Simulating Vegetation Growth in Arid Geographies

May 2017

Edward R. Bennigsen
Supervised by Jonathan Shapiro

Third Year Project Report

School of Computer Science
University of Manchester

Abstract

Vegetation in arid geographies has been shown to grow in cluster patterns, and these cluster patterns can be described using a power law. Power laws are a hallmark of self-organisation, and point to local facilitation driving large-scale change.

This local facilitation has been modelled previously [1], and this project re-implements these models. The results of the original paper are replicated, and then the model is extended to include a competitor species. The relationship between competition and self-organisation, as well the relationship between competition and dominance, is investigated.

The development of the implementation of the models is discussed, followed by a presentation and analysis of experimental results from the simulation application. Competition is found not to disrupt self-organisation, and dominance is found to be linked to competition. Finally, there is some reflection on the project as a whole.

Acknowledgements

Thanks to my project supervisor, Jonathan Shapiro, for all his help, advice, and support. Thanks also to my family for listening to me talk about vegetation and giving me a place to rest and revise from. Finally, to my cactus Rupert for the solidarity he provided and the first-hand insight into vegetation growth in arid geographies.

Contents

Chapter 1

Introduction

The purpose of this project is to investigate and simulate vegetation growth in arid geographies. A large part of the modelling and analysis focuses on self-organisation and how local facilitation leads to global phenomena. It replicates and builds on previous work in this field, and then extends those ideas to investigate how competing species fare in an arid environment and how this affects self-organisation. Understanding how vegetation grows and is maintained is important for anti-desertification efforts and hence the technical modelling here has a degree of real-world applicability. This report deals with the project as a whole, from modelling the phenomenon, to developing an implementation, presenting and analysing experimental results, and evaluating the project.

1.1 Self-organisation in vegetation growth

The bulk of the previous work in the field used in this project comes from a paper published in Nature in 2007, "Positive feedbacks promote power-law clustering of Kalahari vegetation" by Scanlon et al. [1]. The paper analyses vegetation patterns in transects of the Kalahari Desert, using data acquired from satellite imagery. An example of this can be seen in Figure 1.1. The satellite image shows that the vegetation is arranged in clusters, i.e. vegetation is more likely to be present near other vegetation. These clusters can be described using an inverse cumulative probability distribution $P(A \geq a)$: the probability that a given cluster will be of size A greater than or equal to a. When this is graphed against cluster size a, a power law emerges.

A power law is expressed generally as $y \propto x^k$ and is more specifically defined here as below:

$$P(A \geq a) \propto a^{-\beta} \tag{1.1}$$

Note that the exponent is negative as the likelihood of a cluster being a large cluster decreases as cluster size increases.

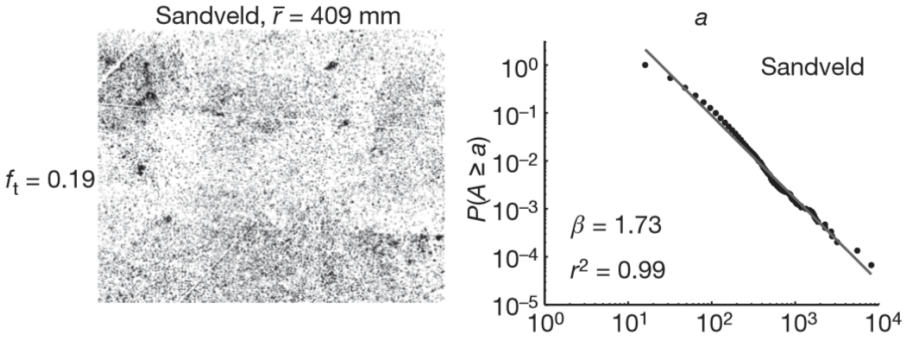

Figure 1.1: Example data and analysis. On the left is the satellite data showing a region of desert near Sandveld. The black pixels are vegetation and the white pixels are desert. The image covers a scale of 2km x 2km at a resolution of 4m. On the right is a log-log graph of cluster size against an inverse cumulative probability distribution of the clusters. As is evident from the straight line fit, a power law is present here (β is the negative exponent).

Power laws in data from natural phenomena are important because they "are a fingerprint of self-organization ... the result of internal dynamic processes driven by local interactions" [2]. Hence Scanlon et al. go on to model the ecosystem, including local interactions, to try and replicate the quantitative data through simulation and show that it is plausible that the power law is arising from self-organisation. The mechanism behind the self-organisation is a process of positive local feedback. Effectively, vegetation is more likely grow near other vegetation, and more likely to die if not near any other vegetation. The properties of the original data are successfully replicated in the original paper using this model. In this project, the model is re-implemented and the power law successfully replicated.

1.2 Competition and self-organisation

One of the aims of the project stated above is to investigate how competition between vegetation species affects self-organisation and growth in the desert. In other words, if there are multiple species competing for the same resources, will they still self-organise or will the competition be so destructive that any global phenomena such as clustering will no longer be present? Does local negative feedback outweigh local positive feedback?

As previously discussed the measure for self-organisation used here is a power law, and so graphs for this are generated to investigate if power laws are still present with competition. The competition is implemented in such a way that the level of destructive feedback between the specie can be varied. This leads to graphs investigating the relationship between competition and dominance, i.e.

as competition is increased, does one species dominate?

Further details on the models can be found in the next chapter.

Chapter 2

Modelling the ecosystem

As mentioned previously, the application written for this project re-implements the model described in "Positive feedbacks promote power-law clustering of Kalahari vegetation" by Scanlon et al. [1], and implements an extension of the model that includes a competing species. This section explores both Scanlon et al.'s model, as well as the extended competitor model.

2.1 Scanlon et al's model

Scanlon et al. model the desert as a regular grid of cells. Each cell is of one of two types, vegetation (v) and desert (o). The grid is randomly initialised so a certain percentage of the cells are vegetation (also just called trees in the paper), and the rest are desert. Rules define transitions from one state to another, carried out over a number of "years" or "ticks". The transition laws are based on a global factor (simulating rainfall) and a local factor, namely vegetation density in the region. The grid for the model in the paper is 500x500 cells in size.

The model described is an example of a (two-dimensional) cellular automaton. Cellular automata were first described by John von Neumann and Stanislaw Ulam in the 1940s & 1950s [3], but rose to prominence with Conway's Game of Life. [4] in the 1970s.

Every tick, 20% of the cells are randomly selected for transition. The probability of transition (from the cell's current state to desert or vegetation) is calculated based on global and local factors, and the cell is transitioned based on that probability.

