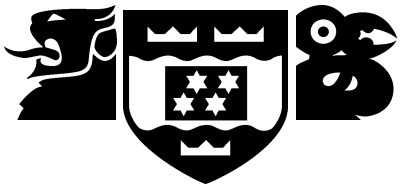
VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui



Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@mcs.vuw.ac.nz

Evolution of Topographical Mappings in the Brain

Jeromé Dolman

Supervisor: Marcus Frean

October 24, 2003

Submitted in partial fulfilment of the requirements for
Bachelor of Science with Honours in Computer Science.

Abstract

The brain contains many areas that are connected together by topographical mappings (where neighbours in one region project to neighbours in another region), yet the mechanisms by which they form are still not fully understood. The retinotectal projection contains one of these mappings, and provides a good basis for studying as extensive study has been conducted on it.

This project aims to find some mechanisms for the formation of these mappings by means of computational modelling and heuristic search. We propose a new biologically plausible representation for these mechanisms and show that it is capable of representing the Tea Trade Model, a known mechanism. We also find a novel new model in this representation via automated search.

Acknowledgements

I would like to pass my many thanks to the following people:

- First and foremost to my supervisor Marcus Frean for all his advice and guidance throughout the course of the project. A special note of thanks for his help formulating the new representation.
- To Donald Gordon and Justin Rens for always being there to answer technical questions and bounce various ideas off.
- To Bunna Ny for all his help proof-reading this report.
- To the inhabitants of Memphis – Annie, Justin, Donald, Bunna, Simon, Urvesh, Ben, Will and Richard – for their company and support during the year.
- To my brother Kashi for his support through the year.

Contents

List of Figures

Chapter 1

Background

The brain consists primarily of cells called *neurons*, richly connected to one another by virtue of their highly elongated cellular processes known as axons (as in Figure 1.1). The tip of the axon branches out to form connections to many other neurons in which the actual connection is called a synapse. The synapse is the small gap between the two cells over which an electro-chemical signal passes as the brain processes information. The pattern of connectivity is not random: rather, it is common for the neurons in one extended region to connect to another region in a *topographical* fashion in which cells that are neighbours in one region project to cells which are neighbours in the other. The best studied of these mappings is in the *retinotectal* system which connects the eye to the visual processing centre of the brain.

Much AI research has focused on how the strength of connections between neurons should vary based on activity, and most algorithms such as Hebbian learning and Self-Organising-Maps fall into this category. However these learning algorithms have only focused on the last stage of neural development, a process which Geoffrey Goodhill describes as [12]:

1. neural induction and neural-tube formation
2. pattern formation and regionalization
3. establishment of connectivity
4. activity-dependent refinement of connections

The first two stages set up the spatial structure of the neurons without any connections, and are not considered here. The third stage of this developmental model sets the stage for all the learning that AI algorithms do, and it is the formation of a topographic mapping in the retinotectal system that provides the focus for this project.

1.1 The Retinotectal System

Behind the retina are neurons known as ganglion cells through which the image seen by the retina is transmitted. The axons from this layer of neurons grow towards the back of the brain (through the optic nerve) and connect to neurons in the optic tectum[1] as shown in Figure 1.2. In order for the visual system to be useful to the animal, these axons must connect in an orderly fashion to the optic tectum such that the image 'seen' on the retina is preserved.

[1]This is the optic processing area found in more primitive creatures – it is simpler and more conducive to study than the optic cortex of humans.

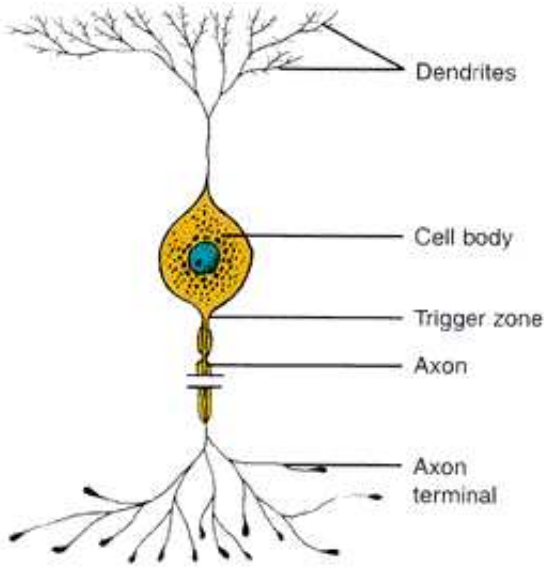


Figure 1.1: A structural representation of a biological neuron. Incoming connections come in to the dendrites; the signal flows to the cell body; when the trigger zone exceeds a certain potential the neuron fires; this signal travels down the axon to become the input to other neurons. *Image adapted from AI@Home [2].*

Although later neural activity refines the map (stage 4 of the above process), it cannot start from scratch but must have a basic mapping to work on – i.e. if this activity stage attempts to refine a randomly initialised mapping, it would be unable to make it topographical. It is the third stage of neural development which sets up a rough mapping which later neural activity will refine. Figure 1.3 shows a diagram of the system along with the orientation that mappings normally form in.

The idea of using chemical markers to assist in formulating these mappings was first proposed by Sperry in 1963 [22] and various models were developed based on this idea (Prestige & Willshaw (1975) [21], Willshaw & Von Der Malsburg (1979) [23], Whitelaw & Cowan (1981), and Gierer (1983;1987)). Sperry proposed that there are gradients of chemical markers in both the retinal and tectal layers, and that it is via these markers that the retinal axons determine where to connect to. In recent years biologists have found chemicals in the brain which play the role of these markers which confirms that marker-based models are indeed on the right track, and gives solid experimental evidence to test them against. These chemicals take the form of the Ephs and ephrins. The Ephs (Erythropoetin-producing hepatocellular) are a member of the tyrosine kinase family of receptors and the ephrins act as ligands ligands[2] to the Ephs[3]. These marker molecules are anchored to the surfaces of the cells, and take the role of providing a signal which indicates whether the synapse between two cells should have a high weight.

For simplicity and tractability, throughout this project I have used a one-dimensional system with a single marker-receptor pair, but it can be scaled up to two-dimensional layers such as a real retina/tectum. In the models described here, the function of these marker chemicals is to give a measure of relative positioning – given the amounts of marker in a set

[2]A ligand is an ion, a molecule, or a molecular group that binds to another chemical entity to form a larger complex.

[3]Historically, these chemical markers were found and named simultaneously by several research groups. Later the Eph Nomenclature Committee was formed to standardize the names [3].

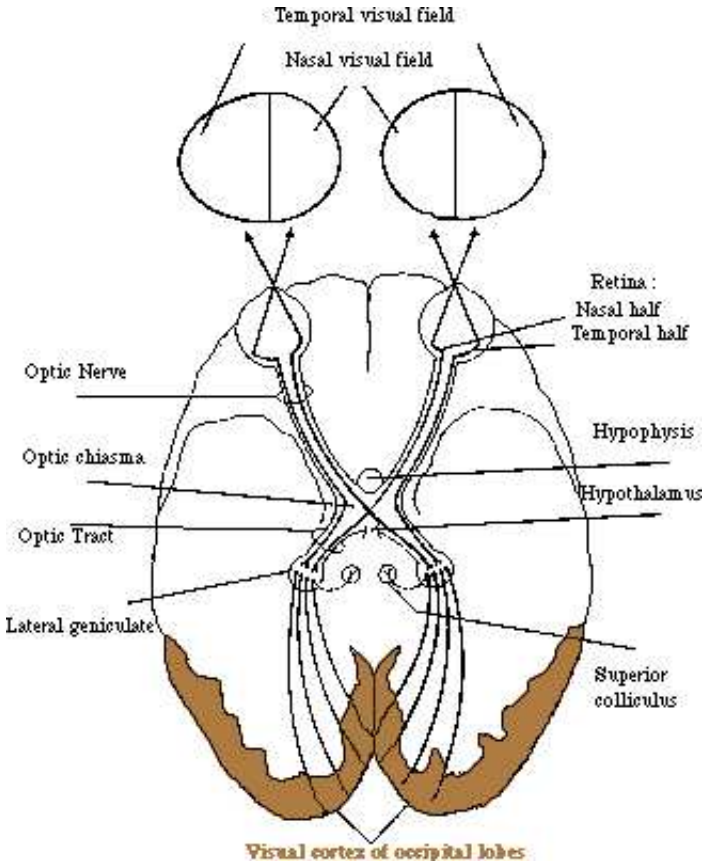


Figure 1.2: The pathways that connect the eye to the brain. *Image adapted from [20]*

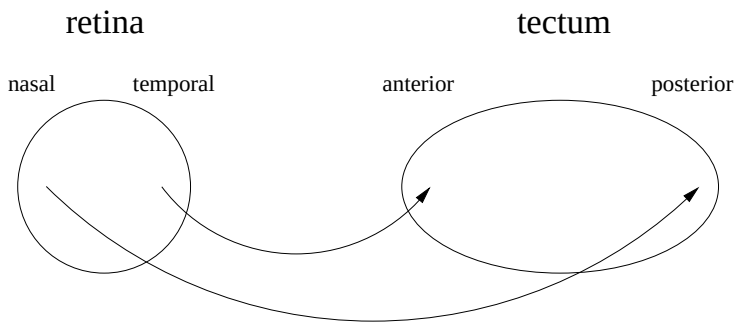


Figure 1.3: The simplified view of the projection from the retina to the tectum as used in this project. Nasal-temporal and anterior-posterior refer to directional axes in the brain.

of cells, you can define an ordering on them.

The system that is used throughout this project is a simplified model of some aspects of the known biology. It is a basic two-layer neural network, where the first layer represents the retina and the second layer represents the tectum. The two layers are completely connected with a weight per connection which represents the synaptic strength (how much signal passes through when the pre-synaptic neuron fires). There is a gradient of retinal marker, denoted R, throughout the retina. In a real brain this might be expected to be an exponential gradient (due to diffusion [10, 11]), but here a linear gradient is used for simplicity [9]. This amount is assumed to be constant within each retinal cell, and is expressed in the growing tip of the axon, thus making it available for use in calculations at the synapse. The tectal cells have certain amounts of tectal marker, denoted L since they are the ligands of the retinal marker. These amounts are affected by the amount of retinal marker in pre-synaptic neurons – this can be either an inhibitory or inductive interaction.

1.2 Topographical Mappings in the Brain

The previous section alluded to ordered, or topographical mappings – a type of mapping that provides a faithful transfer of spatial information. This is vital if, for instance, the same image is to be reproduced at the tectum for neural processing by the animal. Figure 1.4 shows a set of mappings between the two layers in the model retinotectal system; from a set of randomly initialised weights to a perfectly topographical mapping.

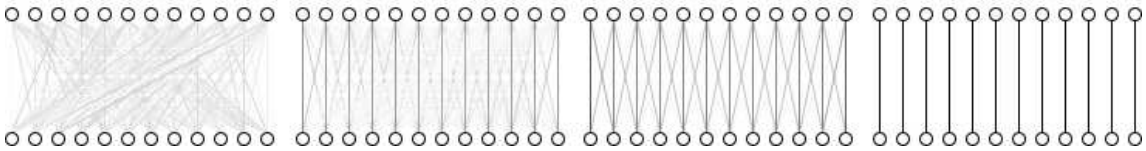


Figure 1.4: A progression from a random mapping (left) to a topographical mapping (right). The two middle ones show more spread out mappings, and the colour of the connection is indicative of it's strength.

