University of Hull
School of Engineering and Computer Science

An Investigation into Alternative Methods for the Simulation and Analysis of Growth Models

by

Lee Richard Frederick Odiam BSc(Hons)

Submitted for the degree of
Doctor of Philosophy in Computer Science

May, 2019

In memory of my Grandmother,

June Christina Branson

Acknowledgements

There are a number of people who I wish to acknowledge, without their help this work would have never been possible an i will be for ever grateful for it and thier help:

- Professor Ken Hawick my supervisor for guiding me through the Ph.D. process and giving me the opportunity to pursue this goal.
- Dr James Walker and Dr David Chalupa who both acted as my technical expert at different points throughout the course of this work.
- My parents for their continued support in everything and constant encouragement.
- The University of Hull School of Engineering and Computer Science for the doctoral student scholarship and the opportunity to pursue this Ph.D. at all.
- All the student and postdoc members of the Computational Science Research Group who helped form a strong support system. Anthony Quinn, Craig Maddra, Liam Stockwell, Daniel Fleming, Russell Billingsley, Mariusz Kosmatka, George Lacey, Annika Schoene, Phininder Balaghan, and Andrew Johnson.
- The Viper High Performance Computing facility of the University of Hull and its support team. Without the use of this facility this work would not have been possible. as a total of $6,021,471$ CPU hours were used during the process of this thesis which is the equivalent of approximately 687.38 years.

Publications

Published

- 3D Printing for Visualisation of the Complex Physical Structures of Agent-Based Simulation Models on Lattices; K.A. Hawick, L.R.F. Odiam and L.A.D. Stockwell; Proceedings of the International Conference on Modelling, Simulation and Visualization Methods (MSV); 2016
- 3D Print Technology for Cellular Agent-Based Growth Models; K. A. Hawick, L. A. D. Stockwell and L. R. F. Odiam; Proc. IASTED Int. Conf on Modelling, Identification and Control, 15-16 February 2016, Innsbruck, Austria, Article 830-037.

Under Review

- Complexity and Scalability of Eden and Invasion Percolation with Various Data Structures; L.R.F. Odiam and K.A. Hawick; Journal of Experimental Algorithmics (Under Review)

In Preparation

- Memristor's for Growth Model Simulation; L.R.F. Odiam and K.A. Hawick and J.A. Walker;
- Component Labelling for the Analysis of Growth Models; L.R.F. Odiam and K.A. Hawick and J.A. Walker;

Acronyms

ABM	Agent Based Model
CA	Cellular Automata
MAS	Multi-agent System
IP	Invasion Percolation
IPR	Random Invasion Percolation
MIP	Meakin Invasion Percolation Model
HIP	Hawick Invasion Peroclation Model
HSH	Hash Table
OCT	Octree
AVL	Adelson-Velsky and Landis Tree
LAT	Lattice
RES	Resizing Lattice
RMRC	Random P_{max} Random P_{chance} MEden Model
RMFC	Random P_{max} Fixed P_{chance} MEden Model
FMRC	Fixed P_{max} Random P_{chance} MEden Model
FMFC	Fixed P_{max} Fixed P_{chance} MEden Model
SPICE	Simulation Program with Integrated Circuit Emphasis
CNN	Convolutional Neural Network
CCL	Connected Component Labeling
TIP	Invasion Percolation With Trapping
NTIP	Invasion Percolation Without Trapping

RP	Random Percolation
HPC	High Percomance Computer
SIANN	Space Invariant Artificial Neural Network
ReLU	Rectified Linear Unit
GPU	Graphics Processing Unit
KDE	Kernel Density Estimation
IQR	Interquartile Range
SRA	Simple Reflex Agent

Abstract

Complex systems are a rapidly increasing area of research covering numerous disciplines including economics and even cancer research, as such the optimisation of the simulations of these systems is important. This thesis will look specifically at two cellular automata based growth models the Eden growth model and the Invasion Percolation model. These models tend to be simulated storing the cluster within a simple array. This work demonstrates that for models which are highly sparse this method has drawbacks in both the memory consumed and the overall runtime of the system. It demonstrates that more modern data structures such as the HSH tree can offer considerable benefits to these models.

Next, instead of optimising the software simulation of the Eden growth model, we detail a memristive-based cellular automata architecture that is capable of simulating the Eden growth model called the MEden model. It is demonstrated that not only is this method faster, up to $12,704$ times faster than the software simulation, it also allows for the same system to be used for the simulation of both EdenB and EdenC clusters without the need to be reconfigured; this is achieved through the use of two different parameters present in the model P_{max} and P_{chance}. Giving the model a broader range of possible clusters which can aid with Monte-Carlo simulations of the model.

Finally, two methods were developed to be able to identify a difference between fractally identical clusters; connected component labelling and convolution neural networks are the methods used to achieve this. It is demonstrated that both of these methods allow for the identification of individual Eden clusters able to classify them as either an EdenA, EdenB, or EdenC cluster, a highly nontrivial matter with current methods. It is also able to tell when a cluster was not an Eden cluster even though it fell in the fractal range of an Eden cluster. These features mean that the verification of a new method for the simulation of the Eden model could now be automated.

Contents

List of Tables

List of Figures

[1] Invasion Percolation

[2] Resizing Lattice

[3] Lattice

There are neither beginnings nor endings to the turning of The Wheel of Time. But it was *a* beginning.

Robert Jordan,
The Eye of the World,
Book 1 of The Wheel of Time

CHAPTER 1

Introduction

OPTIMIZING both the runtime and memory consumption of complex systems simulations is a fast-moving and highly impactful area of computer science especially when the simulation itself is run in a Monte Carlo fashion. Running these type of models in this fashion requires that a large number of runs of the simulation to be performed and as such can take a very long time and massive amounts of memory to compute. In this thesis, the focus will be on the optimisation and new methods of analysis of growth models, a subset of cellular automata that are used in many physical sciences, including physics with the simulation of alloy cooling and dendrite formation. The models which will be focused on here will be that of the Eden growth model, and the Invasion Percolation model. Both of these models are used in the simulation of cancer, and as such their optimisation could aid in the expansion of our current knowledge of cancer by speeding up the rate at which it can be simulated. The Eden model is used for the simulation of the growth of a tumour whereas the Invasion Percolation model is used for the growth of the blood vessels that a tumour grows when it goes through angiogenesis (a process which causes the formation of new blood vessels). Though these are not the only uses of these models, and as such optimisations to them would also offer benefits where ever they are used. It is also hoped that the results of the experiments discussed in Chapter 5 could help inform where these different structures could offer benefits outside of these specific models.

This work will not only investigate methods in software through the application of more modern data structures such as an AVL tree as opposed to the commonly used character array for the storage of these growth model clusters, which will be referred to as the Lattice method. It will also investigate a hardware-based implementation of the Eden growth model that makes use of a memristive agent based cellular automata where a memristor is used for the storage of the state of the cell. It is a commonly known fact that hardware implementations are often considerably faster than their software counterparts, most of the time by thousands of times. Some reasons for this are that 100% of the computational power available is put into the calculation of the simulation also pathways for communication between parts of the system can be optimised for the required communication; this can also reduce the amount of energy that the system requires to perform a calculation or in this case simulation.

Whilst analysing the results of the hardware-based version of the Eden model, seen in Chapter 7 it very quickly became apparent that the use of the fractal dimension, a commonly used method for the validation of a new method for the simulation of the Eden model, was not able to classify individual clusters and instead could only be used to show that a new method produced the cluster within the same fractal range as the old method. This becomes an issue as just because a cluster has a similar fractal dimension does not mean that it is an Eden cluster, it also becomes an issue if the fractal range produced by the new method differs meaning it could be producing two types of Eden cluster or none at all. Therefore, this thesis also presents two new methods for the classification of Eden clusters; these classification methods can be performed with a single cluster and not a range of clusters. The two methods that have been developed make use of an element of Graph theory specifically Connected Component Labelling and a Convolution Neural Network to help classify the cluster and validate a new method.

1.1 Hypothesis

The main research questions that this thesis attempts to answer is whether the current method for the storage of Cellular automata based growth model within the simulation is the most effective or whether the use of more modern methods could offer speed or memory benefits. It also looks at if a memristive hardware-based implementation of the Eden growth model could not only be faster but could also offer additional benefits over the restrictive software implementations of the model.

An additional question arose during this work that is also investigated; this being whether the current methods for the analyses of these models are an effective method and if not what other methods could be developed to solve this issue.

These questions have been summarised into three main hypotheses; these are as follows:

H_1) **Modern data structures which are more sparse can offer both memory and runtime benefits to growth models.**

H_2) **A hardware implementation of the Eden model using memristors can simulate the Eden growth model faster than the software-based version, and the same architecture can simulate different versions of the model.**

H_3) **The use of connected component labelling and convolutional neural network improves that ability to analyse and classify individual clusters of the Eden growth model into the three main classes more accurately than the fractal dimension.**

1.2 Objectives

In order to prove the different hypotheses for this work a set of objectives were developed to help in their testing:

H_1 O_1) **Investigate the amount of the computational domain that the cluster fills for the Eden and Invasion Percolation models as the size of the cluster increases.**

O_2) **Develop an optimal method for the implementation of each of the data structures for the Eden and Invasion Percolation model.**

O_3) **Investigate the effect on the runtime when making use of different data structures.**

O_4) **Investigate the effect on the memory consumption when making use of different data structures.**

H_2 O_5) **Propose a design for a memristive based agent for use in a cellular automaton for the simulation of the Eden model.**

O_6) **Analysis of the fractal dimension of the Standard Eden growth model and compare to that produced by the memristive based Eden growth model.**

H_3 O_7) **Design connected component labelling for the analysis of the Eden growth model.**

O_8) **Test the validity of the connected component labelling for the analysis of the Eden growth model.**

O_9) **Design convolutional neural network for the analysis of the Eden growth model.**

O_{10}) **Test the validity of the convolutional neural network for the analysis**

of the Eden growth model.

1.3 Contributions

This section lists and details the specific novel contributions contained within the pages of this thesis, these novel contributions are as follows:

Investigation into the possible benefits of modern data structures in the simulation of the Eden and Invasion Percolation growth models this is discussed in Chapters 4 and 5. This is an investigation into the use of more modern data structure in growth models specifically the Eden growth model and the Invasion Percolation model and the effect that they have in the total run time and the memory consumption of the simulation. This work uses a range of different models to show that there is a small set of variables for certain models where the currently used method (an array that stores all the cell within the domain at once including cells that do not contain an infected cell) is not the most optimal method that can be used for the running of these model.

Design of a memristive based hardware implementation of the Eden growth model this is discussed Chapters 6 and 7. This is involves the design of both a possible architecture for a hardware implementation of the Eden growth model that makes use of a memristor the fourth fundamental circuit element to store the state of each cell/agent, in addition to the development of a simulation designed to simulate the architecture and allow for experiments into its validity to be performed. The simulation was needed as it was not possible to obtain an adequate number of memristors to fabricate this system.

Analysis of the fill density of Eden and Invasion Percolation growth models based on the change in L size this is discussed in Section 3.4. In order to be able to understand the effects of the size of the domain on the models and how this may affect the runtime of the models the percentage fill of the domain for each of the models that have been used in the investigation allowing for a more detailed understanding of how these model work.

Component Labelling for growth model analysis and classification this is discussed in Section 7.2. With the limitations of the current method used to test the validity of new methods for the simulation of growth models, namely the fractal dimension where in cases such as the EdenB and EdenC model where they are highly fractally similar it becomes near impossible to be able to distinguish between these two model with this method. This contribution involves the use of connected component labelling in a state assigned cluster for analysis of Eden clusters to remove this limitation and to allow for the classification of individual clusters even in a case where the cluster presented are fractally identical.

Image Classification for growth model classification this is discussed in Section 7.3. As with the previous contribution, this is a method for the classification of growth models when other methods, such as the fractal dimension, are unable to do so based off of a single cluster, because of the large overlap in the possible ranges produced by this method. This method makes used of convolutional neural networks to perform image classification on Eden clusters.

Previous to this work the standard method for the storage of these models was that of a dense lattice array in this work we demonstrate that for specific cases this is not the ideal method to use and the different more sparse data structures offer benefits. These benefits could aid the simulation of the model allowing for a version of the model to be run in the same amount of time and memory space on a computer. These results can be found in Chapter 5.

The thesis also demonstrates that significant benefits to the run time of the Eden model, specifically the EdenB and EdenC varients, are able to be gained through the use of a novel memristive architecture. An additional benefit can be obtained through the use of this system, this being the ability to tailor the type of cluster produced through the manipulation of two variable that controls the operation of the model. This is discussed in more detail within Chapter 6.

Previously to this work the method that was used for the analysis of a new method for the simulation of a growth model was to check that the fractal dimension of the two clusters was similar and that the clusters were visually similar too. Here we demonstrate two new novel methods that allow for a more detailed analysis of a different method for the simulation of a growth model. Additionally, this method allows for a better method for the classification of a single cluster, here this allows for the identification of a EdenA, EdenB, or EdenC cluster which would not have always been possible through the use of the fractal dimension analysis. These methods are discussed in Chapter 7.

1.4 Thesis Structure

This section will go into the structure of this thesis and will discuss what each of the different chapters will contain. Chapters 2 and 3 of this thesis after this introductory chapter will begin with a discussion on complexity and complex systems this will then be followed by an introduction to the models that are used in this work. This chapter will discuss in detail each of the models used, the algorithms that describe them, and the implementation of the growth sites list used for each which will be key in the selection of the next site to be infected. The final part of this chapter will discuss the effect of domain size on the cluster specifically how much of the domain is filled by the cluster taking into account the models specific stopping conditions.

Chapters 4 and 5 of this work will then go into the software-based experiments for the simulation of the models where alternative data structures have been tested for their suitability in their simulation. Chapter 4 will begin with an explanation of the different data structures; this will include a discussion on the frequent uses of the structures as well as a description of how they function. Chapter 5 will begin with the results for the effects on the runtime of the model that the data structures have; this will be split into two different sections the first of these will concern itself with the construction time of the structures, and the second part will discuss the total runtime of the models both across a range of domain sizes. All models will be run until they reach their model-specific stopping condition which is defined in Chapter 3.

Chapters 6 and 7 in this thesis will focus on the design and testing of the memristive based Eden growth model system which has been named the MEden method. Chapter 6 will begin with a discussion on the history of the memristor as well as some uses that it is currently being put to, this will then be followed by a discussion on the implementation of the software used to simulate the memristor and its responses to voltage including the equation that describes this behaviour. Next will be a discussion on the design of the agent, along with the algorithm that governs the model. Chapter 7 will begin by comparing the fractal dimension of the EdenA, EdenB and EdenC models against the MEden model in order to demonstrate that this method can produce fractally similar clusters. This will then go into the demonstration of the Connected Component Labelling method as well as the Convolutional Neural Network method showing that these methods can give a better understanding of the clusters that are being produced and can be used to classify individual clusters into one of the three main Eden varients. In the case of the connected component labelling, it will be shown that it can give a quantifiable difference between these models which is not possible with the fractal dimension method.

Finally, in Chapter 8 this thesis will end with the conclusion, this will summarise the contents of the thesis and identifying where within the work each of the objectives is demonstrated to have been completed, and as such showing that the different hypothesis has been proven or disproven. Following this will be a discussion on the planned future work that will be carried out. In addition to the main body of work, this thesis also contains three Appendices which contain the results of the statical analysis performed on the data that makes up the basis of Chapter 5, Appendix A will briefly explain the different statical methods that are used and explain what these methods show. Appendix B will look at the timing data, and Appendix C will look at the memory data.

Progress is Man's ability to
complicate simplicity

Thor Heyerdahl

CHAPTER 2

Complexity and Complex Systems

COMPLEX systems are an highly interesting area of research which present many problems across a vast range fields such as the growth of domains within cooling alloys (Hawick, 1991), Genetic Algorithms (Mitchell, 1998), the spread of an epidemic or violence within an urban area(Burke et al., 2006, Epstein, 2002), and even economic systems such as Bitcoin (Dos Santos, 2017) more example of topics can be seen in Figure 2.1. The most famous of these topics is probably Chaos theory, which studies complex systems that are highly sensitive to changes in the environments of the system or its initial starting conditions, this is often called the butterfly effect.

There are many misconceptions held about chaos theory the most common of these is that it is the study of disorder, this is not true. Chaos theory is the study of order from seemingly chaotic systems (Katherine, 1990). In a situation where a complex system exhibits chaotic behaviours, it would be more accurate to describe the system as a chaotic complex system, as chaos is not a necessity within the system for

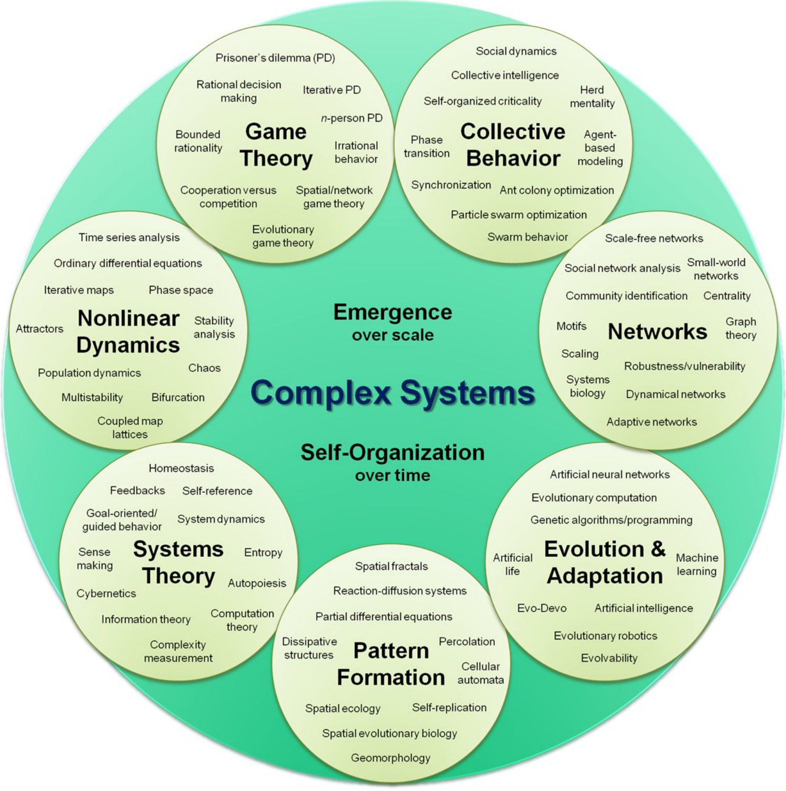


Figure 2.1: Diagram complex system broken down into seven different areas this is not an exhaustive list (Sayama, 2015) but it does give an idea at the range of disciplines that are involved in researching complex systems and complexity. The main areas in which this thesis will focus itself with would fall under the umbrella of Non-linear dynamics and Evolution and Adaptation.

it to be complex.

The study of these various systems falls under the umbrella of Complexity Theory which is a greatly interdisciplinary theory which originates from systems theory. Because of the vast number of disciplines that have an interest in and are affected by complex systems in one way or another the definition of these systems is a highly non-trivial matter, this chapter will attempt to give an overview of what features a complex system exhibits.

2.1 What is Complexity

One of the most common complex systems that we are all in constant contact with is that of the human body. It is made up of a collection of organs each, in turn, is made up of billions of cells which perform various functions. Each of these cells is its self-made up of numerous parts. From looking at the macro scale that of the body its self it is a highly non-trivial matter to determine what is happening on the cellular level, this is due to the nesting of complex systems involved, it took hundreds of thousands of years of experimentation and analysis to begin to understand what is happening within the human body. The brain alone is now considered in many circles to be a complex system(Telesford et al., 2011) that may operate on a quantum level.

As this thesis is in the area of computer science and due to the topics discussed referencing the computational complexity of various data structures. It is essential to understand that complexity in the sense that it is discussed in this chapter does not relate to the computational complexity of an algorithm which classes problems according to their inherent difficulty to compute. There is no one concise definition of what constitutes a complex system(Magee and de Weck, 2004); this is in part due to the vast range of types of complex systems and the diversity of disciplines that study them. Because of this, it is a highly non-trivial task to explain to a person what exactly is meant when the term complex is used; this has lead to a large body of definitions that attempt to solve this problem which seem to end up having more not in common than actually in common with each other. A number of these alternative definitions can be found in an article called What is a Complex System (Ladyman et al., 2012) some of which are listed in Figure 2.2.

This makes clear the difficulty of defining complexity as each of these definitions whereas they may be correct for the specific type of complexity that the author was referring to they either require more information to complete the definition, or they rely on other just as poorly defined terms to back them up the definitions never seem to stand on their own or are uselessly vague. The definition of complexity itself is as complex as the systems that the theory attempts to summarise and as such this work does not attempt to give a definitive definition of complexity. It will merely give a brief description of some aspects on which the majority of definitions agree.

"To us, complexity means that we have structure with variations." (Goldenfeld, 1999)

"Complexity in natural landform patterns is a manifestation of two key characteristics. Natural patterns form from processes that are non-linear, those that modify the properties of the environment in which they operate or that are strongly coupled; and natural patterns form in systems that are open, driven from equilibrium by the exchange of energy, momentum, material, or information across their boundaries." (Werner, 1999)

"Complexity starts when causality breaks down" (Editorial, 2009)

"In recent years the scientific community has coined the rubric 'complex system' to describe phenomena, structure, aggregates, organisms, or problems that share some common theme: (i) They are inherently complicated or intricate; (ii) they are rarely completely deterministic; (iii) mathematical models of the system are usually complex and involve non-linear, ill-posed, or chaotic behaviour; (iv) the systems are predisposed to unexpected outcomes (so-called emergent behaviour)." (Foote, 2007)

Figure 2.2: A selected of different definitions given to the term Complexity from a variety of different academic fields

For this thesis, a system may be considered to be complex if it consists of many interacting agents (an agent is a a discrete object within the simulation that takes in information and act based on a specific set of internal rules), whose properties cannot be wholly understood merely by examining the behaviour of each singular agent; this is due to the emergent properties that these systems often display. This description though can lead to a variety of false conclusions that end up not holding true the first of these is that just because a system contains many interacting parts; it is inherently complex this, however, is not the case. Just because a system is made up of many interacting parts on the microscopic scale that does not mean that it is impossible to sum these interactions up into a much more straightforward system through global quantities on the macroscopic level. It is not necessary to simulate every air particle in a room to understand the system it can easily be significantly simplified and therefore is not complex in nature(Charbonneau, 2017).

In this thesis the models that are to be dealt with could be considered to be Natural Complex model in the way that Charbonneau (Charbonneau, 2017) defines the term, this definition being idealisations of naturally occurring phenomena that whilst the general pattern on the macroscopic scale they are not controlled from this level but instead are the result of smaller individual agents interacting dynamically. A unifying theory of complexity could have vast benefits to our understanding of the universe aiding in the identification and implication of differing complex systems (Playne, 2011) by giving a more concise and specific definition of complexity, as opposed to the somewhat erratic and confusing theory that is currently used, where it could be argued that intuition is as important as part of identifying a complex system as any other.

Below is a detailed description of various aspects that complex systems may exhibit, this is by no means a definitive list more it is to give the reader an idea of some features that make these systems such interesting an unpredictable. These aspects are as follows:

Decentralized Collective is the most common feature in these systems and means that the system could also be considered a leaderless collective wherein the decision-making is distributed out to each member of the collective, making independent decisions based off of an internal rule set instead of following the orders of a leader (Ladyman et al., 2012). The most widely known and encounters example of a decentralised collective would be that of an ant colony, which is capable of incredibly complex global behaviour based upon nothing put simple local interactions. It is such a powerful example that the colony is often discussed as a single entity in and of itself instead of a collection of ants.

Non-Linear Relationships means that even a small change within the system could have a sizeable disproportional response in the system or it is possible that it will have no at all (Lea, 2015). This differs from a linear system where the effect is always proportional to the change. When plotted on a graph, a non-linear relationship is one which does not form a straight line in the positive or negative direction. This can make it very difficult if not impossible to predict the outcome without actually running the system.

Emergent Behaviour which is often merely called Emergence, is when many entities interact in an environment causing the formation of more complex and somewhat unpredictable behaviours, in other words, the whole is greater than the sum of its parts. The entities can vary widely from water freezing into a snowflake or people going about the business moving in a crowd. G. H. Lewes (Lewes, 1875) first defined emergence. The base level mechanisms that cause these emergent behaviours remains unknown.

Feedback can be a significant and useful aspect of a complex system. An entity receives feedback when the way its neighbours interact with it in the future depends on the past interaction with the agent (Ladyman et al., 2012)it can come in both negative and positive forms, which would have a dampening or amplifying effect respectively on the system. Examples of feedback are Craig Reynolds boid system (Reynolds, 1987) wherein the speed and direction of a boid depends on that of its neighbouring boids. The movement of that particular boid will, in turn, affect the other boids in future iterations.

Spontaneous Order is the unforced emergence of order from what is seemingly complete chaos; it is also sometimes referred to as self-organisation. It can be seen in numerous places from, the internet to even the very evolution of life itself (Darwin, 1909).

Hierarchical Organization is the idea that every entity in the system is subordinate to that of a single other entity except for one, this is the same way that modern businesses command structures are organised. Where the undertaking of the task is performed by and under the control of the individual, but the assignment of tasks is governed by a manager.

Nesting a complex system could also be nested this means that there are smaller micro complex systems contained within the larger macro system and that these smaller systems will have an effect on the more extensive system though not all the smaller system need to have the same level of influence as each other over the more extensive system.

An additional interesting aspect of complexity comes in when you look at the systems that are considered to be complex and the fact that they are only able to be considered to be complex when the system has been run for a long time (Bennett, 1985, 1988, 1995, 2003). Regarding an ABM[1] which will be defined later on in this chapter, this means many iterations or time steps have been performed. This again raises another question how long of a history is required. For this reason, it can be necessary to run these types of model for in some cases days and on as large a scale as possible as in some cases certain features may only make themselves apparent in the result of the system when it is simulated over a certain size, this adds an element of scalability to the behaviour that they can display.

[1] Agent Based Model

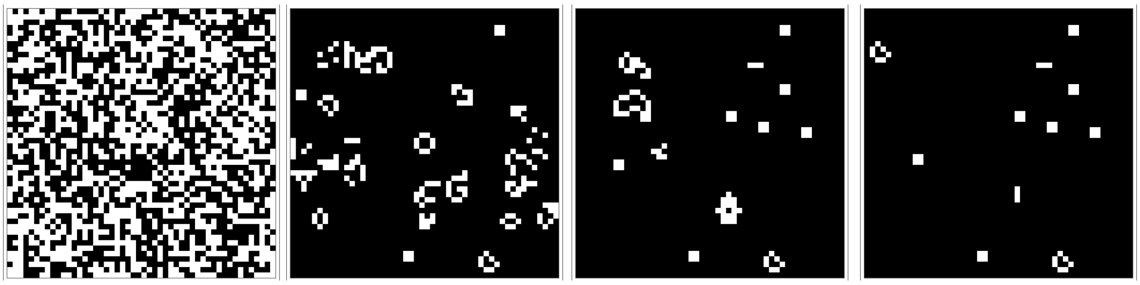


Figure 2.3: Game of Life example showing the model at intervals of 100 time steps from left to right. The first image on the left is the initial starting condition for the simulation, and in the final image on the right, the simulation has stabilised meaning that once reach the model will remain identical for a consecutive iteration.

