SetRuleForce;[266.8959291,41.71603919,15];1>:
(Cell;ES;AND{L2,P4,Z6};{0.1}) |= <SetRuleForce;[266.8959291,41.71603919,15];1>:
(Cell;ES;AND{V1,P4,Z6};{0.1}) |= <SetRuleForce;[243.4349488,52.23875607,15];1>:
(Cell;ES;AND{V12,P4,Z6};{0.1}) |= <SetRuleForce;[243.4349488,52.23875607,15];1>:

% (330,310,290)
(Cell;ES;AND{V4,P4,Z7};{0.1}) |= <SetRuleForce;[328.3344925,42.33677954,15];1>:
(Cell;ES;AND{V42,P4,Z7};{0.1}) |= <SetRuleForce;[328.3344925,42.33677954,15];1>:
(Cell;ES;AND{L1,P4,Z7};{0.1}) |= <SetRuleForce;[297.8198476,37.76789659,15];1>:
(Cell;ES;AND{L2,P4,Z7};{0.1}) |= <SetRuleForce;[297.8198476,37.76789659,15];1>:
(Cell;ES;AND{V1,P4,Z7};{0.1}) |= <SetRuleForce;[266.8959291,41.71603919,15];1>:
(Cell;ES;AND{V12,P4,Z7};{0.1}) |= <SetRuleForce;[266.8959291,41.71603919,15];1>:

% (350,330,310)
(Cell;ES;AND{V4,P4,Z8};{0.1}) |= <SetRuleForce;[351.1815450,53.22020173,15];1>:
(Cell;ES;AND{V42,P4,Z8};{0.1}) |= <SetRuleForce;[351.1815450,53.22020173,15];1>:
(Cell;ES;AND{L1,P4,Z8};{0.1}) |= <SetRuleForce;[328.3344925,42.33677954,15];1>:
(Cell;ES;AND{L2,P4,Z8};{0.1}) |= <SetRuleForce;[328.3344925,42.33677954,15];1>:
(Cell;ES;AND{V1,P4,Z8};{0.1}) |= <SetRuleForce;[297.8198476,37.76789659,15];1>:
(Cell;ES;AND{V12,P4,Z8};{0.1}) |= <SetRuleForce;[297.8198476,37.76789659,15];1>:

% (10,350,330)
(Cell;ES;AND{V4,P4,Z9};{0.1}) |= <SetRuleForce;[7.649200163,67.29003295,15];1>:
(Cell;ES;AND{V42,P4,Z9};{0.1}) |= <SetRuleForce;[7.649200163,67.29003295,15];1>:
(Cell;ES;AND{L1,P4,Z9};{0.1}) |= <SetRuleForce;[351.1815450,53.22020173,15];1>:
(Cell;ES;AND{L2,P4,Z9};{0.1}) |= <SetRuleForce;[351.1815450,53.22020173,15];1>:
(Cell;ES;AND{V1,P4,Z9};{0.1}) |= <SetRuleForce;[328.3344925,42.33677954,15];1>:
(Cell;ES;AND{V12,P4,Z9};{0.1}) |= <SetRuleForce;[328.3344925,42.33677954,15];1>:

% (30,10,350)
(Cell;ES;AND{V4,P4,Z10};{0.1}) |= <SetRuleForce;[20.88120967,82.71375480,15];1>:
(Cell;ES;AND{V42,P4,Z10};{0.1}) |= <SetRuleForce;[20.88120967,82.71375480,15];1>:
(Cell;ES;AND{L1,P4,Z10};{0.1}) |= <SetRuleForce;[7.649200163,67.29003295,15];1>:
(Cell;ES;AND{L2,P4,Z10};{0.1}) |= <SetRuleForce;[7.649200163,67.29003295,15];1>:
(Cell;ES;AND{V1,P4,Z10};{0.1}) |= <SetRuleForce;[351.1815450,53.22020173,15];1>:
(Cell;ES;AND{V12,P4,Z10};{0.1}) |= <SetRuleForce;[351.1815450,53.22020173,15];1>:

% Stop growth in layer 4 (P4) upon reaching generation # (Z#).
(Cell;ES;AND{T1,P4,Z4};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{R1,P4,Z9};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{B1,P4,Z4};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{L1,P4,Z9};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V1,P4,Z9};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V2,P4,Z9};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V3,P4,Z9};{0.9}) |= <SetTags;{C1}>:
(Cell;ES;AND{V4,P4,Z9};{0.9}) |= <SetTags;{C1}>:

%================================================================
% END: Layer 4, Top Right
%================================================================
```

Rule Set for Axon Growth (Fig. 3.4)

```
% The rules used to grow axons from an emitting cell. The axons grow around
% a group of obstacle cells, and connect to a group of target cells
% on the other side.

% LGF Rules; duration of 10000 means LGF last throughout simulation.
(Cell;E;{G2};{0.04}) |= <SetLGF;+;10;10000>:
(Cell;E;{G3};{0.04}) |= <SetLGF;-;15;10000>:

% Axon emissions.
(Cell;E;{G1};{1}) |= <EmitAxon;[20,90]; <GenCone;{H1}>>:
(Cell;E;{G1};{3}) |= <EmitAxon;[340,90]; <GenCone;{H2}>>:

% Application of Rule Force to growth cones.
(Cone;{H1};{0.04}) |= <SetRuleForce;[10,90,5];10.0>:
(Cone;{H12};{0.04}) |= <SetRuleForce;[0,90,5];10.0>:
(Cone;{H2};{0.04}) |= <SetRuleForce;[350,90,5];10.0>:
(Cone;{H22};{0.04}) |= <SetRuleForce;[0,90,5];10.0>:

(Cone;{H1};{10}) |= <SetRuleForce;[330,110,5];8.0>:
(Cone;{H12};{10}) |= <SetRuleForce;[330,110,5];8.0>:
(Cone;{H2};{10}) |= <SetRuleForce;[45,90,5];8.0>:
(Cone;{H22};{10}) |= <SetRuleForce;[45,90,5];8.0>:

% Axon branching.
(Cone;{H1};{1.0}) |= <DivCone;[0,0]; <GenCone;{H12}>>:
(Cone;{H1};{1.0}) |= <DivCone;[90,90]; <GenCone;{H12}>>:
(Cone;{H1};{1.0}) |= <DivCone;[0,180]; <GenCone;{H12}>>:
(Cone;{H1};{1.0}) |= <DivCone;[270,90]; <GenCone;{H12}>>:

(Cone;{H2};{1.0}) |= <DivCone;[0,0]; <GenCone;{H22}>>:
(Cone;{H2};{1.0}) |= <DivCone;[90,90]; <GenCone;{H22}>>:
(Cone;{H2};{1.0}) |= <DivCone;[0,180]; <GenCone;{H22}>>:
(Cone;{H2};{1.0}) |= <DivCone;[270,90]; <GenCone;{H22}>>:
```

Appendix C

Somatosensory Cortex Network Rule Set

The rule set used by the SINSA model to grow the somatosensory cortex network (Sect. 4.2.1) is shown below. The rules are placed in an ASCII text file and are read into the simulator at runtime. The simulator then parses the rule set and displays warnings if there are any syntax errors. After parsing, the rules are compiled into the simulator's internal representation, which is a tree structure. The parser treats all characters on a line following a % symbol as comments. In the file each rule is terminated by a colon.

The initial network state consisted of four "seed" cells each with an initial velocity of 0.0. The initial conditions of these cells are specified in Table C.1.

Table C.1: Initial Conditions of the Somatosensory Cortex Network

Cell #	Position [x,y,z]	Cell Type	Life	Local Time	Tag Set
1	[0.8,0.0,4.5]	ES	9	0.0	{B0}
2	[-0.8,0.0,4.5]	ES	9	0.0	{D0}
3	[0.8,0.0,-4.5]	ES	9	0.0	{A0}
4	[-0.8,0.0,-4.5]	ES	9	0.0	{C0}

```
% The rules used to grow the somatosensory cortex network.

% Grow Thalamic Layer
(Cell;ES;{A0};{0.1}) => <DivCell;[180,90];
<GenCell;ES;{A0};#>; <GenCell;ES;{C01};1>>:  % #1
```

```
(Cell;ES;{C0};{0.1}) => <DivCell;[0,90];
<GenCell;ES;{C0};#>; <GenCell;ES;{C01};1>>:

(Cell;ES;{C01};{0.1}) => <DivCell;[90,90];
<GenCell;ES;{C1};8>; <GenCell;ES;{C02};1>>:

(Cell;ES;{C02};{0.1}) => <DivCell;[270,90];
<GenCell;ES;{C2};8>; <GenCell;E;{C03};0>>:

(Cell;ES;{C1};{0.1}) => <DivCell;[90,90];
<GenCell;ES;{C1};#>; <GenCell;E;{C1};0>>:

(Cell;ES;{C2};{0.1}) => <DivCell;[270,90];
<GenCell;ES;{C2};#>; <GenCell;E;{C2};0>>:

% Grow Cortical Layer
(Cell;ES;{B0};{0.1}) => <DivCell;[180,90];
<GenCell;ES;{B0};#>; <GenCell;ES;{D01};1>>:  % #7

(Cell;ES;{D0};{0.1}) => <DivCell;[0,90];
<GenCell;ES;{D0};#>; <GenCell;ES;{D01};1>>:

(Cell;ES;{D01};{0.1}) => <DivCell;[90,90];
<GenCell;ES;{D1};8>; <GenCell;ES;{D02};1>>:

(Cell;ES;{D02};{0.1}) => <DivCell;[270,90];
<GenCell;ES;{D2};8>; <GenCell;E;{D03};0>>:

(Cell;ES;{D1};{0.1}) => <DivCell;[90,90];
<GenCell;ES;{D1};#>; <GenCell;E;{D1};0>>:

(Cell;ES;{D2};{0.1}) => <DivCell;[270,90];
<GenCell;ES;{D2};#>; < GenCell;E;{D2};0>>:

% Emit Thalamocortical Axons
% After a specified period of time thalamic cells are
% selected at random on regular intervals and given an "E50" tag.
(Cell;E;{E50};*) => <SetTags;{E51,C1}>:                              % #13

(Cell;E;{E50};*) => <EmitAxon;[0,30]; <GenCone;{A1,L1,D1,T0}>>:     % #14
(Cone;{A1};{0.05}) => <SetRuleForce;[0,15,25];3.0>:
(Cone;{A1};{0.1}) => <DivCone;[10,90]; <GenCone;{A11,L1,D1,T1}>>:
(Cone;{A1};{0.1}) => <DivCone;[350,90]; <GenCone;{A12,L1,D1,T1}>>:

(Cell;E;{E50};*) => <EmitAxon;[90,35]; <GenCone;{A2,L1,D1,T0}>>:    % #18
(Cone;{A2};{0.05}) => <SetRuleForce;[90,15,25];3.0>:
(Cone;{A2};{0.1}) => <DivCone;[100,90]; <GenCone;{A21,L1,D1,T1}>>:
(Cone;{A2};{0.1}) => <DivCone;[80,90]; <GenCone;{A22,L1,D1,T1}>>:

(Cell;E;{E50};*) => <EmitAxon;[180,35]; <GenCone;{A3,L1,D1,T0}>>:   % #22
(Cone;{A3};{0.05}) => <SetRuleForce;[180,15,25];3.0>:
(Cone;{A3};{0.1}) => <DivCone;[190,90]; <GenCone;{A31,L1,D1,T1}>>:
(Cone;{A3};{0.1}) => <DivCone;[170,90]; <GenCone;{A32,L1,D1,T1}>>:
```

```
(Cell;E;{E50};*) => <EmitAxon;[270,35]; <GenCone;{A4,L1,D1,T0}>>:   % #26
(Cone;{A4};{0.05}) => <SetRuleForce;[270,15,25];3.0>:
(Cone;{A4};{0.1}) => <DivCone;[280,90]; <GenCone;{A41,L1,D1,T1}>>:
(Cone;{A4};{0.1}) => <DivCone;[260,90]; <GenCone;{A42,L1,D1,T1}>>:

(Cell;E;{E50};*) => <EmitAxon;[0,0]; <GenCone;{A5,L1,D1,T0}>>:      % #30
(Cone;{A5};{0.05}) => <SetRuleForce;[0,0,25];3.0>:
(Cone;{A5};{0.1}) => <DivCone;[0,90]; <GenCone;{A51,L1,D1,T1}>>:
(Cone;{A5};{0.1}) => <DivCone;[90,90]; <GenCone;{A52,L1,D1,T1}>>:
(Cone;{A5};{0.1}) => <DivCone;[180,90]; <GenCone;{A53,L1,D1,T1}>>:
(Cone;{A5};{0.1}) => <DivCone;[270,90]; <GenCone;{A54,L1,D1,T1}>>:

(Cone;AND{L1,D1};{0.7}) => <DivCone;[90,90]; <GenCone;{L1,D2,+}>>:  % #36
(Cone;AND{L1,D1};{0.7}) => <DivCone;[270,90]; <GenCone;{L1,D2,+}>>:
(Cone;AND{L1,D1};{0.02}) => <SetRuleForce;[180,10,25];3.0>:

(Cone;AND{L1,D2};{0.7}) => <DivCone;[0,90]; <GenCone;{L1,D1,+}>>:   % #39
(Cone;AND{L1,D2};{0.7}) => <DivCone;[180,90]; <GenCone;{L1,D1,+}>>:
(Cone;AND{L1,D2};{0.02}) => <SetRuleForce;[90,10,25];3.0>:

(Cone;{L1};{0.7}) => <SetTags;{L0}>:                                % #42
(Cone;{T2};*) => <SetTags;{L0}>:

% Emit Intracortical Axons
% After a specified period of time cortical cells are
% selected at random on regular intervals and given an "F50" tag.
(Cell;E;{F50};*) => <SetTags;{F51,D1}>:                             % #44

(Cell;E;{F50};*) => <EmitAxon;[0,60]; <GenCone;{B1}>>:              % #45
(Cone;{B1};{0.02}) => <SetRuleForce;[0,140,5];1>:

(Cell;E;{F50};*) => <EmitAxon;[60,60]; <GenCone;{B2}>>:
(Cone;{B2};{0.02}) => <SetRuleForce;[60,140,5];1>:

(Cell;E;{F50};*) => <EmitAxon;[120,60]; <GenCone;{B3}>>:
(Cone;{B3};{0.02}) => <SetRuleForce;[120,140,5];1>:

(Cell;E;{F50};*) => <EmitAxon;[180,60]; <GenCone;{B4}>>:
(Cone;{B4};{0.02}) => <SetRuleForce;[180,140,5];1>:

(Cell;E;{F50};*) => <EmitAxon;[240,60]; <GenCone;{B5}>>:
(Cone;{B5};{0.02}) => <SetRuleForce;[240,140,5];1>:

(Cell;E;{F50};*) => <EmitAxon;[300,60]; <GenCone;{B6}>>:
(Cone;{B6};{0.02}) => <SetRuleForce;[300,140,5];1>:

% LGF Rules; duration of 10000 means LGF lasts throughout simulation.
(Cell;E;{D03,D1,D2};{0.05}) => <SetLGF;-;10;10000>:                 % #57
(Cell;E;{C03,C1,C2};{0.05}) => <SetLGF;-;10;10000>:
```

Appendix D

Base Rule Set Used in the Robustness Experiments

The base rule set used in the Robustness Experiments (Sec. 4.2.3) is listed below. The rules are shown as they appear in the ASCII text file that is read, parsed, and then compiled by the simulator at runtime. All characters on a line following a % symbol are comments. Each rule is terminated by a colon.

The initial state of the simulations (trials) of the Robustness Experiments consisted of a single afferent cell positioned at $[x, y, z] = [0, 0, -10]$, a group of 300 cells, some of which were targets for growing axons, positioned in the plane $z = 10.0$, and 250 obstacle cells positioned between the planes $z = -8.0$ and $z = 8.0$. The afferent cell had $CellType$ = A, and $TagSet$ = {A1}. The 300 cells organized into a layer each had $CellType$ = E, and $TagSet$ = {L1}. The obstacle cells each had $CellType$ = I, and $TagSet$ = {O1}. For each of the cells, $Life = 0$, and $LocalTime = 0.0$.

```
% The base rule set used in the robustness experiments.

% Duration of 1000 means LGF lasts throughout simulation.
(Cell;E;{L1};{0.1}) |= <SetLGF;A;30;1000>:

(Cell;A;{A1};{0.1}) |= <EmitAxon;[0,0]; <GenCone;{H1}>>:


% BEGIN: Rules for growth cone divisions.

(Cone;{H1};{0.2}) |= <DivCone;[0,90]; <GenCone;{H2,D0,G2}>>:
(Cone;{H1};{0.2}) |= <DivCone;[90,90]; <GenCone;{H2,D90,G2}>>:
(Cone;{H1};{0.2}) |= <DivCone;[180,90]; <GenCone;{H2,D180,G2}>>:
```

```
(Cone;{H1};{0.2}) |= <DivCone;[270,90]; <GenCone;{H2,D270,G2}>>:


(Cone;{H1};{1.0}) |= <DivCone;[0,90]; <GenCone;{H5,D0,G5}>>:
(Cone;{H1};{1.0}) |= <DivCone;[90,90]; <GenCone;{H5,D90,G5}>>:
(Cone;{H1};{1.0}) |= <DivCone;[180,90]; <GenCone;{H5,D180,G5}>>:
(Cone;{H1};{1.0}) |= <DivCone;[270,90]; <GenCone;{H5,D270,G5}>>:


(Cone;{H2};{0.8}) |= <DivCone;[0,90]; <GenCone;{H3,D0,G3}>>:
(Cone;{H2};{0.8}) |= <DivCone;[90,90]; <GenCone;{H3,D90,G3}>>:
(Cone;{H2};{0.8}) |= <DivCone;[180,90]; <GenCone;{H3,D180,G3}>>:
(Cone;{H2};{0.8}) |= <DivCone;[270,90]; <GenCone;{H3,D270,G3}>>:


(Cone;AND{H3,D0};{0.8}) |= <DivCone;[90,90]; <GenCone;{H4,D90,G4}>>:
(Cone;AND{H3,D0};{0.8}) |= <DivCone;[270,90]; <GenCone;{H4,D270,G4}>>:

(Cone;AND{H3,D90};{0.8}) |= <DivCone;[180,90]; <GenCone;{H4,D180,G4}>>:
(Cone;AND{H3,D90};{0.8}) |= <DivCone;[0,90]; <GenCone;{H4,D0,G4}>>:

(Cone;AND{H3,D180};{0.8}) |= <DivCone;[270,90]; <GenCone;{H4,D270,G4}>>:
(Cone;AND{H3,D180};{0.8}) |= <DivCone;[90,90]; <GenCone;{H4,D90,G4}>>:

(Cone;AND{H3,D270};{0.8}) |= <DivCone;[0,90]; <GenCone;{H4,D0,G4}>>:
(Cone;AND{H3,D270};{0.8}) |= <DivCone;[180,90]; <GenCone;{H4,D180,G4}>>:


(Cone;AND{H5,D0};{0.1}) |= <DivCone;[0,90]; <GenCone;{H6,D0,G6}>>:

(Cone;AND{H5,D90};{0.1}) |= <DivCone;[90,90]; <GenCone;{H6,D90,G6}>>:

(Cone;AND{H5,D180};{0.1}) |= <DivCone;[180,90]; <GenCone;{H6,D180,G6}>>:

(Cone;AND{H5,D270};{0.1}) |= <DivCone;[270,90]; <GenCone;{H6,D270,G6}>>:

% END: Rules for growth cone divisions.


% BEGIN: Rules governing rule-based force for growth cones.

% This rule not varied in the robustness experiments.
(Cone;{H1};{0.1}) |= <SetRuleForce;[0,0,40];1000>:

(Cone;AND{D0,G2};{0.1}) |= <SetRuleForce;[180,8,30];1000>:
(Cone;AND{D90,G2};{0.1}) |= <SetRuleForce;[270,8,30];1000>:
(Cone;AND{D180,G2};{0.1}) |= <SetRuleForce;[0,8,30];1000>:
(Cone;AND{D270,G2};{0.1}) |= <SetRuleForce;[90,8,30];1000>:

(Cone;AND{D0,G3};{0.1}) |= <SetRuleForce;[0,8,30];1000>:
(Cone;AND{D90,G3};{0.1}) |= <SetRuleForce;[90,8,30];1000>:
(Cone;AND{D180,G3};{0.1}) |= <SetRuleForce;[180,8,30];1000>:
(Cone;AND{D270,G3};{0.1}) |= <SetRuleForce;[270,8,30];1000>:
```

```
(Cone;AND{D0,G4};{0.1}) |= <SetRuleForce;[0,8,30];1000>:
(Cone;AND{D90,G4};{0.1}) |= <SetRuleForce;[90,8,30];1000>:
(Cone;AND{D180,G4};{0.1}) |= <SetRuleForce;[180,8,30];1000>:
(Cone;AND{D270,G4};{0.1}) |= <SetRuleForce;[270,8,30];1000>:

(Cone;AND{D0,G5};{0.1}) |= <SetRuleForce;[0,8,30];1000>:
(Cone;AND{D90,G5};{0.1}) |= <SetRuleForce;[90,8,30];1000>:
(Cone;AND{D180,G5};{0.1}) |= <SetRuleForce;[180,8,30];1000>:
(Cone;AND{D270,G5};{0.1}) |= <SetRuleForce;[270,8,30];1000>:

(Cone;AND{D0,G6};{0.1}) |= <SetRuleForce;[0,8,30];1000>:
(Cone;AND{D90,G6};{0.1}) |= <SetRuleForce;[90,8,30];1000>:
(Cone;AND{D180,G6};{0.1}) |= <SetRuleForce;[180,8,30];1000>:
(Cone;AND{D270,G6};{0.1}) |= <SetRuleForce;[270,8,30];1000>:

% END: Rules governing rule-based force for growth cones.
```

Bibliography

[1] Antonelo, E., Schrauwen, B. and Stroobandt, D. (2008). Event detection and localization for small mobile robots using reservoir computing. *Neural Networks*, 21, 862-871.

[2] Arbuckle, D. and Requicha, A. (2004). Active self-assembly. In *Proceedings of the IEEE international conference on robotics and automation (ICRA '04)* (pp. 896-901). New York: IEEE.

[3] Astor, J. C. and Adami, C. (2000). A developmental model for the evolution of artificial neural networks. *Artificial Life*, 6, 189-218.

[4] van den Bergh, F. (1999). Particle swarm weight initialization in multi-layer perceptron artificial neural networks. In *Proceedings of the international conference on artificial intelligence* (pp. 42-45). CSREA Press.

[5] Bishop, J., Burden, S., Klavins, E., et al. (2005). Programmable parts: A demonstration of the grammatical approach to self-organization. In *Proceedings of the IEEE international conference on intelligent robots and systems (IROS '05)* (pp. 3684-3691). New York: IEEE.

[6] Bitam, S., Batouche, M. and Talbi, E. (2010). A survey on bee colony algorithms. In *Proceedings of the 2010 IEEE international symposium on parallel & distributed processing, workshops and Phd forum (IPDPSW '10)* (pp. 1-8). New York: IEEE.

[7] Bonabeau, E., Theraulaz, G., Deneubourg, J-L., Franks, N., Rafelsberger, O., Joly, J.-L. and Blanco, S. (1998). A model for the emergence of pillars, walls and royal chambers in termite nests. *Philosophical Transactions of The Royal Society*, 353, 1561-1576.

[8] Bonabeau, E., Dorigo, M. and Theraulaz G. (1999). *Swarm intelligence: From natural to artificial systems*. New York, NY: Oxford University Press.

[9] Braendler, D. and Hendtlass, T. (2002). The suitability of particle swarm optimization for training neural hardware. In *Proceedings of the fifteenth international conference on industrial and engineering, applications of artificial intelligence and expert systems* (pp. 190-199). Springer-Verlag.

[10] Brown, T., Kairiss, E. and Keenan, C. (1990). Hebbian Synapses: Biophysical mechanisms and algorithms. *Annual Review of Neuroscience*, 13, 475-511.

[11] Buckley, C., Fine, P., Bullock, S. and Di Paolo, E. (2008). Monostable controllers for adaptive behaviour. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5040, 103-112.

[12] Buhl, J., Sumpter, D., Couzin, I., Hale, J., Despland, E., Miller, E. and Simpson, S. (2006). From disorder to order in marching locusts. *Science*, 312, 1402-1406.

[13] Cai, X., Zhang, N., Venayagamoorthy, G. and Wunsch II., D. (2004). Time series prediction with recurrent neural networks using a hybrid PSO-EA algorithm. In *Proceedings of the international joint conference on neural networks (IJCNN '04)* (pp. 1647-1652). New York: IEEE.

[14] Cangelosi, A., Parisi, D. and Nolfi, S. (1994). Cell division and migration in a "genotype" for neural networks. *Network: Computation in Neural Systems*, 5, 497-515.

[15] Carvalho, M. and Ludermir, T. (2007). Particle swarm optimization of neural network architectures and weights. In *Proceedings of the 7th international conference on hybrid intelligent systems (HIS '07)* (pp. 336-339). New York: IEEE.

[16] Cavagna, A. and Giardina, I. (2010). Large-scale behaviour in animal groups. *Behavioural Processes*, 84, 653-656.

[17] Chval, J. (2002). Evolving artificial neural networks by means of evolutionary algorithms with L-systems based encoding (Research Report). Prague, Czech Republic: Czech Technical University.

[18] Chauvin, Y. and Rumelhart, D., (Eds.) (1995). *Backpropagation: theory, architectures, and applications*. Hillsdale, NJ: Lawrence Erlbaum.

[19] Delgado, A. (2000). Control of nonlinear systems using a self-organizing neural network. *Neural Computing & Applications*, 9(2), 113-123.

[20] Deneubourg, J.-L., Goss, S., Franks, N., Pasteels, J.-M. (1989). The blind leading the blind: Modelling chemically mediated army ant raid patterns. *Journal of Insect Behavior*, 2, 719-725.

[21] Deneubourg, J.-L., Aron, S., Goss, S. and Pasteels, J.-M. (1990). The self-organizing exploratory pattern of the Argentine ant. *Journal of Insect Behavior*, 3, 159-168.

[22] Dorffner, G. (1996). Neural networks for time series processing. *Neural Network World*, 6, 447-468.

[23] Dorigo, M., Caro, G. and Gambardella, L. (1999). Ant algorithms for discrete optimization. *Artificial Life*, 5, 137-172.

[24] Dussutour, A., Fourcassié, V., Helbing, D. and Deneubourg, J.-L. (2004). Optimal traffic organization in ants under crowded conditions. *Nature*, 428, 70-73.

[25] Eggenberger, P. (1997). Creation of neural networks based on developmental and evolutionary principles. In W. Gerstner, A. Germond, M. Hasler, J. Nicoud (Eds.), *Proceedings of the international conference on artificial neural networks (ICANN '97)* (pp. 337-342). Berlin/Heidelberg, Germany: Springer.

[26] Elizondo, E., Birkenhead, R., Góngora, M., et al. (2007). Analysis and test of efficient methods for building recursive deterministic perceptron neural networks. *Neural Networks*, 20, 1095-1108.

[27] Elman, J. (1990). Finding structure in time. *Cognitive Science*, 14, 179-211.

[28] Engelbrecht, A. (2005). *Fundamentals of computational swarm intelligence*. West Sussex, England: John Wiley & Sons Ltd.

[29] Engelbrecht, A. and Ismail, A. (1999). Training product unit neural networks. *Stability and Control: Theory and Applications*, 2, 59-74.

[30] Fahlman, S. and Lebiere, C. (1990). The cascade-correlation learning architecture. In D. S. Touretzky (Ed.), *Advances in neural information processing systems II* (pp. 524-532). San Francisco, CA: Morgan Kaufmann.

[31] Farkaš, I. and Miikkulainen, R. (1999). Modeling the self-organization of directional selectivity in the primary visual cortex. In Willshaw, D. and Murray, A. (Eds.), *Proceedings of the international conference on artificial neural networks (ICANN '99)* (pp. 251-256). London, England: IEE.

[32] Filho, C., Neto, F., Lins, A., Nascimento, A. and Lima, M. (2008). A novel search algorithm based on fish school behavior. In *Proceedings of the IEEE international conference on systems, man and cybernetics (SMC '08)* (pp. 2646-2651). New York: IEEE.