These mappings crop up in many areas of the brain, between any two areas where a spatial topography needs to be reproduced. Other than the retinotectal projection, examples of these include the connectivity between the hippocampus and thalamus, in the Vomeronasal projection in the olfactory system, and many more. Since they occur in so many places there is now a lot of study into how they form in the developmental brain, and it is in this role as a mechanism for axon guidance that the Ephs and ephrins (among others) occur most often [4].

Conceptually, creating a mapping of this sort is trivial: for each retinal neuron a connection is made to the neuron at the corresponding location in the tectum – i.e. a neuron 10% of the way along the retina will connect to a neuron 10% of the way along the tectum. This would provide the desired mapping, but it requires omniscient knowledge of the entire network to determine where '10% of the way along the tectum' is. This explicit location information is not thought to be available in a developing brain, making a model such as this one non-biologically plausible. There must however be a way that these mappings are formed in biological systems using only locally available information (such as concentration/presence of chemicals).

Prestige & Willshaw (1975) [21] gave a method involving Sperry's marker theory which operates along these lines. In the early developmental stages the retinal and tectal layers undergo parallel processes of differentiation such that both layers have identical marker gra-

dients, then when the connections form, they do so between neurons with similar marker concentrations. While this is a little more plausible, it seems far-fetched that two independent layers could form identical gradients (especially if they are different sizes).

There is also significant research showing that these mappings can reform after surgical intervention [14]. For instance if the tectum is cut in half, the mapping will change so that the entire retina maps to what is left of the tectum; if the connections are cut and the tectum is reversed half-way through formation the mapping will form backawards etc.

1.3 The Tea Trade Model

This model for topographical mapping formation derives it's name from an anecdote relating it to the colonial trade in tea between India and England: There are various tea plantations scattered around India, each growing a different type of tea. For simplicity, suppose that there are only a few of them and their locations are known. The tea processing plants would then use leaves from all the different plantations in amounts inversely proportional to the distance away. e.g. If there were only two plantations, and a processing plant was located equidistant to both, its blend would be 50:50 of the two plantations; while if it was closer to one its blend would contain more of that one. The final stage is that the blends are exported to England. If a tea connoisseur in England then analyses one of these blends and determines the proportions of different types in it and which plantations these came from, they would then be able to give an exact location of the tea processing factory by triangulation.

This way of identifying spatial location form the basis of the Tea Trade Model (TTM) as proposed in Willshaw & Von Der Malsburg (1979) [23].

The tea type concentrations correspond to marker concentrations, and these can be used to extract spatial information. The model assumes that a gradient of retinal marker is already set up – this could happen earlier (perhaps as the initial topology is formed) by having one or more chemical sources from which the marker diffuses. The amount of tectal marker is induced by the amount of retinal marker at the neuron (a weighted sum of pre-synaptic amounts), and the weight of a connection between two neurons is related to the marker amounts in the pre- and post-synaptic neurons. *Chemical induction* refers to the fact that the amount pre-synaptic retinal marker can affect the amount of tectal marker.

The TTM can be described as the 3-tuple:

$$\langle \Delta L, \delta, \Delta w \rangle$$

where the three elements are the change in tectal marker ΔL, diffusion of tectal marker δ and change in retinal marker ΔR. These are found using Equations 1.1 to 1.3. These contain parameters which can be tweaked for optimal performance: tectal marker update rate ρ, tectal diffusion rate β, weight update rate η and mapping width k. The TTM is normally run with $\rho = 0.3$, $\beta = 0.2$, $\eta = 0.3$ and $k = 0.1$.

$$\Delta L_j = \rho(r_j - L_j) \tag{1.1}$$

$$\delta_j = -\beta(L_{j-1} + L_{j+1} - 2L_j) \tag{1.2}$$

$$\Delta w_{ji} = \eta e^{-\left(\frac{L_j - R_i}{k}\right)^2} \tag{1.3}$$

This model is run in an iterative fashion:

1. First the system is initialised: the retinal gradient is set up; the weights and tectal marker amounts are randomised.

2. The new tectal marker amount is calculated for each tectal cell:

$$L'_j = L_j + \Delta L_j + \delta_j$$

3. The new weight is calculated for each connection:

$$w'_{ji} = w_{ji} + \Delta w_{ji}$$

4. The new weights are normalised by ensuring the sum of weights out of a retinal cell is one:

$$w'_{ji} = \frac{w'_{ji}}{\sum_k w'_{ki}}$$

5. The tectal marker amounts and connection weights are set to these values

6. Steps 2-6 are repeated.

This process is repeated until either stability is achieved or the user stops it. An example of the TTM in action on a network is shown in Figure 1.5.

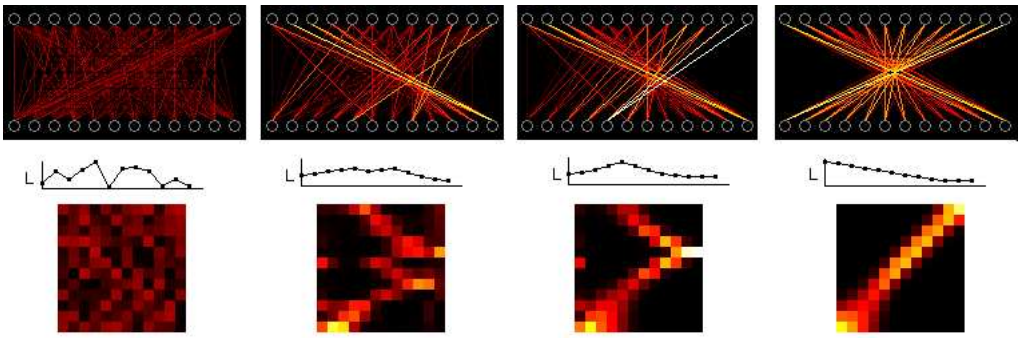


Figure 1.5: A simulation showing a network as the Tea Trade Model is applied to it, starting from a set of randomised weights (left) and ending at a good mapping (right). The top and bottom diagrams show the weights of connections; the middle one shows the amounts of tectal marker in each tectal neuron.

Watching the TTM in action, it becomes apparent that it works by means of an interplay between the weight updates and the tectal marker updates. Once the final mapping has formed it can be seen that it produces strong connection weights between two neurons with similar marker amounts – once the mapping has stabilised the tectal marker is in a gradient matching that of the pre-synaptic retinal marker.

Chapter 2

Evolving Formulae

While the TTM is one mechanism for forming topographical mappings in the brain, it would seem that there could be other mechanisms which also form them. To investigate this, we need a search method and a model representation.

This chapter presents evolutionary computation as a search method and gives a brief history of it, then describes the representation that is used for evolving new models.

2.1 Genetic Programming

When problems become intractable by normal search methods, a new approach is needed. Inspiration has come from nature in the form of evolution as it is so familiar to all of us. Some of the first work in this area was by John Holland et al. at the University of Michigan in the 1960s and 1970s, culminating in Holland's 1975 book *Adaptation in Natural and Artificial Systems* which presented the genetic algorithm as an abstraction of biological evolution [18]. The genetic algorithm (GA) approach it described is a method for creating a new population of individual solutions from an old population by means of Darwinian natural selection and biologically-inspired genetic operators.

Each individual in the population is represented by a *chromosome* which consists of a string of *genes*, each of which is an instance of an *allele*. The allele is the atomic unit, and could be one of the bases in our DNA (A, T, G, C), a bit (0 or 1) or a character depending on the domain. The new population of individuals (chromosomes) are created through biologically-inspired selection operators of *crossover*, *mutation* and *inversion*. They take individuals from the old population that have been selected to reproduce via a selection process that will choose fitter individuals more often than the weaker ones. The crossover operator exchanges sub-sequences of two individual's chromosomes to create a new chromosome, roughly imitating biological recombination between two haploid (single-chromosome) organisms; mutation randomly changes allele values for genes; and inversion reverses subsequences of genes.

GAs are fine if the model being searched can easily be represented as a non-structured string, but often there is more structure to a model that cannot be represented safely as a string. For example, a function can be written as $sin(y/x)$, but bit-wise or character-based mutations could make it un-parsable and meaningless. In his 1992 book John Koza proposed Genetic Programming (GP) as an extension to GAs in which the genes are replaced by program structures [16]. These structures are most often represented as lisp-like expression trees, and the genetic operators of *crossover* and *mutation* work on these trees. Crossover takes two parents, chooses a random subtree from each and swaps them to create two new individuals. Mutation takes a single parent and generates a new individual by copying it

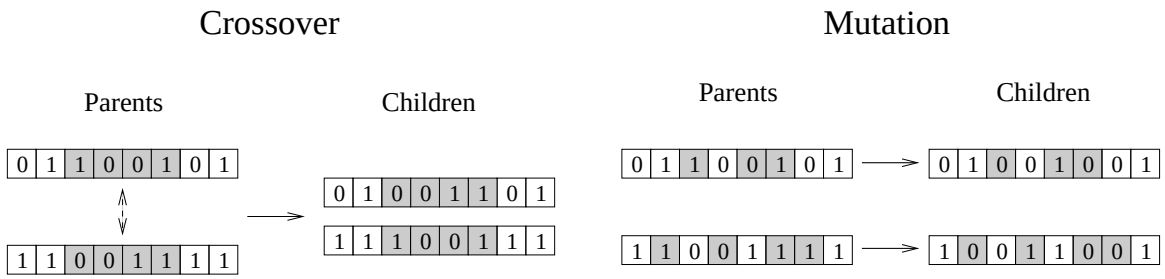


Figure 2.1: Genetic Algorithms operators in action. In crossover the two grey sequences are swapped; in mutation the grey genes are flipped.

with some mutations – these may include changing a single operator or replacing a subtree with a randomly generated one. Figure 2.2 shows examples of these on mathematical function trees.

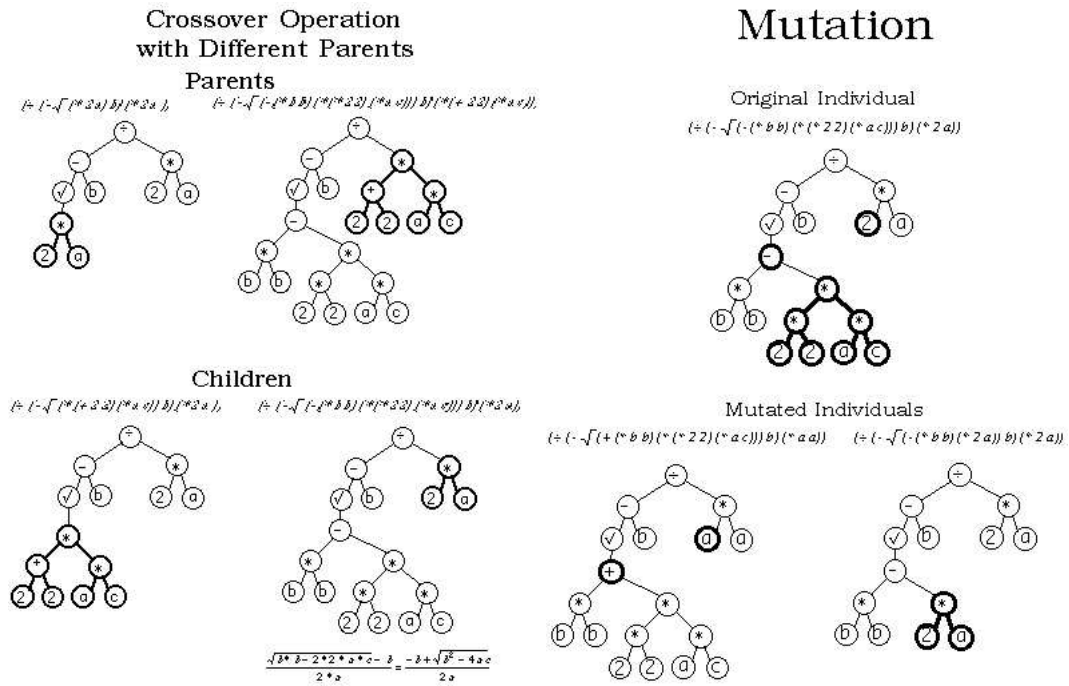


Figure 2.2: Genetic Programming operators in action. This shows how crossover and mutation work to produce new individuals. *Images courtesy of 'The Genetic Programming Tutorial' [1]*

One major advantage to Evolutionary Computation is that due to its random searching, it can come up with solutions which people are very unlikely to think of. This is shown especially well in the diverse creations that the Golem Project have uncovered using genetic programming for simple robot propulsion mechanisms [17]. It is this ability to find way-out new models that this project utilizes.