2.2 Agent-based Modelling and Cellular Automata

ABMs and CA[2]s are potent forms of computational modelling (Bonabeau, Macal and North, 2006) and in the field of complexity science can be an invaluable tool in the simulation of these systems. They give the ability to not only visualise the system but allow for a more natural way to grasp the effect of changes to the systems. They also give a way in which to analyse a system, which in reality is either infeasible or impossible to measure. Take for instance in the case of an epidemic, in order to work out the best way to help restrict the spread of the illness, it would require the processing of massive amount of data regarding previous cases, which would require both the data and computational power to be able to process what would be thousands of terabytes of data. However, with ABM's, it is possible to just run a few thousand simulations of the epidemic each with different initial conditions and parameters that the model may have and to see the results of any intervention. It should be noted that the results from the simulation will only ever be as accurate as the simulation itself and so the validation of these model is an essential aspect of a model and due to the stochastic nature of these simulations will require many runs to get an idea of the average behaviour of the system.

Both CA's and ABM's can be traced back to the Von Neumann CA (Von Neumann, 1966) which was a theoretical idea of a robot that could build copies of its self. This

[2] Cellular Automata

idea was later developed by Stanislaw Ulam who suggested that the Von Neumann machine could be built on paper with the use of a collection of cells in a grid similar to that of a chess board, this later became the first CA. One of the next advances in the field was the zero-player game named Life or as it is more commonly known as Conway's Game of Life (Gardner, 1970) an example of this can be seen in Figure 2.3. Four straightforward rules govern the game of life. From these, a high level of complexity emerges on the macroscopic level; these rules are as follows:

1. Any live cell with fewer than two live neighbours dies as if caused by under population.

2. Any live cell with two or three live neighbours lives on to the next generation.

3. Any live cell with more than three live neighbours dies, as if by overpopulation.

4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

The Game of Life emerged from Conway's keen interest in Von Neumann's idea of a self-replicating machine, but he found that the system Von Neumann designed was very complicated with numerous possible states. Conway wanted to see if he could simplify the idea, and from that desire emerged the Game of Life. This was different from the Von Neumann Machine in that whereas the Von Neumann Machine consisted of a possible 29 states, the agents in the Game of Life could be in only be in one of two possible states, dead or alive, and had straightforward rules to govern the switching between these states. When this model was first devised Conway lacked access to a computer with the ability to visualise it. Because of this, the Game of Life was first simulated on a wéiqí board a game more commonly known as Go in the more western parts of the world.

One of the first simulations to be noted as an actual ABM was thought up by Schelling in his 1971 paper Dynamic models of segregation (Schelling, 1971). The model involved the simulation of the dynamics in racially mixed neighbourhoods.

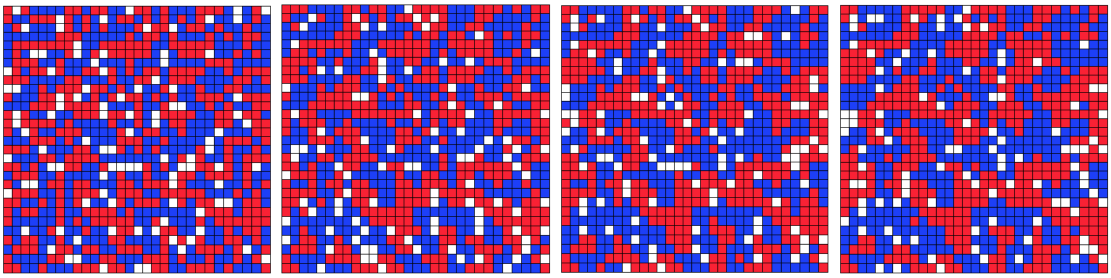


Figure 2.4: The Schelling Segregation Model on lattice with a 30% threshold for neighbour preference and 10% of the grid being empty to give the agents space to move in. The image show the first four iterations, an iteration is considered as each agent making the decision as the move or not. This already show the formation of spinodal like features in the simulation similar to those shown in the Cahn-Hilliard-Cook Model (Hawick, 1991, Playne, 2011), which is used to show domain growth with a quenched alloy.

The results of the simulation showed that even a low level of preference for neighbours could lead to a high level of segregation. The model is defined by a simple rule “If Neighbour count is higher than threshold move to random location otherwise stay still". The model can be simulated both in a lattice and in a Euclidean space. An example of the Schelling model in a grid-based environment can be seen in Figure 2.4.

Schelling’s model exemplified the idea of ABM be a collection of autonomous agents interacting with one another in an environment within which they can move. With the use of this model, Schelling showed that a small preference for your neighbours to be similar to you in some way, in the case of the model the same colour could if given enough time lead to total non-violent self-segregation. Later on, in 1980 a Prisoners Dilemma tournament was held by a political scientist named Robert Axelrod to find the most effective strategy for the game. The prisoner dilemma is an interesting piece of game theory where two people Mr A and Mr B get arrested, each of them is held in solitary confinement and offered a deal giving them the opportunity to either betray the other by testifying against him or to remaining silent(Mitchell, 1995, Wooldridge, 2002) and refuse to testify in any way. The results of what will happen in each of the possible outcomes can be seen in Table 2.1. The goal for each of the agents is to attempt to minimise its loss; this is where the game theory aspect of the model comes in to play, in attempting to figure out if it is worth the risk to stay silent.

In order to attempt to figure out the best way to minimise total prison sentence

Mr. B Mr. A	Silent	Betray
Silent	-1 -1	0 -3
Betray	-3 0	-2 -2

Table 2.1: Prisoners dilemma scoring showing the punishment received for each agent when each combination of Silent and Betray are made. This show how there is no chance of reward only to minimize loss.

over multiple occurrences of this dilemma, Schelling collected strategies from researchers all over the world and pitted them against one another in a 2D spatial grid to find the most effective strategy. One of the strategies that presented itself to be highly effective also happened to be one of the simplest the tit-for-tat strategy (Axelrod and Hamilton, 2008, Szilagyi, 2012) which Axelrod noted was an evolutionarily stable strategy (Smith and Price, 1973). This strategy means that the agent remembers what the other agent did last time they interacted and does that to them. In other words, if the last time the two agents interacted Mr.B betrayed Mr.A, Mr.A would now betray Mr.B. Like complex systems, ABM's are utilised across numerous disciplines like the social sciences (Axelrod, 1997, Epstein, 2002, Epstein and Axtell, 1996, Fukuyama, 1998).

After the work done with the Schelling model and the Game of Life, there was a Cambrian-like explosion of models. More people realised how they could be beneficial to them in their different areas of research. Due to the multi-disciplinary nature of ABMs, it has become critical that highly intuitive, rapid, and simple ways to develop these models where created, as not everyone can program to the level required to allow them the quickly produce these simulations. To this end frameworks which make use of a Domain Specific Language (DSL) such as NetLogo (Tisue and Wilensky, 2004a,b) and more recently Java Agent Development Framework (JADE) (Bellifemine and Poggi, 2000, Bellifemine et al., 1999) and Multi-Agent Simulator Of Neighbourhoods (MASON) (Luke, 2005, Luke et al., 2004) have been made which further lead to the rise in use and number of ABMs.

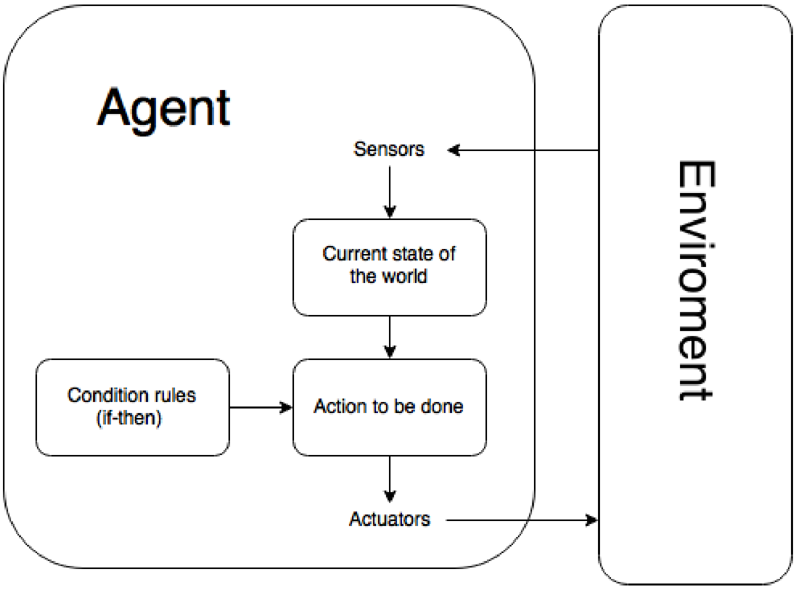


Figure 2.5: An Example of a simple reflex agent showing the sensors taking in information about the environment the agent the builds a view of the worlds applies its if-then rules and then decides on an action which causes the actuators to act on the environment.

The section has discussed many different ABMs but what exactly is an ABM. To define what precisely an ABM it is first necessary to define what an agent in and of itself is. This is by no means as convoluted a definition as with that of complexity, in its purest form an agent is an autonomous entity which observes its environment through sensors and acts upon that same environment with actuators (Franklin and Graesser, 1997). These agents are often called a SRA[3]. A diagram of the working of an SRA can be seen in Figure 2.5, they contain no knowledge or memory of any kind, the most common example of this type of agent is a thermostat, it's sensors reads the temperature of the environment, then the actuators turn on or off the heating. There are other forms of agent such as utility-based agents, Goal-Based Agents, and Learning Agents but all of these agents have the same basic set-up as the SRA with sensors and actuators in one form or another the difference between them is the internal workings of the agents, where or not they learn, have memory, interact with other systems just to name a few.

CA and ABM's are similar and are often confused though they do have differences that make them different from one another. Both of them involve the use of individ-

[3] Simple Reflex Agent

ual agents interacting with one another and or the environment, with the agent each having an internal state that is often represented as a finite state machine. The most significant difference between the two systems is in a CA the agent's location is fixed meaning that the agents' neighbours will never change throughout the simulation hence a simulation such as the Game of Life is a CA whereas as the Schelling model where the agents move is classed as an ABM.

ABM's are also very similar to MAS[4]. The main difference between them is their focus (Niazi and Hussain, 2011). MASs focus on the agent design with the agents being more intelligent but there being less of them, however, in an ABM the focus is put on the interaction between agents which are less intelligent, but there is a vast number of them. The difference between the two is very blurred with both having many aspects in common making it often difficult to distinguish between the two, though for this work these differences are not significant.

2.3 Neighbourhood Masks

With both ABMs and CAs actions having possible effects on their environment as well as being based on the current state of the environment there needs to be a mechanism for the agent to view its environment. In order to limit the amount of information that the agent is reacting too as would occur in a real system, this area is referred to as the agents neighbourhood. There are different forms that this neighbourhood can take, this depends on the type of mask that is applied to the agent. There are two main variants of domain grids, regular and asymmetric these can be seen in Figure 2.6.

On a regular Lattice, two masks are the most common one used when implementing both CAs and ABMs (Hawick, 2013, Hawick et al., 2016b, Kehoe, 2015, Niazi and Hussain, 2011). These are the Von Neumann and Moore Neighbourhoods, The Von Neumann Neighbourhood is often called the nearest neighbour (NN) method and

[4] Multi-agent System

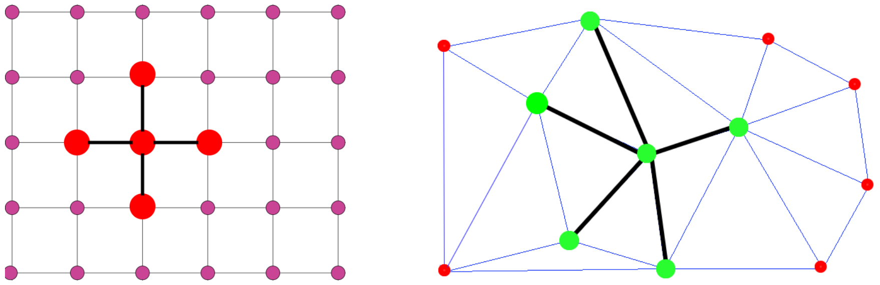
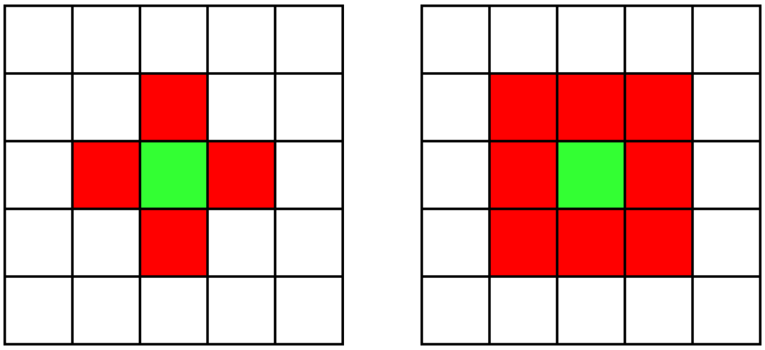


Figure 2.6: Regular (left) and asymmetric (right) mesh/lattice computational domain.



(a) Von Neumann

(b) Moore

Figure 2.7: Neighbourhood examples in 2D for the Von Neumann and the Moore Neighbourhoods with a view radius of one. Some models make use these neighbourhoods with an increase view distance, such as the Civil Violence model (Epstein, 2002), but these are the most common versions of these two neighbourhoods.

can be seen in Figure 2.7a this is the neighbourhood style used in the Game of life which was discussed earlier on with this chapter and is also the neighbourhood that is implemented in the models used for this work. The cell only looks at the cells directly above, below, to the left, and to the right of the cell in question in 3D, this adds the cell in front and behind the cell making for a total of 6 neighbouring cells. Finally, the Moore neighbourhood adds in the diagonal cells forming a square around the cell, and this can be seen in Figure 2.7b in question in 3D this square becomes a cube surrounding the cell.

With asymmetric computational domains such as the one seen in Figure 2.6 it can become challenging to predict how the mesh will look compared as so a simple mask is not applicable. In the regular mesh, it is straightforward to predict the neighbour count as shown in Figure 2.6 it will always be 4, if we consider the mesh to wrap, if not only the edges and corners with have less, with three and two respectively, whereas in an asymmetric mesh the number of neighbours can vary greatly. For example in the cluster highlighted in Figures 2.6 the total neighbour count is 5 and the position of each neighbour is much less predictable. This would need to be taken into account when designing the structure used with this type of mesh. The use of a graph system would be the most applicable in the case of an asymmetric domain allowing for straightforward neighbour communication. The models within this thesis make use exclusively of the regular grid with a neighbourhood style of Von Neumann/Nearest Neighbour.

2.4 Initial Conditions

With both ABMs and CA the initial conditions of the model can drastically affect the end state of the model and as such could be considered to be just as important if not more than each of the time steps. For example, in the case of Conway's Game of Life, some initial configurations will lead to an empty environment. Some will lead to a completely stable unchanging environment, but it will still contain some cells that exist within the alive state. Finally, there are even initial conditions that can lead to an eternal Game of life that will never stop iterating with each time step being different in some way from the previous one.

These initial conditions can consist of a number of things that include the positions and state of the agents within domain, the size of the domain, as well as an variable use to tailor the behaviour of the agents within the model, and even the seeding value for the random number generator that is used for stochastic selections within the model. The storage of these initial condition enables a model to be re-run without the need to store each of the specific steps. Also, but excluding the seeding value from the stored initial is allows for the level of variance in the model to be calculated in the case of stochastic growth models such as those discussed within this work. These conditions can be either random such as with the Schelling Segregation model, or

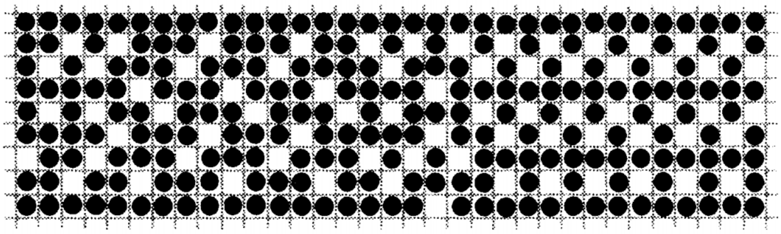


Figure 2.8: The first Garden of Eden for Conway's Game of Life. Which was found in 1971 by Roger Banks (Gardner, 1983).

they can be more predetermined as with the models that are discussed in Chapter 3.

An interesting aspect about initial conditions comes in with the study of conditions that are unable to be produced from any previous time step given the rules of the model. This means that this specific set up can only exist as an initial condition (Ceccherini-Silberstein and Coornaert, 2017, Gardner, 1983), this is called a Garden of Eden. A Garden of Eden consists of the whole of the domain that the model is running within and is not just a small subset of the domain, however for every garden of Eden within that specific garden can be found a finite number of what are called orphans, these also have no previous state within the model. An example of a Garden of Eden for the Game of Life is shown in Figure 2.8.

Gardens of Eden are not only possible in 2D CAs they are also possible within higher dimensions, but the discovery of them becomes increasingly difficult with each increasing dimension. The Garden of Eden theorem of Moore and Myhill states that any CA based on a square grid of N dimension has a Garden of Eden if and only if it has twins. Which are two finite patterns that have the same successors whenever one is substituted for the other one. These configurations not only exist in a grid-based system but the can also exist of different network styles (Machi and Mignosl, 1992). As mentioned earlier the cells within CA do not move this differs from ABMs, for this reason, it is much less likely that a garden can exist within an ABM.

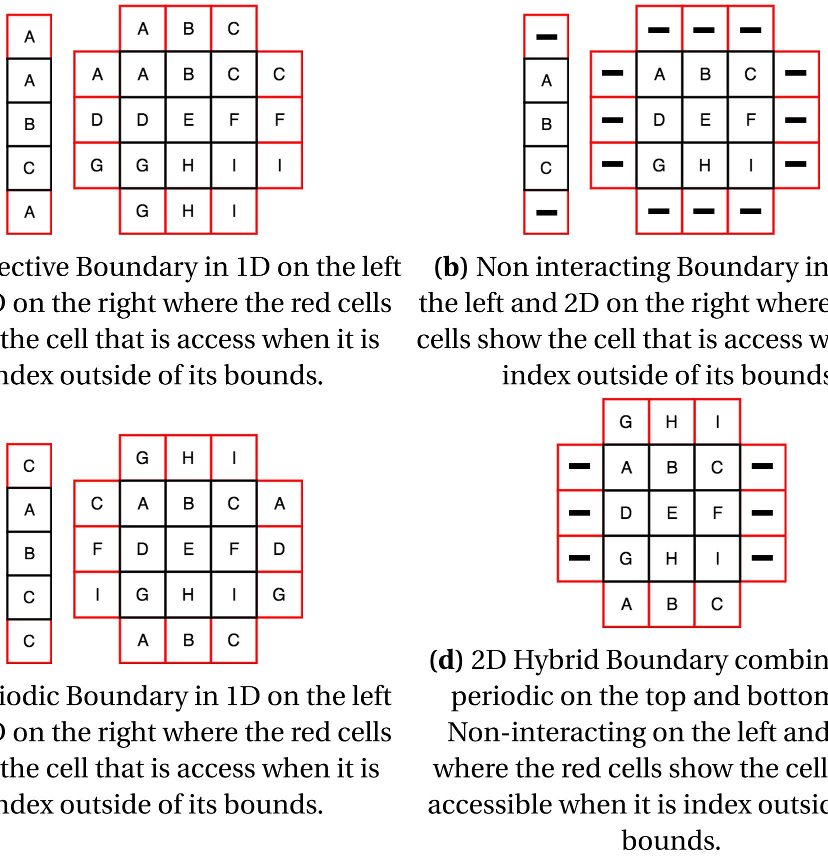
The search for these patterns is a non-trivial matter especially for CAs with more than one dimension such as the Game of Life and Schelling's segregation model for one dimensional CA it is possible to find the garden of Eden for the model with a polynomial algorithm but for model with 3 or more dimensions there is no guaranteed algorithm that will terminate having found a Garden of Eden, but even so there have been successful searches.

2.5 Boundary Conditions

When simulating these models the domain that the model exists in must at some point have an end because even with methods that will be discussed here on the growth of the domain the will reach a point where the computer that the simulation is being run on is unable to store any more data. So in addition to the rules of the model and the initial conditions, there must be rules that govern the boundaries of the model. With models that can be classified under the umbrella of site exchange models wherein the agents in the system can exchange states with one another.

The most commonly used boundary condition is that of a periodic boundary this allows for the approximation of an infinite domain, this is achieved by wrapping the boundary in all directions as seen in Figure 2.9c in one dimension this would mean the array $[A, B, C, D]$ would hit A again when it attempts to step past D this is very easy to do with the application of the modulo function. By taking the modulo of the index against the size of the dimension, this will wrap the index around the domain. In a domain with size 128 if attempting to index cell 129 the modulo function will wrap this around to 1 $(129 \% 128 \ = \ 1)$. One of the issues that can arise in the use of the type of structure is with the self-interaction of the models this can lead to the destruction of the patterns that can be formed within the model meaning that it no longer represents accurately the system that attempted to be simulated. The easiest way to deal with this is to increase the size of the domain to reduce the effect of this self-interaction.

The next type of boundary condition is a non-periodic boundary condition; this is when there is no wrapping making a limited domain. There are many ways to handle



(a) Reflective Boundary in 1D on the left and 2D on the right where the red cells show the cell that is access when it is index outside of its bounds.

(b) Non interacting Boundary in 1D on the left and 2D on the right where the red cells show the cell that is access when it is index outside of its bounds.

(c) Periodic Boundary in 1D on the left and 2D on the right where the red cells show the cell that is access when it is index outside of its bounds.

(d) 2D Hybrid Boundary combining the periodic on the top and bottom and Non-interacting on the left and right where the red cells show the cell that is accessible when it is index outside of its bounds.

Figure 2.9: These show an example of how the boundary differs for different conditions in both 1D and 2D when a Von-Neumann neighbourhood is the one that is being used.

this two of which will be discussed here these being a reflective and non-interaction. The first of these, the reflective boundary is used when the system being simulated also has a boundary as this can imitate bouncing off of this boundary such as with water in a jar, an example of this can be seen in Figure 2.9a. This can also be implemented reasonably easily by merely checking that the index is higher than the boundary of the domain and take the difference away from the index.

The final boundary condition is that of a non-interacting boundary method, in this case, it is as if the model exists within a void and if it attempts to step outside of this void nothing happens. It also is called an absorbing boundary as if an entity crosses this boundary the boundary absorbs it and as such no longer exists for interaction within the domain. This condition can also be used in conjunction with the stopping condition so that if the model meets the edge of the domain, the simulation stops.

Each of these different boundary conditions that have been discussed in this section can be used together to form a hybrid boundary, an example of this can be seen in Figure 2.9d. Where the top and bottom of the domain using a periodic-boundary and the left and right using an absorbing-boundary.

2.6 Summary

This Chapter has discussed complex systems which are a rapidly growing multidisciplinary area of research; it also identifies aspects that a complex system would have, though because of the vast area it doesn't attempt to give one definitive definition but describes how it is used here. This was done because of the variety of definitions that exist within the different domains that study these systems. It also contains a detailed history of ABM as well as CA both of which are tools that are used in the simulation of complex systems. They are tools that have many similarities between them, though are distinct. The history included a few examples of some key CAs and ABMs such as the Game of Life and Schelling Segregation model.

This is followed by a discussion on the way in which agents within these simulations interact with one another this includes the neighbourhood masks, or topology of the agents be this a four neighbour Von Neumann or an eight neighbour Moore neighbourhood. Finally, the last part of this chapter discussed the boundary conditions for a model and the importance and effect of these conditions which define what type of interaction occurs at the edge of the domain if any.

Complex systems continue to inspire research interest all over the world with more and more researchers coming to see the benefit of studying these systems, as well and making use of the tools used to simulate them such as CA and ABMs. With this increase in interest, it becomes increasingly important to ensure that the models being run are doing so in the most time and memory effective methods possible as this can lead to a significant increase in the speed at which this type of research that can be performed. It may not be possible to optimise for both memory and timing, and as such, it is possible that a smaller memory footprint might have to be traded for a faster runtime or vice versa, this all depends on the limitations that the person running the model might face.

Growth models are a form of CA that are being used in a variety of different fields the main one of concern for this work being cancer research with the application of models such as the Eden Growth model and the Invasion Percolation model for things such as tumour growth and when a tumour goes through angiogenesis and becomes vascularized (this is when a tumour forms new blood vessels from already existing ones, this is a critical step that transitions a tumour from being benign to being malignant). The following chapter will discuss these models in detail, focusing on some specific applications of these models as well as the methods that can be used for there simulations including the handling of background data that is required.

Those who cannot learn from
history are doomed to repeat it.

George Santayana

CHAPTER 3

Growth Models

IN various scientific disciplines the study of growth in non-equilibrium systems is a highly interesting and active research area with many impact areas. One of the most useful tools for this type of research is that of growth models. Hayes (Hayes, 1979) in an article on different growth models starts off by quoting Douglas Adams Hitch-hikers Guide to the Galaxy "We have it on good authority that the earth is a gigantic computer, built to calculate the answer to some ultimate question (or vice versa)". He then goes on to state that as you look around the world, the idea seems a more and more plausible.

In the purest sense, there are two types of growth models the first is a model with a purely stochastic mechanism behind them which will be the class of model that are of concern within this thesis. The second class of growth model is ones with a more energetic base such as temperature (Herrmann, 1986). This includes models such as the Potts model which is a being a generalisation of the Ising model and used in the study of ferromagnetism as well as in computational biology to model the collective behaviour of cellular structures (Szabó and Merks, 2013).

Growth models can be a somewhat broad term that may require some more defining in order for it to be fully understood. Many fields make use of the term growth model, the largest area in which it is used is in economics. In this field, they are used to try to simulate and predict changes in the economy. In this thesis, the term growth model will not refer to this type of model but instead will refer to models that are used for the simulation of biological systems such as tumours, the roots of a plant, or other similar systems. Many of these models have found numerous application across a wide range of field separate from their intended purpose.