[33] Fleischer, K. and Barr, A. (1994). A simulation testbed for the study of multicellular development: The multiple mechanisms of morphogenesis. In C.G. Langton (Ed.), *Artificial life III (Vol. XVII of the SFI studies in the science of complexity)* (pp. 389-416). Redwood City, CA: Addison-Wesley.

[34] Fleischer, K. (1995). *A multiple-mechanism developmental model for defining self-organizing geometric structures* (Dissertation). Pasadena, CA: California Institute of Technology.

[35] Franks, N., Gomez, N., Goss, S. and Deneubourg, J.-L. (1991). The blind leading the blind in army ant raid patterns: Testing a model of self-organization (Hymenoptera: Formicidae). *Journal of Insect Behavior*, 4, 583-607.

[36] Frean, M. (1990). The upstart algorithm: A method for constructing and training feedforward neural networks. *Neural Computation*, 2, 198-209.

[37] Gaing, Z. (2003). Particle swarm optimization to solving the economic dispatch considering the generator constraints. *IEEE Transactions on Power Systems*, 18, 1187-1195.

[38] Gao, Y. and Er, M. (2005). NARMAX time series model prediction: feedforward and recurrent fuzzy neural network approaches. *Fuzzy Sets and Systems*, 150, 331-350.

[39] Gaspar, J. *Wasp March 2008-3*. 13 Mar. 2008, commons.wikimedia.org/wiki/Category:Pictures_and_images.

[40] Gaspar, J. *Wasp March 2008-10*. 12 Apr. 2008, commons.wikimedia.org/wiki/Category:Pictures_and_images.

[41] Ge, H., Liang, Y. and Marchese, M. (2007). A modified particle swarm optimization-based dynamic recurrent neural network for identifying and controlling nonlinear systems. *Computers and Structures*, 85, 1611-1622.

[42] Goodhill, G., Gu, M., and Urbach, J. (2004). Predicting axonal response to molecular gradients with a computational model of filopodial dynamics. *Neural Computation*, 16, 2221-2243.

[43] Goodhill, G. and Xu, J. (2005). The development of retinotectal maps: A review of models based on molecular gradients. *Network*, 16, 5-34.

[44] Gracias, D., Tien, J., Breen, T., Hsu, C., and Whitesides G. (2000). Forming electrical networks in three dimensions by self-assembly. *Science*, 289, 1170-1172.

[45] Gross, R., Bonani, M., Mondala, F., Dorigo, M. (2006). Autonomous self-assembly in swarm-bots. *IEEE Transactions on Robotics*, 22, 1115-1130.

[46] Grove, E. and Fukuchi-Shimogori, T. (2003). Generating the cerebral cortical area map. *Annual Review of Neuroscience*, 26, 355-380.

[47] Gruau, F. (1993). Genetic synthesis of modular neural networks. In S. Forest (Ed.), *Proceedings of the 5th international conference on genetic algorithms (ICGA '93)* (pp. 318-325). San Francisco, CA: Morgan Kaufmann.

[48] Gruau, F., Whitley, D., and Pyeatt, L. (1996). A comparison between cellular encoding and direct encoding for genetic neural networks. In *Proceedings of the 1st annual conference on genetic programming (GECCO '96)* (pp. 81-89). Cambridge, MA: MIT Press.

[49] Grushin, A., Reggia, J. (2006). Stigmergic self-assembly of prespecified artificial structures in a constrained and continuous environment. *Integrated Computer-Aided Engineering*, 13, 289-312.

[50] Grushin, A., Reggia, J. (2008). Automated design of distributed control rules for the self-assembly of pre-specified artificial structures. *Robotics and Autonomous Systems*, 56, 334-359.

[51] Gueron, S., Levin, S. and Rubenstein, D. (1996). The dynamics of herds: From individuals to aggregations. *Journal of Theoretical Biology*, 182, 85-98.

[52] Gunduz, A., Ozturk, M. and Sanchez, J. (2007). Echo state networks for motor control of human ECoG neuroprosthetics. In *Proceedings of the 3rd international IEEE/EMBS conference on neural engineering (CNE '07)* (pp. 514-517). New York: IEEE.

[53] Haessly, A., Sirosh, J. and Miikkulainen, R. (1995). A model of visually guided plasticity of the auditory spatial map in the barn owl. In J. F. Lehman and J. D. Moore (Eds.), *Proceedings of the 17th annual meeting of the Cognitive Science Society* (pp. 154-158). Hillsdale, NJ: Lawrence Earlbaum.

[54] Hartland, C. Bredeche, N. and Sebag, M. (2009). Memory-enhanced evolutionary robotics: The echo state network approach. In *Proceedings of the IEEE congress on evolutionary computation (CEC '09)* (pp. 2788-2795). New York: IEEE.

[55] Hartland, C. and Bredeche, N. (2007). Using echo state networks for robot navigation behavior acquisition. In *Proceedings of the IEEE international conference on robotics and biomimetics (ROBIO '07)* (pp. 201-206). New York: IEEE.

[56] Hassibi, B., Stork, D. and Wolff, G. (1993). Optimal brain surgeon and general network pruning. In *Proceedings of the IEEE international conference on neural networks (ICNN '93)*, vol. 1, (pp. 293-299). New York: IEEE.

[57] Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2nd ed.). Upper Saddle River, NJ: Prentice-Hall.

[58] Helbing, D., Farkas, I. and Vicsek, T. (2000). Simulating dynamical features of escape panic. *Nature*, 407, 487-490.

[59] Hentschel, H. and van Ooyen, A. (1999). Models of axon guidance and bundling during development. *Proceedings of the Royal Society (London) B*, 266, 2231-2238.

[60] Hildenbrandt, H. Carere, C. and Hemelrijk, C. (2010). Self-organized aerial displays of thousands of starlings: A model. *Behavioral Ecology*, 21, 1349-1359.

[61] Hollldobler, B. and Wilson, E. (1990). *The ants*. Cambridge, MA: Harvard University Press.

[62] Holzmann, G. and Hauser, H. (2010). Echo state networks with filter neurons and a delay&sum readout. *Neural Networks*, 23, 244-256.

[63] Hotz, P., Gomez, G. and Pfeifer, R. (2002). Evolving the morphology of a neural network for controlling a foveating retina - and its test on a real robot. In Standish, R., Bedau, M. and Abbass, H. (Eds.), *Artificial life VIII: Proceedings of the eighth international conference on artificial life* (pp. 243-251). Cambridge, MA: MIT Press.

[64] Honda, H. (2003). Competition between retinal ganglion axons for targets under the servomechanism model. *Journal of Neuroscience*, 23, 10368-10377.

[65] Itoh, Y. and Adachi, M. (2010). Chaotic time series prediction by combining echo-state networks and radial basis function networks. In *Proceedings of the 2010 IEEE international workshop on machine learning for signal processing (MLSP '10)* (pp.238-243). New York: IEEE.