A general-purpose evolution engine was used for finding new models here (an overview of the engine code can be found in Appendix A). An overview of the process can be seen in Figure 2.3, and details of how each part was customised follow.

Fitness Criteria: Defines how 'good' each individual is, and is described in detail in the *Fitness Function* section below.

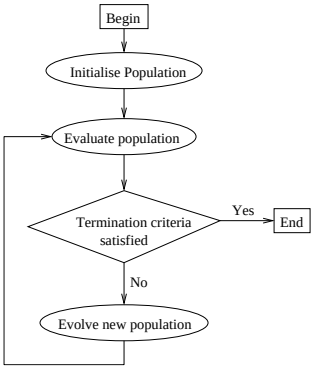


Figure 2.3: The Genetic Programming process.

Evolution: Generally Genetic Programming uses both crossover and mutation operators, and John Koza asserts in his book that crossover is all that is needed and that mutation plays no role. However, Fuchs shows in [5] that mutation can perform just as well, if not better than crossover for some problems. Due to the sensitivity of the functions defining models in this representation, this lead was followed, and only mutation was used to generate new individuals from the current set.

Since the representation contained function trees, there were several types of mutation used for these. They were applied to the trees in a recursive fashion, and there was no upper limit on the number of mutations that could happen at once. The most common type was tweaking a numerical terminal – this involved adding a small random number (normally distributed with mean of 0 and small standard deviation) to the terminal. The second type of mutation was to change an operator – this would change e.g. a $+$ node to a $\div$ node, while keeping the same children (the cardinality of the operator was preserved, and so a binary $+$ node would never be replaced by a unary node such as $\surd$).

Due to the design of the evolution engine, the population at each generation had to be a new set of individuals. To ensure that once a good individual was found it would never be lost, the top 5% of the population was always reproduced directly with no mutations into the new population.

Termination Criteria: Since this problem could go on searching forever, and it is not known ahead of time how long it will take to find a solution, the termination criteria was generally omitted. This meant that the engine would run undefinitely until the user determined either that it had found something or that it was not going to on that run.

Population size: This is a parameter that needs to be set in the engine, and there is no easy way to determine what it should be. While it would seem that the larger the population the better (more model space is being searched in each generation, there is a trade-off between population size and running time. Since it takes about 2 seconds to evaluate the fitness of each individual, the process slows down dramatically as the population size is increased. However it has been shown that small population can in fact outperform large ones [7]. Gao shows that a lower bound on the population size can be derived from the complexity of the problem [6], and Monsieurs *et al.* describe a method by which population size can be decreased while maintaining diversity in the population [19].

2.2 Model Representation

Since Genetic Programming is being used to search for new models, a tree-based representation is needed. Inspiration for this representation can be taken from the structure of the TTM which is described by the 3-tuple of formulae $\langle \Delta w, \Delta L, \delta \rangle$: the new representation keeps the two formulae from this, but constrains the diffusion mechanism and adds decay of tectal marker.

Inline with the theme of biological plausibility, tectal marker diffusion and decay is taken as a process that always happens (in the same way the retinal marker is always initialised – it is part of the underlying system). These are controlled by diffusion rate β and decay rate φ and their combine effect is defined by Equation 2.1. The result of this is added to the result of the ΔL formula to determine the complete effect on tectal marker amount. Diffusion only makes sense physically if it is constrained to a range between 0 and $\frac{1}{2}$, and similarly with decay between 0 and 1, so the model restricts the two parameters to this range.

$$\delta_i = -\beta \left(L_{i-1} + L_{i+1} - 2L_i \right) - \varphi L_i \tag{2.1}$$

The ΔL and Δw formulae are represented by expression parse trees in which nodes are operators and terminals are variables or constant values. The formulae the trees represent are evaluated by setting the current value of variables and evaluating each node recursively; the result is the value returned by the root node. Since the mechanisms are supposed to use only local information, each formula only has a small set of variables available for use: ΔL can contain L (amount of tectal marker in this neuron) and r (induced amount of retinal marker in this neuron); Δw can contain R (amount of retinal marker in pre-synaptic neuron), L (amount of tectal marker in post-synaptic neuron) and w (current weight of this connection).

$$\langle \Delta w, \Delta L, \beta, \varphi \rangle \tag{2.2}$$

Thus a model is represented by the 4-tuple of Equation 2.2 where ΔL, Δw are expression parse trees and β, φ are numeric values. Figure 2.4 shows what the two TTM formulae look like in this representation.

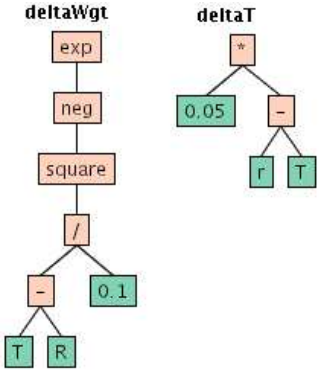


Figure 2.4: The formula tree representation of the TTM.

This model is applied to a network in the same way as the TTM, except that now δ is a set formula that is parameterised by part of the model.

1. First the system is initialised: the retinal gradient is set up; the weights and tectal marker amounts are randomised.

2. The new tectal marker amount is calculated for each tectal cell:

$$L'_j = L_j + \Delta L_j + \delta_j$$

3. The new weight is calculated for each connection:

$$w'_{ji} = w_{ji} + \Delta w_{ji}$$

4. The new weights are normalised by ensuring the sum of weights out of a retinal cell is one:

$$w'_{ji} = \frac{w'_{ji}}{\sum_k w'_{ki}}$$

5. The tectal marker amounts and connection weights are set to these values

6. Steps 2-6 are repeated.

As with the TTM this process is repeated until either stability is achieved or the user stops it.

Since these formula trees are to be used in Genetic Programming, the mutations need a set of operators that they can use in the function trees when generating new individuals. A minimal (yet hopefully complete) set of operators was used: $+$, $-$, $\times$, $\div$, exp, ^{2}, $\sqrt{}$. All of these perform as usual except division, for which *protected division* is used: it returns 0 if the denominator is 0, and performs normal division otherwise (to avoid individuals that die with divide-by-zero errors). The formulae could have numeric terminals and variable terminals – each formula has a set of variables that it is allowed based on local information: ΔL can contain L and r; Δw can contain R and L.

2.3 Fitness Function

A core part of any evolutionary computation paradigm is the ability to rank individuals based on how good they are. This is done by defining a function that assigns a *fitness* to individuals proportional to how well they perform. For ease of analysis and model comparison it is good to constrain the fitness values to between 0 and 1, where 0 denotes a useless model and 1 is a perfect solution. Often it is easier to calculate the fitness via an intermediate error measurement which determines how far from a valid solution the model is. This error may range from zero upwards. It can then be mapped onto the fitness via Equation 2.3. For example an error of zero gives a fitness of 1; and as the error increases the fitness decreases, reaching 0 in the limit $E = \infty$.

$$f = \frac{1}{1+E} \tag{2.3}$$

Since the fitness of a model is determined by how good the final mapping is, the fitness function ranks the mapping that a model produces after it has stabilised (or run for some large number of generations if it did not stabilise). Through the process of finding a good fitness function several different ways of calculating it were tried, and this section describes each of them and why they were unsatisfactory. Each method has the same basic structure where an error measurement is calculated to measure the fitness. The methods are:

1. The first method was to test the 'image' that came out the other side of the mapping by setting its input and feeding this forward – some examples are shown in Figure 2.5. A perfect mapping should reproduce the input at the tectum, and thus the error measure is defined as the sum-squared difference between the input in and the output out (where the retina and tectum are the same size):

$$E = \sum_i (in_i - out_i)^2$$

Since it is perfectly valid for a topographical mapping to form in a crossover fashion rather than straight (contrast the right-most topographical mapping in Figure 2.5 to the other two), this error measurement is calculated in both orientations, and the minimum of these is taken. Indeed the TTM will come up with either one of these based on slight variations in initial conditions, so it is important that a fitness function recognise it.

While this measurement was excellent for well-formed mappings such as those generated by the Tea Trade Model, it was not very good at rewarding partial solutions. This is a key part of any hill-climbing based method as it allows the algorithm to hone in on solutions by gradually stepping closer and closer.

2. Here it is beneficial to return to the focus of the project: finding a topographical mapping. Thus, a fitness function should be testing for the *topographical-ness* of the mapping. The easiest way to test this is to generate a sample mapping W and calculate the error measure as the sum-squared difference between the mapping weights and the sample mapping:

$$E = \sum_j \sum_i (W_{ij} - w_{ij})^2$$

Figure 2.6 shows a sample of perfect mappings, and it highlights one of the main drawbacks of this approach: what should k be? If it is too small the target mapping will be very narrow, which means that sub-solutions (mappings which are topographic but with a wide k) will not be rewarded appropriately. If it is too large the target mapping will be very wide, in which case better solutions will not be rewarded.

Similarly to the first method, this one is specific to forming a mapping of a particular orientation, so the error measurement is calculated in both orientations, and the minimum of them is used.

As with the first method, this one does not reward partial solutions. Figure 2.7 shows the results of some mechanisms which are on their way to producing a topographical mapping, but will not be rewarded by this fitness function.

3. Since the previous approach didn't work, it is wise to return to the original definition of a topographical mapping. In particular, the *'neighbours project to neighbours'* part – there should be a way of directly checking this, and it doesn't need to be related to a concept of being globally topographical. Goodhill provides an overview of several ways this can be measured in a general case of the pre- and post-mapping layers are of different dimensions. The solution presented here has been adapted from these [15].