There where a large number of different models that could have been selected for use in these test the reason for the selection of these two in particular other then their use in the fields of cancer research. Comes down to the similarity between then for all intents and purposes these models can be handled nearly identically with the difference being how to handle to growth sites list in the adding to a removing from. This means that the effect of change the storage of the cluster can be handled in the same way between all of the different models with only a couple of differences such as the use of an additional state as in the case of the EdenA model (this is discussed in more detail later on within this chapter).

This chapter will start off with a discussion of the history of each model, along with how they are currently being used in different fields. After which there will be a description of the rules of each of the models along with the algorithm describing the running of the model. The Eden and IP models ,both of which are asyncronous in there infection steps, were selected because of their similarities; this meant that the same types of data structure could be made to work with both of them with only a little change to allow them to run as efficiently as possible. This will allow for the identification of whether a particular data structure is better suited for a variety of growth models or if it is just for one specific model that it stands out. Growth models can have exciting features such as their asymmetries which can change as the model

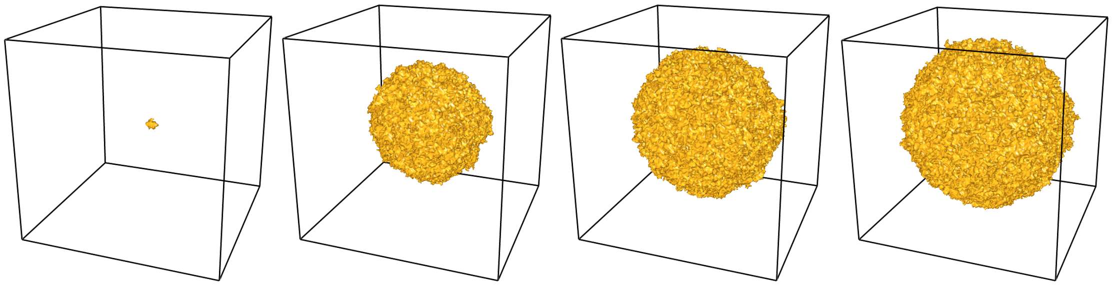


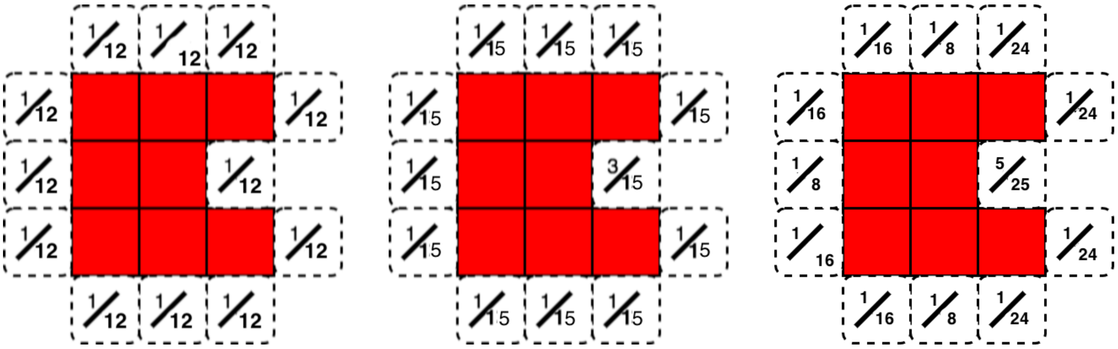
Figure 3.1: Eden growth model in 3D on a lattice with an L size of 128 at time steps 100, $200,000$, $400,000$, and $600,000$.

grows (Hawick, 2016).

3.1 Eden and Screened Eden Growth Models

The Eden growth model was first conceived of in 1961 (Eden, 1961); its intended purpose was to simulate the growth of organic substances specifically bacteria within a culture medium with adequate food for the cluster to grow. It differs from other growth models such as Diffusion Limited Aggregation (Witten and L.M., 1981) and the Ballistic Particle Deposition (Hayes, 1979). In that, whereas these models fire particles into the domain and thus would be considered closer to an ABM with each particle being a single agent. In the Eden growth model, however, the cells within the domain are themselves agents and exists within one of two states without moving through the system making Eden more of a CA than an ABM, and instead of adding agents the to the domain the already existing agents instead switch their states.

Another feature of the model is the fact that the surface of the cluster exhibits fractal-like properties (Jullien and Botet, 1985) making it interesting to various fields within the physical sciences. One application of the model currently is in the simulation of large-scale cancer like clusters (Hawick and Scogings, 2009). An interesting piece of work done on this subject was published in Nature in 2015 (Waclaw et al., 2015) wherein a large scale Eden Growth Model was developed. This model included the way in which tumours can disperse within a short range, as well as including the



(a) EdenA (b) EdenB (c) EdenC

Figure 3.2: Example of the probability distribution in the EdenA, EdenB, and EdenC show the different probability distribution for each of the variant with the same cluster. These different version of the Eden model where selected so give an understanding of how significant the handling of the growth site is.

chance for the cell to mutate in different ways. Allowing for the researches to simulate the death of tumours with the chance that some of the cells would be immune to the treatment; this allowed for the simulation of how tumours can become resistant to treatments such as chemotherapy and radiotherapy as well as how the cluster may grow after such treatments. The use of these simulations allowed them to conclude that the restriction of cellular migrations in tumours could lead to improvements in the battle against cancer significantly reducing the growth rate of tumours.

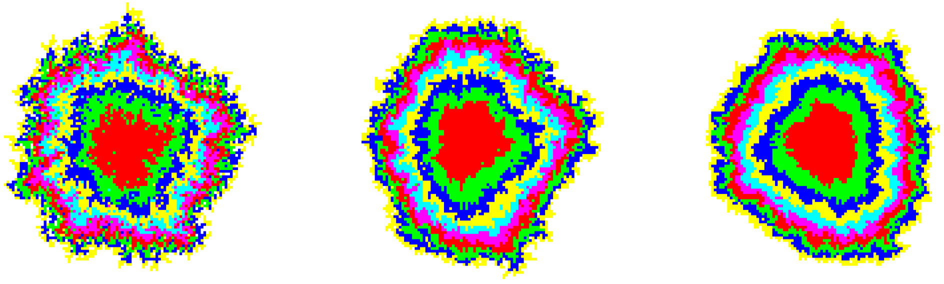
Another adaptation of the model (Wang and Bassingthwaighte, 1997) as the development of an off-lattice version of the model. This simulated the growth of a Euclidean-based (not limited to a grid instead it has free movement within a floating-point based euclidean space) Eden model on the surface of a sphere. This would make it possible to simulate the way that tumours might grow on a curved surface such as the ball of a joint or a person's' skull. It was also an attempt to deal with the inherent non-isotropic nature of the model. Due to that fact that the model tends to live on a regular mesh even with its stochastic nature, it tends to grow along the axis of the grid (Thompson, 1942) biassing the possible antisymmetry of the model.

The Eden model is most commonly represented on an M-site 3D matrix or Lattice where $M = L_x \times L_y \times L_z$ and L represent the size of a specific dimension; with the dimension being shown by the subscript by either the letter x, y, or z. The cells can be in one of two possible states infected or empty within a more advanced version of the simulation the infected sites could have the chance of having multiple possible values depending on the type of infection present in the cell (Waclaw et al., 2015). There are three main types of Eden models each with different ways of selecting which growth site will be the next one to be infected by assigning a different probability to the cells based on the method. This work will make use of three different standard Eden model rule sets, the Eden A, B, and C rule sets.

The EdenA is the simplest of all three of the variants. In this version, each of the neighbouring cells to the cluster has an equal chance of being selected for infection. EdenB has it such that each of the edges of the cluster has the same chance of spawning a cell, meaning that if one uninfected cell has three infected neighbours, it will have three times the chance to become infected as compared to a cell with only one infected neighbour. The third and final variant of the Eden growth model that is used within the set of Standard rules is the EdenC model, this version of the model can be the most difficult to implement. In this version, one of the edge cells is randomly selected, and one of its uninfected cells is then in turn randomly selected and set to be infected. Examples of the probability distribution of each of these variants on identical clusters can be seen in Figure 3.2 and a 2D visualisation of the style of clusters these rules produce can be seen in Figure3.3.

For all models that have been written for this work whenever a random number is discussed this is obtained through the built-in rand method in the standard template library of C++, this may not the best of the pseudo-random number generators that are available, but it can generate a highly uniform distribution and is very commonly used, hence its use here.

The main three variants of the Eden growth model the EdenA, EdenB, and EdenC each produce quantifiably different clusters which can be seen in Figure 3.3 where one of each of the cluster is shown in a 2D form. Each of these clusters where generated from the same initial infection seed (a single infected cell in the centre of the



(a) EdenA **(b)** EdenB **(c)** EdenC

Figure 3.3: Examples clusters grown with the different rules sets for the Eden growth model in the EdenA model, EdenB, and EdenC. All the clusters were grown with the same initial conditions including the seeded value to the random number generator. The only difference being the handling of the selection for the next site to be infected. The colours in this model represent the time step at which each cell was infected with the colours starting with red this shows the colours with nine different colours changes, with the colour change ever 500 steps.

show grid) with identical seeding values fro the random number generator used, Each of the cluster where run to 2000 iterations and coloured to show the effect growth rate with the cluster changing colour about every 335 iteration's.

The EdenA rule set produces a cluster with the lowest density of the three the change of the colour in this image shows the flow of time within the model with the colours changing from Red to Green to Blue to Yellow to Cyan to Magenta and then back around to Green as the simulation runs. This shows how some cells can remain uninfected at the centre of the cluster for a considerably long time before eventually becoming infected this is what leads to the lower density, with the clusters shown the EdenA cluster having 58 uninfected internal cells compared to the EdenB's 10 and the EdenC's 2. As the cluster moves through the rules from A to C the clusters that are grown become increasingly dense with EdenC only having very minimal mixing of the colour bands in the model showing that it grows much more uniformly than the other two clusters especially the EdenA.

The Eden Growth Model follows a straightforward set of rule first an initial cell within the computational domain is set as infected, this is placed in the centre of the domain to give the maximum possible space for the cluster to grow within. The list of possible growth sites is then built from uninfected cells with the neighbourhood

Algorithm 1 Eden Growth Model

```
plant seed cell in the centre of an $M$ matrix
check seed add its empty neighbour sites to growth sites list $G$
while current time-step < max time-step do
    generate random integer $R$ $[0,$ size of $G)$
    set $R$ cell in $G$ to infected
    delete $R$ cell from $G$
    check neighbours of last infected cell and add empty cells to $G$
end while
```

range of infected ones. One of these sites is then selected and set as infected. The process then repeats until some pre-set stopping condition is met. A pseudo-code representation of the algorithm used for this work can be found in Algorithm 1.

An interesting feature of the Eden growth model is the surface of the model the centre the model tends to be a solid especially in the case of the EdenC whereas as stated earlier the surface of the models exhibits fractal-like properties(Freche et al., 1985, Meakin et al., 1985, Wang and Bassingthwaighte, 1997); this means that as with a dendritic or tree-like structure, such as the Invasion Percolation Model which is discussed in the following section of this chapter, it will exhibit self similarity(Sauer and Schroer, 1987). This can be measured through the use of the calculating its fractal dimension; this will be discussed in more detail ion Chapter 6.

One of the issues that can arise with the use of the square lattice-based grid for the computational domain, this can cause the model to exhibit anisotropy meaning that the model tries to growth along the axis of the domain (Freche et al., 1985, Meakin, 1988); this leads to the model itself being skewed in shape. Through the use of a noise reduction algorithm it is possible to make this more evident (Meakin, 1988) this can be seen in Figure 3.4. Even though this anisotropy affects the shape of the cluster that is grown this does not mean that it is pointless to use and is still a widely used domain for this style of model, so long as this issue is considered when making use of the model its effect can be compensated for.

Aside from the three already discussed versions of the Eden growth model, there is one additional version that will be included as it shows highly interesting behaviour

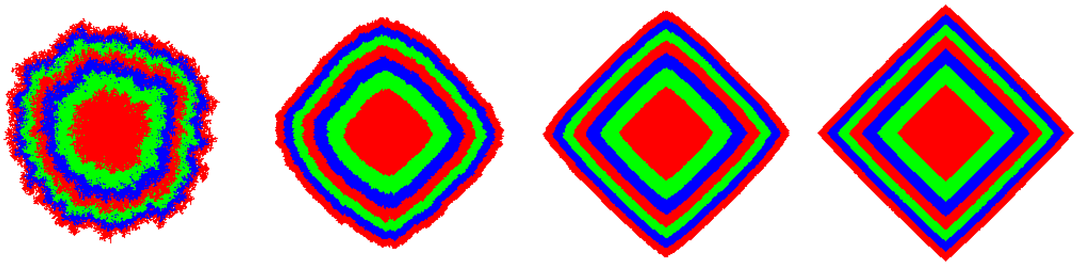
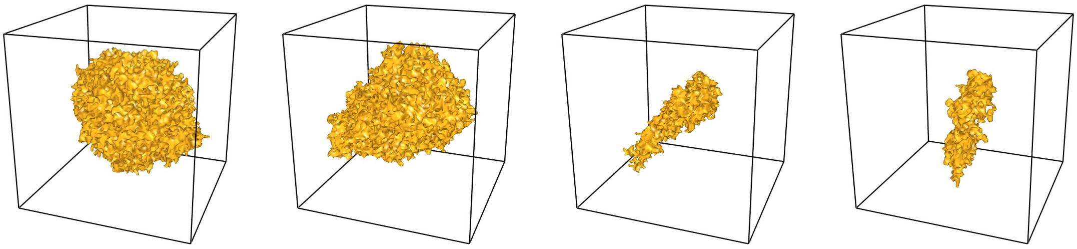


Figure 3.4: Eden growth model run with the use of a noise reduction algorithm to emphasize the anisotropy of the model. The picture shown above are obtained through the use of a method discussed in (Meakin, 1988). this method makes use of a smoothing variable (m) the picture above make use of different smoothing values from left to right of 0, 10, 100, 1000. All the images where obtained by running the model to the same number of infected cells and on the same size grid with the same initial seed value for each of the runs. The colours in the images of the cluster represent different time steps the cells where infected on.

depending on the screening factor that is applied to it, it is this screening factor that gives the model its name the Screened Eden growth model. The Eden growth model as shown in Figure 3.1 tends to generate very similar clusters with relatively low variance within the model. There have been many different methods for the screening of the model in order to allow for a higher level of variance to the model have been thought up over the years.

One of these methods involved the number of free paths to the edge of a domain (Jiang and Gang, 1989). In this method, the growth sites are considered to be all sites that are the nearest neighbour to an infected cell and can be connected to the edge of the domain with a straight line without obstruction from any infected cells. Cells are then grouped by the number of these paths that they contain and then are selected based off of the probability value for each cell. The method used here different in that the growth probability is based on the distance a cell it from the centre of mass of the cluster (Hawick, 2016, Hawick et al., 2016b, Xie et al., 2011); this is done with Equations 3.1 and 3.2. This is an interesting version of the model as the clusters that it is capable of generating can exhibit a wide range of different behaviours based upon the specific beta value that is applied to it. The effect of different beta values can be seen in Figure 3.5.

As this variant is so different from that of the Standard version of the Eden Model



(a) $\beta = 0.1$ **(b)** $\beta = 0.3$ **(c)** $\beta = 0.7$ **(d)** $\beta = 0.9$

Figure 3.5: Screened Eden Growth Model grown on a $(64 \times 64 \times 64)$ grid with different β values 0.1, 0.3, 0.7, and 0.9 from left to right respectively. Shown at the point when the cluster first touches the edge of the domain. β represents the screening value for the model which decides how much of an influence the radius of gyration has on the growth of the model

$$p = (r_{qmax} + 1 - r_q)^{-\beta} \tag{3.1}$$

where:

p is the growth probability for the cell
r_{qmax} is the distance of the further cell from the centre of mass
r_q is the distance of the current cell from the centre of mass
β is the screening value for the model in the range $[0, 1)$

$$r_{cm} = \frac{1}{N}\sum_{i=1}^{N} r_i \tag{3.2}$$

where:

r_{cm} represents the centre of mass
N is the number of cells within the cluster
r_i is the 3D coordinate for the $i^t h$ cell

Algorithm 2 Screened Eden Growth Model

```
plant seed cell in the centre of an $M$ matrix
check seed add its empty neighbour sites to growth sites list $G$
while stopping condition not met do
    calculate distances rq of Q sites to the centre of mass
    randomly select cell i from Q with rq chance
    set site i as infected
    update growth sites list
    recalculate the centre of mass
    check neighbours of last infected cell and add empty cells to $G$
end while
```

```
void update_cm(const int index) {
vec3 pos = fromK(index);
for (int i = 0; i < N_dims; ++i) {
r_cm[i] *= N;
r_cm[i] += pos.v[i];
r_cm[i] /= N + 1;
}
N++;
}
```

Figure 3.6: This shows the code used for the calculation of the centre of mass of the infected cluster this is only used for the Screened Eden growth model. The D variable seen in the code is equal to the number of dimensions, 3 for a 3D world 2 for 2D worlds. This means that the same code can be used for an N-dimensional cluster. This code was specifically extracted from the Lattice version of the Screened Eden model though it same algorithm is used for all the different version of the model.

the algorithm used for its generation is more complicated, this algorithm can be seen in Algorithm 2. The initial steps are mostly the same here as for the other 3 Eden methods where this version is different is in the addition of the calculation of the probabilities for each of the cells within the growth sites list and the section of the next cell based off of these probabilities because of this the Screened Eden model is much slower to calculate.

The most computationally expensive part of the Screened Eden model is the calculation of the centre of mass of the clusters. The naive method would be to iterate through all the items within the data structures each iteration and add together the 3D coordinates of the model and then divide each of the three values by the number of infected cells that there are within the domain. The issue with this method is that it

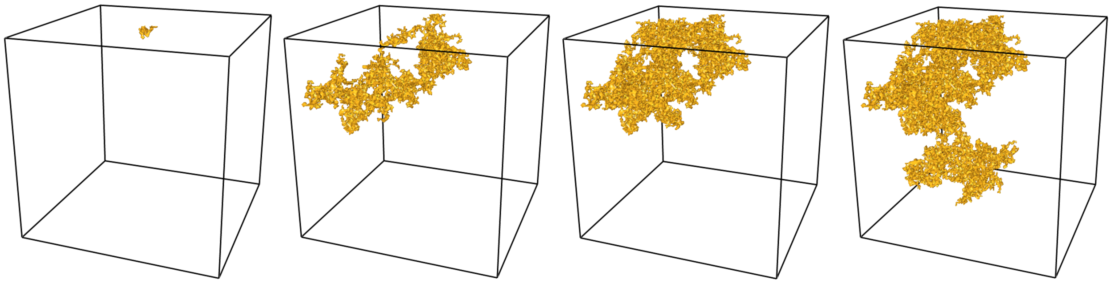


Figure 3.7: Standard IP model with the percolation threshold randomly assigned in 3D on a lattice with an L size of 128 at time-steps 100, $10,000$, $20,000$, and $30,000$.

is very slow, especially with larger domains. Instead, the method that will be used here will be to keep track of the centre of mass and then when a new cell has been infected the code in Figure 3.6. This works by first converting the k-index of the cell into its 3D coordinate, this is discussed in detail in Chapter 4 after this each of the 3 coordinates of the centre of mass the x, y, and z are multiplied by the number of infected cells from the previous iteration the respective coordinate from the newly infected cell is then added to it and then the centre of masses value will be divided by the actual number of infected cells. The final step of this method is to increment the number of infected cells ready for the next iteration.

3.2 Invasion Percolation

IP model (Wilkinson and Willemsen, 1983) which can be seen in Figure 3.7 was developed to simulate the flow of immiscible fluid through a porous medium such as dirt, stone, or sponge among other such media (Hawick, 2014, 2011), it is capable of simulating multiple different fluids at once. It was an advancement on a previous model, RP[1] which has many connections to the Ising Model (J., 2016) as well as other Potts models (Potts, 1952). The original method for this simulation was for that of static mediums whereas IP explicitly took into account the transport process taking place during the process of the event attempting to be simulated.

There are many applications that the IP model has been used for. One exam-

[1] Random Percolation

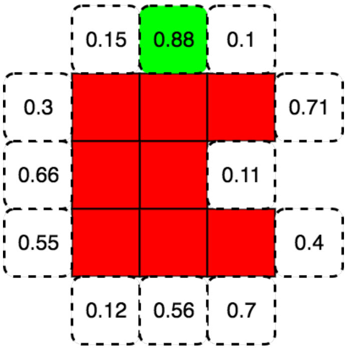


Figure 3.8: Invasion percolation probability distribution with the invaded cells represented in red and the probabilities for the neighbouring sites to become invaded shown. The green cell indicates the next to be invaded.

ple application for the IP model is for the simulation of fracking (Norris et al., 2014), which is the process of injecting liquid at high pressure into different mediums, for example, subterranean rocks and boreholes. It is used for the extraction of oil and to open fissures. Another interesting use of the IP model is that of the modelling of coastal erosions (Hawick, 2014); this was done by combining the IP model with that of the Kawasaki model for site exchange (Hawick et al., 2016a, Kawasaki, 1966). This effect can have a significant impact on different environmental issues, and it can be challenging to simulate with other methods. It can be expressed with a simple set of rules; this makes it ideal for the use of CAs due to the simplicity and relatively computationally inexpensive calculations that are required.

Another and more recent application of the IP model is in the same area as the Eden Growth model that being cancer research. Unlike with the Eden growth model though instead of simulating the growth of a tumour, the IP model is used for the simulation of when a tumour goes through a process known as angiogenesis or vascularisation (Baish and Jain, 2000, Dobrescu and Ichim, 2009) this is when new blood vessels are formed from pre-existing blood vessels; this is a hazardous thing with cancer as this means that a tumour has a food source and can now spread throughout the body with much greater ease. By being able to simulate this is is possible to figure out ways to be able to slow down or even stop this process.

The IP model has two main variations that being TIP[2] and NTIP[3]. The type that has been implemented here is that of the NTIP. Regarding the IP, model trapping means that while the simulation is run if at any point the invader encircles a section of the defender this encircled an area of defenders becomes a forbidden zone to the invader (Ebrahimi, 2010). TIP is a more complicated version of the model with it requiring an additional check to identify if any part of the defender has become encircled. This can be done on the Lattice with algorithms such as the Hoshen-Kopelman (Al-futaisi and Patzek, 2003, Hoshen, 1997).

The difference in the models allows for the simulation of compressible defenders in the case of NTIP and incompressible in the case of TIP. The inclusion of trapping on IP can have a significant effect on the structure of the model when it is run in 2D as it can significantly alter the long-range scaling and even the local structure of the mode. However, this is not true for higher dimensions when the effect of trapping becomes increasingly short range as such the experiment here. This version IP model will not take into account trapping as in a 3D domain the chance of trapping occurring is very low and only likely in small sections and as such only serves to slow down the simulation due to the additional computation need to check for trapped cells.

The IP model like the Eden model has a straightforward set of rules. On each iteration of the model, the cell with the highest probability that is a neighbour to a currently invaded cell is set to be invaded. An example of an IP neighbourhood can be seen in Figure 3.8. By changing how this probability is assigned throughout the system, the shape and size of the cluster formed can be drastically altered allowing for the simulation to mimics different mediums and fluids.

Algorithm 3 shows the method used for the simulation of the IP model the only difference between the different IP models discussed here is the equation used to get the percolation threshold value for a cell. With this algorithm, the domain is first generated, and then the seed is added to the domain this seed can consist of multiple

[2] Invasion Percolation With Trapping

[3] Invasion Percolation Without Trapping

infected sites. After this, the growth sites list is then updated, the method for adding to the growth sites list for the IP model is discussed in detail later on in this Chapter. The simulation then iterates through each time step until the desired stopping condition is met. A single time step involves a few operations the first of these is to select the cell in the growth sites list with the highest probability and then set this cell to be invaded, after this the nearest neighbours to the newly invaded are checked to see if they are empty, if so they are then added to the growth sites list. The growth probability for the cells is assigned to the cell just before it is added to the growth sites list; this means the cells that are never added to this list do not have an assigned growth probability reducing the computation needed to compute the cluster speeding up the overall simulation.

Algorithm 3 Invasion Percolation

```
plant infected seed $S$ in the centre top of $M$ matrix
check $S$ empty neighbour sites generate growth probability and add to growth site list $G$
while stopping condition not met do
    pick the highest probability cell $C$ from $G$ and set as infected
    delete $C$ from $G$
    check $C$ empty neighbour sites generate growth probability and add to growth site list $G$
end while
```

The Initial conditions for the IP model can be one of two different styles the first of them is with a single invaded cell at the top of the domain which will be the one that is used here. The second condition that is often used is that of a blanketing of one side of the domain with a small amount of randomness to it; this randomness means that there will be peaks to the seed and not just a uniform sheet of invaded cells. With this version, every cell on that side would be iterated through a column of the invaded cell would be placed here within a range of 0 and N, where N is the maximum number of cells. The different starting conditions can have a drastic effect on the shape of the cluster that is produced as the second method will have a large selection of initial growth sites at the start allowing it to form multiple dendrites possibly.

One of the most critical aspects of the IP model is that of the invasion probability;

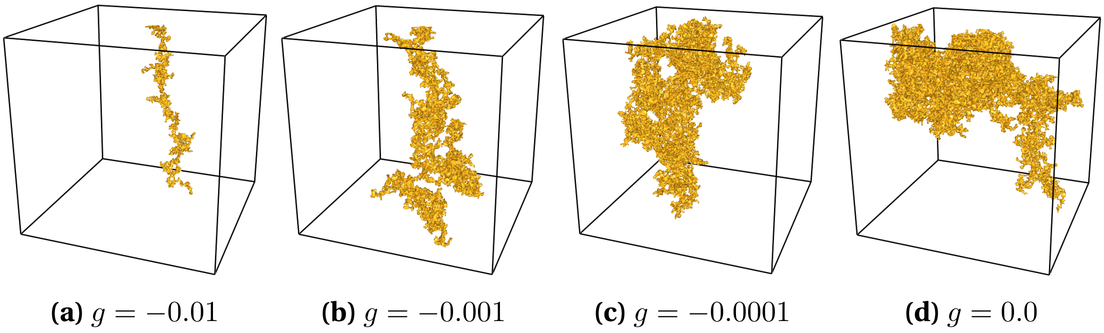


Figure 3.9: The Invasion Percolation model with the Meakin method this shows the model run until the stopping condition of touching the bottom of the domain is met. This also shows how when a beta value of 0.0 is used the Meakin model produces a Random Invasion Percolation cluster.