The global factor, designed to simulate rainfall in the region, states that there is a limit on how many vegetation cells the grid can support, e.g. 30% vegetation. It is set up in such a way that the model tries to maintain a constant level of vegetation: too many vegetation cells and they will be more likely to die, too few and they will be more likely to grow. This means that the vegetation cannot die out or expand to cover the whole grid. Another way of saying this is that the model tries to align overall fractional vegetation cover,

f_v, with the number of vegetation cells the grid can support, f_v^*. The transition probability can be expressed as equations: $P(v) = \rho_v + (f_v^* - f_v)/(1 - f_v)$ and $P(o) = \rho_v + (f_v - f_v^*)/f_v$, where $P(x)$ is the probability of transitioning to state x, and ρ_v is the local factor explained below. The fraction in terms of f_v and f_v^* is the global factor aligning the two variables.

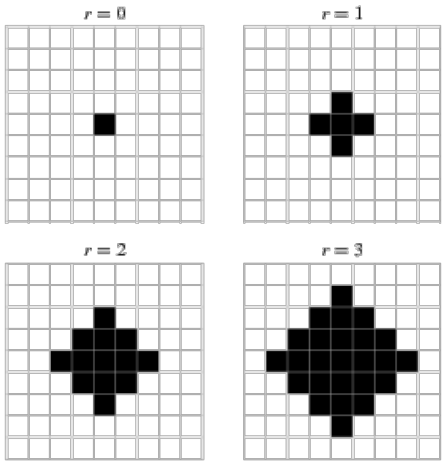

Figure 2.1: Image describing the von Neumann neighbourhood. The von Neumann neighbourhood is all cells at a Manhattan distance of r or less. It can be described mathematically as $V(r) = 2r(r + 1) + 1$ [5].

The local factor is modelled by using the neighbourhood vegetation density ρ_v. Vegetation density is a weighted function of distance, d from the target cell. Distance is measured using the Manhattan distance, i.e. the shortest path along orthogonals between two points. The distance is weighted using the Pareto-like function $(1/d)^k$, where k is the weighting factor. k was set to 3.0 for all simulations in the original paper and in most of this project's implementation's simulations. As k increases, greater weight is placed on the vegetation closer to the target cell. This is defined using the following equation:

$$\rho_v = \sum_{\Omega_{i,j,M}} (1/d)^k x_{i,j} \Bigg/ \sum_{\Omega_{i,j,M}} (1/d)^k \tag{2.1}$$

$\Omega_{i,j,M}$ represents all positions i, j in a von Neumann neighbourhood (explained in Figure 2.1). of distance or radius M. $x_{i,j}$ equals 1 for a vegetation cell and 0 for a desert cell. M is important for performance concerns when implementing the equation in code: if it is too high, it will take too long to calculate the density. See Subsection 3.4.1 for a discussion of this.

The paper points out that the power-law clustering is not contingent on the Pareto weighting scheme, and describes other neighbourhood weighting metrics (exponential and linear). An exponential weighting metric was also implemented and, as described, the clustering was still present.

2.2 Extending Scanlon et al.'s model to include a competitor species

For the extended model, a competitor vegetation species is introduced into the simulated ecosystem and its effect on vegetation growth and clustering is measured. The addition of competition adds negative local feedback to the positive local feedback in Scanlon et al.'s model. A large part of the investigation into competing species in the project centres around seeing if the negative local feedback destroys the self-organising behaviour that emerges from positive local feedback.

There are two competing vegetation species, v_1 and v_2, that behave identically to each other, as well as desert cells o. The grid is initialised with a certain proportion of v_1 and v_2 cells, with the remaining cells being v_o. The transition probability is based on the equation described in the original model, with the global factor remaining the same and density ρ_v being part of the local factor. ρ_v remains the same, except that it is calculated individually for v_1 and v_2 so is better described as ρ_{v_1} and ρ_{v_2}. A competition factor c is also introduced. The equation for transition probability is hence defined this like:

$$P(v_1) = \rho_{v_1} - (c * \rho_{v_2}) + (f_v^* - f_v)/(1 - f_v) \tag{2.2}$$

c (set between 0 and 1) controls the level of negative feedback between the two species: the higher it is the less likely the species are likely to grow near each other, and vice versa. $P(v_2)$ is equivalent but v_1 and v_2 are switched. Additionally, $P(o) = 1 - P(v_1) - P(v_2)$.

Note the differences between the two models (apart from competition). In the extended model, the combined vegetation coverage is still maintained, however per-species coverage is not. In this way, one species can dominate the ecosystem, even driving the other species to extinction. This is a deliberate feature, indeed the relationship between dominance and the competition factor c is explored in Section 4.5. Another difference is the definition of $P(o)$ and the fact that it no longer attempts to maintain the global constraint as well. It was easier to define it as above and it made no apparent difference to maintaining the constraint in practice.

Chapter 3

Development of the simulation application

The project was undertaken from September 2016 to March 2017, and a significant amount of the time spent on it was allocated to writing, testing, and debugging code. This section explains the choice of languages and platform, issues that arose during development, and how the software was tested and evaluated.

3.1 Platform and language choice

The original paper did not describe or provide the implementation of their model, so the best way to implement it and the extended model had to be decided. Considering a platform meant considering what the software artefact was designed to achieve and what platform was best suited to achieve this. The programming language was the main area of consideration, but supporting tools such as operating system, development environment, and version control system, were also considered.

3.1.1 Language choice

Effectively, the model consists of a large data structure (the grid of the cellular automaton) which is acted upon many times per tick. The probability calculations involve looking at a large number of cells for each calculation. Thus the language chosen needed to be able to handle large, custom data structures and be fast enough to perform hundreds of floating point calculations per tick and still maintain a reasonable running time. As the original model was being implemented, and then extended, the language also needed to offer good code re-use.

Initially Scala was considered. Scala is a functional language and as such methods have no side-effects. This can help in reducing software defects. Scala

is also highly parallel which is useful for trying to maximise performance and speed, and is object-oriented so code re-use is easier. I did not have much previous familiarity with this language, but learning a new language was part of the motivation.

Java was another possible language choice. Java is the language I was most familiar with and as such development would not involve a large amount of time learning the language. Java is object-oriented as well, and Java 8 introduced functional-like features such as streams so it can partly enjoy some of the benefits that Scala possesses. Importantly, Java has Maven for package management, making importing libraries much easier. As the project was being developed across multiple computers, not having to deal with classpath management was a large benefit. I was also very familiar with JUnit, Java's standard unit testing suite. Use of unit testing was desirable for its many benefits; see 3.5 for more details.