The function captures the fact that mappings between pairs of neurons are topographical, even if they are not connecting the 'correct' parts. Due to its focus on locally

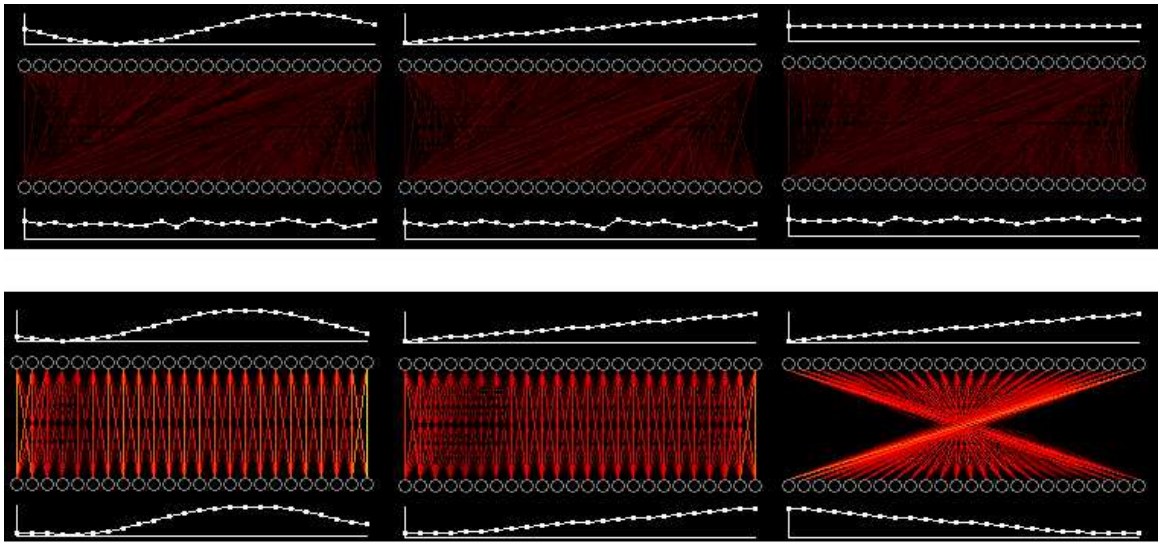


Figure 2.5: Feeding an image through the network: the top row shows bad mappings and the lower row shows good ones

Figure 2.6: Varying k to demonstrate the different width mappings. The width will effect the image projected onto the tectum: if it is narrow, the image will be crisp and precise; if it is wide the image will be blurred and smoothed.

Figure 2.7: Several piece-wise mappings which contain several locally-topographic areas, but are not globally-topographic.

topographical areas it doesn't need to differentiate between mappings of different orientations.

Unlike the previous two methods which can be calculated in a single step this method is an entire algorithm, the steps of which are:

(a) The average tectal position that each retinal neuron connects to is calculated:

$$\mu_i = \frac{\sum_j w_{ij} \times j}{\sum_j w_{ij}}$$

(b) The spread of connections is calculated (if each retinal neuron connects to all tectal neurons or to two separated one this will be high):

$$\psi_i = \sum_j w_{ij} \times |\mu_i - j|$$

(c) The error is then calculated in two steps: the mapping target error is determined by the average distance between the target of neighbouring retinal neurons; the spread error is the average spread. ξ serves to punish a mapping if some parts of it 'turn around' – if the two neighbouring retinal neurons project to tectal positions on the same side of this neurons projected position. This is shown by:

$$\begin{aligned}
\xi_i &= \begin{cases} 1.5 & \text{if } sign(\mu_i - \mu_{i-1}) = sign(\mu_i - \mu_{i+1}) \\ 1 & \text{otherwise} \end{cases} \\
E_\mu &= \frac{\sum_i \left((|\mu_i - \mu_{i-1}| - 1)^2 + (|\mu_i - \mu_{i+1}| - 1)^2 \right) \times \xi_i}{\max(i)} \\
E_\psi &= \frac{\sum_i \psi_i}{\max(i)}
\end{aligned}$$

(d) The final error is the product of these two (adding 1 to the spread error is necessary since it can be 0, and this destroys the fitness measurement).

$$E = (1 + E_\psi) \times E_\mu \tag{2.4}$$

Equation 2.4 shows the final error metric used to determine the fitness of individuals, and it is used throughout the project. It's ability to give intermediate fitness values to intermediate mappings is shown in Figure 2.8 – it is worth noting that the fitness is (almost) constantly increasing with the number of iterations.

2.4 Results

The search for new models was done in two main parts: first we started with a population of TTM individuals, and then with a population of randomly generated individuals.

In the random-start runs, absolutely nothing was found. Despite being run on several computers for about a month (and being restarted several times to try different points in the

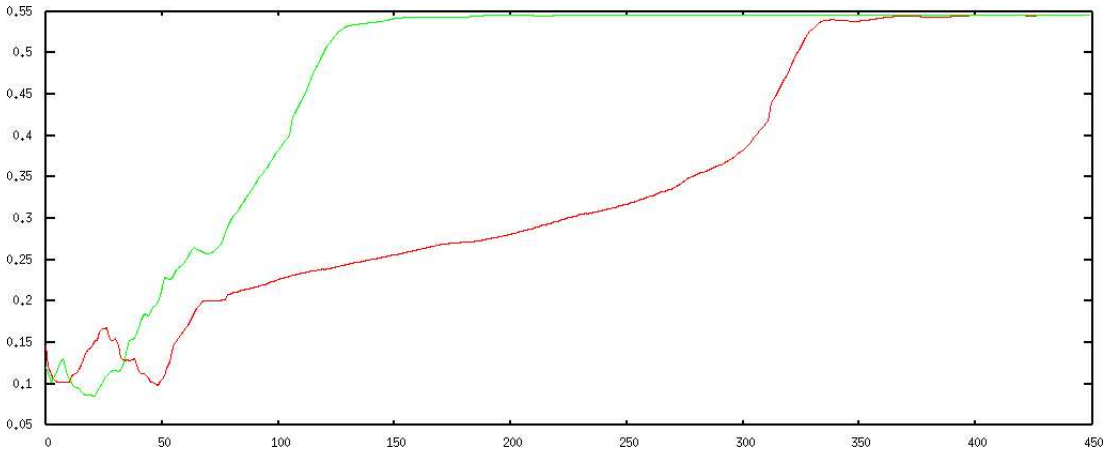


Figure 2.8: A plot showing the fitness value assigned to a mapping as the TTM is applied to it. Each line shows a run with different starting conditions. The x-axis is the number of iterations; the y-axis is the fitness of the mapping at that point.

model-space), it never came up with a model that formed a topographical mapping – not even the TTM.

In the TTM runs, the initial population was filled with individuals representing the TTM. This did not find any new models with a different formula structure to the TTM, but did refine the numerical values in it. This is really what would have been found if the representation was just a set of numbers representing parameters in the TTM formulae and some other search on these numbers alone was used.

2.5 Fitness Landscape

At this point, an explanation as to the failure of GP to find anything was needed. It took many hours of searching, and the culprit was finally found by investigating the relative fitness of individuals close to a known solution. Since the TTM is known to work, it was used as the base model. From here the fitness of models a single mutation away was measured and analysed. Two separate sets of data were gathered here: mutations involving the numeric terminals in the formula, and mutations involving a significant structural change. A significant structural mutation is defined as one that changes an operator or part of the tree so that it represents a different function.

Figure 2.9 shows the proportion of generated individuals in each fitness range. The green plot shows that numeric terminal mutations generate individuals with a decent fitness spread across all ranges. The red plot shows that significant structural mutations generate individuals with very low fitness. The fitness distribution of the red plot is in fact the same as that of randomly generated individuals, and this is exactly the reason that GP could not find anything. If it came across one of these individuals that is close to the TTM at any point (by a mutation from a current individual) the fitness function would give no indication that it was in the right neighbourhood, and the individual would be thrown away. Due to this, the GP engine would need to be exceedingly lucky to run across a working model accidentally!

This highlights the sensitivity of Genetic Programming to the fitness landscape of the domain. If the fitness landscape is not smooth enough and does not give any indication that a solution is nearby, it has no way of focussing the search around it. This problem is more general than just Genetic Programming though, and applies to any hill-climbing based search method.

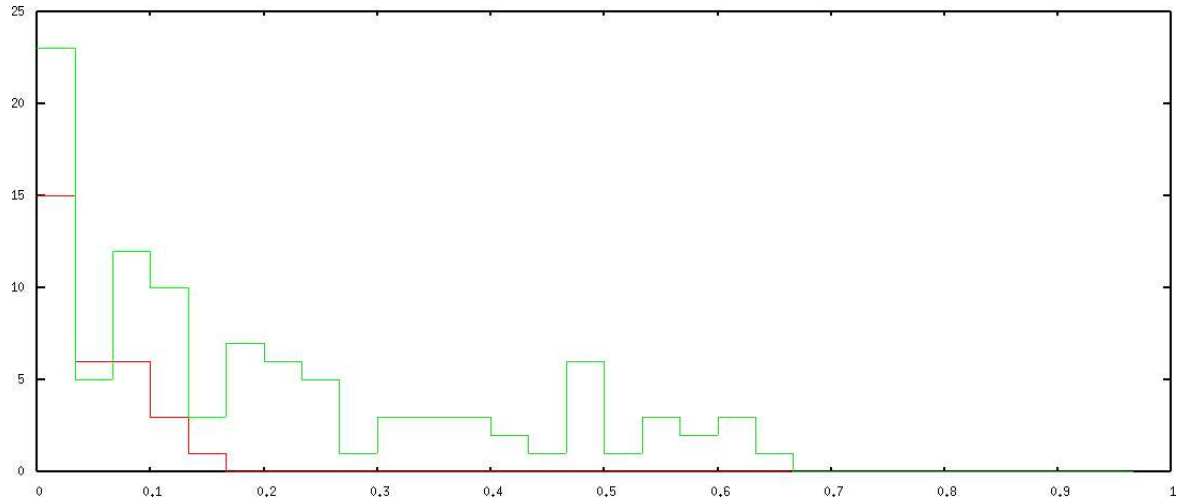


Figure 2.9: A plot showing the number of individuals of each fitness value after one mutation from an individual which has fitness 0.55. The green line is mutations of numeric terminals and the red line is mutations involving significant structural change. The x-axis is the fitness value of the individual; the y-axis is the number of individuals with that fitness.

Thus, a new model representation and/or search method is needed. The next chapter reformulates the problem to find a new representation, and shows that it is more suited to hill-climbing search than this representation.

Chapter 3

An Alternative Approach

The previous chapter considered a formula-tree representation, and illustrated that its fitness landscape is far from smooth or continuous. It proceeded to show that this discontinuity causes substantial problems for hill-climbing in that space.

Consequently, a new representation is needed that:

- includes the TTM as one case (as it is a known solution)
- has sufficient flexibility that we might hope to find *other* solutions.
- constrains all solutions to biological plausibility
- is sufficiently smooth for hill-climbing search to be an effective heuristic

This chapter addressed these by presenting a new representation derived by myself and Marcus Frean. The TTM is shown to be an instance of it, and finally we show that it is sufficiently smooth for hill-climbing.

3.1 A New Representation

This section steps through the derivation of a new parameterised representation, stating goals and assumptions made along the way and the final parameters derived.

Since one of the original goals of the project is to find a model that only uses local information, the view of the network is restricted to that shown in Figure 3.1.

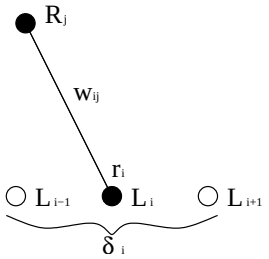


Figure 3.1: The view of the network relevant to the update rules.