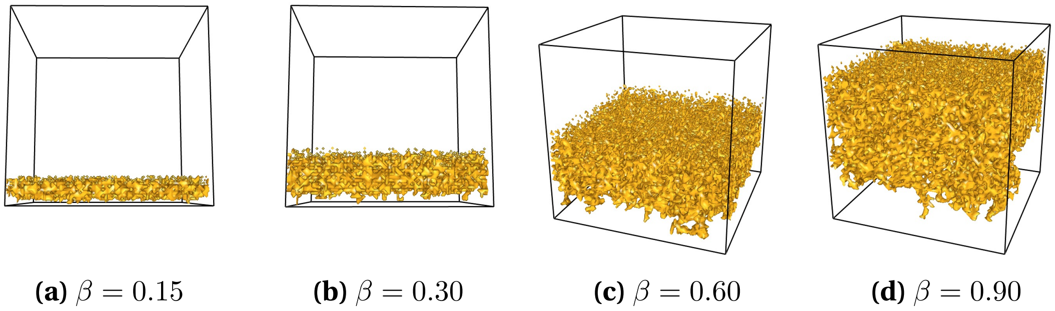


Figure 3.10: The Invasion Percolation model with the Hawick method for the distribution of growth probabilities for the cells. Showing the model run to till to stopping condition of touching the bottom of the domain is met with the beta values 0.15, 0.30, 0.60, and 0.90 starting off in the top left and ending in the bottom right. It should be not that is all cases for this version the domain is filled above the golden surface. This part of the simulation was not shown as this allow for the roughness of the surface formed to be more easily seen. β here as with Screened Eden model is the bias values for the model and is in the same range as for the Screened Eden model

this can have a significant effect on the structure of the model as such there as three methods for the calculation of this value that have been used within this work in order to give a broader understanding of the effects that these different data structures have on this model. The different methods are first Random wherein each cells bond threshold is randomly assigned within the range of $[0, 1]$ to make the IPR[4] an example of the cluster type of cluster produced by this method can be seen in Figure 3.7.

The second method used will be referred to as the Meakin Model or MIP[5]s method for bond threshold and comes from a paper on the Invasion Percolation model in a

[4] Random Invasion Percolation

[5] Meakin Invasion Percolation Model

$$P_i = U_i + g \cdot h_i \tag{3.3}$$

where:
P_i is the percolation threshold for the cell
U_i is a random number in the range of $(0, 1)$
y_i is the current height coordinate
g represents the gravitational gradient within the simulation

$$P_i = U_i \times \beta - (1 - \beta)\frac{L_y - y_i}{y_i} \tag{3.4}$$

where:
P_i is the percolation threshold for the cell
U_i is a random number in the range of $(0, 1)$
β is the erosion coefficient in the range $(0, 1)$
L_y is the total height of the domain
y_i is the y coordinate of the current cell

destabilising gradient (Meakin et al., 1992). This can be seen in Equation 3.3, the specific effect of different g values can be seen in Figure 3.9 this forms the MIP. When the Meakin Model is supplied with a g value of zero, it will produce standard Random Invasion Percolation clusters as when $g = 0$ the h_i value is cancelled out leaving behind only the random value U_i, this can be seen in Figure 3.9 where the clusters increase in size as g decreases.

The third and final method that will be used will be referred to as the Hawick method the equation for this can be seen in Equation 3.4. This specific formula was derived from one presented in a paper on flood incursion and coastal erosion (Hawick, 2014). It should be noted that the coordinate system used in this implemented is such that the top front left-hand corner is considered to be the origin of the simulation at the coordinates $(0, 0, 0)$. The effect of various p values can be seen in Figure 3.10 this forms the final version of the IP model the HIP[6].

[6] Hawick Invasion Peroclation Model

3.3 Growth Sites

The growth sites list is a crucial aspect of the growth models discussed within this work. The list is how the simulation stores all of the cells that are neighbouring to the infected/invaded cells. The most brute force way in which this list can be handled is for it to be rebuilt every iteration which is a highly computationally expensive method with a computational complexity of $O(N)$ as it would have to look at every single cell within the domain. Because of this more optimal methods have instead been selected for use here where a persistent list will be updated by adding and removing cells as needed.

The first of the methods that will be discussed is that of the simple growth sites list. In the case of the Eden growth sites model the simple growth sites list consists of a dynamic array that will store the k-index of the required cells, for the work discussed here the dynamic array that will be used will be that of the C++'s Standard Template Library Vector (Cppreference, 2017). New cells are then just added to the end of the list. When taking a cell from the list in order to obtain the next cell that is going to be infected a random integer in the range of $[0, N)$ where N is the number of cells contained within the list; is generated this is then used to index the list for the required index. After this, the cell is removed from within the list.

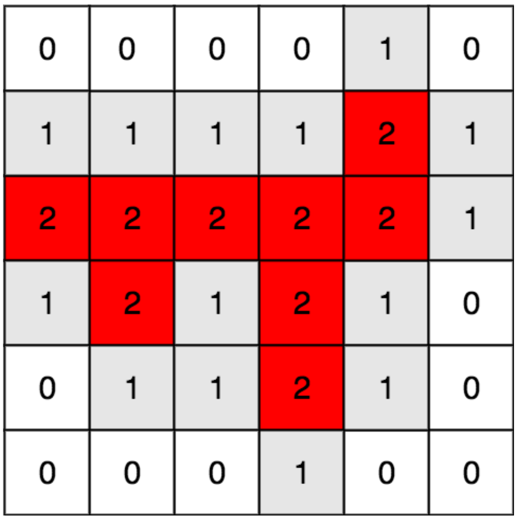
3.3.1 Eden Growth Sites Handling

The three main variants of the Eden growth model the EdenA, EdenB, and EdenC models each require a slightly different method for the handling of their growth sites this is due to the different infection probabilities that must be made use of to be able to simulate each of them.

The Eden A variant is the fastest of the three models that have been implemented; this is due to a few optimisations that the model can make use of. The first of these is with the model being able to make use of an additional cell state giving a total of three states for this model. These states represent Empty (an empty cell), Neighbour (a cell that has already been added to the growth site list), and Infected (a cell that has been infected). The addition of the Neighbour state allows for a fast way to identify cells that have already been added to the growth sites list, this way it can be ensured that every cell within the growth sites list is unique, an example of this can be seen in Figure 3.11. This is handled through a different method for each of the data structures that are used; this is discussed in more detail later on in Chapter 4 for each of the different structures. This additional state is also applied to the Screened Eden growth model as the growth probabilities are not based on the number of neighbouring cells but instead are based on the position of the growth site relative to the clusters centre of mass.

Because of this additional state guaranteeing that a cell can only exist within the growth sites list once it is easy to optimise the removal process from this list. A graphical representation of this process can be seen in figure 3.2. When getting the next index to be infected the first step is to randomly select a cell from within the growth sites list and store the value of the cell. Next step is to set this cell's value to that of the value of the last cell in the list and delete the last cell from the list. This method makes use of the design of the C++ vector dynamic list and a feature that it has specifically the pop_back function which is a fast method for the removal of the last cell in the list; this function works by merely decrementing an iterator in the structure that indicates the end of the list.

With the EdenB model, the optimal method for handling the growth sites is different from that of the EdenA variant. In the EdenA model there is the application of an additional state to show that a cell has already been added to the growth sites list, this additional state cannot be used in the case of the EdenB model; this is because the addition of a single site multiple times allows for the chance of a cell being selected from the growth list with chance proportional to the number of infected cells

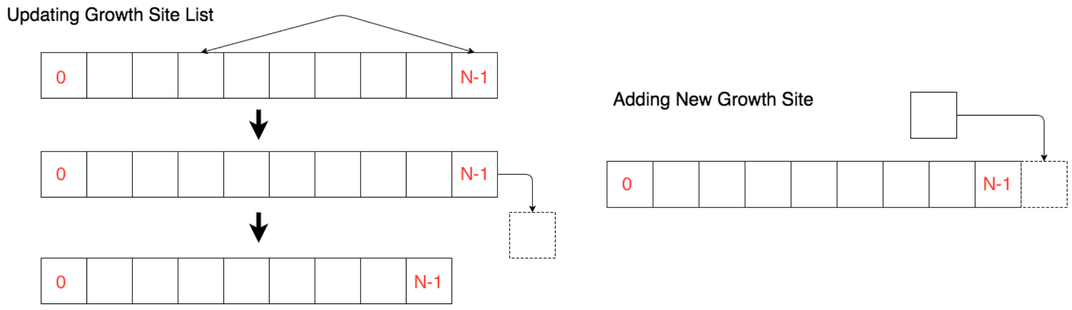


0	0	0	0	1	0
1	1	1	1	2	1
2	2	2	2	2	1
1	2	1	2	1	0
0	1	1	2	1	0
0	0	0	1	0	0

Figure 3.11: Example of a cluster grown with an additional state 0 shows empty cell state two show an infected cell and state 1 shows a neighbour cell. The addition of the additional state is used to avoid adding a cell to the growth sites list more than once. When a cell is added to the list, its state is updated to 1 and then when infecting to 2. This method is used in the EdenA model, The Screened Eden Growth Model and the Invasion Percolation Model.

that surround it as shown in Figure 3.2. This means that when removing a growth site from this list unlike the EdenA model, there is a chance that it will be necessary to remove more than one value. It is possible that there might be six entries in the list that are in need of being removed as such the removal for a site is similar to that of the EdenA, but it iterates through the list and checks if the current cell matches the index of the cell that has been infected. If so it performs the removal process shown in Figure 3.12 and discussed above. This iteration over the list means that the removal process from the EdenB model is considerably slower than the EdenA version with a computational complexity of $O(N)$.

There is another method that could have been used which is similar to the method used for the Screened Eden model, which is discussed in more detail later on within this section. Briefly, the way this method would work is with the addition of an extra variable which can be called e within the growth sites list cell which stores the number of times this cell that has been added to the list. So instead of adding another version of a cell that already exists within the list, this number would be increased this in combination of the sum of this additional variable which can be call E can allow for the correct probability distribution within the list. When selecting a cell and random



(a) Update **(b)** Add

Figure 3.12: The growth sites handling methods for the Eden a and the Screened Eden models. The Update function is used to obtain the next site to be infected and the Add function is to add a new possible growth site to the list.

number between 0 and E is generated and then you iterate through the list taking each cells e values away from the generated value until it got below zero. This method would also make use of the growth sites cell state, so it is apparent without always having to search through the list that a cell has been added or not. This method was not used in this work because it ended up being slower than the method that was selected as it required not only a full search of the list when adding a new item but also an iteration over this list when selecting the right cell.

The EdenC model is the slowest of the three unlike with the other two versions wherein the growth site list would store the cells that are the nearest neighbours to the cluster in this version the list stores that cells that are on the edge of the cluster. Hence for this method, the growth sites list will be referenced to as an edge list. When a new cell is infected, it is then added to this edge list, and the list is then updated to check that the cells contained within are still on the edge of the cluster. The selection process for the next cell to be infected is a bit more involved in this case than in the cases of the EdenA and EdenB model. First, a random cell is selected from the edge list, then the neighbour cells are checked to see if they are empty, and from this selection of empty cells, a random cell is then selected to be infected.

The Screened Eden model makes use of the same additional cell state that the EdenA model uses and as such the same removal process from the list. The only difference here is the selection of the growth site instead of it being randomly selected based on the number of cells in the list the random number U that is generated for the selection is in the range of [0, P] where P is the sum of all the cells within this lists probability. Once this number is gained the list is then iterated through with each of the cells with the lists own probability being added to a checker variable, and the selected cell is the first cell that causes the checker variable to become greater than or equal to that of U.

3.3.2 Invasion Percolation Growth Sites Handling

In the case of the IP model the list is commonly stored sorted low to high by the growth probability value that each cell, as the IP model always infects the cell with the highest probability it is a waste of memory to store a cell within the growth sites list multiple times as such IP makes use of the same 3 state system as with the EdenA model and the Screened Eden Models. In the conventional method, a cell will be added to the end of the growth site list and then sorted into its correct position within the list with the cell with the highest probability at the bottom or top of the list depending on which is the easiest to remove from. This method can be very slow as it is possible that a cell will have to be compared against all other cells within the list to be sorted. Because of this there is another method that can be used, this method was first thought up by Masson (Masson, 2016, Masson and Pride, 2014) in this paper he makes use of a list structure which is treated in a way similar to that of a binary search tree. This tree has two functions that it can perform these are the Addition and Update functions.

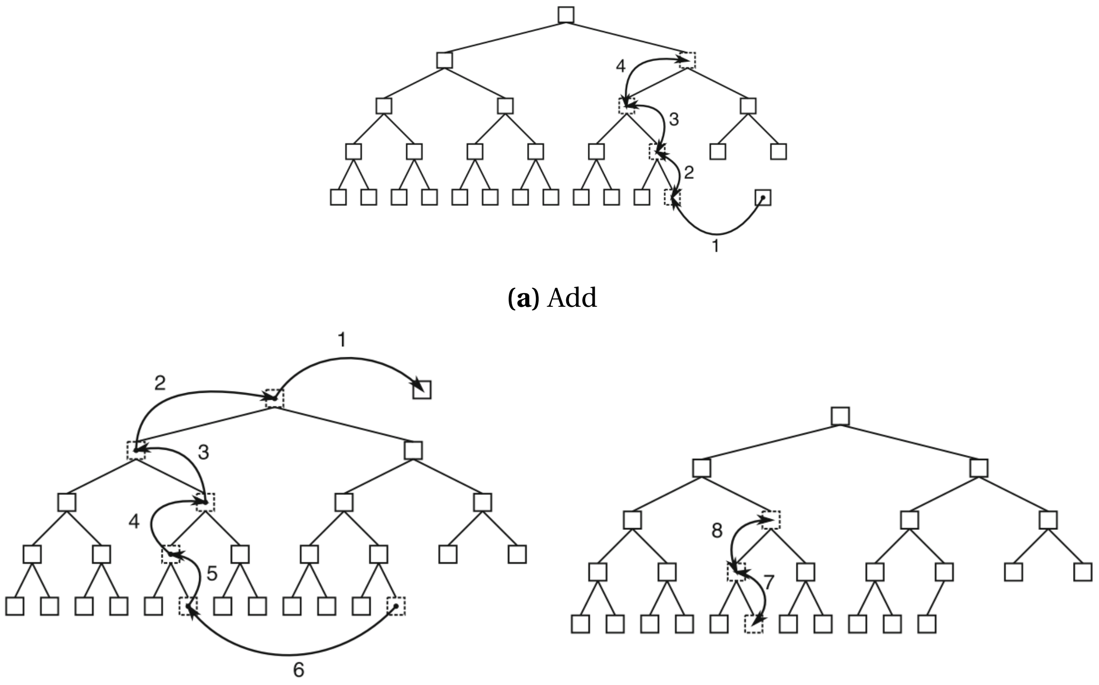
The first of these functions to be discussed is Addition. This function allows for a new cell to be added to the list. Once a cell is added to the list it is then sorted comparing the current cell to its parent node and if its percolation threshold is higher than that of the parent the two are swapped this process is then repeated until either the cells becomes the root node in the list, or it meets a parent node whose growth probability is greater than or equal to its own. A cell is considered to be the parent of another cell based on a simple rule. If the current cell is N, then it has a left child at position $2N$ within the list, and right child at a position of $2N+1$ within the list and its parent is at $\lfloor(N/2)\rfloor$. This addition process can be seen in an example in Figure 3.13a.

The second function is the Update function, this function allows for the obtaining of the index of the next cell to be invaded and removes the cell from the list. The inner working of the Update function is a little more complicated than the Addition function. As the tree is sorted based on the percolation threshold of the cell, the cell with the highest probability always ends up being the first node in the list. So the first step in this process is to store the value of the root element and remove it from the list, next is to this space must be filled and so its children nodes are checked to make use of the method mentioned before and the one with the highest chance is placed in its new position. Then continue down the tree comparing parent to child until it reaches a node with no child. At this point, the final cell in the list is then placed into the final empty slot within the tree. This cell must then be sorted into its correct position within the structure and so is compared with its parent and swapping if need continuing up the list is the same manner as with the additional function until it is correctly positioned. This process can be seen in Figure 3.13b.

In this tree the left child is found in the $2N$ cell where $index$ is the index of the parent node, and the right node is found in the cell $2N+1$ because of the way in which Fortran indexes its array, that being arrays start at one and not 0 as with languages such as C++, when indexing these cells within the list it is necessary to minus one from this index but when performing the test with the length of the array to determine whether your current node has a child node you have to make use of the original index value as would have been used in the Fortran version of the function.

3.4 Fill Density

When testing the effectiveness of the different data structures with these models, it is essential to understand how these models are commonly run and to what extent the



(a) Add

(b) Update (left to right)

Figure 3.13: IP Add Branch and Update Procedures procedure (Masson and Pride, 2014). This method is only used for the Invasion Percolation model.

cluster will fill the computational domain, as this could help in explaining the different results that are obtained in Chapter 5. With these models the likelihood that they will be run to total grid completion is very low, in most cases, they have a stopping condition that is not necessarily concerning the number of iterations for which the model runs. For example with the Eden growth model, the most common stopping condition is when the cluster reaches the edge of the computational domain; this is because if it was allowed to continue to grow after this point, the cluster could begin to interact with its opposite side causing a possible distorting of the result or with a non-periodic boundary it would flatten out the cluster; this will be the stopping condition that is used here for all the variants of the Eden model including the Screened Eden model. The IP model has two common stopping conditions though one that is used depends on the initial condition of the simulation if the initial condition has the centre cell being infected then the IP model has the same stopping condition as the Eden model, but this is not a common way to use this model in 3D. Instead, because the initial condition that is used here where the first cell to be infected will be in the

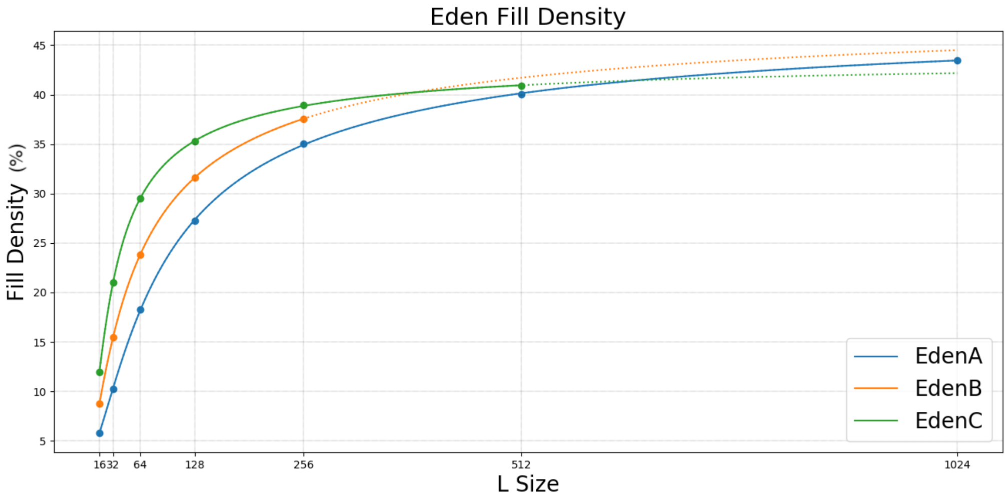


Figure 3.14: This shows the 5PL curve of best fit for the Eden growth model with the line extended to the 2048 dimension size for all three versions of the model this shows an issue with the predicted data due to the A-line overtaking that of the B and C this goes to show that the 4PL data is underestimating the B and C fill levels.

top centre of the domain the stopping condition will be when the cluster has reached the bottom of the domain, the model will also have a non-periodic domain boundary on all sides of the domain.

This section will start with a discussion of the Eden Growth Model and its three variants followed by the Screened Eden growth model. After with it will then move on to discussing the IP model with the three growth probability methods the have been discussed earlier on in this chapter these being the Random, Hawick, and the Meakin variants of the IP model.

The Eden growth model shows an increase in the percentage fill of the domain as the size of the domain increases, this effect can be seen in Figure 3.14. The clusters that are grown on these larger grids also have a greater chance of becoming more spherical than on a smaller grid. Due to the time required to run the EdenB and EdenC model, there is a limit to the amount of data that can be discussed here as such the dotted line that can be seen in Figure 3.14 represents the predicted curve past the point of the measured data.

The Screened Eden Growth models fill density has a much more interesting range

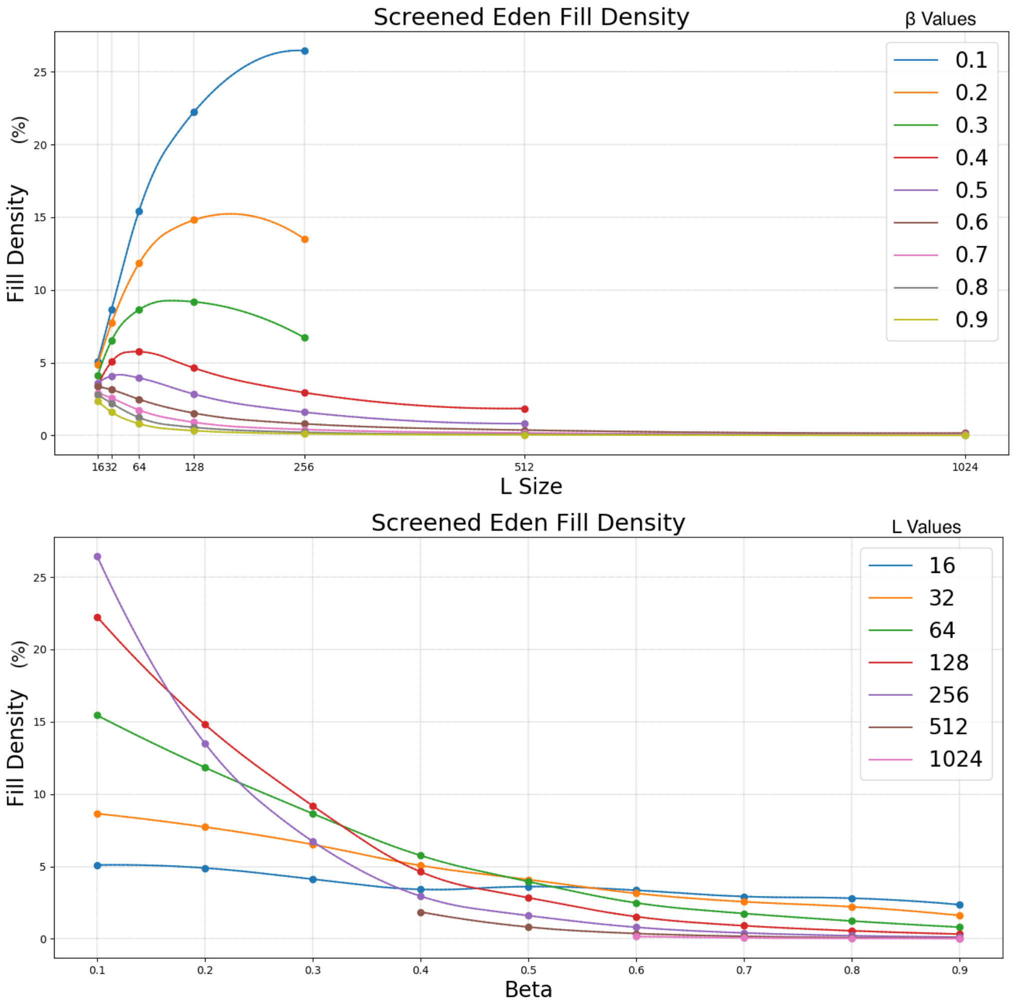


Figure 3.15: Screened Eden Growth Model fill density graph. This shows the fill for a range of beta values from 0.1 to 0.9 in increments of 0.1. A beta value of 0 has not been used as this would must produce an EdenA cluster.

of variance to it then the unscreened versions of the Eden growth model. At the smallest tested L size of 16, the screened Eden growth model closely relates to that of the EdenA model of the growth sites. In these tests we ran the test for beta values from 0.1 to 0.9 in increments of 0.1, the reason for not testing a beta value of 0.0 is because this would produce the same results as that of the EdenA model this is due to the way in which the growth sites are handled. The results for the fill experiments showing the median for the range produced can be seen in Figure 3.15 where two graphs as seen the top graph shows the fill density for different beta values plotted against the domain size and the graph on the bottom shows the fill density for different domain sizes plotted against the beta values.

In order to compute these clusters in a manner that allowed for large enough batches a HPC[7], which is commonly called a Supercomputer, named VIPER which is located at the University of Hull was utilised in order to compute the clusters in a practical time frame but due to restrictions on the runtime of a program on this system it was not possible to obtain a full range of fill data for the Screened Eden model as with the EdenB and EdenC so the only thing that can be done is to make some assumption on how these curve may act, but this will not be used in the actual analysis as this could very easily mislead and lead to a highly incorrect conclusion.

The Screened Eden model is a fascinating model when looking at the domain fill as with lower values of beta it starts off like the Eden model where they increase in the size of the domain in fact when a beta value of 0 is used the model acts precisely like that of an EdenA model only with much more computation going on within the model slowing it down considerably. However, as soon as the beta value increases from zero even to a value as low as 0.1 the change in the model is very apparent when looking at the fill. At an L size of 256 the EdenA model takes up a median of 35.03% of the domain, and the Screened Eden model takes up only 26.45% of the domain, and this decrease in the fill of the domain continues as the beta value increases which can easily be seen when looking at the bottom graph shown in Figure 3.15.

The Screened Eden model exhibits interesting behaviour for beta value in the range on 0.2 to 0.5 wherein as the L size increases, though the fill first starts of increasing eventually it begins to rapidly drop off in some instances such that by the time the L size reaches 1024, meaning a domain of $(1024 \times 1024 \times 1024)$, the fill percentage for the beta values that there is data for have already reached well below 1%. Though from 0.6 and up it doesn't have this initial increase in the fill. For the smaller beta values of 0.1 to 0.3, it was only possible to obtain fill values for up to an L size of 256 due to the runtime of the model at this point. However, it can be assumed that the 0.2 and 0.3 values would follow a similar pattern as the higher beta values slowly decreasing past this point as the L size increases. Unfortunately, it is not possible to

[7] High Percomance Computer

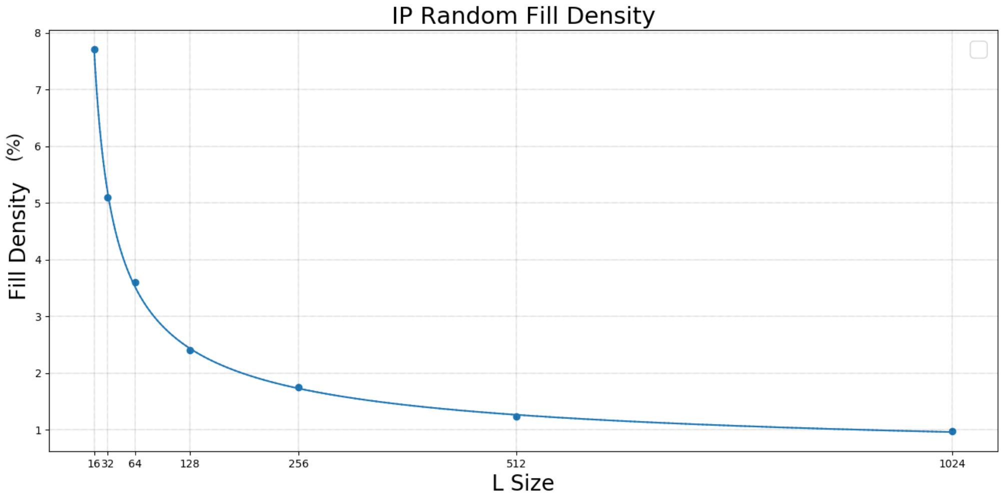


Figure 3.16: Graph showing the fill density for the Invasion Percolation model where the percolation threshold is set by the random method.

gain to an understanding of how the 0.1 beta value will progress as the domain size increases; it could very likely follow the general trend that the standard Eden model does or at some point it could begin to decrease like with the higher beta values for this model.