C++ was the final language considered. I had some previous experience with C++ but not as much as Java. C++ does not use a virtual machine like Java and Scala and as such can be very fast. It is object oriented and has powerful data structures.

In the end Java was chosen for multiple reasons. Firstly, learning a new language (in the case of Scala) or learning a lot more about a language (C++) would take up valuable development time. Secondly, the supporting ecosystem for Java (Maven and JUnit) would make working on the project much easier. All the languages examined possessed features allowing for good code re-use and large, fast data structures so this was not an issue. There were some concerns over the speed and scalability of Java, however this was seen as unlikely to be a large problem.

3.1.2 Platform choice

Once Java was chosen as the primary language, IntelliJ was chosen as the IDE. It has excellent tools for Java including static code checking, syntax highlighting, and auto-completion. It is also cross-platform, which is important because the code was developed on Windows and Linux.

Git was chosen for version control for its power and flexibility, and GitHub to act as a host server for the code. Using a version control system allowed the use of branchesand code history, useful when developing a large project. Hosting the project on GitHub meant that transporting code between development machines was much easier.

I used Kanban for project management. Kanban is an agile methodology designed to divide work up into discrete tasks and ensure they are completed properly after passing through a number of stages [6]. Trello, an online Kanban "board", was used to host the tasks and track their progress.

Part of the project involved creating images of the grid, exporting data to text files, and creating graphs of the data. It was initially thought that this would be implemented using Java libraries and part of the motivation for using Maven was so libraries to do this could be easily imported. Creating images

and exporting data to text files turned out to be possible using the standard libraries and so did not require external libraries. Graphing libraries were looked at for Java, but having decided a graphical interface was not necessary these were ignored. Instead, Gnuplot was chosen.

Gnuplot is an easy-to-use graphing application powered by a command line interface, and possessing a scripting interface. It is available on Windows and Linux so fitted the cross-platform requirement. By writing scripts, data from the application could be easily and repeatedly graphed. Switching to Python for graphing was briefly considered, but decided against as graphing scripts had already been written in Gnuplot and the issues with Gnuplot were not as large as previously thought.

3.2 Code style and structure

Whilst the model is relatively easily described through text and equations, translating it into code is not as simple. As Java is an object-oriented language, code constructs are generally thought of as "nouns", i.e. objects. Some of the concepts in the model, such as the grid, lend themselves to this, but constructs like the density equation do not. To address this, generally an object was made solely for a function (for example a DensityMetric class was created for the density equation), or the function was added as a method to an object (the equations for calculating the transition probabilities were attached to the relevant simulation classes).

Having read Clean Code by Robert C. Martin [7], I used the guidance there as a basis for code style. This emphasises, amongst other stylistic advice, small objects and the single responsibility principle (SRP). This often meant splitting up a larger object into smaller functional units. Code readability was a goal throughout, with detail often abstracted. This included making the code structure roughly follow the model structure outlined in the paper, making cross-referencing between the two easier. Readable code is much easier to maintain, especially important on a large project conducted over several months.

From the beginning of the development process, the code was designed to be easy to extend. I began with the original model, and because of my modular, class-based approach, I could easily add features to support the extended model. For instance, a base Cell class was created, which was extended to create a Vegetation class, and later a CompetitorSpecies class. Furthermore, code that behaved identically between the two models (e.g. cluster analysis) was re-used.

3.3 Application structure and usage

The simulation application really consists of several applications working together. This section describes these sub-applications

3.3.1 Simulation

The simulation part of the application runs in three distinct modes: one for simulating the Scanlon et al. model, one for simulating the competitor model, and one for simulating multiple runs of the competition model to investigate dominance. These modes all work in roughly the same way. They take parameters defining the grid size, proportion of cells to update each tick, number of years to run the application for, proportion of species to initialise, and a density metric. In addition, the competitor model simulation allows you to specify the competition factor.

The single run modes initialise and run the simulation, then output the data. This is done in the form of `.data` files describing the clustering probability distribution and an image of the grid in `.png` format. The clustering probability distribution has calculated first.

The multiple run mode does several runs of the competition mode, increasing the competition factor each time. The fractional step to increase the factor by can be set, and it will take the factor from 0.0 through to 1.0. Dominance is calculated each simulation run (see Section 4.5 for more details). This mode does not output images or data files for individual runs, rather it outputs a data file containing competition factor against dominance over time.

3.3.2 RunCombiner

This application allows multiple runs to be combined. It was originally implemented quickly, so it simply appends together the datafiles it is given. Initially, it was thought that this would be needed but it turned out to not be, so it was rarely used and not improved from its simple first implementation.

3.3.3 FitScriptCreator

The Gnuplot scripts did not support parameterisation, so they had to be done via this application. Subsection 3.4.2 fully describes the necessity and workings of this application

3.3.4 Gnuplot scripts

As mentioned previously, Gnuplot was used for plotting graphs of the data output from the simulation. A number of Gnuplot scripts were written to do this, one for each mode of the simulation application. Scripts were also created to easily setup the graphs (axis titles etc) and save the graphs as images.

3.4 Issues faced during development

Development was relatively smooth but had issues along the way. This section discusses these issues and how they were overcome.

3.4.1 Performance

The main issue faced was that of performance. To aid with iteration and testing, the simulation has a functional requirement in that it should run in a short amount of time (less than five minutes). A long simulation time would have vastly increased the length of the feedback loop, slowing progress on the project.

von Neumann neighbourhood

The primary speed bottleneck was the vegetation density calculation. Each tick, 50,000 cells are selected for transition (20% of cells on the 500x500 cell grid), and each of those cells requires a density calculation to calculate their transition probability. As evident from Equation 2.1, this requires looking at all cells in the von Neumann neighbourhood of this cell. The number of cells in a von Neumann neighbourhood of radius/range r can be expressed as such: $V(r) = 2r(r + 1) + 1$ [5]; it is of the order $O(n^2)$. Hence wisely choosing a maximum value for r (or M as it is called in Equation 2.1), is an important performance consideration.

In the density equation, the distance is weighted using a Pareto-like function, $(1/r)^k$, so as the distance increases, cells contribute exponentially less to the density. This means that as the number of cells increase exponentially, they are exponentially less important. As $k = 3.0$ in all simulation runs, there is a point at which cells make a negligible contribution to the density. Through some empirical experimentation and some basic calculations, 10 was decided on as the maximum radius of the von Neumann neighbourhood to be used.

Proportion of vegetation cells

The density calculation also requires knowing the proportion of vegetation cells on the grid. Originally, the application was calculating this every density calculation by a method call on the Grid object that ran through all cells to find the number of vegetation cells. This call took a long time and slowed down each tick considerably.