The goal here is to find a set of rules which define how the network weights and tectal marker amounts are updated each iteration. As this is to use locally available information only, the two rules are restricted to the following quantities:

- The weight update rule can depend on: R (retinal marker in pre-synaptic neuron), L (tectal marker in post-synaptic neuron), w (current weight).

- The tectal marker update rule can depend on: L (current amount of tectal marker), r (induced amount of retinal marker) and δ (diffusion of tectal marker).

3.1.1 Tectal Marker Update

There are some assumptions here about the underlying physical system:

- There is a background amount of tectal marker, denoted L_{rest}.
- The tectal marker will diffuse across neighbouring neurons with a diffusion rate of β. Chemical diffusion is the process by which a chemical will flow from areas of high concentration to areas of low concentration as a result of the kinetic properties of the particles. The end result (with no other influences) is that there will be a uniform distribution of chemical. In this case, it represents the marker chemical spreading out across neurons in a layer.

  The following formula represents the operation of diffusion by adding the amount flowing into this neuron from neighbours to the amount flowing out to those neighbours. The diffusion per timestep, denoted δ is calculated by:

$$\delta_i = \frac{1}{2}\beta\left(L_{i-1} + L_{i+1} - 2L_i\right)$$

- The amount of retinal marker induced in each tectal neuron is a weighted sum of the amounts in pre-synaptic neurons:

$$r_i = \sum_j w_{ji} R_j$$

The amount of tectal marker is either induced or inhibited by this r, which gives:

$$L \propto \alpha r$$

where α is between -1 and 1. If $\alpha < 0$ then the incoming retinal marker will inhibit the amount of tectal marker; if $\alpha > 0$ then the incoming retinal marker will induce more tectal marker. If $\alpha = 0$ the tectal marker will remain at a constant amount determined by L_{rest}. From this we get the target tectal marker amount:

$$L_i^* = L_{rest} - \alpha r_i \tag{3.1}$$

The delta rule allows us to move a towards this amount with rate ρ, and this can be combined with diffusion to give the change in tectal marker amount:

$$\Delta L_i^{new} = \rho\left(L_i^* - L_i\right) + \delta_i \tag{3.2}$$

Which is then added to the current amount to give the new tectal marker amount:

$$L_i^{new} = L_i + \Delta L_i \tag{3.3}$$

Thus the first four parameters of the new model are defined: L_{rest} is the background amount of tectal marker; β is the tectal marker diffusion rate; α is the rate at which L follows r; ρ is the update rate of the tectal marker.

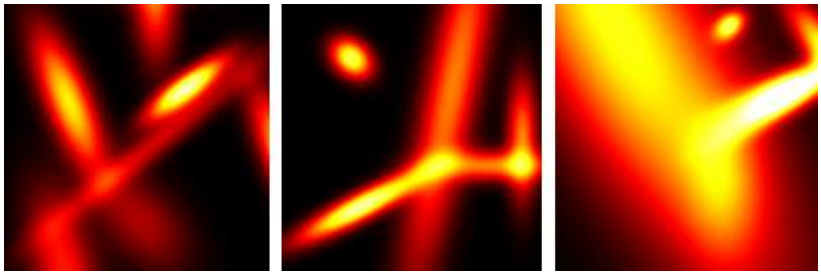


Figure 3.2: A couple of conceptual two-dimensional surfaces that *could* be used in the $(R, L) \rightarrow wgt$ function. The weight a connection would get is the colour of the pixel where its pre-synaptic R is the x-axis and post-synaptic L is the y-axis – black corresponds to 0, red to 0.3, yellow to 0.6 and white to 1

3.1.2 Weight Update

A weight update rule is a function that maps an (R, L) pair to a new weight value (via either a target weight or a change in weight function). Since it is a two variable function, it is easiest to think of it as a surface in two dimensions. Conceptually, this mapping could be anything from a simple plane to a multi-modal mixture of functions as in Figure 3.2, which makes the task of parameterisation somewhat difficult as it would be hard to capture all this complexity in a few parameters[1].

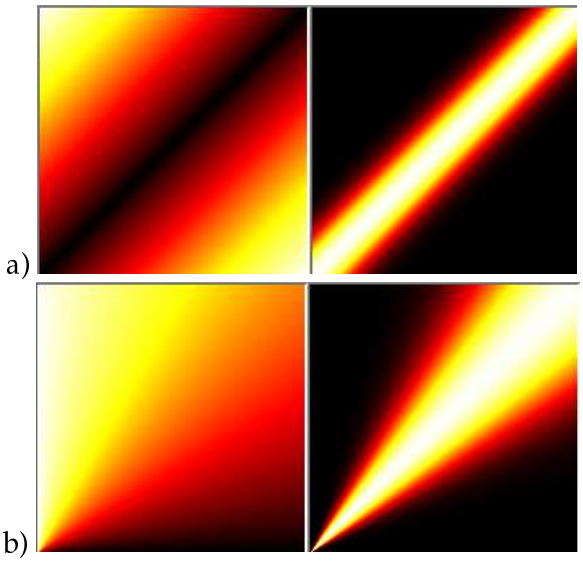


Figure 3.3: $(R, L) \rightarrow w$ mappings: a) The mapping in the TTM; b) The mapping in the parameterised model. In each of these the left hand box shows the $(R, L) \rightarrow f$ mapping and the right hand box shows the full $(R, L) \rightarrow w$ mapping

However the TTM suggests a way to restrict the space of possibilities further. In the TTM the difference between R and L is transformed by a Gaussian (a function of the form $e^{-\frac{x}{\sigma}}$) to the final weight. It would make sense however if the actual amounts of these quantities is irrelevant, and rather it is the ratio between the two that is important. Consequently the (R, L) pair can be transformed to a one-dimensional variable analogous to the angle, scaled to be between 0 and 1:

[1] Just imagine the complexity for a moment: $a.R + c.RL + d.R^2 + \ldots + f.\log L + g.e^{\frac{-(h-R)^2}{2i^2}} + \sin(L) \ldots$

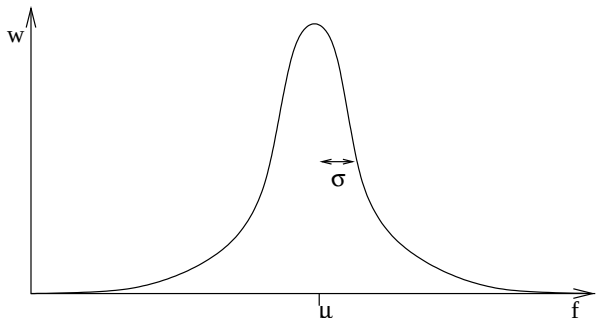


Figure 3.4: The Gaussian mapping from f to new weight values for the weight update rule

$$\begin{aligned} \theta_{ji} &= \tan^{-1} \frac{R_j}{L_i} \\ f_{ji} &= \theta_{ji} \times \frac{2}{\pi} \end{aligned}$$

The problem has now been reduced to finding a one-dimensional function from this f to the new weights – a considerably simpler task than the two-dimensional mapping needed earlier. Conceivably, any type of function could be chosen for this (and perhaps one could be found using genetic programming), but for simplicity a uni-modal Gaussian distribution was chosen since it is capable of the TTM. A more complex mapping could be used at some stage later on to try finding more models. This Gaussian can be described by a mean μ and standard deviation σ as the following equation, and is shown in Figure 3.4.

The target weight can be calculated by:

$$w^*_{ji} = e^{\frac{-(f_{ji}-\mu)^2}{2\sigma^2}} \tag{3.4}$$

The delta rule can be applied to move towards this target weight with rate η:

$$\Delta w_{ji} = \eta \left(w^*_{ji} - w_{ji} \right) \tag{3.5}$$

Which is then added to the current weight to get an un-normalised new weight:

$$w'_{ji} = w_{ji} + \Delta w_{ji} \tag{3.6}$$

Normalisation of weights plays a large role in Artificial Neural Networks. There are two main ways it can be enforced: additively or multiplicatively, each of which leads to qualitatively different weight patterns [13]. Additive normalisation leads to a few connections with the maximum possible weight and the rest with zero weights, whereas the multiplicative version leads to a more spread out mapping with connections having intermediate weights. Thus the multiplicative version is used here the mapping requires having some weights in the intermediate ranges.

Normalisation is often done entirely at either the pre-synaptic or post-synaptic neuron, but rather than pre-specifying one of these, a combination of both could be used. This is done here, controlled by a parameter γ determining how important each type is, i.e. if $\gamma = 1$ it is entirely pre-synaptic; $\gamma = 0 \rightarrow$ post-synaptic; $\gamma = 0.5 \rightarrow$ half-half. The exact equation for it is:

$$Z_{ji} = \gamma \sum_k w'_{jk} + (1-\gamma) \sum_k w'_{ki}$$

This is then applied to give the normalised new weight value:

$$w_{ji}^{new} = \frac{w'_{ji}}{Z_{ji}} \tag{3.7}$$

3.2 Summary of the New Representation

The above derivations have led us to a model that is characterised by a set of formulae and a set of parameters. We will assume the formulae and search the space of parameters for those that lead to topographical mappings.

To calculate the new tectal marker amounts:

r_i	=	$\sum_j w_{ji} R_j$	retinal markers make their way down axons and accumulate in tectal cells
δ_i	=	$\frac{1}{2}\beta\left(L_{i-1} + L_{i+1} - 2L_i\right)$	diffusion in tectum
L_i^{new}	=	$\rho\left(L_{rest} + \alpha r_i - L_i\right) + L_i + \delta_i$	tectal markers induced by presence of retinal markers

To calculate new weight values:

f_{ji}	=	$\tan^{-1}\frac{R_j}{L_i} \times \frac{2}{\pi}$	the $(R, L) \rightarrow$ ratio mapping
w_{ji}^*	=	$e^{\frac{-(f_{ji}-\mu)^2}{2\sigma^2}}$	the target weight
w'_{ji}	=	$\eta\left(w_{ji}^* - w_{ji}\right) + w_{ji}$	the un-normalised weight
w_{ji}^{new}	=	$\frac{t_{ji}}{\gamma\sum_k t_{jk} + (1-\gamma)\sum_k t_{ki}}$	normalising the weights

These formulae involve the following 8 parameters:

Parameter	Range	Description
α	$[-1, 1]$	The rate at which L follows r
β	$[0, \frac{1}{2}]$	Diffusion rate of tectal marker
ρ	$(0, 1]$	Update rate of tectal marker
L_{rest}	$[0, 1]$	background amount of tectal marker
μ	$[0, 1]$	mean of the Gaussian in $(R, L) \rightarrow w$ transform
σ	$(0, \frac{1}{2}]$	standard deviation of the transformation Gaussian
η	$(0, 1]$	learning rate of the weights
γ	$[0, 1]$	normalisation location

These formulae and the values assigned to the 8 parameters essentially specify a type of learning algorithm that can be iteratively applied to a network. The effectiveness of this algorithm can then be measured the same way as described in Chapter 2, and we now have a small piece of 8-dimensional space to search for new models in.

3.3 Searching

As in Chapter 2, the goal here is to find models using the new parameterised representation. Neil Gershenfeld [8] gave an excellent overview of general search and optimization techniques from which the following has been derived.