From here on the second model type, the IP model will be discussed, with each of its different version discussed in the order of Random, Meakin, and finally Hawick. The Random version and the Meakin model are the two that are most closely related to one another with the Meakin model able of reproducing the random model by feeding it a beta value of 0. Unlike the Eden growth mode; as the L size increases the fill percentage of the Random IP model decreases. At the smallest L size of size, these versions of the IP model take up more of the domain then any of the different versions of the Eden growth model. This model very quickly tends towards having a fill percentage of less than 1%.

The Meakin version of the IP models was testing with the beta values -0.1, -0.01, -0.001, -0.0001, -0.00001, and -0.000001. The fill density of each of the beta values increases as the value tends to 0. This model gives the lowest of the fill values out of all of the different models, and as such it will be interesting to see if this effect which of the different data structures will be the best suited in this case

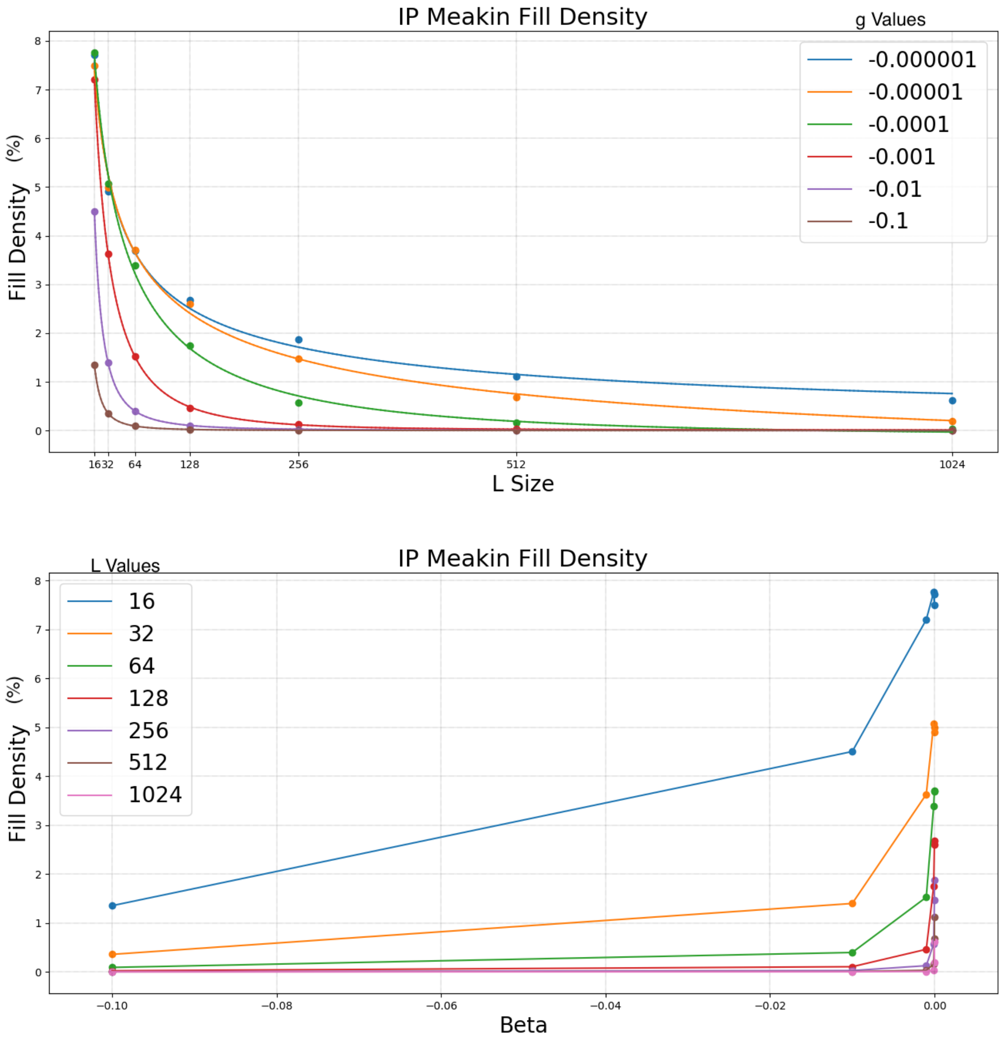


Figure 3.17: Graph showing the fill density for the Invasion Percolation model where the percolation threshold is set by the Meakin method. With the top graph showing the effect of the g values of -0.1, -0.01, -0.001, -0.0001, -0.00001, and -0.000001. The bottom graph show the effect of the change in domain size (L size shown in the ke

The Hawick method for the IP model is the final version that will be discussed this version of the model acts much differently than all the other growth models that have been discussed. This method was first used to simulate coastal erosion on a 2D simulation. This means that the model wants to fill as much of the domain as it can as it makes its way to the edge of the domain much more than the other version of the IP model. The fill density for different beta values in the range of 0.1 to 0.9 can be seen in Figure 3.18. This model gives the largest fill density of any other model discussed within this thesis with a maximum fill density at an L size of 1024 with a beta value of 0.1 where the mean density is 95.7% with a minimum of 95.5% and a maximum value

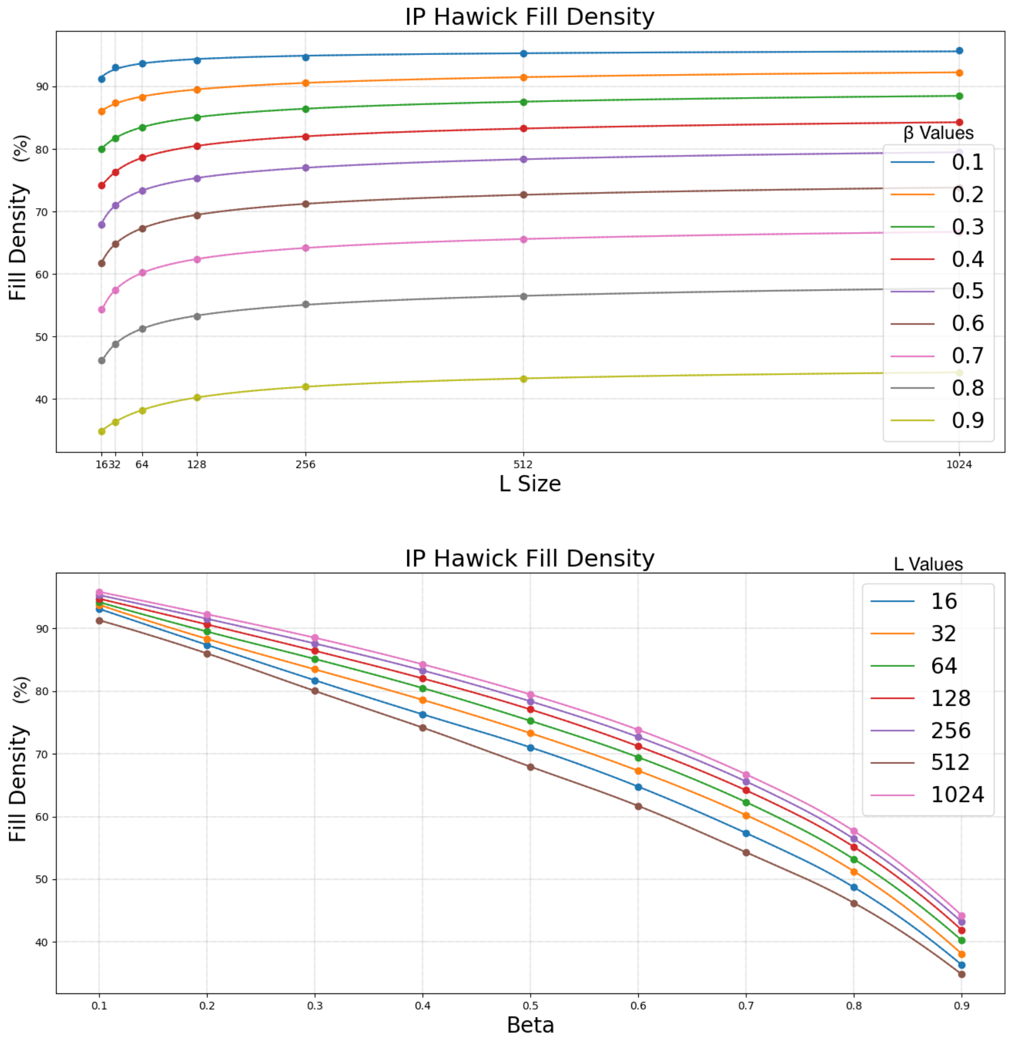


Figure 3.18: Graph showing the fill density for the Invasion Percolation model where the percolation threshold is set by the Hawick method. With the top graph showing the effect of the β values in the range of 0.1 to 0.9 in increments of 0.1. The bottom graph show the effect of the change in domain size (L size shown in the key)

of 95.9%. Though there is a broad range in the fill percentages for this model with a beta value of 0.9 giving a mean value of 44.3% at a dim size of 1024 which is only 1% more than the EdenA growth model.

The results in these experiments will be useful in the analysis of the various run times of these models with the different data structures as it will allow for a prediction on whether a different type of model that is not discussed here stands a chance of benefiting from one of the methods discussed. The results here show that amount of the domain that is filled but the cluster as the domain grows in size is considerably different for each of the model, whereas some increase as the domain grow as with the Eden models, others decrease such as the Meakin model and in other cases more interesting behaviour is exhibited as with the Screened Eden model.

3.5 Summary

This chapter has gone into the History of the different types of Eden growth model and IP model that will be used for testing the effects of different data structures. It also discussed how these models have been used in different fields including cancer research where the Eden growth model is used for the simulation of a tumour its self and the Invasion Percolation model is used for the simulation of when a tumour gores through angiogenesis and starts to grow blood vessels.

This chapter then also discussed the different methods that have been implemented for the handling of the growth sites of the different models including a method that will be referred to as the Masson method (named after the person that invented it (Masson, 2016, Masson and Pride, 2014)) for the simulation of the Invasion Percolation model which involves the use of a binary tree inspired method for the storage sorting and selection of the next cell that will be infected in the domain, this is the method that is used for all the different variants of the Invasion Percolation model that this work will concern itself with.

The final part of this chapter went into the effect that the domain size has on the sparsity of the different models. This information is an essential part of the investigation into the application of the different data structures as it will allow for a better understanding of why a particular structure may be the best in one situation but not in another. This will mean that the data here can be applied to a broader range of models that are not discussed here if the model domain fill is known it could give a good idea of which data structure might be the best to use.

When human judgement and big data intersect there are some funny things that happen.

Nate Silver

CHAPTER 4

Data Structures

DATA structures are an abstract method for the storage of data in an organised manner so that it can be used efficiently by the program. Data structures make use of a variety of different abstract data types depending on the type of structure in addition to the purpose for which it will be used. These data types specify the multitude of operations that can be performed on the data structure in addition to the computational complexity of the structure, which is described with the aid of a form of notation call Big O notation. Big O describes the number of actions performs in the worst case scenario of a specific algorithm based off of the scaling of the data supplied to the algorithm. The effect that different Big O values have on the runtime of an algorithm can be seen in Figure 4.1.

$O(1)$ describes an algorithm that will always be executed at the same time regardless of the size of the data that is input such as accessing a specific value from an array. The notation means that the algorithm has a fixed complexity where only one operation is needed to complete the task, though the 1 can be substituted for a

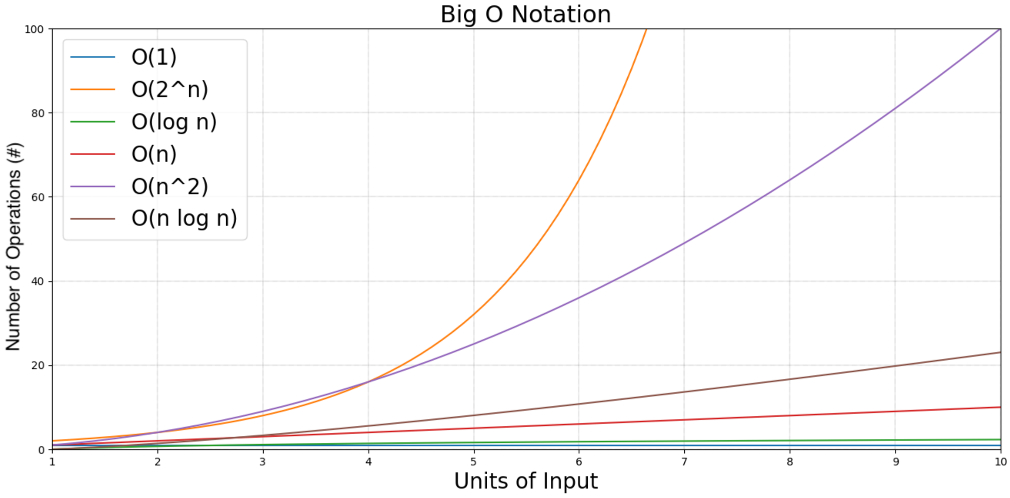


Figure 4.1: Graph showing the effect that increases units of input on the number of operations an algorithm has to perform for different Big O values. Big O notation is used to describe the complexity of an algorithm as the size of the data input into it grows in size.

number, in this case, this would mean an increased number of fixed operations are required. $O(N)$ describes an algorithm where the grows linearly in fashion with the growth in the size of the input data, such as checking an array to see if it contains a specific value. There is a wide range of different computational complexities such as logarithmic and exponential.

One of the main areas of study within computational complexity concerns itself with the problem of P vs NP which is a currently unsolved problem. It asks whether every single problem which can be checked within polynomial time can also be solved within polynomial time (is it $P = NP$ or $P \neq NP$). An answer to this problem could have a wide-reaching impact on the whole of society. If it was found that $P = NP$ this would have both beneficial and detrimental impacts on the world. One negative drawback would be to encryption which relies on the fact that it is very complicated to crack an efficient solution would make most of the currently existing cryptography useless and would mean that the field would have to look into alternate solutions if it was even possible to perform encryption anymore.

Different data structures are most aptly suited to various types of applications. The specialisation of the data structure is highly dependent on the structure itself some are very highly specialised with very few other uses, and some are very generalised. For instance, hash tables are a type of data structure which is very useful in the field of compiler implementation for the lookup of identifiers but can be generalised to many other tasks, such as is done here.

They provide a optimal means to store and organize data such as with databases which can use numerous structures such as an SQL based databased which makes use of the relational model, organizing the data into one or more tables with a unique key identifying each row, rows represent instances while columns represent values attributed to that entry. Another example would be a graph-based system such as the one used but packages such as Neo4j which uses a property graph model, this is a model in which the graph contains a collection of connected entities called nodes which can hold any number of attributes and are represented in the system as key-value-pairs these values are then connected to each other through the use edges to demonstrate a relationship between the nodes which is determined by the label on the edge.

Many data structures are designed to allow the system to access and store the data from anywhere within the memory of the computer; this allows for the structure to be made dynamic allowing it to grow and shrink in size as needed. This is often achieved through the use of pointers which are an object whose value refers to the location of another value, obtaining the value from the pointer is known as dereferencing the pointer. There are a large variety of languages that support the use of pointers such as C++ and Java, which are two of the most widely used programming languages to date. The creation of all data structures requires the writing of a set of algorithms to allow for them to be instantiated and manipulated through a set of functions that govern adding to, removing from, and searching the data structure. The efficiency of a specific data structure is intrinsically linked to these functions and as such the speed of the structure is limited by these functions as well as the format of the structure.

There would have been a different path to the speed up of these simulations but

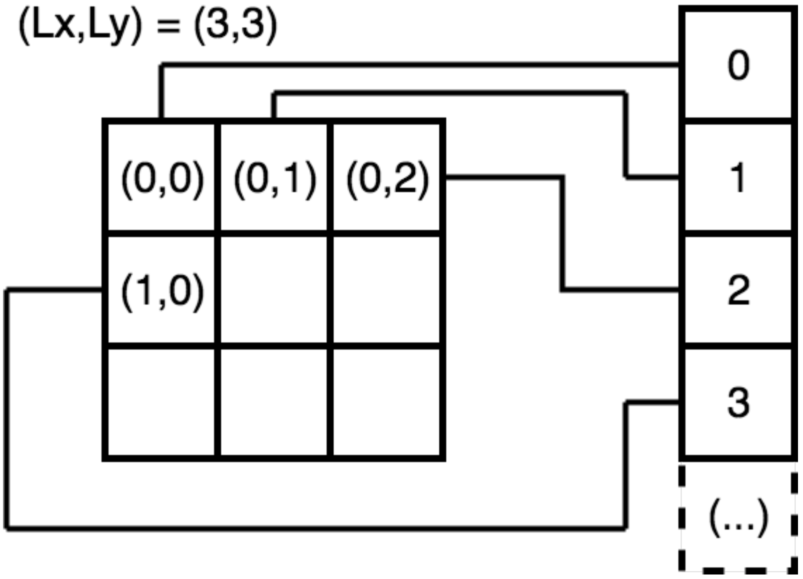


Figure 4.2: Example of k-indexing from a 2D grid with an L size of 3 in both the x and y dimensions over to a 1D array which shows the resulting k indexed values for each of the coordinates. This method can be used for the flattening any N-dimensional array so long as the size of each of the dimensions of the domain are known.

it would have come with what was felt to be a significant issue that of parallelization. A GPU[1] based implementation of the system could have offered significant speed ups over the methods investigated here. However there has already been work done in the parallelization of this model (Machta and Greenlaw, 1994) but due the asynchronous nature of the model moving to the GPU for speed up can significantly effect the resulting structures even if the a fractally similar (this is discussed in Chapter) 7.

Even the highly specialised data structures which seem only to be fit for a singular purpose are sometimes able to be used in other situations and can even offer great benefits such as a reduced memory consumption or a faster search time. This is where the interest for the work done here comes in as if simulations can be sped up through the application of these alternate data structures it could be of great benefit to researches who make sure of these models, by allowing them to perform experiments much more rapidly then before.

[1] Graphics Processing Unit

4.1 Lattice

The Lattice structure consists of a simple character-based array and is the standard data structure that is used for the simulation of growth models. It involves the use of either an N dimension array or one vast array that is indexed in such a way to act as an N-dimensional array. This can be done through the use of k-indexing, which can take an N-dimensional, in this case a 3D coordinate, within the computational domain and flatten it into a single dimension for indexing as well as take a k-indexed value and convert it back into an N-dimensional coordinate which is useful in edge detection in the models, a visualization of this method can be seen in Figure 4.2. The equation for the flattening of an array from N dimensions to a k-index can be seen in Figure 4.1 and the code that is used for reverting this flattened value from a k-index to an N-dimensional coordinate can be seen in Figure 4.3, This method is also used for all the other structures apart from the Octree the reason for this will be discussed in Section 4.3.

$$k(x, y, z) = x + (L_x \times y) + ((L_x \times L_y) \times z) \tag{4.1}$$

where

x, y, z represent the 3D coordinates for the specific cell whose probability is being generated

L_x is the length of the x dimension

L_y is the length of the y dimension

There are many benefits to the use of this method as opposed to the use of a 3-dimensional vector storing the x, y, and z. The main benefit of this method is that it does not matter how many dimensions there are to the grid; this will work just as well with a 2D grid as it will with a 7D grid. An additional benefit is that in that case where the index itself needs to be stored as with the AVL[2] tree (discussed in the following

[2] Adelson-Velsky and Landis Tree

```
const int N_dims = 3;

int * fromK(const int k, const int * L){
  int *x = new int[N_dims];

  int lp = L[0];
  for(int i = 1; i < N_dims; i++) lp*=L[i];

  for (int i = N_dims - 1; i >= 0; --i) {
    lp /= L;
    x[i] = (k / lp);
    k %= lp;
  }

  return x;
}
```

Figure 4.3: The code used in the experiments to convert a k-index values into an N dimension coordinate, where N_dim is the number of dimensions. In the case of these models this is 3.

chapter) this method allows for the coordinate to be stored with a single value instead of the multiple that would otherwise be needed reducing the memory consumption for each new cell with the system.

The computational complexity of the search of a single neighbouring cell for each of the data structures varies quite a lot. In the case of the lattice structure due to the ability to index the array with 3D coordinate or a flattened version of these coordinates into a single k value it has the lowest computation complexity being that of $O(1)$ this means that it only ever makes one check every time it needs to see if a cell is empty. But this doesn't mean that it will always be the fastest as it is possible for the grid and the growth sites list to be of such a size that it may not all be contained within the RAM of the system and so must be loaded back on to the RAM from virtual memory before it can be checked which is a relatively slow process and could drastically slow down the searching of the grid depending on how often this has to be done.

4.2 Balanced Binary Search Tree

For the first of the tree based structures that we have implemented for this work, we have made use of the Binary Search Tree (BST) (Knuth, 1998c, Leiserson et al., 2009) seen in Figure 4.4 specifically a balanced binary search tree. In a BST each node contains the nodes value and two pointers to other nodes; these are often called the left and right child nodes. The left node will point to a node whose value is less than the current nodes value and the right node with a point to a node whose value is higher than the value of the current node. An example of the logical structure of a binary search tree can be seen in Figure 4.4.

The binary search tree is probably one of the most widely used data structures in computing and tends to be one of the first about which a computer scientist will learn. A widespread use of BSTs that most people benefit from is within a router; here they are used for the storage of the routing table which stores the different routes to particular network destinations as well as values that can be of benefit such as the distance. They are also used in a computers heap for the implementation of priority queues (García et al., 1999) which advise the scheduling process within an operating system. The final method that will be brought up here is that of Binary Space Partitioning (Naylor, 1998) which is a method that is used in 3D rendering algorithms; it was used in Games such as Doom and Quake. This method involves the division of space within the environment in half based off of the position of the walls; this is then stored in a binary tree structure. This method allowed for an efficient method to avoid overdrawing in a scene allowing for the rendering of more advanced graphics.

Figure 4.4 is an example of a balanced BST, meaning that no one side of the tree being significantly large then the other. This balancing is a significant aspect of the tree which can lead to an enormous impact in the performance of the tree when both inserting into the tree and searching the tree for a value, in the worst cases this can lead to the BST taking the form of a singularly linked list instead of a BST.

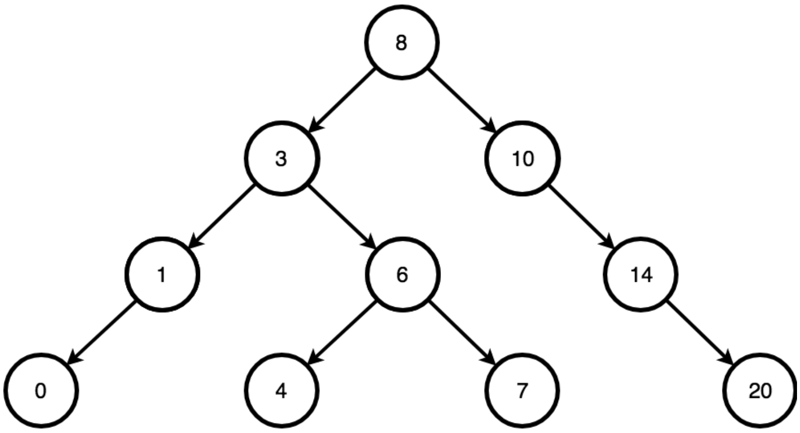


Figure 4.4: An example of a simple binary search tree with 10 nodes and a height of 3.

A balanced BST has a computational complexity of $O(\log N)$ for both the searching of the structure and the insertion into the structure. Whereas a singularly linked list has a computational complexity of $O(N)$ this can show the level of effect that this can have on the performance of the tree if it is filled in an unbalanced manner. Due to the way in which these clusters can grow this can mean that a standard binary search tree can very quickly become unbalanced and thus lead to inflated insertion and search times. This means that a balancing algorithm should be used in order to ensure that the tree remains balanced and gives an optimal insertion and search time, especially in the case that presents itself here where the cells that will be added to the tree will have unpredictable values. When a balancing method has been implemented the name of the tree changes the two most common are the Adelson-Velskii and Landis (AVL) tree (Adelson-Velskii and Landis, 1962, Knuth, 1998a) and the Red-Black tree (Knuth, 1998b, Leiserson et al., 2009).

Each of these two versions of a balancing BST takes a different approach. The first of them the AVL tree makes use of the tree height to keep balance within the tree. In an AVL tree, the height of the two child subtrees can differ by at most one. If at any point this rule is not met the tree will begin to rebalance itself. This rebalancing is done through the use of tree rotations; this can be done with single rotations or double rotations. There is a simple set of rules for the AVL tree to help decide which type of rotation should be performed. The algorithm that performs the sorting can

Algorithm 4 This describes how the AVL is rebalanced so that it maintains its $O(\log N)$ search time.

```
if tree is left heavy then
    if trees left sub-tree is right heavy then
        Do a double rotation
    else
        Do a single rotation
    end if
else if tree is right heavy then
    if trees right sub-tree is right heavy then
        Do a double rotation
    else
        Do a single rotation
    end if
end if
```

be seen in Algorithm 4.

A Red-Black tree is similar to an AVL tree in the way that it fixes an unbalanced tree through the use of single and double rotations. The way in which these trees differ is in how they decide whether the tree is unbalanced Red-Black tree makes use of a colouring mechanic with two colours, Red and Black which is where it gets its name from. Each node can be one of these two colours; there are a few conditions that are needed for a red-black tree to work correctly. The first is that the root node is set to black (though this rule is sometimes omitted), all the nodes that do not contain a value a set to black. The next condition is that if a node is black, then both of its children are red. The rule that decides if the tree needs to be rebalanced states that every path from a particular node (N) to a descendant leaf must have the same number of black nodes not including the N node.