[66] Jaeger, H. (2001). The echo state approach to analysing and training recurrent neural networks. *Fraunhofer Institute for Autonomous Intelligent Systems (AIS)*, GMD Report 148.

[67] Jaeger, H. and Hass, H. (2004). Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304, 78-80.

[68] Jiang, F., Berry, H. and Schoenauer, M. (2008). Supervised and evolutionary learning of echo state networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5199, 215-224.

[69] Jing D., Venayagamoorthy, G. and Harley, R. (2009). Harmonic identification using an echo state network for adaptive control of an active filter in an electric ship. In *Proceedings of the international joint conference on neural networks (IJCNN '09)* (pp. 1098-7576). New York: IEEE.

[70] Jones, C. and Matarić, M. (2003). From local to global behavior in intelligent self-assembly. In *Proceedings of the IEEE international conference on robotics and automation (ICRA '03)* (pp. 721-726). New York: IEEE.

[71] Juang, C. (2004). A hybrid of genetic algorithm and particle swarm optimization for recurrent network design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34, 997-1006.

[72] Jung, J. and Reggia, J. (2006). Evolutionary design of neural network architectures using a descriptive encoding language. *IEEE transactions on evolutionary computation*, 10, 676-688.

[73] Kalay, A., Parnas, H., Shamir, E. (1995). Neuronal growth via hybrid system of self-growing and diffusion based grammar rules: I. *Bulletin of Mathematical Biology*, 57, 205-227.

[74] Karaboga, D. and Basturk, B. (2008). On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, 8, 687-697.

[75] Karsai, I. (1999). Decentralized control of construction behavior in paper wasps: An overview of the stigmergy approach. *Artificial Life*, 5, 117-136.

[76] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of the IEEE international conference on neural networks* (pp. 1942-1948). New York: IEEE.

[77] Kiranyaz, S., Ince, T., Yildirim, A. and Gabbouj, M. (2009). Evolutionary artificial neural networks by multi-dimensional particle swarm optimization. *Neural Networks*, 22, 1448-1462.

[78] Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4, 461-476.

[79] Klavins, E. (2007). Programmable self-assembly. *IEEE Control Systems Magazine*, 27, 43-56.

[80] Klavins, E., Ghrist, R., and Lipsky, D. (2004). Graph grammars for self-assembling robotic systems. In *Proceedings of the IEEE International conference on robotics and automation (ICRA '04)* (pp. 5293-5300). New York: IEEE.

[81] Kohonen, T. (2001). *Self-organizing maps*. New York, NY: Springer.

[82] Kollmann, D. *Fischschwarm*. 13 Jun. 2011, commons.wikimedia.org/wiki/Category:Pictures_and_images.

[83] Kunz, H. and Hemelrijk, C. (2003). Artificial fish schools: Collective effects of school size, body size, and body form. *Artificial Life*, 9, 237-253.

[84] Lapizco-Encinas, G., Kingsford, C., and Reggia, J. (2009). A cooperative combinatorial particle swarm optimization algorithm for side-chain packing. In *Proceedings of the IEEE swarm intelligence symposium (SIS '09)* (pp. 22-29). New York: IEEE.

[85] LeCun, Y., Denker, J., Solla, S. (1990). Optimal brain damage. In D. Touretzky (Ed.), *Advances in neural information processing systems II* (pp. 598-605). San Francisco, CA: Morgan Kaufmann.

[86] Lendasse, A., Verleysen, M., de Bodt, E., Gregoire, P., and Cottrell, M. (1998). Forecasting time-series by Kohonen classification. In M. Verleysen (Ed.), *Proceedings of the 6th European symposium on artificial neural networks (ESANN '98)* (pp. 221-226). Brussels, Belgium: D-Facto public.

[87] Lett, C. and Mirabet, V. (2008). Modelling the dynamics of animal groups in motion. *South African Journal of Science*, 104, 192-198.

[88] Lin, X., Yang, Z. and Song, Y. (2009). Short-term stock price predication based on echo state networks. *Expert Systems with Applications*, 36, 7313-7317.

[89] Lopez-Bendito, G. and Molnar, Z. (2003). Thalamocortical development: How are we going to get there? *Nature Reviews Neuroscience*, 4, 276-289.

[90] von der Malsburg, C. (1973). Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14, 85-100.

[91] Marchand, M., Golea, M. and Rujan, P. (1990). A convergence theorem for sequential learning in two-layer perceptrons. *Europhysics Letters*, 11(6), 487-492.

[92] Martin, C. and Reggia, J. (2010). Self-assembly of neural networks viewed as swarm intelligence. *Swarm Intelligence*, 4, 1-36. DOI: 10.1007/s11721-009-0035-7.

[93] Nembrini, J., Reeves, N., Poncet, E., et al. (2005). Flying swarm intelligence for architectural research. In *Proceedings of the IEEE swarm intelligence symposium (SIS '05)* (pp. 225-232). New York: IEEE.

[94] Omran, M., Engelbrecht, A. and Salman, A. (2005). Particle swarm optimization method for image clustering. *International Journal on Pattern Recognition and Artificial Intelligence*, 19, 297-322.

[95] van Ooyen, A. (1994). Activity-dependent neural network development. *Network: Computation in Neural Systems*, 5, 401-423.

[96] Oubbati, M, Kord, B. and Palm, G. (2010). Learning robot-environment interaction using echo state networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6226, 501-510.

[97] Ozturk, M. and Principe, J. (2007). An associative memory readout for ESNs with applications to dynamical pattern recognition. *Neural Networks*, 20, 377-390.

[98] Payton, D., Estkowski, R. and Howard, M. (2005). Pheromone robotics and the logic of virtual pheromones. *Lecture Notes in Computer Science*, 3342, 45-57.

[99] Pearlmutter, B. (1989). Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1, 263-269.

[100] Pearson, J., Finkel, L. and Edelman, G. (1987). Plasticity in the organization of adult cerebral cortical maps: A computer simulation based on neuronal group selection. *The Journal of Neuroscience*, 7, 4209-4223.

[101] Prokhorov, D. (2005). Echo state networks: Appeal and challenges. In *Proceedings of the international joint conference on neural networks (IJCNN '05)* (pp. 1463-1466). New York: IEEE.

[102] Prusinkiewicz, P. and Lindenmayer, A. (1990). *The algorithmic beauty of plants*. New York, NY: Springer.

[103] Pulakka, K. and Kujanpa, V. (1998). Rough level path planning method for a robot using SOFM neural network. *Robotica*, 16, 415-423.