Evolutionary techniques such as those explored in the previous chapter are best when the problem is not well defined and there is not much locality in the space being searched (the fitness landscape in this case). Since this parameterised representation is a set of continuous numeric parameters, there is a wider set of methods that may be applicable. While it could be searched using the evolutionary techniques previously explored, they contain a lot of overhead that can be avoided here (management of populations and selection etc.). A simpler way to search is using the Metropolis algorithm as proposed by Metropolis *et al.* in 1953. The method as it applies to this problem is as follows:

1. Start with a random initial model M (a random set of parameters)
2. Calculate its fitness f
3. Generate a new model M' by adding a random amount to one of its parameters
4. Calculate the fitness f' of M'
5. Keep this new model with probability $p = \min(1, \frac{f'}{f})$
6. Repeat steps 3-6 until termination criteria is reached (or run indefinitely if in interactive mode)

Simulated Annealing is an extension of this algorithm which adds a coefficient analogous to temperature T which gradually decreases over time. The idea is that early on when T is high the solution is jumping around accepting every random change it gets, and as T decreases the solution will slow down and spend more time in high-fitness states. It does this by modifying the probability of moving to a new model even if it has lower fitness than the current one. The change is made by slowly decreasing the probability that a solution with lesser fitness gets chosen as time goes on. The main problem with this though is that when the search is being run in interactive mode (running indefinitely until the user stops it), there is no way to define how quickly this random-ness coefficient should decay.

3.4 Tea Trade Model Revisited

The Tea Trade Model (TTM) was introduced in section 1.3, with chapter 2 describing an approach to representing it with explicit formulae. Since the model has been shown to work, it can be used to indicate that this parameterisation is on track by determining the parameters for the TTM. First, a set of values were chosen through comparison of the original TTM formulae and the parameterisation, then these values were refined using the search method described above as a check that the TTM is in fact a local maxima in this fitness space.

The parameters that correspond to a local optima in the fitness landscape are:

$$
\begin{aligned}
\alpha &= 0.45 \\
\beta &= 0.2 \\
\rho &= 0.09 \\
L_{rest} &= 0.0 \\
\mu &= 0.7 \\
\sigma &= 0.035 \\
\eta &= 0.35 \\
\gamma &= 0.88
\end{aligned}
$$

3.5 Stability Analysis

Given a set of parameters describing a model, it would be nice to find out how stable these are with respect to changes in each parameter. This would be particularly useful in determining how conducive the system is to search by hill-climbing, just as was done for the GP fitness-landscapes in Section 2.5.

A tool was developed to explore the stability of a model in which each graph shows a scatter plot of fitness of individuals as the parameter is varied while keeping all others constant (a screenshot is shown in Figure 3.6). Due to the slightly chaotic nature of the system, a particular set of parameters almost never leads to the same fitness value when started with different initial weights. One reason for this is that an iterated map such as this is very sensitive to initial conditions, and the network is initialised with random (small) weights and tectal marker amounts. This variation in starting conditions can lead to either the system not converging on that run, or taking many more iterations to converge (e.g. 3000 iterations for bad starting conditions whereas some runs take 350 iterations). Another reason is that a particular set of parameters may lead to a system which oscillates through a sequence of states as in Figure 3.5 – each of these states have a different fitness.

Figure 3.6 shows the stability of the parameterised Tea Trade Model. Despite the wide range of fitness values at each point, the most important value is the maximum, as it represents what the model can do given the right starting conditions. Thus, this analysis focuses on the upper bound of the plots. What they show is that the model is stable with respect to β, ρ, η and to a lesser degree L_{rest} and γ, but very sensitive to changes in α, μ and σ. Looking at these, the most stable ones are the update rates – this hints that the system could be updated at any speed without loss of accuracy. Also the current values are the peaks in these plots – which is good since the model is the result of running the search. This sensitivity will give an indication of how likely it is to find this model while searching randomly through model-space.

It is worth noting that this is by no means an extensive search through model-space – it is only moving out in a single direction for each graph it plots, and eight-dimensional space is rather large. This analysis will not even show the fitness of a model in which two parameters are changed simultaneously. It also takes a long time to generate as each data point takes about two seconds to calculate.

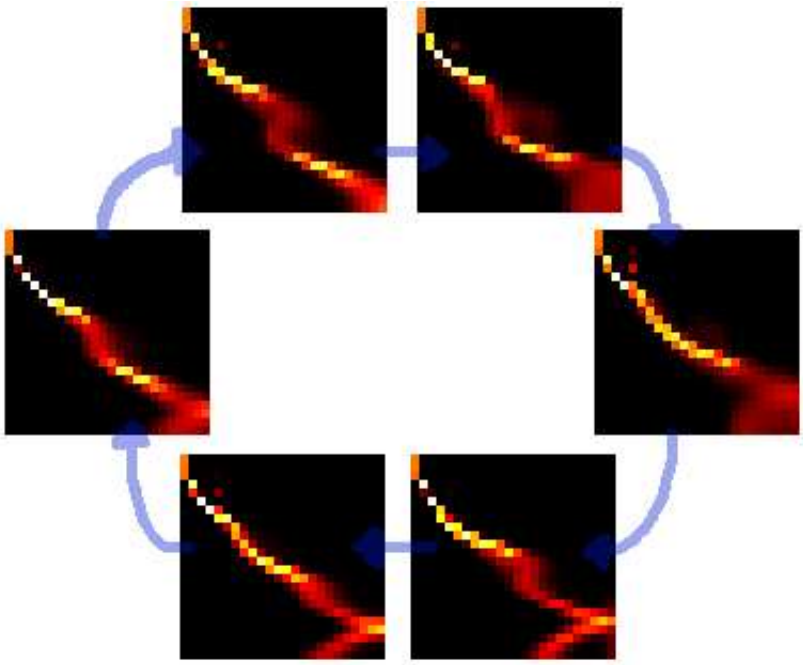


Figure 3.5: A model which leads to an oscillating set of mappings – these are a set of states that run clock-wise.

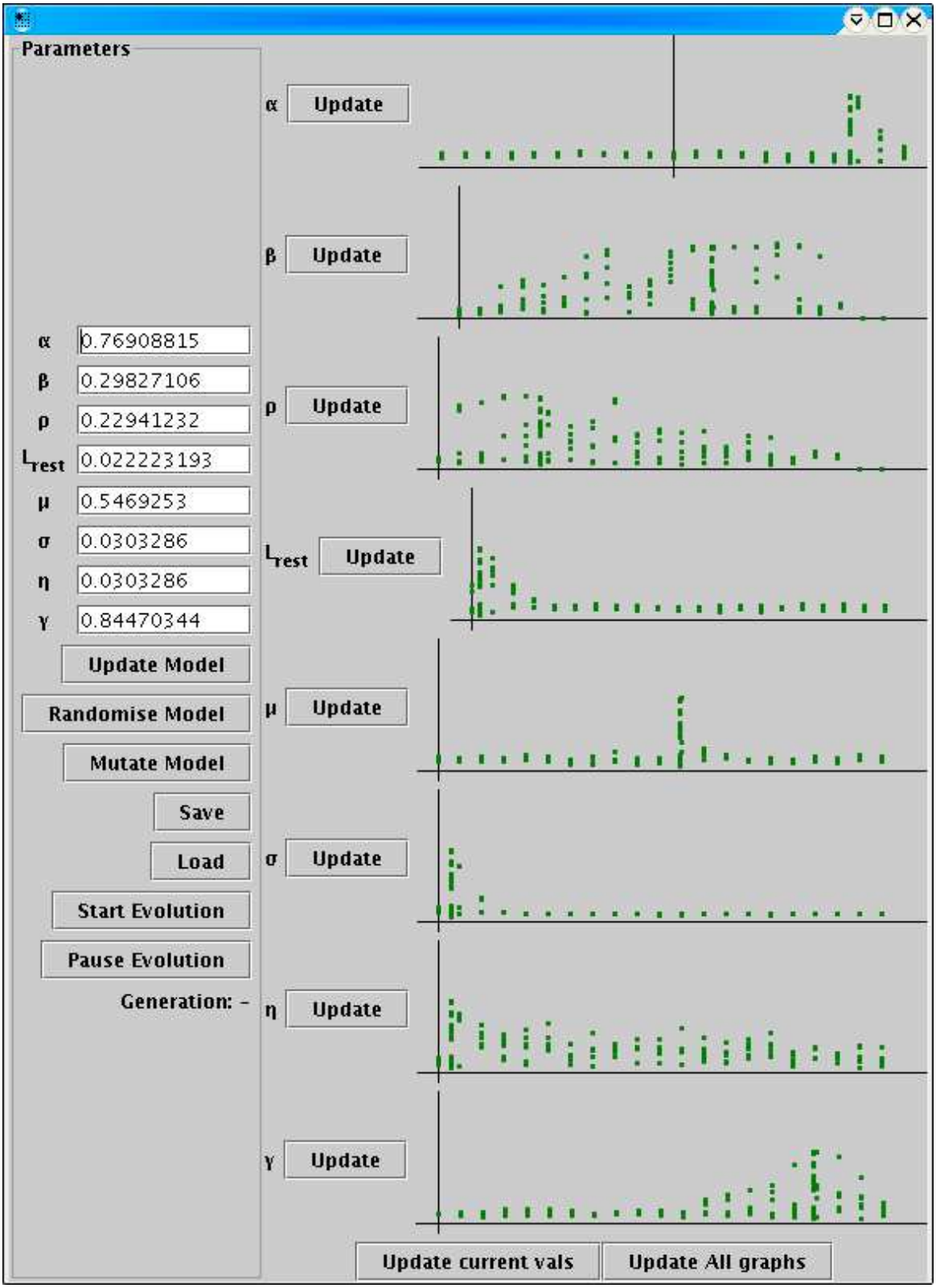


Figure 3.6: Stability Analysis of the parameterised version of the Tea Trade Model.

Chapter 4

Discovery of a Novel Mechanism

Once the parameterised representation had been formulated, we went about using it to search for models. On the first night that it was running, it discovered a new model with a set of parameters that we'd never dreamed of – this chapter analyses that model and shows what we've found out about it.

4.1 Parameters

This model is described by the following set of parameters, with the TTM parameters alongside for comparison:

α	$=$	-0.3	$=$	0.45
β	$=$	0.08	$=$	0.2
ρ	$=$	0.002	$=$	0.09
L_{rest}	$=$	0.0	$=$	0.0
μ	$=$	0.8	$=$	0.7
σ	$=$	0.003	$=$	0.035
η	$=$	0.8	$=$	0.35
γ	$=$	0.9	$=$	0.88

When it is run on a network, this model does not appear to do anything for a while, then it does all its work in a relatively short time period – Figure 4.1 shows a sequence of network states through this process. It forms a perfect topographical mapping such as this one about 60% of the time, forming disjointed topographical mappings such as those shown in Figure 2.7 the rest of the time.