Both of the trees have their benefits and drawbacks. For example because an AVL is more rigid in the way that it manages the tree it is possible for it to offer faster lookups over a Red-Black tree this makes it a better choice for systems that are search heavy, whereas the slightly more lax rules of the Red-Black tree make them more of a benefit in insertion heavy situations. There is also the space difference between an AVL tree and a Red-Black tree with the AVL tree it is needed to store the specific height

```
inline int Masked_AVL::MASK(int x) {
  return (x < 0 ? -x : x);
}
```

Figure 4.5: AVL Mask which is used to help in the handling of the additional state which is stored as a negative value. This allows for this storage without increasing that memory consumption of the data structure.

data of the node mean a significant increase in the amount of stored data. However, with a red-black tree, there is a useful trick that can be used to avoid having to store extra data to determine the colour of the node. If the data that is being stored in the tree is always guaranteed to be higher than one the state of the node can be stored in the sign bit of the value, this would mean for example a black node could be denoted by a negative number and a red by a positive number or vice versa. In this case, the AVL tree was selected as the growth models used here will be more search heavy then they will be insertion heavy.

For the EdenA and the IP models, there is the requirement that a cell upon being added to the growth sites list in order to ensure that each cell has only been added to the growth sites list once this fact must be represented in the structure that stores the cluster. In order to do this with an AVL, the state of the cell is stored within the sign bit of the index value for the node. If a cell is negative, the cell is considered to be an empty neighbour cell whereas if positive it is considered to be a filled infected/invaded cell. To make the correct comparison between the values in the tree, the value must be positive this ensures that the node is inserted into the correct location within the tree so that when it becomes infected and is set to a positive value, it does not cause a significant change to the tree which could be a time-consuming process. To this end the absolute of the value must be used in the comparison, a few different methods for calculating this where tested, these included the std built in abs function as well as bit shifting, But the fastest method that was tested ended up being a simple inline function that contained a ternary operator that checks if the value is less than 0 and if so returns the positive else it just return the original value fed into it, this function can be seen in Figure 4.5.

There is one issue that arises from this use of the sign bit to store the state of the cell (neighbouring or infected). This issue occurs if the number 0 is added to the structure as a neighbouring value; this is because it is not possible to store a negative 0 value in C++. The solution to this that has been implemented is a straightforward one when adding a value to the structure it is incremented by one, and then if the cell is a neighbouring cell it is set to the negative value after the correct position with the tree has been found. However, this must be taken into account when searching through the tree for a value, and this is done through the use of the same mask used in the insertion.

Except for the initial seed and in the case of the EdenC model, every cell will be added to the structure first as a neighbouring cell this allows for the code to be further simplified for use in that the state of the cell need not be passed to the structure. When inserting a cell if a null node is reached when searching for the values position the negative of that value is added else if the negative of the value is found it is merely set to be the positive version of itself.

4.3 Octree

Octrees (Meagher, 1980, 1982) are an example of a spatial data structure, similar to that of kd-trees (Duncan et al., 2001, Husselmann, 2014) or R-trees (Guttman, 1984). The Octree is the 3D equivalent of a quadtree which subdivides a 2D plan into four quadrants recursively until the desired resolution is reached. In the case of the Octree, a 3D space is subdivided into eight octant's recursively until the desired resolution is reached an example of this recursive storage with increasing resolution the further down the tree it goes can be seen in Figure 4.6, the further down the tree a node is smaller the amount of space it refers to. Each node refers to a specific 3D point which is the centre of the octant. The octree was first discussed in 1980 (Meagher, 1980). Since its conception octrees have become a common spatial structure used in many different areas.

In simulations such as flocking an example of which is the boids model (Husselmann, 2014, Husselmann and Hawick, 2011, Reynolds, 1995) or other n-body style simulations (Bédorf et al., 2012) they can be used to reduce the number agents within the environment that need to be iterated through to calculate values such as the cohesion vector in the case of a boid simulation. This is be done by assuming that agents who are outside of the same octant as the current agent are too far away to have any significant effect on the current agent, this can be a considerable optimization to these types of simulations allowing for thousands of individual agents to be easily simulated in real time and with the addition of the massive levels of parallelization that GPUs can supply this can become millions of agents. This same idea is also used in video games to reduce the number of objects that the games have to calculate collision detection with (Jime et al., 2001).

Another powerful use for octrees is in the rendering of 3D objects through the use of ray-tracing (Laine, 2011). Ray tracing is a rendering technique that generates an image by tracing the rays of light as pixels in an image it can produce very detailed render with a high level of realism. But it is a very computationally costly method, meaning that it can be challenging to use effectively in a real-time sense and tends to be used for pre-rendered scenes such as with Disney films where they make use of their Hyperion software as in the case of the film Big Hero 6 which required the use of a compute-cluster without which the film would have been an impossibility to render in a timeframe that would have made it feasible to make. Octrees can improve the performance of this making it possible to render large volumetric data sets that can consist of millions of voxels; a voxel is to 3D space what a pixel is in a 2D bitmap, such as with the work in by Nvidia on GigaVoxels (Crassin et al., 2009). It is because of things like this that octree is very popular in the field of volume rendering (Knoll, 2006) and 3D space carving (Kutulakos and Seitz, 2000).

As an Octrees is filled up, it is possible to be able to delete all child nodes from an

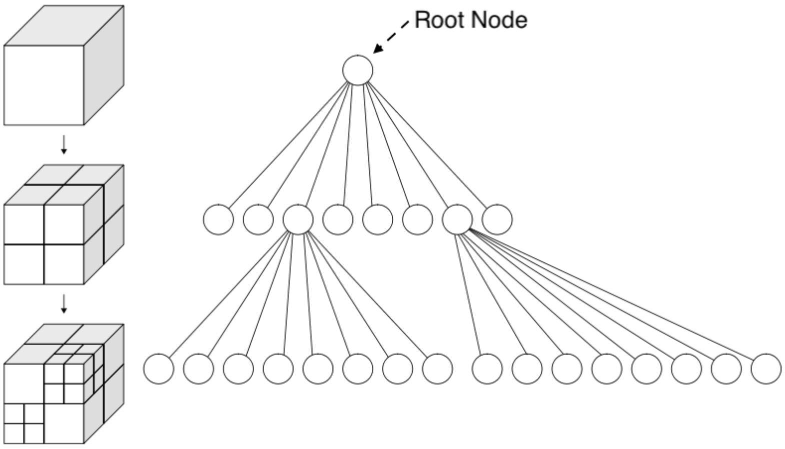


Figure 4.6: An example of an octree in both a cubic grid based representation and a tree based structure (Wikipedia, 2018). This shows how the further down the tree a node is the higher the resolution of the node.

octant that is wholly filled allowing for a sizeable possible reduction in the amount of memory consumed. The search time of the structure, as well as its memory usage, differs from a lot of other structures because of this as is doesn't grow consistently as cells are added to the structure. For example, with the AVL the more prolonged the simulation runs for and the more cells that are infected, the larger the tree becomes increasing search time as well as memory usage. With an Octree, this memory consumption and search time will reach a sort of critical mass where once this happens the number of nodes within the structure will start to reduce as a culling of the data points from the structure will begin until the entire domain is filled wherein the structure will consist of a single node.

Both the search and insert functions just like the with binary trees are recursive functions. The search and insertion functions in the octree functions very similarly to the AVL with one significant difference. This difference is in how the next child node to visit is selected; this is done by making use of the function shown in Figure 4.7 this allows for a quick and easy way to figure out which of the child octant's contains or would contain the cell being searched for. The function will fall down the structure until it hits a null node meaning that the cell is not in the tree and in the case of the insert function should be added as an infected node meaning the index is within the tree. After a cell has been added to the octree it must go through what can be consid-

```
int Octree::get_octant(const vec3 &pos, const vec3 &pos2){
  int oct = 0;
  if (pos.v[0] > pos2.v[0]) oct |= 4;
  if (pos.v[1] > pos2.v[1]) oct |= 2;
  if (pos.v[2] > pos2.v[2]) oct |= 1;
  return oct;
}
```

Figure 4.7: The method used for the selection of the correct child octant the searched for point lays in the case of the search function in or should lay in the case of the insert function.

```
void Octree::calcualte_depth(cosnt int dim){
  int temp = 2, max_depth = 0;
  while (temp != dim){
    temp = 2;
    max_depth++;
    for (int i = 0; i < max_depth - 1; ++i) temp *= 2;
  }
  max_depth++;
}
```

Figure 4.8: Code used for the calculation of the max depth for the octree. This code requires that the L size of the domain be not on the same on all sides but also that it is a power of 2 number such as 16 which is 2^4 or 128 which is 2^7.

ered and clean up phase where, as the recursive function start to go back up the nodes each nod will check all of its child nodes, and if each of these nodes are infected then the child nodes are deleted, and the current node is set to be infected.

When inserting a variable into the octree, this method keeps count of the current depth of the tree and will only allow the cell to be added to the structure when the depth has matched the max depth. This max depth variable is to ensure that when a cell is initialised, it is at a point with the structure of the octree where that cell only takes up one single cell. The calculation for the max depth is dependant on the L size of the domain and can be seen in Figure 4.8. At an L size of 128 would give a depth of 8 and an L size of 1024 would give a depth of 11.

The additional state cannot be stored in the same way as with the AVL tree where the negative of the index means that a cell is just a neighbouring cell. In the octree, the cells centre point is stored in the form of a vector3, which is a structure containing three values storing the x,y, and z coordinates of these centre point for the cell. Because of this, an additional value has been used to store this information, in this case, a char has been selected to define the state. A boolean value could have also been used instead as they both take up the same amount of space within the memory of the system and the read-write times are no different. The only reason that a char was selected over a boolean value was just down to simple preference.

4.4 Hash Table and Hash Set

Hash Table and Hash Sets are examples of a class of data structure that are commonly used and only subtly different from one another. They work through the implementation of what is called an associative array; this structure maps keys to values within itself. The hash table gets its name from the hash function that is used to work out the index of the value within the structure making the decision which bucket to place the value into. A bucket can be thought of as a collection of values whose hashes are the same. If the hash function that is used were perfect every value within the table with a different key value would be stored into separate buckets. However, the imperfection of these functions is why these collisions occur within the table causing multiple items to be stored within the same bucket. An example of this can be seen in Figure 4.9. The simpler the type of key that is used the easy it is to avoid collisions.

An analogy for the Hash Table that can make it easier to be able to visualise how it works mentally is that of a filing cabinet. Think of the draws in the cabinet as the buckets in the hash table, each of these draws can contain numerous files within it, however, the more files that a draw contains, the harder it is to find the specific file for which is being searched. The system for deciding which draw the files go in, be that numerical or alphabetical is analogous for the hashing function. The Hash Set is very similar only in their case the container cannot have duplicate keys which are excellent for the use case here as this is not wanted in the first place. In a hash set, the value of the element is its key instead of it being a key, value pair system.

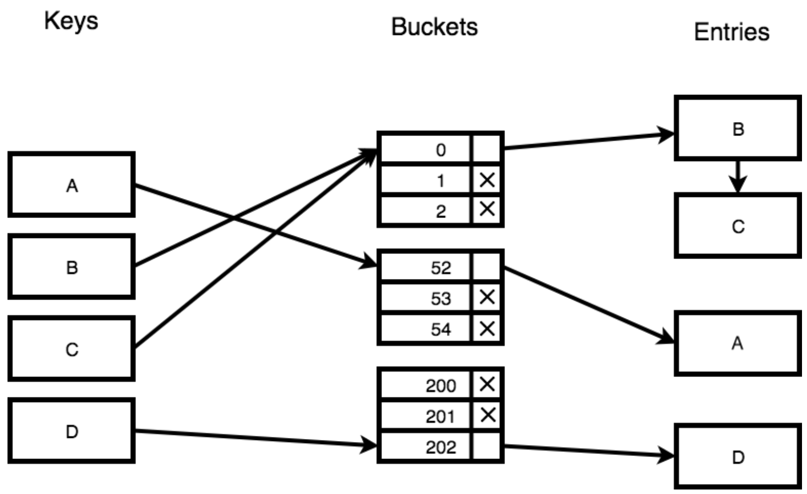


Figure 4.9: Visualization of the abstract structure of a hash table, showing how the key is placed in a bucket and then into a linked list within the bucket. This shows what happens if the hashing algorithm cases a collision and multiple items end up stored in the same bucket this can be seen with items B and C.

To implement these structures in the models C++'s standard template library was used. For the Hashtable the unordered map has been used in for models such as the Eden-A and IP models when the state of the cells exist in three states instead of two those three states being infected, uninfected, and neighbour, as it works through the storage of a key pair value in the case $< int, bool >$, the integer is used in storing the index of the cell, and the boolean is used to tell if the cell is a neighbour or infected if an index is not found within the structure then it is considered to be in the non-infected non-neighbouring state. This means that neighbouring cells can be ignored when adding to the growth site to keep the correct probably in the growth sites structure.

The second type that has been used is that of an unordered set though this is used for the Hash Set. The set is used for models such as the EdenB where the neighbouring cells are being added to the growth sites structure multiple times, or in cases like the Eden-C where the neighbouring cells are not added to the growth sites list instead the edge cells to infected cells are, this means that there would not be multiple additions of the same value because a cell can only be infected a single a time.

The selection of the hash function can have a massive impact on the efficiency of the structure. There are a wide variety of hash functions that have been thought

```
struct KeyHash{
  unsigned long operator()(const int& key) const {
    return reinterpret_cast<unsigned long>(key) % TABLE_SIZE;
  }
};
```

Figure 4.10: An example of a possible hash function that can be used when the key in that table or set is in the form of an integer as in the case that is discussed in this work.

up especially in the field of cryptography, but the functions used by Hashtables and Hashsets are very different to these. The main reason for this is that the speed of the function is critical and even in the case of some of the fastest cryptographic hashing function these are very sluggish in comparison to the types used in hash tables the reason for this is that the security of the data is not the focus.

In the case of the unordered map and set in C++'s standard template library the hash function is decided on based on the type of key that is used. These functions are different because as in the case of the key if it is a string, this would have to be converted to a numerical value so that it can be used as the index. However, in this case, the key value that is being used is already a numerical value with it being the k-index of the cell in question, and as such, the hash function can be, and the index value itself can be used without much if any processing. For example in the case of a $< string, string >$ unordered map the default hash function turns the string *"Lee Odiam"* into the integer key value of $5,280,821,831,415,455,054$, whereas in the case of an $< int, bool >$ the hash function only performs a modulo operation on the key passed to it an example of the type of hash function that can be used to achieve this can be seen in Figure 4.10.

A critical statistic with a hash table is that of the load factor the equation for this can be seen in Equation 4.2. The load factor for a hash table can inform as to the speed of the hash table, if the load factor is over one it means that there are buckets contained within the table that contain multiple values and below that as the load factor tends to zero the more empty buckets that are contained within the hash table. It is ideal to use a hashing function that keeps the load factor as close to 1 without

$$L_f = \frac{E}{B} \tag{4.2}$$

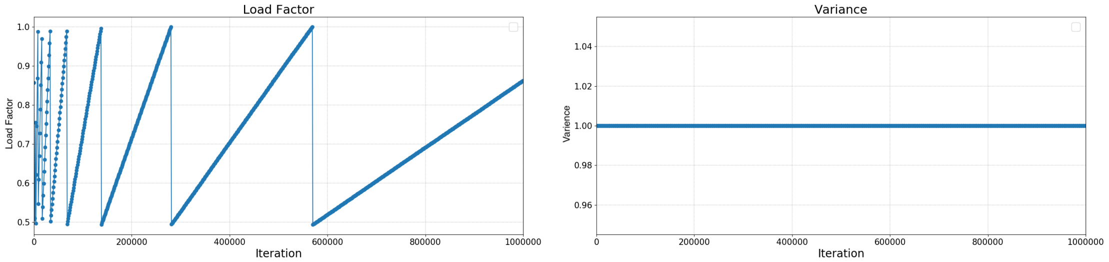
where:
L_f is the load factor
E the number of entries
B the number of buckets

$$V = \frac{B_f}{E} \tag{4.3}$$

where:
V is the variance
B_f is the number of filled buckets in the hash table
E is the number of entries in the hash table

ever going over it as possible, as when the load factor is over one this means that it is certain that more the one entry is in a single bucket at some point in the table and if the load factor is very low this means that the table is consuming a lot of unnecessary data. One issue with the use of the load factor is that if it has a value of less then one this might lead you to think that the hashing function is perfect and each filled bucket has only one item in it, but this is not that case this is where the variance of the hashtable comes into play.

Variance helps give a better understanding of the structure of the table as it can identify situations such as where the table has 100 entries within it as well as 100 buckets this would mean a load factor of one which would give the impression that it is perfectly balanced, but it is possible that all of those entries are contained within a single bucket meaning that the hash table is now just a linked list and would function incredibly slowly. In this case, it would tell you that the hashing function that is being used is not fit for purpose. Variance is calculated in a very similar way to the load factor, and this can be seen in Equation 4.3. In this case, a variance of one is ideal as this would mean that every entry is in its own bucket and as such this would offer the best possible search and insertion time. A variance of less than one says that collisions are going on within the structure. If the variance of the structure is very low, it infers that



(a) Load Factor

(b) Variance

Figure 4.11: Graphs showing the load factor and variance for the Invasion percolation model as it is run for one million iterations. The load factor graph tells that each time the hash table is resized the size that it is resized by increases

the hashing function currently in use is not ideal for the situation. Though it is not always possible for the variance to be 1.0, this is however ideal.

A hash table can be thought of as having a $O(1)$ insertion and search time, but because it is needed to run a hashing function, this can still be slower the insertion and time of an array. This $O(1)$ search and insertion time can quickly change if the hashing function that is used starts to assign the same index to different keys which is possible this can be seen in Figure 4.9 where B and C are assigned to the same bucket, this is often done with a linked list, in this case, all the entries must be iterated through in order to find if the searched for entries exists or to add the new entry in the needed position. It is easy to avoid this when the data that will be entered or the range of data that will be entered is known as with the case being looked at here. In this case, it is possible to make use of what is known about the data to allow for a perfect hashing function.

In order to test whether the default hash function used by the unordered map and set was perfect, it was essential to analyse the load factor and variance of the structures when being used for this method. The results of the tests can be seen in Figure 4.11. The Load factor of the table and set fluctuated when the structure approached a load factor of around 0.99 the structure would be resized reducing its load factor to around 0.5. The highest the load factor ever reached was 0.99915, and the lowest the load factor ever reached was 0.492581 this shows that the number of empty cells within the structure is kept in balance with the time it would take to resize the structure. The Variance for these implementations is a critical graph, and it can be seen in Figure 4.11 it stayed perfectly flat meaning that each cell within the structure will only ever contain one value. This data was gathered from 100 different runs of different models, and the results had a standard deviation of 0 showing that this was the case for all the models with which the structure is used.

4.5 Domain Resizing

One way in which to deal with the additional memory usage caused from having empty cells when using of the Lattice structure is to make use of Domain Resizing; this means that the structure would only start off small and would increase in size when the cluster reaches the edge of one of the boundaries. This would mean that in the case of a model such as the IP model with the Meakin method used for the calculation of the percolation threshold when the beta value become more lower and the cluster stops coming close to the edge of the domain a large amount of space could be saved. This section will discuss the two methods that have been developed for the domain resizing, symmetrical and asymmetrical, as well as the type of data structure that has been decided on for this and why it was selected.

The first of these two methods that have been developed is that of the symmetrical domain resizing this is the method that is used for all of the Eden model variants (A, B, C, and Screened). In this method when the cluster within the domain touches one of the edges each of the sides of the domain have the same number of cells added to them making the domain grow symmetrically; this means that the domain keeps its cube shape. In order to achieve this the L sizes that describe the length of each of the x, y, and z dimensions of the domain are doubled or in other words $new_L = old_L \times 2$ the structure that contained the cluster is the resized to this new size.

After the resizing of the structure is done it is necessary to recentre the cluster by repositioning all the cells within the domain to their new positions in order to avoid over rewriting an already infected cell with an uninfected value and destroying the cluster that has been formed it is best to start at the end of the structure and decrement the index value for each cell until the beginning of the structure is reached. Because all the new instantiated cells are initialised with a zero value, it is only needed that the infected cells go through this repositioning process and are then set to a zero value. The code that is used to find the new position for a cell within the domain can be seen in Figure 4.14. In this method the first step that must be performed it to obtain the 3D coordinate of the cell within the domain before it had been resized; this is why the $initial_L$ value is passed through to the fromK method show in Figure 4.14. After this, an offset value must be calculated this is easy to obtain in the case of the symmetrical resizing as it is merely $\frac{initial_L}{2}$

The second type of resizing used is that of asymmetrical meaning that the resulting domain does not have to be a perfect cube and instead would be described as a cuboid. This method is the one used in the case of the IP model for all of its variants, Random, Hawick, and Meakin.

Unlike with the Symmetrical domain growth the calculation of the change to each side it a more complicated process. The first step is to identify which side of the domain the cell is touching; this is then placed into an offset array which can be seen in Figure 4.12. In this code depending on which side of the domain is being touched the offset can be set to a negative or positive number the reason for this so that the direction of the domain that needs to be altered can be easily identified; this aspect of the code makes more sense when the positioning code that is seen in Figure 4.15 is taken into account. Though in the case of the asymmetric resizing it must be ensured that the domain does not grow too far in one direction as this can affect the clusters grown, that is the purpose of the increase and max_increase arrays seen which make sure that the domain is limited in each direction. This limit is not needed for the bottom of the domain as the stopping condition of the model will make sure that

```
bool IP_RES::touch_edge(const int x, const int y, const int z){
  // Reset resizer so that the return function will work
  for (int i = 0; i < 3; ++i) {
    resizer[i] = 0;
  }

  // Touching the left or right
  if (increases[LEFT] <= max_increases[LEFT] && x >= L_x - 1){
    resizer[0] = increases[LEFT];
    increases[LEFT] *= 2;
  }
  else if (increases[RIGHT] <= max_increases[RIGHT] && x <= 0){
    resizer[0] = -increases[RIGHT];
    increases[RIGHT] *= 2;
  }

  // Touching the bottom - stoping condition takes care of
  // domain over resizing -> (L_y == max_L && y == max_L - 1)
  if (y >= L_y - 1){
    resizer[1] = increases[BOTTOM];
    increases[BOTTOM] *= 2;
  }
  // No touching top as IP shouldnt grow up in this case

  // Touching front or back
  if (increases[FRONT] <= max_increases[FRONT] && z >= L_z - 1){
    resizer[2] = increases[FRONT];
    increases[FRONT] += increases[FRONT];
  }
  else if (increases[BACK] <= max_increases[BACK] && z <= 0){
    resizer[2] = -increases[BACK];
    increases[BACK] *= 2;
  }

  // returns true if the domain needs resizing
  return(resizer[0] != 0 || resizer[1] != 0 || resizer[2] != 0);
}
```

Figure 4.12: Offset calculation for asymmetrical domain resizing. The negative values are used to indicate the diferect of growth of the domain this is then combined with the code in Figure 4.15 to reposition the cell with the newly sized domain. The negative numbers are used to tell which side of the domain needs to be adjusted. This code only needs to be run once per resize. The increase size, and maximum increase values are calculated in Figure 4.13.

```cpp
void IP_RES::set_resize_limit(int dim){
  int d = (dim / 2) - resize_val;
  int counter = 0, val = resize_val;

  while (d > 0){
    d -= val;
    val += val;
    counter += resize_val;
  }

  for (int i = 0; i < 6; ++i){
    increases[i] = resize_val;
    max_increases[i] = counter;
  }

  /// This is because the IP models domain does not grow up
  increases[BOTTOM] *= 2;
}
```

Figure 4.13: This calculates the values for the increase and maximum increase array to stop the domain becoming lopsided and growing too much in one direction as this would mean that the seed location would no longer be in the centre of the domain top. This code is designed in such a way that the minimum domain size if 16 and that all domain must be a power of two which is how these models will be tested.

```cpp
void EdenA_RES::update_kindex(int &i, const int initial_L){
  vec3 pos = fromK(i, initial_L);
  pos.v[0] += offset;
  pos.v[1] += offset;
  pos.v[2] += offset;
  i = flatten(pos.v[0], pos.v[1], pos.v[2]);
}
```

Figure 4.14: Code to calculate the new position in the domain for an infected cell after the domain has been resized. In this case, the offset will be equal to half the resize value this is to keep the cluster centred as both sided have grown by half the amount of the resize value.

```cpp
int IP_RES::get_newk(const int index){
  vec3 pos = fromK(index, initial_L);
  pos.v[0] += (resizer[0] < 0 ? -resizer[0] : 0);
  pos.v[1] += (resizer[1] < 0 ? -resizer[1] : 0);
  pos.v[2] += (resizer[2] < 0 ? -resizer[2] : 0);
  return flatten(pos[0], pos[1], pos[2], dims);
}
```

Figure 4.15: This code is used in the case of the asymmetrical domain resizing to obtain the new k index for the cell after the domain has been resized it works together with the code shown in Figure 4.12. This code will be run for each of the cells within the domain that are not set to zero/empty.

it cannot overgrow in this direction.

The final step in the resizing of the domain process for both the symmetrical and asymmetrical method is to reposition all the cells that are within the growth sites list, as they currently store the indexes that they would have had in the smaller domain. It is a merely a matter of iterating through the list and applying the same functions that are used to reposition the cells that are within the domain. Other than the resizing process used the different resizing versions will act identically to the standard Lattice model in how it stores the state of the cells within the domain.

The resizing method is only needed for the Lattice version of the model as the other data structure are inherently capable of handling the resizing by just setting them to have a massive domain size from the start, even if this is not the full size of the domain being used. The only limiting factor would be the memory of the computer, as through the use different variables for the storage of the state such as a *unsigned long long* it would be possible to store a maximum index of $18,446,744,073,709,551,615$.

4.6 Summary

This chapter has detailed the different data structures that will be tested. It has discussed some of the common uses for the structures such as how binary search trees are commonly used in scheduling within a computer or even in some games for a method to reduce the amount of work done in rendering a 3D world. It has also discussed the algorithms that will be used and how they have been designed to take advantage of the knowledge of the specific domain, such as the hashing algorithm that is used in the hash table and set. The final part of this chapter discussed the implementation for the domain resizing method that is used discussing both the symmetrical method and the asymmetrical method for this along with in which model each of the different methods will be used.

In the following chapter, Chapter 5 the results from the experiments that have been run will be discussed. The Chapter will go into the effect that the different structures have on the model and will aim to show points at which specific structures can offer benefits through a comparison of the memory consumption and the runtime of the models.

It is a capital mistake to theorize
before one has data.

Arthur Conan Doyle

CHAPTER 5

Data Structure Results

BEFORE it can be stated whether a newly developed method is superior or inferior to a previous method, whether that be regarding the runtime of the method or in the memory consumption of this method it is essential to perform much testing on the method. This involves gathering data from multiple runs of the system to be able to identify general trends in the effect of the new method; this is especially important with models such as those that are used here which have a high level of randomness which can have a significant impact on the runtime of the system. This chapter will demonstrate that in certain situations there is a benefit in the application of alternate data structures to the standard Lattice method that is most commonly employed for the simulation of growth model clusters.