To fix this, the Grid class was changed to keep track of the number of vegetation cells consistently, through one large calculation on initialisation, and then adding or subtracting one on get/set calls when a vegetation cell was added to or removed from the grid. This greatly improved running speed, but had to be changed again when multiple types of vegetation cell were added via the extended model.

3.4.2 Gnuplot paramaterisation

After writing graphing scripts using Gnuplot, it was discovered that they were not parameterisable. That is, they do not support passing in of data filenames. This stopped there from being a universal graphing script that could have data passed into it. To fix this, a Java interface for Gnuplot was considered, as well as the possibility of switching to Python and associated libraries, but neither of

these solutions seemed satisfactory as they would still require rewriting a lot of code. Instead, I a simple script was written in Java to do the parameterisation. The script takes in a data filename, and makes a copy of the graphing script with the data filename inserted. In this way every data file has a custom graphing script for it.

3.4.3 Probability distribution

The final major issue was that for several weeks or even months the probability distribution did not work. The graphs created from the data did not fit a power law because the probability distribution was not programmed correctly. This turned out to be due to a misunderstanding of how the distribution was constructed, and it was fixed. A more detailed explanation of the distribution itself is available in "Section 4.1 Probability distribution".

However, it was not immediately obvious that it was the probability distribution code that was at fault. The other alternative was the simulation itself had been programmed wrong, or that the model was simply faulty. As the original model was peer-reviewed and published in Nature, that was unlikely. A large review of the simulation implementation was conducted, followed by a review of the probability distribution code. Eventually, by thinking about the distribution in more detail, the error was brought to light. Despite the time appearing wasted, the examination of the simulation application did cause a number of other errors to be found and fixed.

3.5 Testing and software evaluation

. As mentioned previously, JUnit, Java's automated unit testing suite, was a major motivation for using Java. JUnit was used heavily throughout development via the Test Driven Development (TDD) methodology.

TDD involves writing tests for a method before the bulk of code, in order to specify and validate the method[8]. Tests are written to handle general case scenarios as well as edge cases. In this way, code can be written iteratively, first to handle the general case, and then modified later to fulfil the edge case tests. In some cases an edge case may be unforeseen and a bug found at a later date through other testing. When this happens, extra tests are added to cover the edge case and the code is modified to pass these tests.

Importantly, the tests also act as the specification of a method. This means that ensuring the test are comprehensive and correct is very important, but it does mean that once code passes all the tests for a method it is completely correct. This speeds up development because code is shown to be correct through automated testing, and the developer is not left worrying whether the code is fully correct or not. Of course, the specification could be wrong or be changed at a later date, but the issue then is with the specification, not the code itself.

Automated testing greatly improved the code-writing process for me. I had confidence that my code worked as intended, and when I changed code I had a

ground truth to reference. If a piece of modified code failed to pass its unit tests, I knew that either the code was incorrect or that it had changed definition and needed different tests. This was especially helpful for mathematical functions like the density equation. I hand-calculated answers to specific test data, and wrote tests based on those answers. When a larger piece of code was not working, I had assurance that the smaller units were working correctly as defined, and thus it must be an issue with my specification rather than implementation.

TDD was mostly followed throughout the development process, however not always. Writing detailed tests before writing a method can be time-consuming, and as the deadline approached code was increasingly written before tests, or without tests at all. This is not ideal, rather an unfortunate side-effect of finishing a project within time constraints.

Automated testing was one way of evaluating the software, and performance was another way. Tests ensured the software ran correctly, and the individual simulation run time was kept under the five minute limit. Once the models were implemented, and both of these requirements were met, the project was close to completion.

Chapter 4

Presentation and analysis of experimental results

This chapter discusses the experimental results acquired from running the final implemented models. The results are presented and analysed for both model types, along with discussions on the probability distribution used and truncated power laws.

4.1 Probability distribution

This section describes the probability distribution used in more detail. As discussed in Chapter 1, the probability distribution $P(A \geq a)$ is an inverse cumulative probability distribution. It describes the probability that a given cluster will have size A greater than or equal to a. Generally a distribution like this is created by adding up individual probabilities, i.e. all $P(A)$ where $A = a$, but it is calculated slightly differently here to improve performance, the difference being that they are only normalised to fit between 0 and 1 until the end of the calculation.

Initially, the power law did not seem to be present because the distribution was not reaching the same orders of magnitude as the examples given in Scanlon et al. (the probabilities were not low enough). It was initially thought that the issue was with the simulation, but this was shown to probably not be the case. This encouraged a closer examination of the distribution and it was found that it had been misinterpreted.

Say there is single cluster of size 6 in the grid. Surely $P(A \geq 6) = 1$? However, this is not the case. There are actually "hidden" clusters within that cluster: 6 of size 1, 3 of size 2, and 2 of size 3. Hence the probability of a cluster being of size 6 is much lower. Once was this fixed in the code implementation, the probability distributions created matched more closely those found in the original paper. This confusion did cause a delay of several weeks in the project however.

4.2 Replication of Scanlon et al.'s results

Replicating the Kalahari model from Scanlon et al.'s paper was the first goal for the project, and this was achieved successfully. The simulation application produced numerical data describing the clustering, a graphing script for that numerical data, and an image of the grid post-simulation. All of the project's simulations were run for 500 years, and at the end of those years the data was produced.

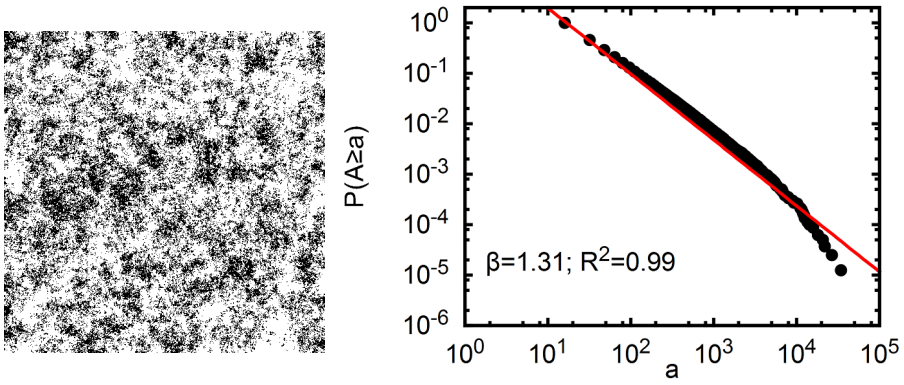

Figure 4.1: Left: Graphical representation of the grid after 500 years of simulation. The black pixels are vegetation and the white pixels are desert. Right: Log-log graph of cluster size a against the inverse cumulative probability distribution over cluster size $P(A \geq a)$.