The truly bizarre part of this model's behaviour is the zipping action that moves the connection into the right place. It moves along each retinal neuron

4.2 Analysis

The Stability Analysis method described in section 3.5 can be used to analyse this model, the results of which are shown in Figure 4.2. Unfortunately not much is known yet about how this model works – the following is an overview of what has been determined so far

- Since α is negative and L_{rest} is 0, the tectal marker amount in each tectal neuron will start positive (due to initialisation) and slowly decrease until it stabilises at a negative amount. The rate ρ at which it does this is very low though, so this will be a slow

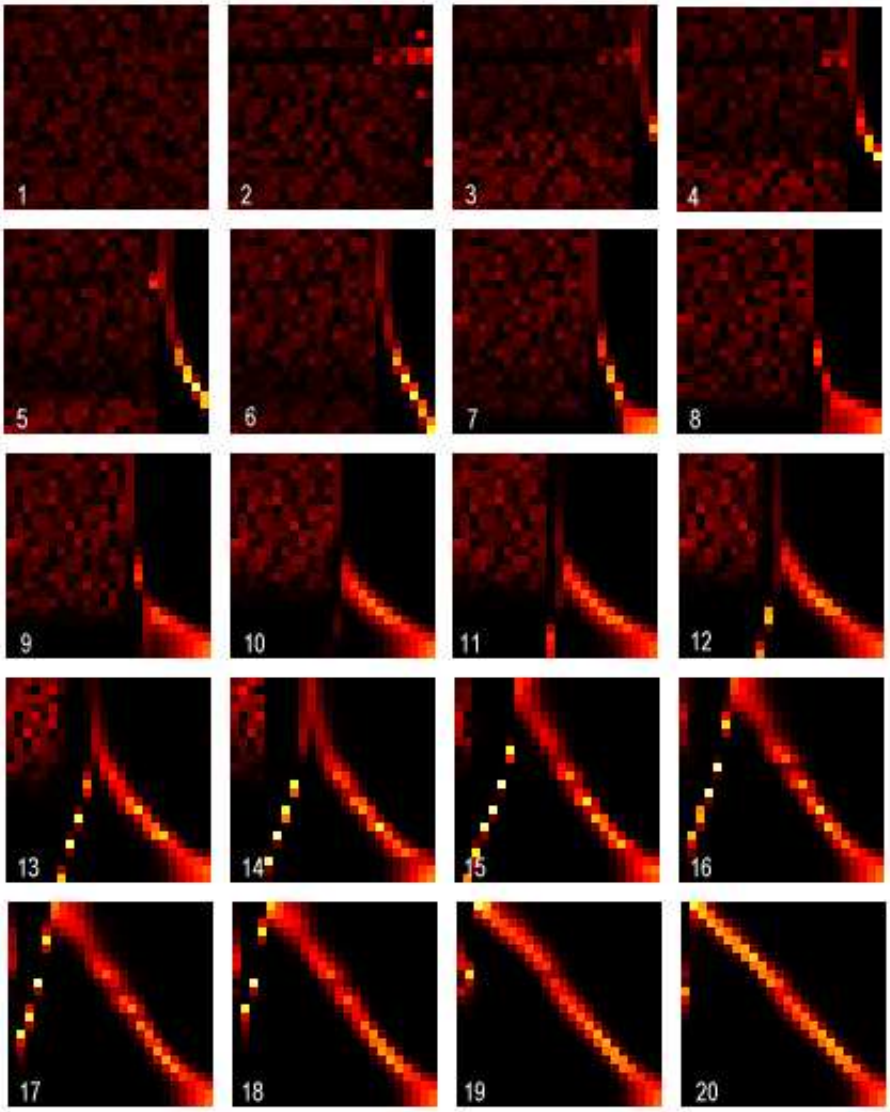


Figure 4.1: Network states at various stages of applying the model. Beginning at iteration 0, it immediately settles into state 2, then nothing happens until shot 3 at iteration 115. From here each state is at an interval of about 20 iterations, until it stabilises in state 19 at iteration 500. The final stabilisation into state 20 is done by state 850, after which nothing happens

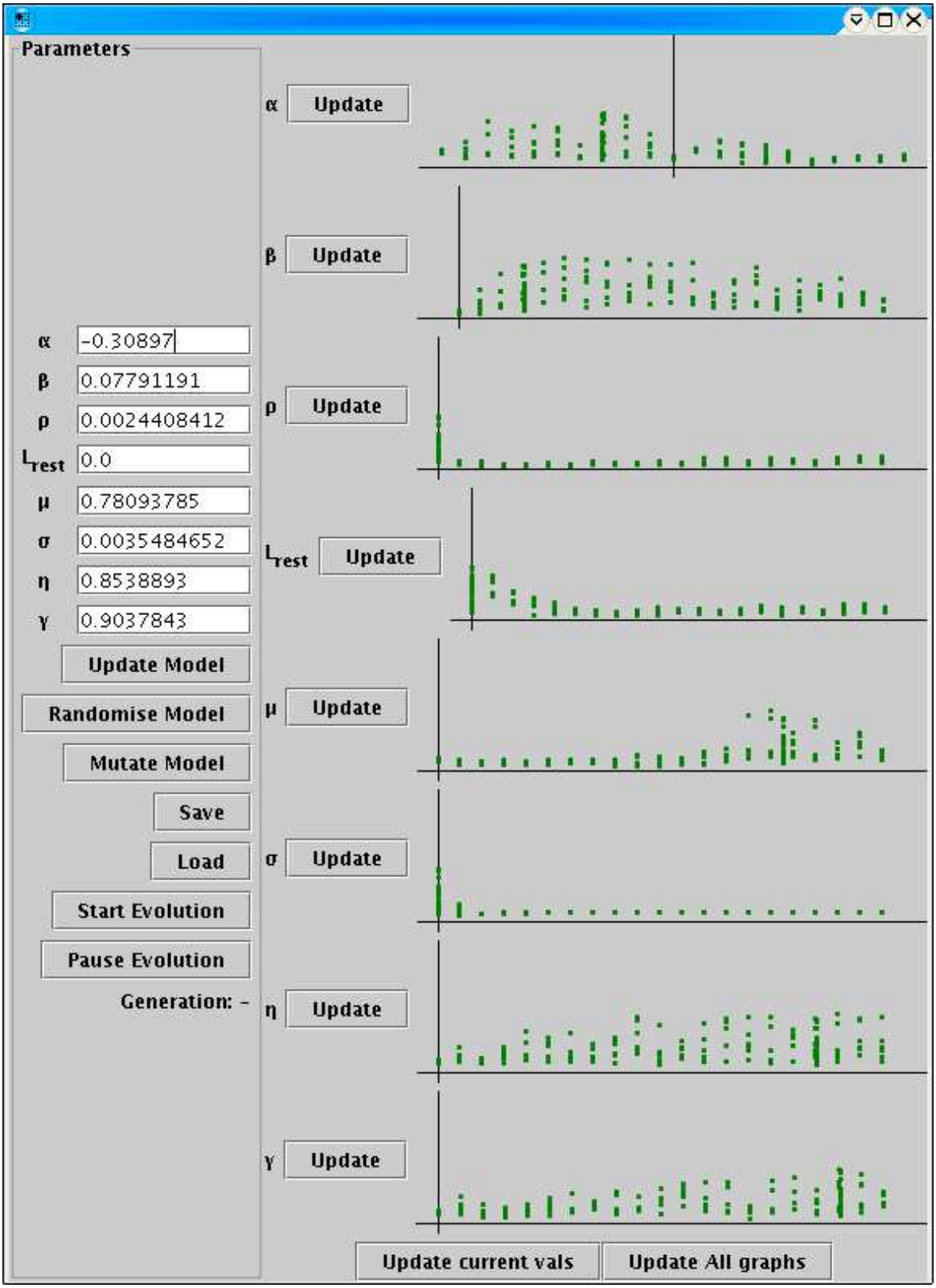


Figure 4.2: Stability Analysis of the new model.

process. The stability analysis shows that the model is quite stable with respect to α but not as stable with respect to L_{rest}.

- It will run with almost any amount of tectal diffusion, as long as it is non-zero. Since this rate β is so much higher than the tectal update rate ρ it will serve to equalise all the tectal marker amounts and keep them at an equal level to each other.
- μ and σ specify that the weight mapping only gets values when the ratio L/R is within a small range of 0.8. When the Δw function is approximately 0, nothing happens to the weights – they remain where they were previously set to. The model is very sensitive to these parameters, and will not work if they are modified.
- Since the weight update rate η is so much higher than the tectal marker update rate ρ, the weights will fall into place quickly as the tectal marker slowly moves through the range determined by μ.
- The model does not appear to be very sensitive to where normalisation is calculated, but it performs best around when closer to (but not at) the retina.

Since the model is within the legal parameter ranges, according to the derivation above it should be biologically plausible. It uses a novel approach to forming the mapping, yet unfortunately it is still not entirely understood. From the behaviour that has been observed above, and the fact that it goes about the mapping formation in a brand new fashion, it holds some tantalizing opportunities for future research.

Chapter 5

Conclusion

The first approach was the use of Genetic Programming with a simple representation of a mechanism. All attempts to find new models (or even the Tea Trade Model) with this approach met with failure, and indicated the need to find a new approach. In the process it highlighted the fact that Genetic Programming can only be used in model-spaces which are smooth enough and give some indication (via the fitness solution) of partial solutions. The representation being used here didn't satisfy this condition – in fact we showed that all models a single mutation away from a known solution (the Tea Trade Model) had the same fitness distribution as randomly generated models.

Consequently, a new representation was formulated: a model in this representation is an 8-tuple of real-valued numbers. Due to constraints put on the through the formulation, every point in this bounded 8-dimensional space corresponds to a biologically plausible model. It is known that the TTM is represented by a point in there, and the hope is that several more points correspond to currently unknown mechanisms which result in formation of a topographical mapping. We showed through analysis of the fitness landscape that this representation can be searched by standard search methods such as Metropolis algorithm or Simulated Annealing. Using these search methods a novel new mechanism was found, and a brief overview of how it works is given.

The overall goal of this project was the automated searching for mechanisms for topographical mapping formation, and in this regard it was a large success.

5.1 Future Work

This project focused on the mappings formed in the initial stages of brain formation, but there are also significant experimental results regarding the regeneration of them after surgical interference. It would be worth investigating these models and representations to find out whether they show this regeneration.

Some other research has indicated that the mapping may be partially formed en route to the tectum, and that chemical axon guidance cues play a role in this – this could be explored as an extra part of the mechanism.

The Tea Trade Model and new model found can form a mapping of either orientation depending on initial conditions, but experimental evidence indicates that mappings form in a particular orientation (as indicated by Figure 1.3. This would suggest that there is some cue in the biological system that sets this orientation up – perhaps this could be a chemical gradient at the neuron formation or axon guidance stages.

Appendix A

Software systems

Over the course of the project I wrote various systems to help with understanding the system and developing new ones. A major choice early on in the project was the development language(s) that would be used, and the choice came down to either C++ or Java since they are the high-performance languages (relatively) and I am familiar with both of them. I decided on using Java for all coding for a couple of reasons:

- It's easy to write. Especially compared to C++, it removes the hassle of working with pointers and memory management, thus speeding up development significantly.
- Graphical User Interfaces are easy to build. The Swing libraries in Java are simple to use and extend – this was of great importance in building all the program interfaces and visualisation tools.
- I was far more familiar with Java at that point in time, so I could get down to writing the tools faster.

Unfortunately equivalent code will run slower in Java than C++, but this was a sacrifice I was willing to make in order to get the ease of writing and interface design. Overall the code for the project totals 13,000 lines of Java, is contained in a well-organised package hierarchy, and has build tools configured to easily build it all.

A.1 Visualisation

It was very important throughout the project to see what the various algorithms/models were doing, so part of all coding was the development of a set of graphical visualisation widgets. The core ones relate to displaying the network and various quantities in it, and are shown in Figure A.1. The top display shows the neural network and the weights of all the connections. Each connection is drawn with a colour corresponding to it's weight – a 'fire' colormap is used here: black is the background colour, and denotes a weight of 0, and the progression goes from black through red through yellow to the maximum weight, 1, of white.