With compilers becoming more and more intelligent such that they are now able to attempt to minimise the computation time by taking advantage of language or processor specific features and tweaking the code that has been passed into them. As

Node Name	CPU	RAM	MPI Version
Computing Node	2 x 14 Broadwell E5-2680v4 (2.4–3.3 GHz)	128 GB DDR4	(OpenRTE) 1.8.8

Table 5.1: The specification for the nodes on the University of Hull's Viper HPC in the gather of the data discussed within this chapter.

some of these features are highly processor dependent, the results that have been gathered and that will be discussed in this chapter have been done with all compiler optimisations turned off; this has been done in order to give a level a playing field as possible making the results more generalisable allowing for the results to be as broadly applicable as possible hopefully.

In order to obtain the results shown in this chapter in a decent time frame, a high-performance computer was explicitly utilised, the University of Hull's Viper HPC. This system has been used in conjunction with OpenMPI, which is an open source message passing interface that allows for these experiments to be run in parallel in an easy to develop and efficient manner. The system distributes instances of the program to different CPU's, with multiple slave processes running the experiments and the master process collecting and outputting all the data. The specification of the HPC node that was used can be seen in Table 5.1. One issue did arise with an update to the VIPER system the effect of this limited the range of L sizes that could be tested.

There will be some shorthand made use of for the representation of the different data structures. LAT will be used to represent the array Lattice structure, AVL will represent the AVL tree, HSH[1] will be used for the Hashtable and the hash set the models where each of these will be used is discussed in Section 4.4, OCT[2] will be used for the octree, and finally RES will be used for the resizing domain system discussed in Section 4.5 and will be used for both the symmetrical and asymmetrical.

In Section 3.4 the fill density of the various models at increasing domain sizes has discussed. This aspect of the models will be key in the understanding of the results

[1] Hash Table

[2] Octree

discussed here. The aim of this chapter is to investigate two points whether or not for a specific growth model as the size of the domain increases does the type of data structure that is best suited for the storage of the cluster in terms of memory consumption and/or run time also change. Also to investigate whether or not the Lattice method for the storage of the cluster is the best method to use in the storage of these sparse clusters

The reason for the use of the different models is due to two reason the first of these is that in terms of the programming of the algorithms the only difference between the models is the handling of the growth sites and the selection of the next infected cell this different in the selection of the cells leads to different fill densities as shown in Section 3.4.

These more modern data structures discussed in the previous chapter such as the AVL have the benefit of only having to store the cells that make up the cluster itself and not all of the empty cells in the domain. This theoretically offers the possibility for a significant reduction in the maximum amount of memory that the simulation consumes. Additionally, the aim is to show that as the fill percentage of the domain decreases even though these more modern data structures possibly have a slower search and insertion time the lack of need to initialize all of the cells in the domain as with the Lattice model means that they can still offer significant benefits in both memory consumption as well as run time.

There are three main expected outcomes from this chapter these are as follows:

E1 It is expected that for models with a sparse domain fill such as the MIPs model the LAT structure will become more and more of a hindrance to the run time and memory consumption of the model whereas with mode the end up filling more of the domain such as the HIP model the LAT remains the best choice.

E2 It is also expected that for models such as the HIPs model where the percentage of the domain that is filled by the cluster it very high that the LAT structure will remain to be the best structure to use for the storage of the cluster whilst simulating its growth.

E3 It is expected that the beneficial effect of the alternative modern data structure should become increasingly pronounced especially for models such as the MIP model where, as the size of the domain increases the percentage of the domain that ends up infected decreases meaning an increase in the sparsity of the model increasing the beneficial effect of the more modern sparse data structures

These expectations will be refered to by their key (**E1**, **E2**, **E3**) in the summary of this chapter.

The structure of this chapter will be start off the with a section discussing the effects on the total run time of the cluster that the data structure has; this section will begin with the construction time for each of the data structure and will subsequently be followed by a discussion of the overall runtime of the model in relation to the size of the domain. The second part of the chapter will focus on the memory consumption of the different data structures for the models and will be a compared to the timing results to see where any possible speed increases offered by any of the structures also give memory consumptions or come at the cost of using additional memory.

5.1 Timing Data

In this section, the timing results for the various data structures discussed in the previous chapter for all of the different models that have been discussed in Chapter 3 will be detailed. This section will discuss two main aspects involved in the runtime of these models the first of these things that will be looked at is the construction time of the individual structures for a range of domain sizes. The second aspect will be the complete runtime of the models with the different data structures for the same range

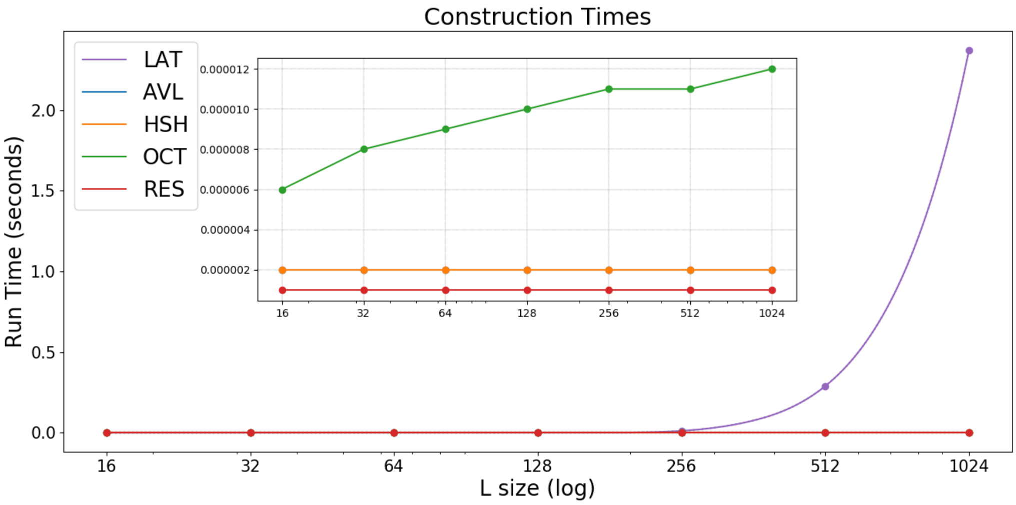


Figure 5.1: This graph shows the median of the construction time for the different data structures each of the pointers are gathered from one thousand runs of the construction. The line representing the construction time for the AVL tree lies underneath the orange line for the Hash Table and due to the draw order is not visible. The Hashtable and hash set are both represented with a single line as they both had the same construction time. The floating box in the centre of each of the graph shows a zoomed in area sharing the same units as the main graph for both the x and y axis

of domain sizes as in the construction time. In this work the domain sizes that will be looked at will all be $(L = 16)$, $(L = 32)$, $(L = 64)$, $(L = 128)$, $(L = 256)$, $(L = 512)$, and finally $(L = 1024)$ and all of the data will be gathered from 100 runs of the simulation. It should be noted that there will be cases that due to the time it took to run the models or the memory consumption of these larger domain sizes being so high it became infeasible to run the experiments and some points may be predicted values. All predicted values that are used will be represented by a dashed line in all graphs instead of a solid line in order to avoid any confusion about computed and predicted values. These predicted values will be obtained through the use of the *interp1d* function that is built into SciPy and is an interpolation function used to perform a curve of best fit.

5.1.1 Construction Time

The construction times for the various data structures can be a critical part of the total runtime of a model especially as methods used get faster, and it can become a limiting factor. In some cases, it is possible for this part of the program to be the most time-consuming aspect of the whole model. This section will discuss in detail the construction time for the different data structures. A set of simple experiments where run where the specific data structure was constructed 1,000 times in order to gain an accurate measure of the range of the time it takes to construct each of the data structures, in the case of the LAT structure this also requires an iteration over the structure to zero out all the cells contained within it to ensure that there is no conflicting data. The medians for each of the structures across a range of L sizes can be seen in Figure 5.1.

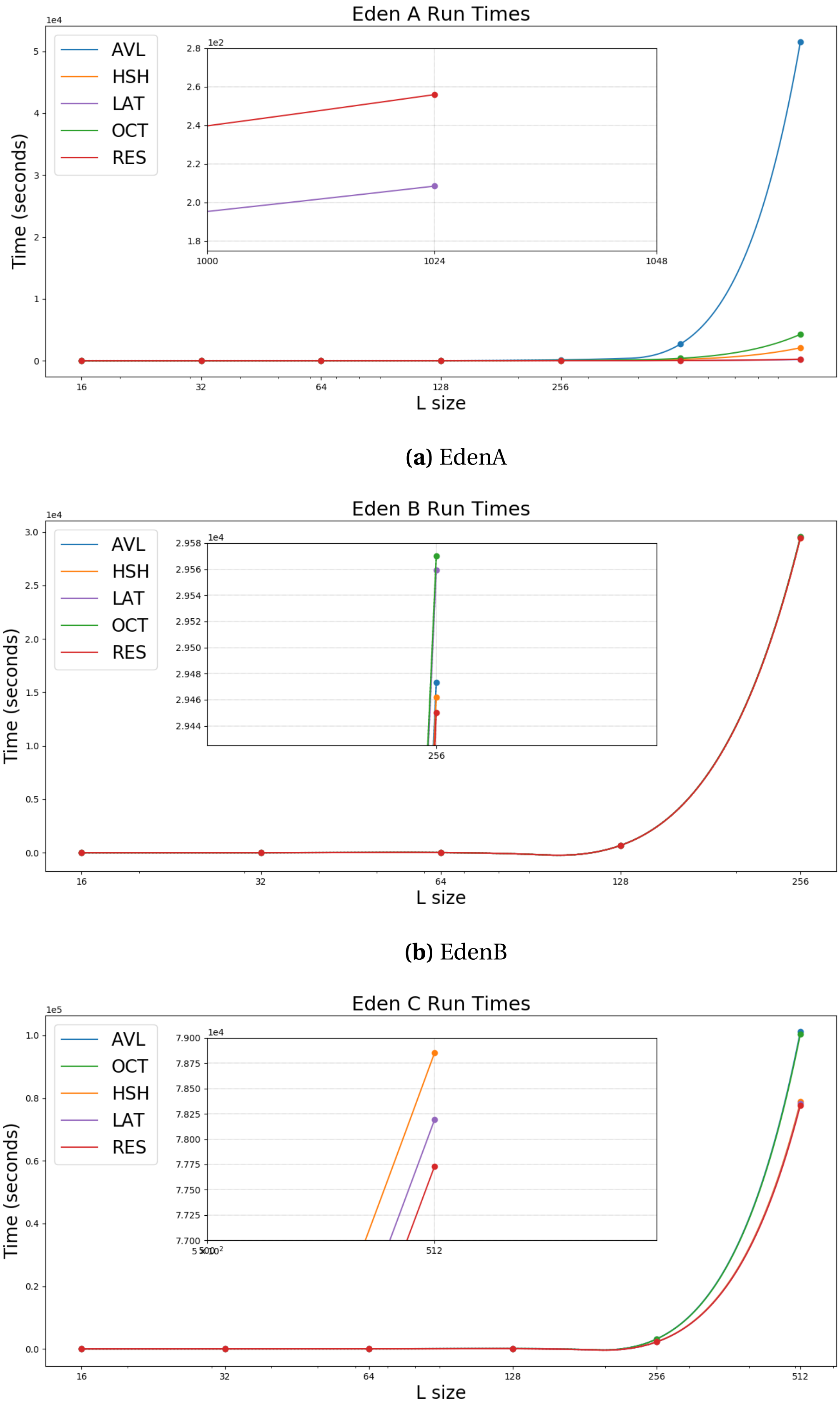
As the size of the domain increases, it is evident that the size of the domain has the most significant impact on the LAT model out of all the structures. This makes perfect sense as when the LAT model has initialised it is necessary to mark out an array vast enough to contain all the cells and also zero out each of these cells in order to ensure that whatever memory is contained within doesn't affect the growth of the cluster; this gives the construction of the LAT method a computational complexity of $O(N)$. The LAT structure differs from the others in that they only have to deal with a much smaller amount of memory, in the beginning, this is due to that way that the structures grow as the simulation runs. There is only one of the other structures that are also affected by the size of the domain, the OCT structure, this is not because it is marking out additional memory as with the LAT but is instead due to that calculation of the max depth for the tree, the method used for this is discussed in Chapter 4.

The AVL tree and the Resizing Domains methods both have the same median construction time, however they do have completely different distribution populations which can be demonstrated through the use of a number of different statistical test the Mann-Whitney U (Mann and Whitney, 1946) test the Kolmogorov–Smirnov test (Dodge, 2008) and the Vargha-Delane effect size (Vargha and Delaney, 2000) measure or A measure the results of these test can be seen in Appendix B in Tables B.3 along with a comparison for all the other structures and an explanation of the algorithms can be found in Appendix A. In this case, they both have the same minimum construction time of $1\mu s$ this could be due to a limitation in the timing library that was used (Standard Template Library chrono). The difference between these two is that the AVL tree has a maximum of $3.6\mu s$, and the Resizing has a maximum of $4.6\mu s$. When this is taken into account along with the statistical tests to show that the distributions of the two are from different populations it is possible to assume that the AVL tree is the better of the two in terms of construction time, this is also an excellent example of how the medians of a distribution is not the be all and end all in terms of analysing the difference between two distributions.

5.1.2 Standard Eden Growth Model

The focus in this section will be on the three main versions of the Eden growth model the EdenA, EdenB, and EdenC models, and they will be discussed in that same order. The Screened Eden model has been excluded from this section and will be discussed within its own section due to the number of different variables within the system and the vastly different results that the model can exhibit and possible exciting behaviour depending on the β value and the structure used.

The runtimes for the five different data structure methods tested for the EdenA model can be seen in Figure 5.2a, these results show shows that the LAT version of the system is the fastest of the five different structures; this is true for all the tested L sizes. The slowest data structure that was tested by a large margin was the AVL tree with it being nearly 250 times slower than the LAT structure. The LAT model has a median runtime of about 208 seconds for when the domain reaches an L size of 1024 with a median construction time of around 2 seconds at this size for this structure this shows that most of the runtime for this model is spent in the actual simulation of the



(a) EdenA

(b) EdenB

(c) EdenC

Figure 5.2: Graphs showing the run times for the three main types of Eden model the EdenA, EdenB, and EdenC model. This model is inclusive of the construction time of the data structure. The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

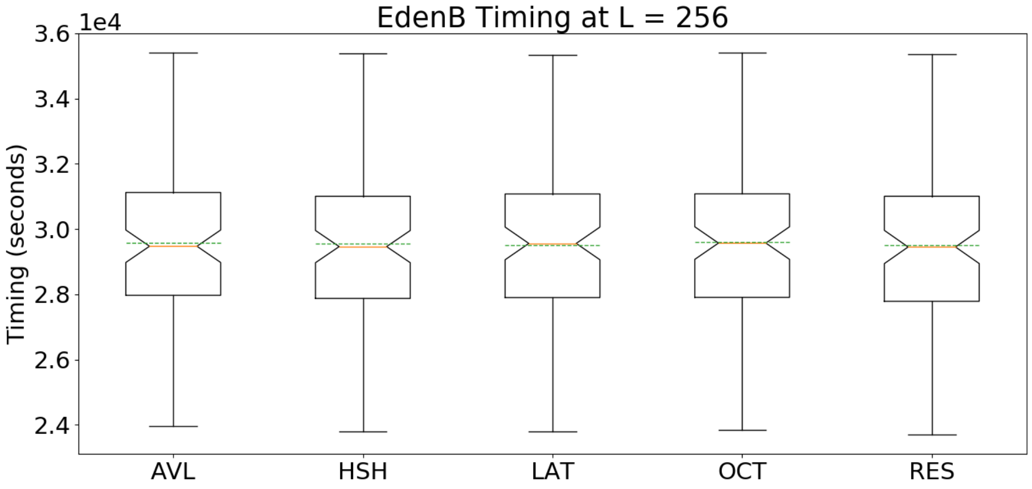


Figure 5.3: EdenB L=256 Runtime box plots. These show that there may not be any difference between each of the 5 different methods for the EdenB model.

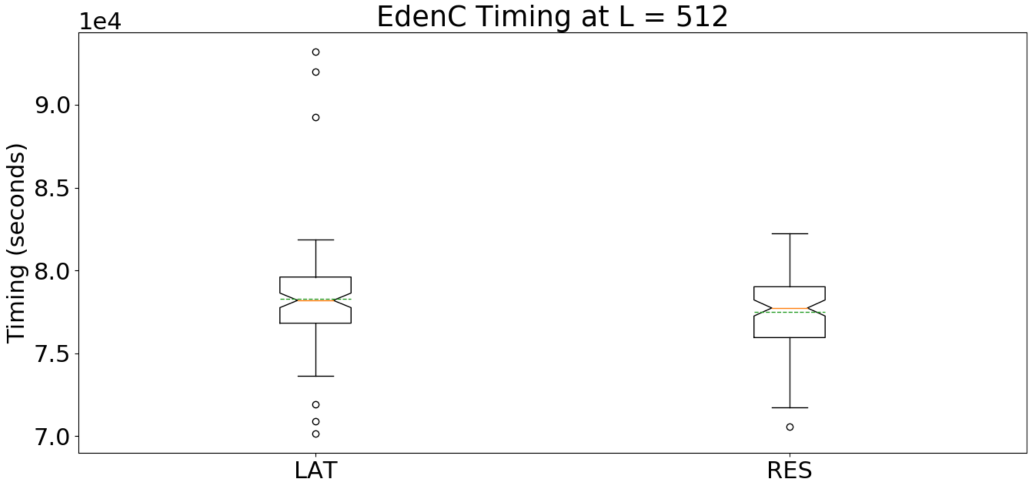


Figure 5.4: EdenC L=512 Runtime boxplots for the LAT and RES methods. This shows that there is significant overlap between these two methods though it is apparent that benefits in the total runtime can be obtained through the use of the RES method over the LAT method.

cluster with the construction time taking less than 1% of the total runtime.

However, the median run time for the domain resizing method is very close to that of the standard LAT with a runtime of approximately 255 seconds though there is very minimal overlap between the two structures with only extreme values in the LAT reaching high enough to match the resizing domain method and the results of statically test which can be seen in Tables B.4 and B.8 shows that the difference between the Resizing at the LAT methods is statistically significant. This difference in speed means if the aim of a run was to grow a cluster of a specific size instead of calculating the specific size of the domain that would be needed the resizing method could be used to ensure that the cluster is not affected by the edge of the domain while only causing minimal effect on the runtime of the model.

The second of the three Eden model that will be discussed is the EdenB model. The graph showing the runtime of this model can be seen in Figure 5.2b. The EdenB model is an interesting example of the use of different data structures for the simulation of growth models. When looking at Figure 5.2b it would be easy to conclude that the Resizing method is the best, but this is not the case. As can be seen in the non-zoomed-in section of the graph these run times are very similar to one another with it being difficult to identify the difference between them only by zooming into the graph the difference can be easily seen. The big question is whether these differences in the medians is a significant difference or not.

In order to be able to answer this question, a few things will have to be looked at. The first of these is the boxplot of the different data structure run times after this two different statistical tools will be put to use the first is the Mann-Whitney U test which is a non-parametric test that can be used to calculate the chance that two samples are drawn from the same populations with the same median the results for this can be seen in Table B.8 in Appendix B. The second of the methods that will be used is the A measure which is a measure of the scientific significance what this allows for the testing of whether the difference between two distributions with similar medians is significant. The results for this can be seen in Table B.7 in Appendix B.

When the box plots that are seen in Figure 5.3 are taken into account, a different story from what the graph tells becomes evident. Whereas there is a slight difference in the medians of the different data structures runtimes when the full range of the data is taken into account this begins to look less significant and with the results of the statistical analysis which is seen in Tables B.7 and B.8 becomes highly apparent that it is not possible to state that a statistically significant difference exists between any of the various the data structures in the case of the EdenB model. This means that in the case of the EdenB model it does not matter which of the different structures are used when looking at the at the EdenB model in terms of the runtime of the model.

The third and final version of the standard Eden growth model the EdenC will now be discussed the result of the runtime experiments can be seen in Figure 5.2c. The EdenC is more like the EdenA model than it is the EdenB model and it is possible to draw a more specific conclusion from the data that was obtained than in the case of the EdenB model. In this case, the RES method is the fastest of the five different methods that have been tested; this is closely followed by the LAT model with a difference of 7.67 minutes for the median run times for an L size of 512. Even though this result may seem like a good result when the full range of the data is taken into account and statistically analysed it is shown that the difference between these to methods is only small, this can be seen in Table B.10. What this means is that whereas the RES method can run in a shorter amount of time with a lower median when the results are plotted on a box and whisker system which can be seen in Figure 5.4, the high whisker of the RES model does reach slightly passed that of the LAT model.

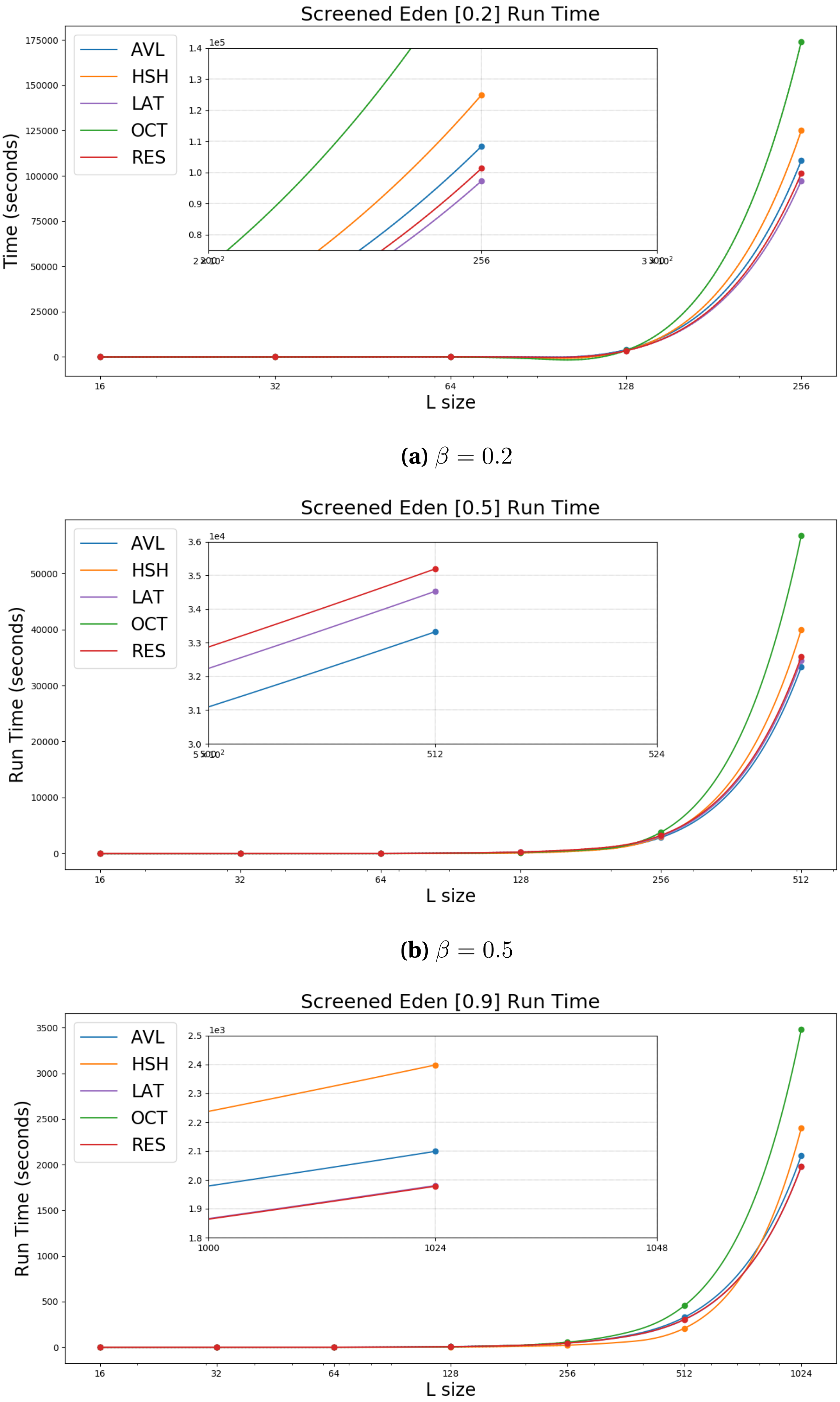
This doesn't mean that the RES method is pointless even though the RES method does only offer a small positive effect over the LAT version there is a positive effect meaning that this method is a better method and will save time even if only a small amount, when running the simulation of the model for the number of times that will be needed to get a understand of the behaviour of the model in the use case even small savings in timing can eventually add to be a significant saving. For example, with this work, the model was run 100 times if these methods where run serially it would have taken a total of 89.72 days to run with the LAT model whereas with the RES method it would have taken only 88.7 days. Though it is true that it is unlikely that these type of models would be run serially, it is possible, this does go to show that the RES method does offer a significant speed-up over the LAT method over many runs and this saving will only increase the more times it is run.

5.1.3 Screened Eden Growth Model

The Screened Eden growth model is the most interesting of the Eden models that are discussed here due to the possible behaviours that it is capable of displaying through the tweaking of the beta value. This is the first of the model to be discussed that will make use of a range of beta values in order to be able to give a good understanding of the effect that this variable has on the runtime of the model and to see if different setting have different optimal data structures, these values are $\beta = 0.2$, $\beta = 0.5$, and $\beta = 0.9$ (defined in Chapter 3).

With Screened Eden model the higher the beta value used, the smaller the cluster that can fit in the domain as it increases the clusters bias to grow along the radius of gyration. Unfortunately due to an issue with the experiments timing out due to a time limit that VIPER imposes on all jobs it was not possible to get a full range of data for the 0.1 version of the Screened Eden growth model at the L size of 256 and so instead the 0.2 was used instead.

The runtimes for the Screened Eden model with a beta value of 0.2 can be seen in Figure 5.5a this cluster is the one that is the closest to that of the EdenA model out of all the different beta values that are discussed. For this model, the order from fastest to slowest of the data structures somewhat reflect that of the EdenA model, with the LAT and RES methods being the two fastest with very minimal significance between



(a) $\beta = 0.2$

(b) $\beta = 0.5$

(c) $\beta = 0.9$

Figure 5.5: The run time for the Screened Eden model with a range of β values these being 0.2, 0.5, and 0.9. These can be seen in this order from top to bottom respectively. The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

the two distributions meaning there is a high chance that they come from the same population. The HSH method is next in the lineup, and this structure only has a small level of significance when compared to the AVL tree, all of the statistical tests showing the significance of the result compared to each other can be seen in Tables B.13, B.14, and C.12. Though the level of significance of these results is low, it was only possible to get a result for up to an L size of 256, and it is like that the spread of these results will increase with high L sizes making the data structure decision more and more critical.