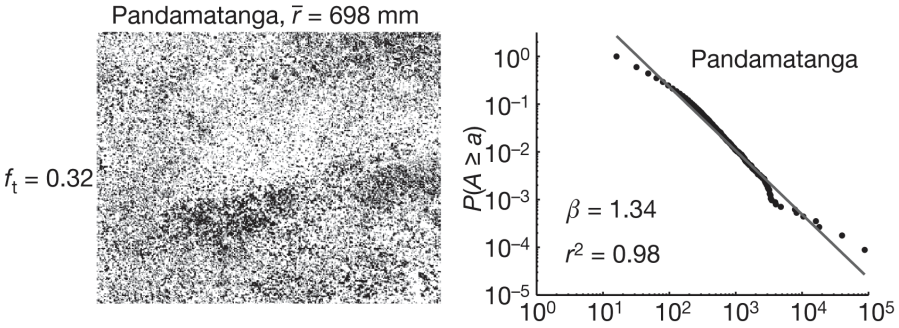

Figure 4.2: Satellite image and cluster analysis graph of the Pandamatanga region of the Kalahari

Figure 4.1 contains the image output and a graph produced from the graphing script. See caption for more detail. The equation used for line of best fit for the graph is $ka^{-\beta}$. This is derived from Equation 1.1 in Chapter 1. β and R^2 are both given to two decimal places. The global rainfall constraint f_v (called

f_t in the paper) is 0.32.

Figure 4.2 shows real data taken from Scanlon et al. As we can see, the data is similar to the simulation run with the same parameters. The clustering patterns in the simulation appear more random and there are less large clusters[1]. The graphs are also reasonably similar, with β being almost equal in both cases. However, the clusters from the satellite data reach a larger area than the clusters from the simulation.

If we compare the simulation results with Scanlon et al.'s simulation results (seen in Figure 4.3, the results are very similar. Both have a similar β, and both differ from the satellite data in that their clusters do not reach the same scale. The images also look similar.

When a simulation is run with just the global constraint and no local constraint, power-law clustering does not emerge. This shows that it is the local positive interactions that drive self-organisation

Hence we can see that the simulation application successfully replicates the behaviour found in Scanlon et al., which in turn replicates the real-world clustering phenomenon. In this way, the first goal of the project is met.

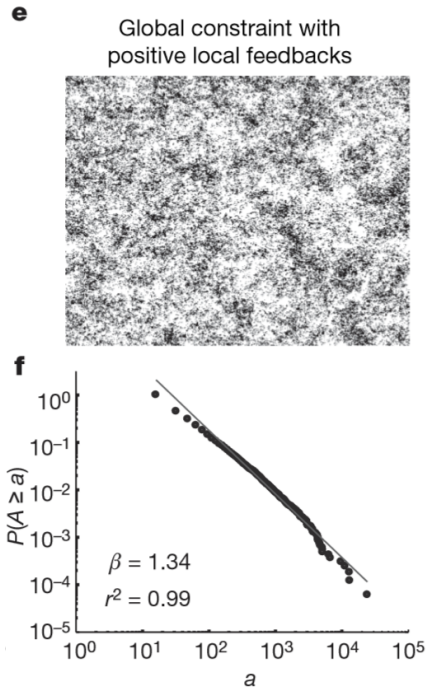

Figure 4.3: Grid image (e) and cluster analysis graph (f) of a simulation from Scanlon et al. where $f_v = 0.32$

4.3 Truncated power laws

As seen in both this project's and the Scanlon et al.'s simulation graphs, the cluster sizes achieve do not match the satellite data. This causes a deviation from the power law at higher points. This implies that the simulation is not as effective as a increases. However, a probable explanation for this is the boundary conditions of the grid [9]. The satellite data is a grid taken from a continuous space, and the vegetation and desert continue beyond the image. In the simulation, nothing exists outside the edges of the grid. Any cell on the edge of the grid has 2 or 3 neighbours rather than 4. Hence different behaviour

[1]The simulation image may appear darker or to have more vegetation. This is due to a slightly different colour palette to the satellite image rather than a quantitative difference

is expected here

To model this, a truncated power law can be used for the line of best fit, as seen below. The equation used for line of best fit for the graph is instead. The k term is just a constant found through linear regression, but the $exp(-a/c)$ term is slightly more complicated. It allows the end of the graph to curve on the log-log graph, in order to fit the data better, i.e. it allows the power law to be truncated or the line to deviate from the law. This idea from was taken from "Spatial vegetation patterns and imminent desertification in Mediterranean arid ecosystems" by Kéfi et al. [10].

$$ka^{-\beta} * exp(-a/c) \tag{4.1}$$

Figure 4.4 shows this truncated power fitted to the data from the simulation in Figure 4.1. The curve fits the data more closely and R^2 is high enough to round to 1. This shows that the truncated power law better describes the simulation data.

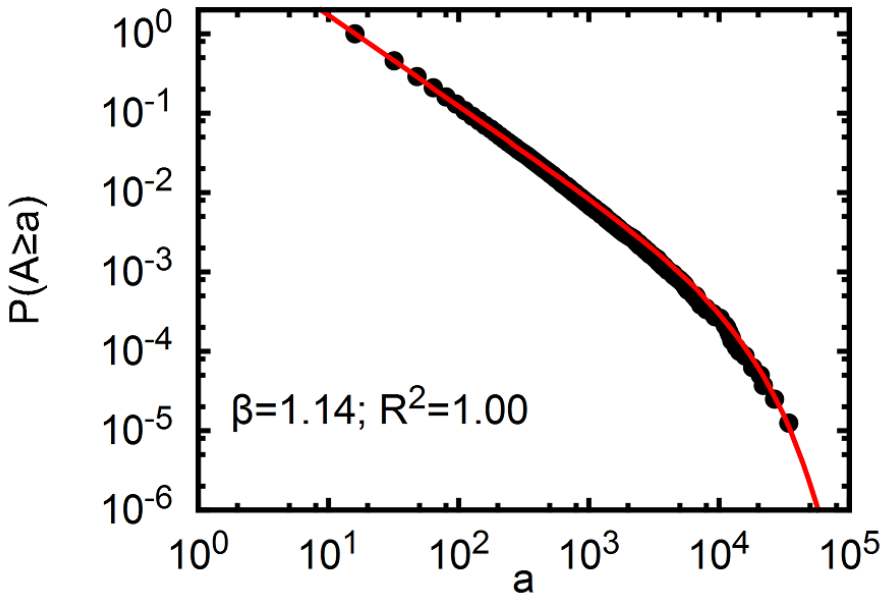

Figure 4.4: Cluster analysis graph with truncated power law. Same data as Figure 4.1

4.4 Competition and clustering

After Scanlon et al.'s results were successfully replicated, the model was extended to include a competing species of vegetation, as described in Section 2.2.