[104] Puskorius, G., Feldkamp, L. and Davis, L. (1996). Dynamic neural network methods applied to on-vehicle idle speed control. *Proceedings of the IEEE*, 84, 1407-1420.

[105] Rae, A. *Red-billed quelea flocking at waterhole*. 11 Jun. 2007, commons.wikimedia.org/wiki/Category:Pictures_and_images.

[106] Rahmat-Samii, Y., Gies, D. and Robinson, J. (2003). Particle swarm optimization (PSO): A novel paradigm for antenna designs. *The Radio Science Bulletin*, 305, 14-22.

[107] Reggia, J. and Martin, C. (2009). *Self-assembly of a neural network*. College Park, MD: Univ. of Maryland, Dept. of Computer Science. http://www.cs.umd.edu/~reggia/martin.html.

[108] Reynolds, C. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4), 25-34.

[109] Ritter, H., Martinetz, T. and Schulten K. (1992). *Neural computation and self-organizing maps*. Reading, MA: Addison-Wesley.

[110] Rodriguez, A. and Reggia, J. (2004). Extending self-organizing particle systems to problem solving. *Artificial Life*, 10, 379-395.

[111] Rust, A., Adams, R., Schilstra, M. and Bolouri, H. (2003). Evolving computational neural systems using synthetic developmental mechanisms. In S. Kumar and P. Bentley (Eds.), *On growth, form and computers* (pp. 353-376). New York, NY: Academic Press.

[112] Şahin, E., Girgin, S., Bayindir, L. and Turgut, A. (2008). Swarm robotics. In C. Blum and D. Merkle (Eds.), *Swarm intelligence* (pp. 87-100). Berlin, Heidelberg: Springer.

[113] Salmen, M. and Ploger, P. (2005). Echo state networks used for motor control. In *Proceedings of the 2005 IEEE international conference on robotics and automation (ICRA '05)* (pp. 1953-1958). New York: IEEE.

[114] Shukla, A., Tiwari, R. and Kala, R. (2010). *Towards hybrid and adaptive computing: A perspective*. Studies in computational intelligence, 307. Berlin, Heidelberg: Springer-Verlag.

[115] Siegelmann, H., Horne, B. and Giles, C. (1997). Computational capabilities of recurrent NARX neural networks. *Systems, Man, and Cybernetics, Part B: Cybernetics*, 27, 208-215.

[116] Sperati, V. Trianni, V. and Nolfi, S. (2010). Evolution of self-organised path formation in a swarm of robots. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6234, 155-166.

[117] Spitzer N. (2006). Electrical activity in early neuronal development. *Nature*, 444, 707-712.

[118] Sousa, T., Silva, A. and Neves, A. (2004). Particle swarm based data mining algorithms for classification tasks. *Parallel Computing*, 30, 767-783.

[119] Sutton, G., Reggia, J., Armentrout, S., D'Autrechy, C. (1994). Cortical map reorganization as a competitive process. *Neural Computation*, 6, 1-13.

[120] Theraulaz, G. and Bonabeau, E. (1999). A brief history of stigmergy. *Artificial Life*, 5, 97-116.

[121] Tong, M., Bickett, A., Christiansen, E. and Cottrell, G. (2007). Learning grammatical structure with echo state networks. *Neural Networks*, 20, 424-432.

[122] Tsukada, T., Tamura, T., Kitagawa, S. and Fukuyama, Y. (2003). Optimal operational planning for cogeneration system using particle swarm optimization. In *Proceedings of the IEEE swarm intelligence symposium (SIS '03)* (pp. 138-143). New York: IEEE.

[123] Vesanto, J. (1999). SOM-based data visualization methods. *Intelligent Data Analysis*, 3, 111-126.

[124] Vesanto, J. and Alhoniemi, E. (2000). Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3), 586-600.

[125] Voss, M. and Howland III., J. (2003). Financial modelling using social programming. In *The IASTED International Conference on Financial Engineering & Applications* (pp.1-10). Calgary, Canada: ACTA Press.

[126] Wang, K., Huang, L., Zhou, C., and Pang, W. (2003). Particle swarm optimization for traveling salesman problem. In *Proceedings of the international conference on machine learning and cybernetics* (pp. 15831585). New York: IEEE.

[127] Weigend, A., Rumelhart, D. and Huberman, B. (1991). Generalization by weight-elimination with application to forecasting. In *Advances in neural information processing systems, vol. 3* (pp. 875-882). Mateo, CA: Morgan Kaufmann.

[128] Werfel, J. and Nagpag, R. (2006). Extended stigmergy in collective construction. *IEEE Intelligent Systems*, 21, 20-28.

[129] White, P., Zykov, V., Bongard, J. and Lipson, H. (2005). Three dimensional stochastic reconfiguration of modular robots. In *Proceedings of robotics: Science and systems* (pp. 161-168). Cambridge, MA: MIT Press.

[130] Whitehead, B. and Choate, T. (1996). Cooperative-Competitive genetic evolution of radial basis function centers and widths for time series prediction. *IEEE Transactions on Neural Networks*, 7, 869-880.

[131] Whitesides, G. and Gzybowski, B. (2002). Self-assembly at all scales. *Science*, 295, 2418-2421.

[132] Wieland, A. (1991). Evolving neural network controllers for unstable systems. In *Proceedings of the international joint conference on neural networks (IJCNN '91)*, vol. 2, (pp. 667-673). New York: IEEE.

[133] Xu, D., Lan, J. and Principe, J. (2005). Direct adaptive control: An echo state network and genetic algorithm approach. In *Proceedings of the international joint conference on neural networks (IJCNN '05)* (pp. 1483-1486). New York: IEEE.

[134] Xue, Y., Yang, L. and Haykin, S. (2007). Decoupled echo state networks with lateral inhibition. *Neural Networks*, 20, 365376.

[135] Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87, 1423-1447.

[136] Yates, P., Holub, A., McLaughlin, T., et al. (2004). Computational modeling of retinotopic map development to define contributions of EphA-EphrinA gradients, axon-axon interactions, and patterned activity. *Journal of Neurobiology*, 59, 95-113.

[137] Zhang, C. and Shao, H. (2000). An ANNs evolved by a new evolutionary system and its application. In *Proceedings of the 39th IEEE conference on decision and control* (pp. 3562-3563). New York: IEEE.

[138] Zhang, J., Zhang, J., Lok, T. and Lyu, M. (2007). A hybrid particle swarm optimization-back-propagation algorithm for feedforward neural network training. *Applied Mathematics and Computation*, 185, 1026-1037.

[139] Zhao, L. and Yang, Y. (2009). PSO-based single multiplicative neuron model for time series prediction. *Expert Systems with Applications*, 36, 2805-2812.