The next two plots show a sample input to the network and the output that comes from the standard weighted sum of the inputs. These plots show what typical activity is going to look like for the given mapping, and give a feel of how good it is (if the same image appears at the tectum it is a topographic mapping).

The next three plots show the amounts of marker chemicals in each part of the network, and are labelled accordingly. The R plot is the amount of retinal marker in the correspond-

ing retinal neurons; r is the amount of retinal marker induced in the corresponding tectal neurons; T is the amount of tectal marker in the corresponding tectal neurons.

The last box is an alternate way to view the connection weights. Using the same colour scheme as above, the colour of a point in the (i, j)th cell shows the weight between the ith retinal neuron and the jth tectal neuron. This visualisation shows all the weights at once (in the network diagram all the connections cross over and hide each other), and it demonstrates overall structure a lot more clearly. For instance if there is a section which is brightly coloured and connected, it means that this section is locally topographic. If there is a full diagonal line of high weights, this represents a globally topographic mapping.

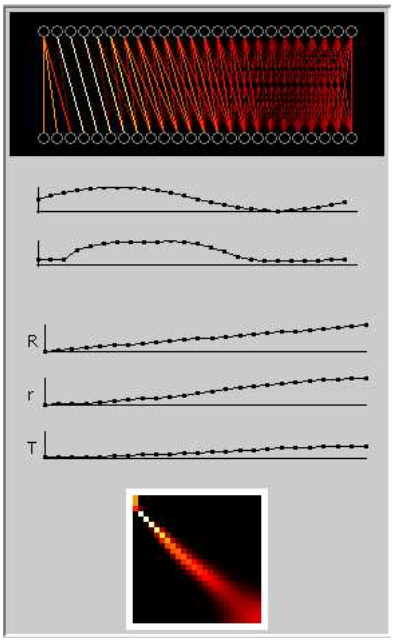


Figure A.1: The collection of widgets used to display a network and all it's attributes

A.2 Simulation Framework

Since this project was all about understanding the mechanisms involved in retinotectal map formation, it was necessary to simulate the system. I designed a set of classes and programs for this which separated the functionality in a way that made it easy to add new representations and to watch a model at work. Screenshots showing the final program in formula-model mode and parameterised-model mode are shown in Figure A.2

A.3 Evolution Engine

This is a general purpose evolution engine which is easily extended to a new application. It has been designed in a modular system in which a new problem is designed by writing Java classes which implement a few interfaces, and the engine takes over and handles the rest. It follows the flow diagram in Figure 2.3, and needs the following parts of the process to be defined for the application:

Individual: What constitutes an individual in this application – this is the representation that is evolved and selected over time.

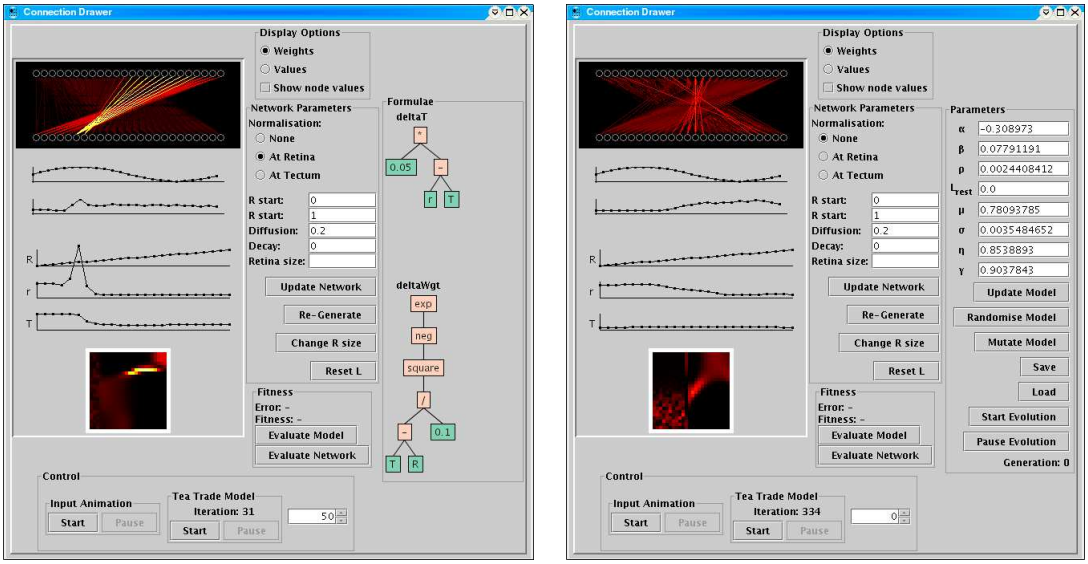


Figure A.2: Screenshots of the program in action. The left shows the formula-model mode, and the right shows the parameterised-model mode.

Population size: How many individuals should be created for each population - a number passed to the engine.

Fitness Criteria: Defines how 'good' each individual is: evaluates the individual and assigns a fitness value between 0 (bad) and 1 (good) – this concept is discussed in detail in Section 2.3.

Termination Criteria: This is how the evolution process knows that it is finished. Depending on parameters passed, the engine will run for a set number of generations, until the fitness of the best individual reaches a certain level, or run until told to stop.

Evolution: Each problem must define it's own evolution strategy – given a population of individuals, this will generate a new population through whatever genetic operators may be applicable.

The engine also has a plugin interface for watching where it is up to. This is done through defining callbacks with Java interfaces – a bit of your code is called every time a certain event happens in the engine, or it reaches a stage in it's process. A couple of general pre-built plugins come with the engine for logging it's progress and saving the current best populations to file as the engine runs.

This engine has also been used in several other contexts, and it's extendability and ease of use was well proven.

A.4 Utility

A.4.1 Trees

This is an entire framework for representing generic tree structures. It contains a base interface which defines a tree node, and a set of derivative interfaces that different types of tree (such as mutable trees, parse trees etc.) must conform to. It also contains some highly customisable UI widgets for displaying these trees such as the one shown in Figure A.3. A

very nice feature of this is that the layout is dynamic, and when a new one is selected the components will slide into the new position rather than snapping to it – this leads to far more understandable visualisations.

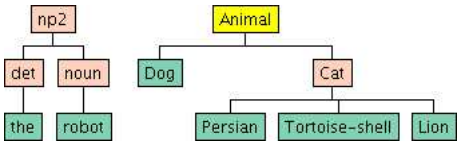


Figure A.3: Trees drawn by the tree displaying widget

A.4.2 Maths

An extensible symbolic mathematics engine using a formula tree representation. This is built on top of the tree library, so any tool or widget that can operate on them will work on these maths trees. The main advantage of this is the visualisation aspect – any formula can be displayed very easily once it is created.

The value of an equation represented by a tree is found by recursively evaluating the root node: it will evaluate it's child nodes and apply the operator it represents to the values found.

Most of the time these expression trees are wrapped in a `Formula` object which adds the ability to have named variables in the formula, substitute these variables at evaluation time, and to simplify the tree. The simplification process can recognise and simplify redundant expressions such as adding 0, multiplying or dividing by 0 or 1, and even dividing an expression by itself. It can also condense subtrees that only contain numeric terminals as these will always evaluate to the same value.

So far it has the base arithmetic module which defines most standard operators ($+$, $-$, $\times$, $\div$, $\sqrt{}$, ^{2}, $\sin$, $\exp$) and a boolean logic module which can do all boolean operations ($\vee$, $\wedge$, $\neg$, $\overline{\vee}$, $\overline{\wedge}$, $\oplus$).

A.4.3 Parsing

This is a reasonably generic parsing engine which can take a grammar and parse input in either LL(1) or bottom-up natural-language-parsing style. A grammar is represented by a set of data structures, and can be read in from a file. This is used primarily for parsing user input of equations.

Bibliography

[1] The genetic programming tutorial. `http://www.geneticprogramming.com/Tutorial/index.html`.

[2] Forrest Briggs. Spiking neural networks, 2000. `http://www.aiathome.com/snndocs.html`.

[3] Eph Nomenclature Committee. Unified nomenclature for eph family receptors and their ligands, the ephrins. `http://eph-nomenclature.med.harvard.edu/cell_letter.html`.

[4] Barry J. Dickson. Molecular mechanisms of axon guidance. *Science*, 298(6):1959–1964, 2002.

[5] Matthias Fuchs. Crossover versus mutation: An empirical and theoretical case study. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 78–85. Morgan Kaufmann, 22-25July 1998.

[6] Yong Gao. Population size and sampling complexity in genetic algorithms. In *Proceedings of the Bird of a Feather Workshops(GECCO2003)—Learning, Adaptation, and Approximation in Evolutionary Computation*, 2003.

[7] Chris Gathercole and Peter Ross. Small populations over many generations can beat large populations over few generations in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 111–118, Stanford University, CA, USA, 13-16 1997. Morgan Kaufmann.

[8] Neil Gershenfeld. *The Nature of Mathematical Modelling*, chapter 13, pages 156–168. Cambridge University Press, 1999.

[9] G. J. Goodhill. Gradients for retinotectal mapping. *Advances in Neural Information Processing Systems*, (10):152–158, Dec 1998.

[10] G. J. Goodhill. Mathematical guidance for axons. *Trends in Neuroscience*, 21(6):226–231, 1998.

[11] G. J. Goodhill. A mathematical model of axons guidance by diffusible factors. *Advances in Neural Information Processing Systems*, (10), 1998.

[12] G. J. Goodhill. Models of neural development. In *Encyclopedia of Cognitive Science*, pages 261–267. Nature MacMillan Publishers, 2002.

[13] G. J. Goodhill and H. G. Barrow. The role of weight normalization in competitive learning. *Neural Computation*, 6(2):255–269, 1994.

[14] G. J. Goodhill and L. J. Richards. Retinotectal maps: molecules, models and misplaced data. *Trends in Neuroscience*, (12):529–534, 1999.

[15] G. J. Goodhill and T. Sejnowski. Quantifying neighbourhood preservation in topographic mappings, 1995.

[16] John Koza. *Genetic Programming*. MIT Press, 1992.

[17] Hod Lipson and Jordan B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, August 2000.

[18] Melanie Mitchell. *An Introduction to Genetic Algorithms*. The MIT Press, 1998.

[19] Patrick Monsieurs and Eddy Flerackers. Reducing population size while maintaining diversity. In *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *LNCS*, pages 145–156, 14-16April 2003.

[20] G. Scott Owen. Eye and brain – part of a vision explanation, 1999. `http://www.siggraph.org/education/materials/HyperVis/vision/eyebrain.ht%m`.

[21] M. C. Prestige and D. J. Willshaw. A role for competition in the formation of patterned neural connexions. *Proceedings of the Royal Society of London B*, 190:77–98, 1975.

[22] R. Sperry. Chemoaffinity in the orderly growth of nerve fiber patterns and connections. *Proceedings of the National Academy of Sciences of the United States of America*, 50:703–710, 1963.

[23] David J. Willshaw and Christoph von der Malsburg. A marker induction mechanism for the establishment of ordered neural mappings; its application to the retinotectal problem. *Philosophical Transactions of the Royal Society of London, Series B*, 287:203–243, 1979.