The main goal of the investigation for this model was to see if the power law still held even when competition was introduced.

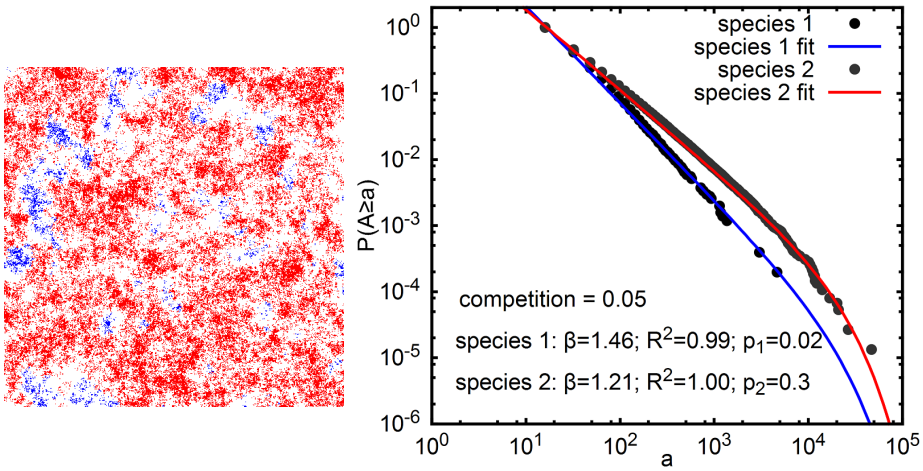

Figure 4.5: Left: Graphical representation of the grid after 500 years of simulation. The blue pixels are species 1 vegetation, the red pixels are species 2 vegetation, and the white pixels are desert. Right: Log-log graph of cluster size a against the inverse cumulative probability distribution over cluster size $P(A \geq a)$.

Figure 4.5 shows the results from a competitor model simulation. The competition factor, c, is set to 0.05 and the model was run for 500 years. To match the Scanlon et al. simulation run previously, f_v is set to 0.32. The initial proportion of species is divided unequally, with the proportion of vegetation species 1, p_1 set to 0.12 and the proportion of vegetation species 2, p_2, set to 0.2. Throughout the simulation, $f_v = p_1 + p_2$, to maintain the global constraint.

The first point of interest in Figure 4.5 is the change in species proportion. p_1 has decreased to 0.02 and p_2 has increased to 0.3. From this we can see that the advantage given to species 2 at the start of the simulation has benefited it, and it is continuing to dominate the environment. The second point of interest is that there is a difference in scales between the two datasets, with species 2 achieving a cluster almost of size 10^5 and species 1 only achieving a cluster around 10^4. This is to be expected as species 1 has considerably less cells than species 2. Numerically, species 1 has 10x less cells than species 1, and its largest cluster is 10x smaller.

Finally, both species' clustering still follows a power law. Both species' data are fit to the truncated power law described in the previous section. Species 1's fit is almost a straight line and the "tail" behaviour does not really begin until after the final datapoint. Species 2's fit has more of a tail-off, probably due to the boundary effect described earlier. The fact that the power-law still holds implies that this degree of competition does not disrupt the positive local

feedback enough to destroy it. In other words the positive local feedback within species is stronger than negative local feedback between species.

What happens when competition is increased? Figure 4.6 describes a simulation with the same parameters as the simulation in Figure 4.5, but with a competition factor of 0.2 instead.

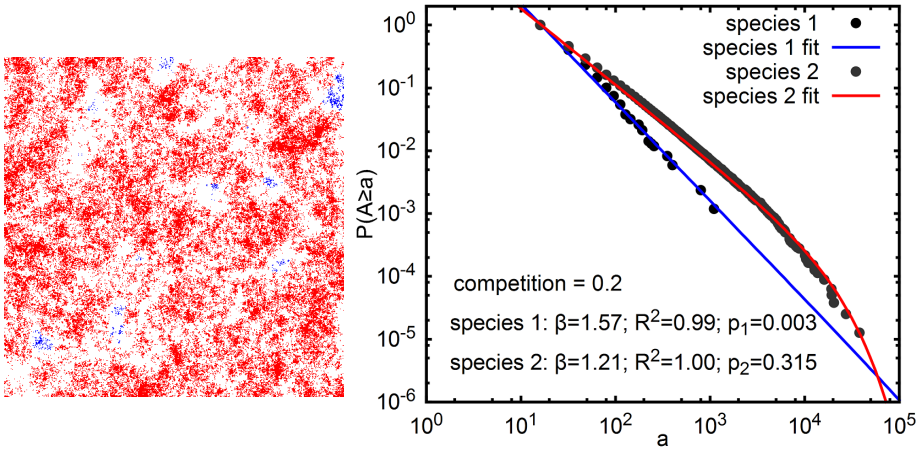

Figure 4.6: See 4.5 for description.

As we can see, there are barely any species 1 type vegetation left: there are 100x fewer cells of species 1 than species 2. Despite this, both species cluster distributions are still following a power law. This is evidence that even as competition increases, power laws are still maintained. In other words, the positive local feedback methods are robust enough to survive competition.

The competition factor c has only reached 0.2 so far. What happens when it is 1.0, when the negative feedback is as strong as the positive feedback? When a simulation is run with the same parameters as previously (but with $c = 1.0$), species 2 totally dominates, with species 1 driven to extinction, i.e. there are no species 1 cells left. This is to be expected because they do not start out with the same proportion of vegetation. Species 2's clusters follows a power law.

The simulation in Figure 4.7 instead starts off both species with an equal share of the vegetation, 0.16 each. The data is produced after 500 years of simulation with a competition factor of 1.0.

From the graph, we can see that, again, both species' clusters follow a power law. There is a higher level of tail behaviour evident here, possibly due to competition. However, the tail behaviour is not so large to point to a full breakdown of the power law at higher cluster sizes. Both species reach a similar scale of cluster, and species 2 manages to take a slight lead in cell proportion. This is due to the random nature of the simulation. From other simulations undertaken, it is evident that small perturbations in the proportions early on can lead to dominance (see Section 4.5).

Overall, there is clear evidence that destructive local feedback via competi-

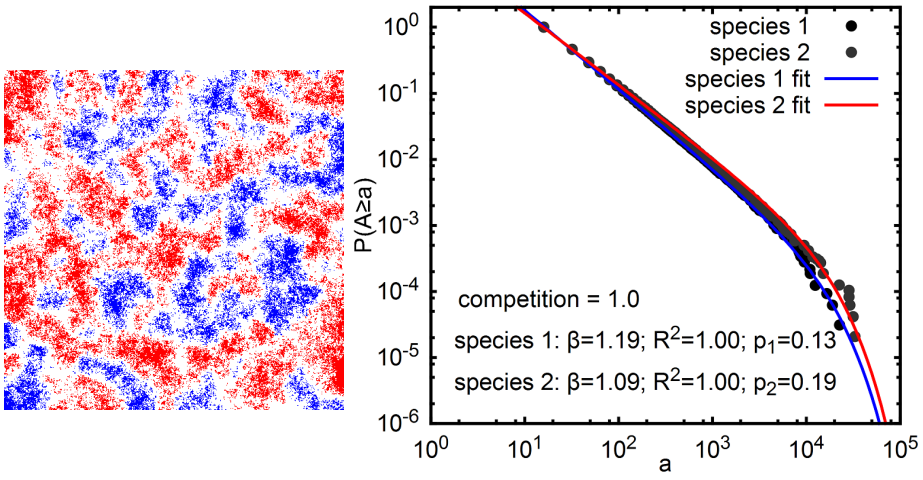

Figure 4.7: See 4.5 for description.

tion does not disrupt the positive local feedback, even when competition is at its maximum. It is possible for one species to die out, given enough competition and a large enough dissimilarity in starting proportions. These experimental results suggest that competing species with similar characteristics will not break down the self-organising behaviour described in the power law. Rather, the species form homogeneous clusters which exhibit strong self-organisation.

4.5 Competition and dominance

The final area of investigation was into competition and dominance. Dominance is defined as the most important species in an ecosystem [11]. A good variable to measure importance, and by extension dominance, is biomass [12]. In the simulation, biomass is represented as the proportion of cells that are of a certain species. Hence we define dominance in the simulation as the species with the greatest proportion of cells.

Rather than a binary dominance measure, a continuous measure, d is used. This is defined as $(n_1 - n_2)/(n_1 + n_2)$, where n_1 and n_2 are the proportion of species 1 and species 2 cells respectively. If species 1 is dominant, the dominance measure is positive. If species 2 is dominant, the dominance measure is positive.

Figure 4.8 shows a plot of multiple runs. As stated in the caption and visible from the graph, species 1 starts out dominating the ecosystem. The competition factor c is increased in steps of 0.05 from 0 to 0.5. As the dominance measure is rapidly approaching 1 by this point, the simulation is cut off to save on time. It is theorised that the relationship between dominance and competition is asymptotic, i.e. $d \propto 1/c$. A function of this form has been fitted to the available data. The high R^2 value implies that there is indeed a relationship, and that it is asymptotic. This function holds true whenever a species starts

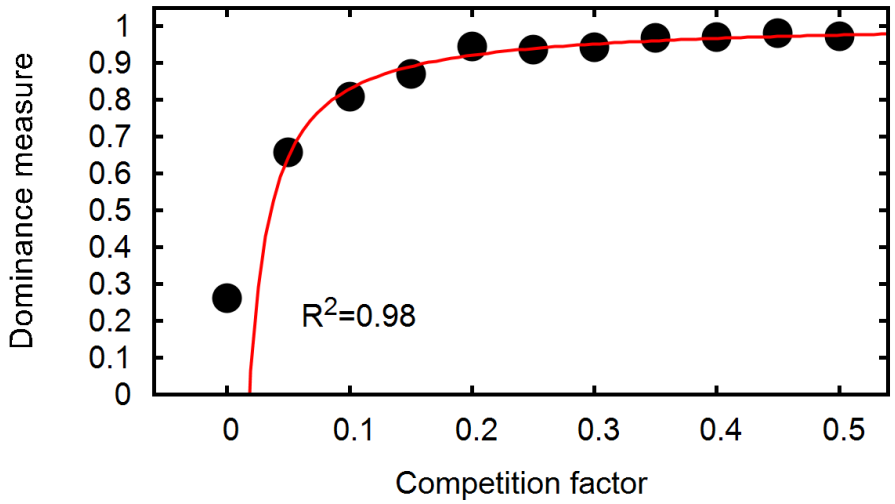

Figure 4.8: Plot where each point represents a simulation run. As competition increases on the x-axis, the dominance measure varies on the y-axis. Initial cell count proportions in each simulation are 0.2 and 0.12 for species 1 and species 2 respectively, and the dominance is calculated after 200 years. The data is fitted to an asymptotic curve.

off as dominant, although the graph would be mirrored on the x-axis if species 2 dominated.

However, what happens when a species does not begin in a position of dominance? The result of such a simulation is described in Figure 4.9. The dominance effectively follow a random walk. Despite one species occasionally taking a larger lead in a simulation, neither manages to achieve a dominance measure of ± 0.3 or greater and most dominance measures are in the range -0.1 to 0.1. The random nature of the plot is to be expected as the grid's initialisation is slightly random (one species might have a slight edge over the over), and an equal number of each species is not necessarily selected for update each year.

This implies that some initial dominance is required for a species to then fully dominate an ecosystem. If a competing species was introduced to a homogeneous ecosystem, and behaved as modelled here, introducing an equal amount of vegetation to the current species would ensure its survival. Further investigation could be done here to study the relationship between dominance at the end of the simulation and initial dominance.

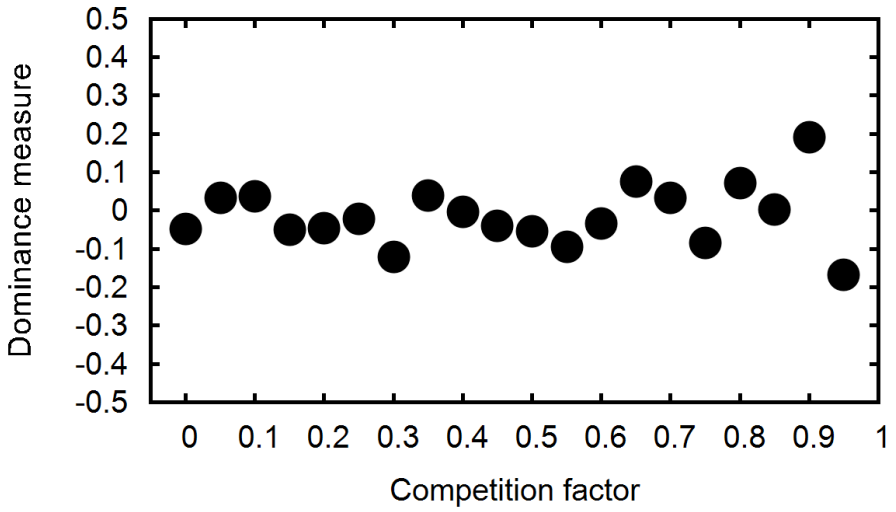

Figure 4.9: Plot where each point represents a simulation run. As competition increases on the x-axis, the dominance measure varies on the y-axis. Initial cell count proportions in each simulation is 0.16 for both species, and the dominance is calculated after 200 years.

Chapter 5

Reflection and conclusion

This chapter includes some personal reflection on the project, including learning and planning, as well as concluding remarks.

5.1 Learning

I learnt a lot from this project. The opportunity to read and replicate world-class papers gave me the opportunity to deep-dive into the specific area. Topics I learnt about not taught on the degree include cellular automata, self-organisation, and constructing mathematical models to simulate.

I also learnt about translating mathematical models into code. The performance requirements required me to critically examine my code in a way I had not done before, as performance had never been an issue previously. Finally, I had the opportunity to learn how to use Gnuplot and its associated scripting language.

The pieces of coursework gave me a good insight into presentation skills, technical writing, and video production.

5.2 Planning, management, and goals

The original goals and plan for the project can be found in Appendix A, and this section reflects on them.

5.2.1 Goals

All four goals have been met. I originally interpreted visualisation as an animated visualisation, and save/load partly as being able to replay these animations. However, during development I realised this was going to be a lot of effort for not much reward. My interests and larger goals lay with the simulation itself, and so I decided to prioritise implementing the original and extended models, and analysing the results.

5.2.2 Planning and management

The project plan was was deliberately structured in a vague way, being much less specific than say a Gantt chart. As I was following Agile principles during the development of my project, I did not think a Gantt chart was a good way to plan the project. Trying to plan out work to the week months in advance is too inaccurate, and from previous experience I would be getting ahead or falling behind (either way deviating from the plan) within no time. The Semester 2 part of my plan was also very open-ended, which does not lend itself to a Gantt chart. Hence I chose to write the plan in the more informal manner you see in Appendix A.

In general, the plan was followed. As mentioned previously, the graphical interface aspect of the visualisation was dropped in favour of a simple command-line visualisation, and later the static images you see in the report. The extension I chose for the model was the competing vegetation species. Originally I wanted to base this off a natural phenomenon but could not find any papers that would fit the project well, so I chose something more fictional. This research and implementation was delayed (shortening my paper search) due to the probability distribution code not working properly as talked about earlier. This moved the extension implementation til after Christmas. For the further improvements in Semester 2 I decided to investigate the relationship between dominance and competition.

5.2.3 Reflection for the future

If I were to start the project from scratch, I would try to further modularise the code using a rule system for transitions. This would make creating novel models much quicker and easier, as components could be mixed and matched without having to write a lot of code.

5.3 Concluding remarks

Overall I feel I met the requirements of the project successfully. All the goals were met, and my extensions to Scanlon et al.'s model produced interesting, novel results. The project was completed on time and I am pleased with the results.

If I were to take the project further, I would continue to investigation competition and dominance, as well extending the model again. Other possibly extensions could include extra species, non-plant life (e.g. humans farming or animals grazing), and seasonal variation.

Ecosystem simulation is a vital tool for understanding the natural world and its complexities. I am pleased to have been able to contribute to this field, and excited about its future.

List of Figures

Bibliography

[1] Todd M. Scanlon et al. "Positive feedbacks promote power-law clustering of Kalahari vegetation". In: *Nature* 449 (Sept. 2007). See supplementary information document also, pp. 209–212.

[2] Ricard Solé. "Ecology: Scaling laws in the drier". In: *Nature* 449 (Sept. 2007), pp. 151–153.

[3] John von Neumann and A. W. Burks. *Theory of self-reproducing automata*. University of Illinois Press, 1966.

[4] Martin Gardner. "Mathematical Games - The fantastic combinations of John Conway's new solitaire game 'life'". In: *Scientific American* 223 (Oct. 1970), pp. 120–123.

[5] Eric W. Weisstein. *von Neumann Neighborhood*. `http://mathworld.wolfram.com/vonNeumannNeighborhood.html`. [Online. Accessed May 2017].

[6] David J. Anderson. *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press, Apr. 2010. ISBN: 0984521402.

[7] Robert C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, 2008.

[8] Beck. *Test Driven Development: By Example*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN: 0321146530.

[9] R White and G Engelen. "Cellular Automata and Fractal Urban Form: A Cellular Modelling Approach to the Evolution of Urban Land-Use Patterns". In: *Environment and Planning A* 25.8 (1993), pp. 1175–1199. DOI: `10.1068/a251175`.

[10] Sonia Kéfi et al. "Spatial vegetation patterns and imminent desertification in Mediterranean arid ecosystems". In: *Nature* 449 (Sept. 2007), pp. 213–217.

[11] R. H. Whittaker. "Dominance and Diversity in Land Plant Communities: Numerical relations of species express the importance of competition in community function and evolution". In: *Science* 147.3655 (Jan. 1965), pp. 250–260.

[12] Qinfeng Guo and Philip W. Rundel. "Measuring dominance and diversity in ecological communities: choosing the right variables". In: *Journal of Vegetation Science* 8.3 (1997), pp. 405–408. ISSN: 1654-1103. DOI: 10.2307/3237331. URL: http://dx.doi.org/10.2307/3237331.

Appendices

Appendix A

Project goals and plan

A.1 Goals

- Simulation
- Visualisation of simulation
- Graphing of simulation
- Simulation configuration (set up and save/load simulations)

A.2 Plan

- Start of Semester 1 to Reading Week (inclusive)
  - Investigate implementation technologies
  - Discuss and decide on a model to implement
  - Implement basic visualisation
  - Implement model
  - Implement manual set up of model
  - Prepare seminar
- Reading week to Christmas
  - Seminar
  - Find graphing library & implement basic graphing
  - Discuss and decide how to extend/change/implement other model
  - Extend/change/implement other model
- Start of Semester 2 to project hand in

- Improve one or more of: visualisation, graphing (classification of graphs potentially), simulation configuration, model(s), other
- Demonstration
- Report
- Screencast