The runtimes for the 0.5 setting for the Screened Eden model can be seen in Figure 5.5b and this value does have a fascinating effect of the runtime of the model. At this setting, the AVL version of the model appears to be the fastest of the five different structures, followed closely by the RES and the LAT version at an L size of 512. However, it is essential to be able to tell whether this difference in the medians is actually significant or not; this is where the data shown in Tables B.16, B.17, and C.15 comes it. If this is taken into account it becomes apparent that the difference between the LAT, RES, AVL, HSH is not very significant at all, even though the AVL has the lowest of the medians; this means that is this case it will come down to the memory consumption of the different structures to help decide which of the different structure is the best suited here. As it is not possible to get the full population for these runs, it is not an accurate measure to say that one of these methods would have a benefit over the others for the larger domain sizes based on the distribution of data that is here. The slowest of the structures was the OCT method which considering the extra amount of organisation that goes into the storage of a value within the structure and the extremely long runtime that is being dealt with here with the AVL method having a median of approximately $3.3e^4$ seconds or 9.3 hours this was to be expected.

The final beta value to be discussed will be 0.9, the results of the experiments for this setup can be seen in Figure 5.5c. Unlike with the previous settings the RES and LAT methods alone are in this case clearly the fastest of the different structure, with there being no real significance between these two different methods and a high chance that they come from the same population; this can be seen in Tables B.19, B.20, and C.18, this means that in this case, it would be possible to make use of either of these methods and gain a very similar runtime for the growth of the cluster. The AVL method is now slower than the RES and LAT method, and there is a small level of significance meaning that unlike with the 0.5 it would be a slower option even if only slightly. The order of the two slowest methods is the same as before with the OCT and HSH methods being the slowest with and a small amount of significance between the AVL and the HSH method. Though runtime is not always the focus and there could be significant memory saving to be made here due to the small size of the clusters produced.

The Screened Eden model is an excellent example of a model where the beta value has a significant impact on the best structure regarding the runtime of the system. With the lower beta values, there is very little difference between the AVL, HSH, LAT, and RES methods will offer benefits in the runtime of the model, meaning that there are many choices here. However, as the beta values increase the AVL tree and the HSH tree start to become slower, then that of the LAT and RES and in the 0.9 version, they are the fastest of the structure. However, this speed could come at the cost of significant memory consumption which might not be worth the trade off in runtime.

5.1.4 Random Invasion Percolation

The Random version of the IP model will be the first one to be discussed out of the three IP models with which this thesis concerns itself. The graph showing the results of the experiments for the runtime of this model can be seen in Figure 5.6. The IPR model at an L size of 1024 has a very similar pttern to that of the EdenA model in terms of the median runtime, with the fastest structure being the LAT; this had a median runtime of 20.12 seconds at $L = 1024$. The slowest of the clustered results was the AVL with a median runtime of 170.61 seconds at $L = 1024$ which is a much less significant increase than in the EdenA model with it only being 8.5 times slower though this is

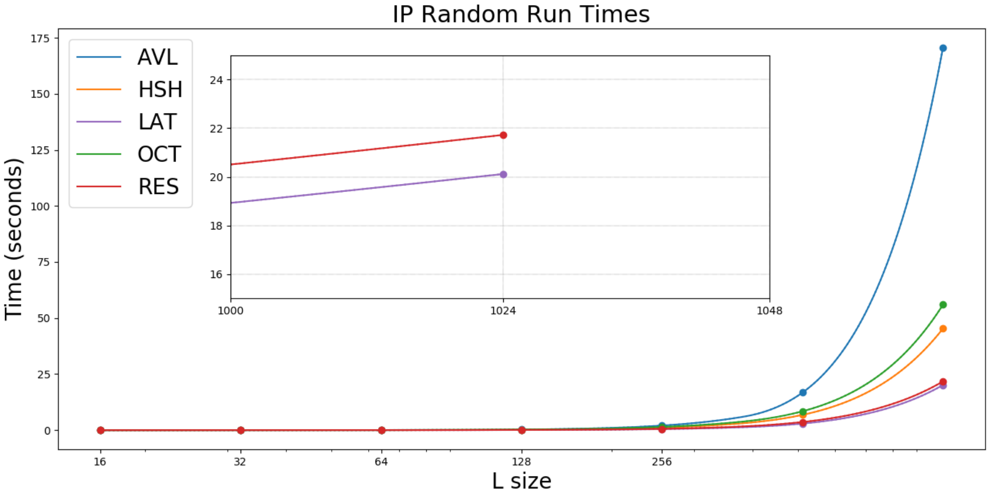
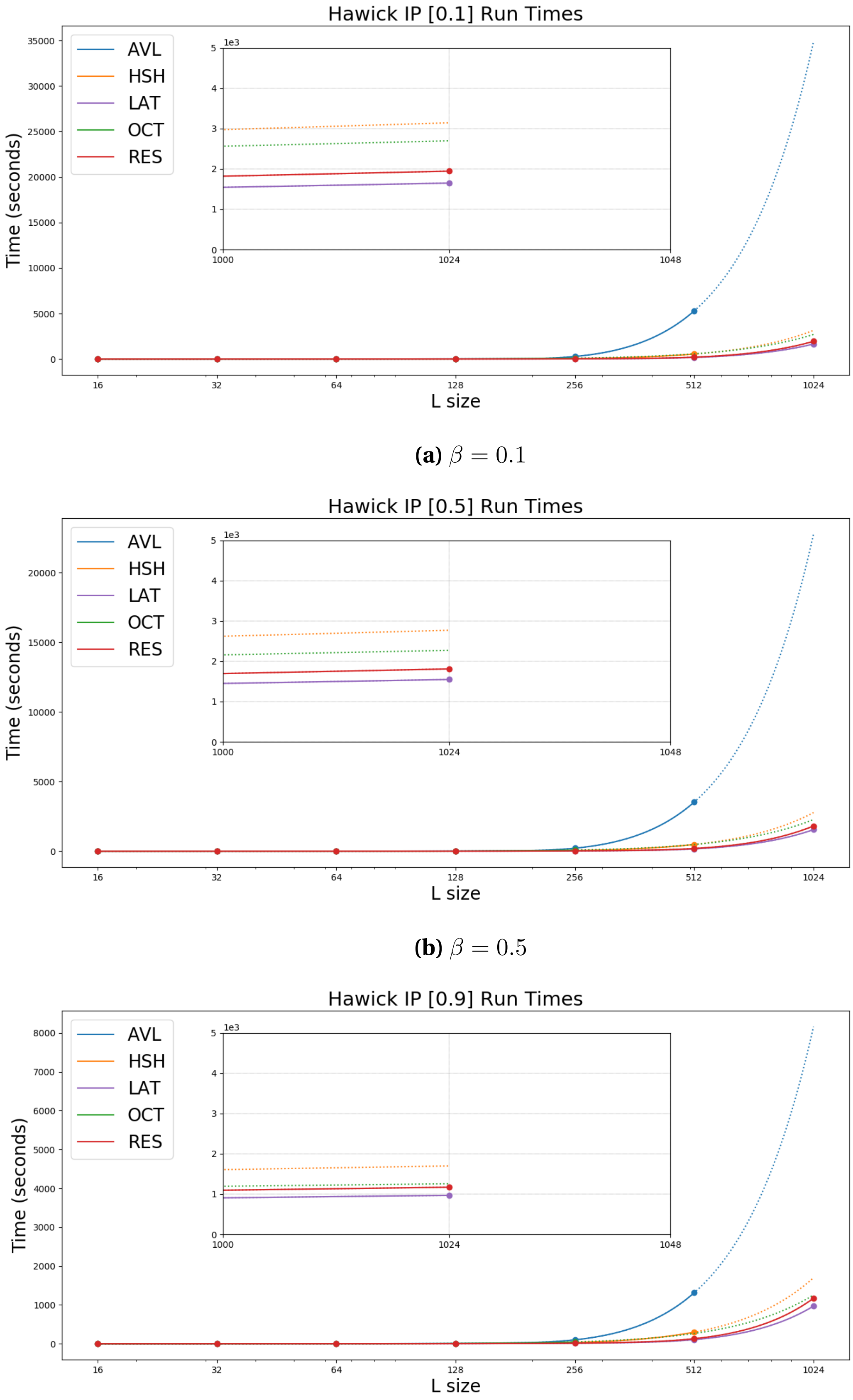


Figure 5.6: The Run times for the Random Invasion Percolation Model.The floating box in the centre of each of the graph shows a zoomed in area sharing the same units as the main graph for both the x and y axis

still not an acceptable difference. As with the EdenA model the RES method is only slightly slower than the LAT one, with a median runtime of 21.72 seconds at $L = 1024$ though when the results the statistical tests are taken into account, seen in Tables B.22 and B.23, and B.24), this difference is only with a small effect with an A-measure of between 0.524 and 0.66 for the all the different L sizes tested as such it is not possible to reject the hypothesis that the distributions of these two samples are the same. This means that it is not possible for an accurate claim whether one of these methods is better than the other, as statistically speaking with the results obtained here they are identical; this is unlike the EdenA model where it was entirely possible to identify that the LAT would be the fastest.

5.1.5 Hawick Invasion Percolation

The Hawick method is probably the least common of the methods out of all these discussed here. It was used here due to the high percentage of the domain that is filled to improve the fill percentage coverage that is shown in this work. The results for the HIP model with three different β values, 0.1, 0.5, and 0.9, can be seen in Figure 5.7. Due to the memory consumption of the AVL, OCT, and HSH, it was not possible to



(a) $\beta = 0.1$

(b) $\beta = 0.5$

(c) $\beta = 0.9$

Figure 5.7: The Run times for the Hawick Invasion Percolation Model with a β values of 0.1, 0.5, and 0.9 from top to bottom respectively.The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

obtain a full range of data. The OCT version was only able to be run up to the L size of 256 and the HSH, and AVL tree could only be run up to the 512 domain size. At the larger domain sizes, these methods would run for a long time only to crash.

This model is the only one with a variable that can be changed to tailor the behaviour that doesn't change which is the fastest of the version of the model between different values given to it. This means that with a value of 0.1 or 0.9 the LAT version of the model is the fastest of the different structures with the resizing method coming in second. However unlike in other models where the different in the runtime between the LAT and the RES method was minimal such as in the case of the EdenA model, in this case with the fastest of the models the 0.9 version there is a time difference of 3.3 minutes and in the slowest case a time difference of 4.9 minutes which when the time for the LAT version of the model is taken into account which is a 16.16 minute run time for the 0.9 is a very significant result.

The significance of this can be seen in the result of the A-measure test which can be seen in Tables B.34, B.37, and B.40. When these are taken into account, it is shown that this difference has a substantial effect between the LAT and the RES versions as well as with all the other different data structures. This means that it is easy in the case of the HIP model to state that the LAT version of the model would be the best structure to use as it is considerably faster especially compared to the AVL tree which has a median runtime of 37.2 minutes which is more than double the median of the LAT model.

This shows that in the case of models such as the HIPs model which are highly filled the optimal choice for the storage in terms of runtime would be that of the LAT structure and considering the fact that it was not possible to get timings for the larger domains for the other structures this shows that this would also be the case in terms of the memory that the model would consume, though this will be discussed in more detail in Section 5.2.

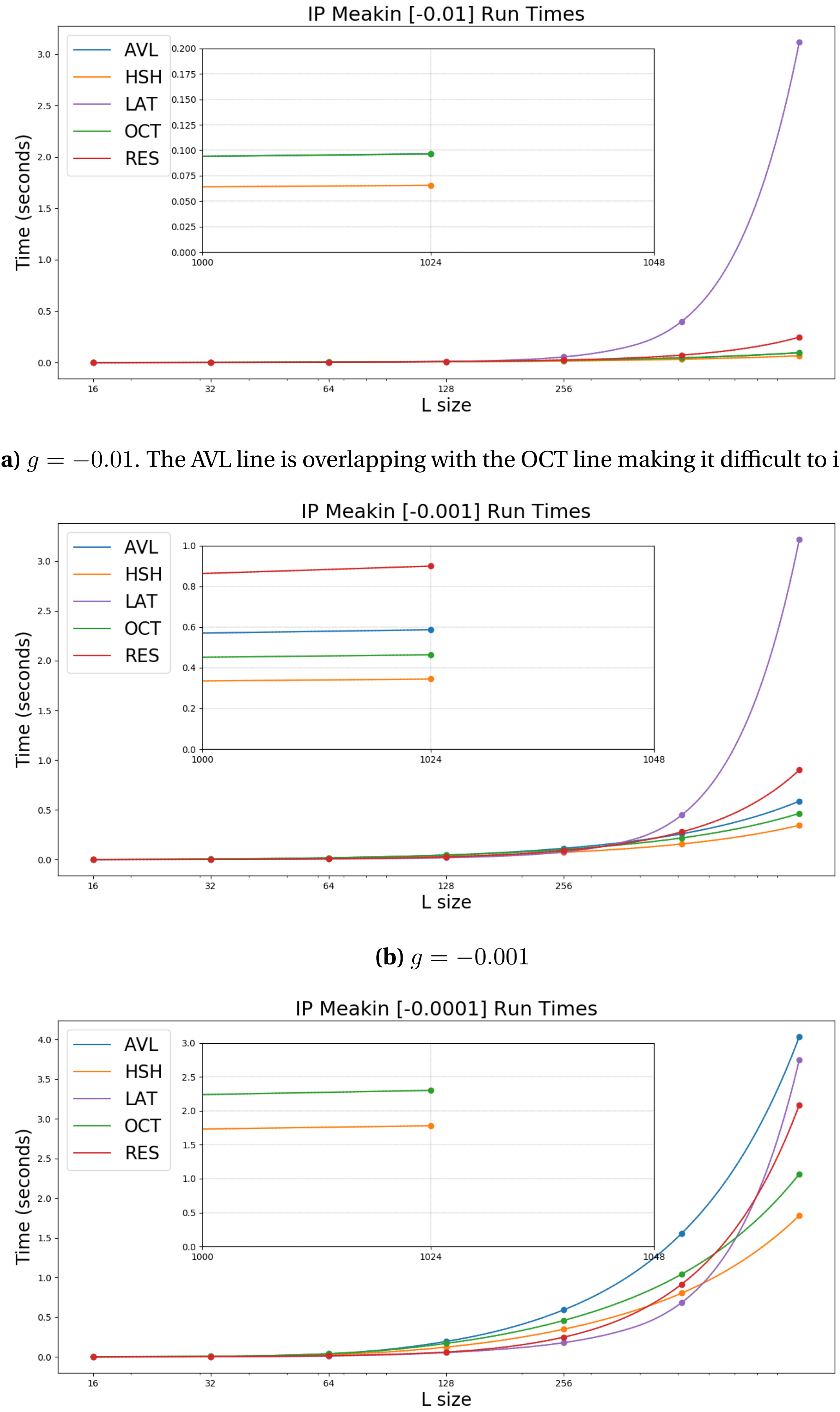
5.1.6 Meakin Invasion Percolation

The Meakin method for the calculation of the percolation threshold can give the most sparely filled domain out of all methods used and discussed for both the Eden and IP model that are detailed within this thesis. The MIP model has been tested for three different g values; these being $g = -0.01$, $g = -0.001$, and finally $g = -0.0001$, these will be discussed in this order. As shown in Section 3.4 as the gravitational coefficient used in the calculation of the percolation threshold tend to zero the fill of the domain increases at with a g value of 0 the model being identical to that of the Random IP model.

The MIP method produces some of the most interesting graphs out of all the different models that are investigated here. This is the model where the construction time of the data structure used has the most significant impact on the runtime of the model, more than that of the search and insert times. The reason for this is due to just how sparsely filled the domain is filled, with the fill density for all the gravitational coordinate tested having a fill of less than 1% when the L size reaches 1024 as seen in Section 3.4.

The $g = -0.01$ version of the MIPs model displays a behaviour that is highly significant, this being the runtime of the LAT method. Unlike with the other models that were studied in this case the LAT method took significantly longer to run with a median run time of 3.12 seconds this is 17 times slower than the next fastest method which is the RES method when the domain reaches an L size of 1024. The reason for this is that the construction time of this data structure at this domain size is 2.37 seconds, which is considerably longer the total runtime of the other methods at this L size. This leads to the conclusion that a high number of successful runs would be able to finish before the LAT method has even initialised its data structure.

In this case the fastest method of the remaining three is that of the HSH method this is followed by the OCT and AVL methods which overlap considerably on this



(a) $g = -0.01$. The AVL line is overlapping with the OCT line making it difficult to identify

(b) $g = -0.001$

(c) $g = -0.0001$

Figure 5.8: The Run time for the Meakin Invasion Percolation Model with β values of 0.01, 0.001, and 0.0001, sorted from top to bottom respectively. The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

graph making it hard to see the AVL method; this is because there is a difference in the medians of only $1.9e^{-4}$ seconds, though before the L size of 1024 the AVL tree was actually the faster of the two by a considerable margin. There is no crossover in the runtime between the HSH and these two different methods this means that the speed up offered by the HSH method is highly significant even when compared to the second fasted methods for this gravitational coefficient; this is also demonstrated by the statistical tests seen in Table B.25.

Even though the LAT method is the slowest, by a significant margin at an L size of 1024, this is not the case for the lower L sizes that were tested. The first L size test where the LAT model became the slowest of the five data structures was when an L size of 256 is reached; This goes to so how significant in this case the construction time of the data structure is on the total run time of the simulation. Though it might not be evident in Figure 5.8 it is interesting to note that the OCT and HSH methods both increase in runtime in a manner that could be considered to be linear though more testing would need to be done on larger domain sizes in order to confirm this.

The next gravitation coefficient setting that was tested was that of $g = -0.001$, and this is very similar to the previous setting that was discussed. However, it takes up to an L size of 512 before the LAT model becomes the slowest of the different data structures; this again goes to show how significant the domain size can be in the simulation of these models. The general order of slowest to fastest at an L size of 1024 is very similar to that previous g value discussed above, with the HSH being the fastest and this being followed by the OCT, the AVL, and finally the RES method. Unlike with the previous setting, in this case, the AVL tree data points are not hidden behind the OCT method in this case the AVL tree is noticeably slower than the OCT method, and it becomes this way from an L size of 32.

The $(g = -0.0001)$ version of the Meakin model continues the trend between the previous two g values that have been used in that as this value decreases the runtime of the LAT method merges more and more into the range of the other methods. In this case, the LAT method is only the second slowest at and L size of 1024 with a median runtime of 3.74 seconds. The slowest of the method, in this case, is the AVL which has a runtime of 4.03 seconds at the same L size. However, it is quite apparent that if it was possible for the simulation to be run on a data structure that was larger than the 1024 max size that was tested that the LAT model would very quickly become the slowest of the different structures from the trend of the data that is shown.

As with the previous data g values that were tested the HSH method is the fastest of the different method at the L size of 1024 with a median run time of 1.78 seconds, which is 2.10 times faster than the LAT method and 2.26 times faster than the slowest method in this case which is the AVL. The HSH method only became the faster of the methods for the 1024 L size the previous test L size of 512 had it in third place behind the LAT which is the fastest and the RES method. As with the LAT method, it is quite apparent that the RES method would very quickly after the range of L size tested become slower than the AVL method and that the timing graph, if tested with large enough L sizes, begin to look more like the previous two g values tested.

As has been shown here as the L size increases with the Meakin method the LAT and RES methods very quickly become the slowest of the five different methods that are tested; this is due to the domain fill that this model exhibits which can be seen in Section 3.4. With the model very quickly approaching less than 1% domain fill, and due to the use of the Masson tree that is used for the handling of the growth sites being so efficient, in this case, the construction time of the specific data structure has much more of a significant impact on the total runtime of the model then the search and insertion into the data structure. Though eventually with a small enough g-value, this model would begin to act in the same manner as the random version of the IP model. This does demonstrate that there are considerable benefits to the runtime of the model to be gained from the use of more modern data structures for specific growth models, especially with larger domain sizes.

5.2 Memory Usage Data

One crucial aspect of any data structure is the maximum amount of memory that it can consume, as this can mean that some data structures are not possible to make use of due to out of memory exceptions. Unlike in the timing section, the data shown in the graph here will not be the median value and will instead be the maximum memory consumption of the system over the 100 runs. The reason for this is that is doesn't matter if a data structure can use a minimal amount of memory if there is a high chance that it could slip into a run that consumes so much memory that it causes the simulation to crash. This section will look into the memory consumptions of the different data structures, calculating the maximum memory consumption of these model is not always a simple matter of keeping track of the number of cells that have been added to the data structure and multiplying them by the node size. In the case of a structure such as the Octree with its ability to cull nodes from its structure as it becomes increasingly filled is it possible for the number of nodes within the structure to decrease between iterations. Also when a node is added to the Octree multiple nodes can end up having to be created to allow for the value to be stored in the correct position within the structure.

With the domain resizing method the number of cells within the domain changes each time the structure resizes itself; this also affects the amount of memory consumed between iterations but depending on the model in question the structure could end up being the same size as the Lattice version which would generate the full domain even though it is possible for a large portion of this never to be used. Because of these issues along with the stochastic nature of the models themselves, it was necessary to run many experiments to calculate a possible maximum memory consumption of each of the structures; this will give an idea of any significant drawbacks.

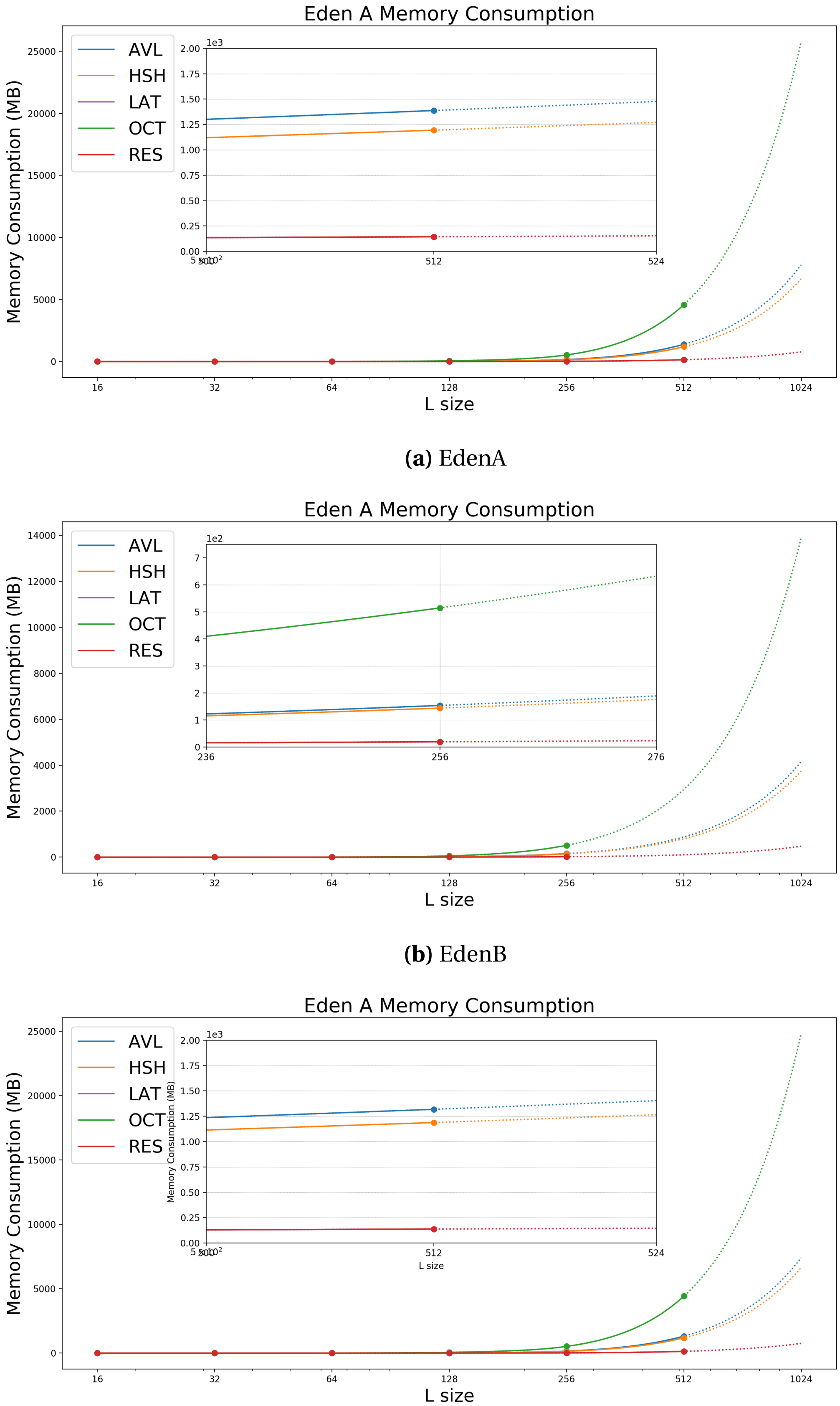
The Hashtable might seem like it poses a bit of difficulty because the fact that a single cell can hold a number values within it, this would mean that it would be necessary to iterate through each bucket and work out its memory usage and add these together. However, in this use case it has been proven that each of the buckets will only contain a single value, hence it is possible to merely multiply the bucket count by the size of a size_t variable for the key and a void pointer for the value in the bucket which works out to be about 16 bytes. The void pointer is a commonly used estimate in the calculation of memory for the C++ std hash table the unordered_map.

The memory consumption discussed here is inclusive of the memory consumption of the growth sites list and not just the size of the data structure as this is a critical aspect of the simulation of these models and can itself end up consuming a considerable amount of memory. Unfortunately, due to a change in the way that VIPER handles virtual memory it was not possible to be able to generate a cluster with an L size of 1024 for the memory consumption experiments for some of the models for which runtime data was able to be gathered. As such there are no 1024 data points for the EdenA, HIP, and the IPR methods.

The structure of this section will be the same as the previous section with it first discussing the three standard Eden methods followed by the Screened Eden method. Then with be the three IP model in the order of IPR, HIP, and finally MIP method. There will be references made the to runtime of the model within this section of the chapter especially in cases where the fastest of the method consumes the most memory; this will be done in order to keep the data in the proper context. The data shown here will make use of the same domain sizes as the timing section and each data point will be gathered from 100 runs of the simulation.

5.2.1 Standard Eden Growth Model

The Eden model has two different types of growth site lists that are utilised for the EdenA, and EdenC models the growth sites list that is used will only ever contain an entry once due to the additional state labelling that is used. The EdenB model, however, doesn't make use of this labelling system as having a single cell in the list multiple



(a) EdenA

(b) EdenB

(c) EdenC

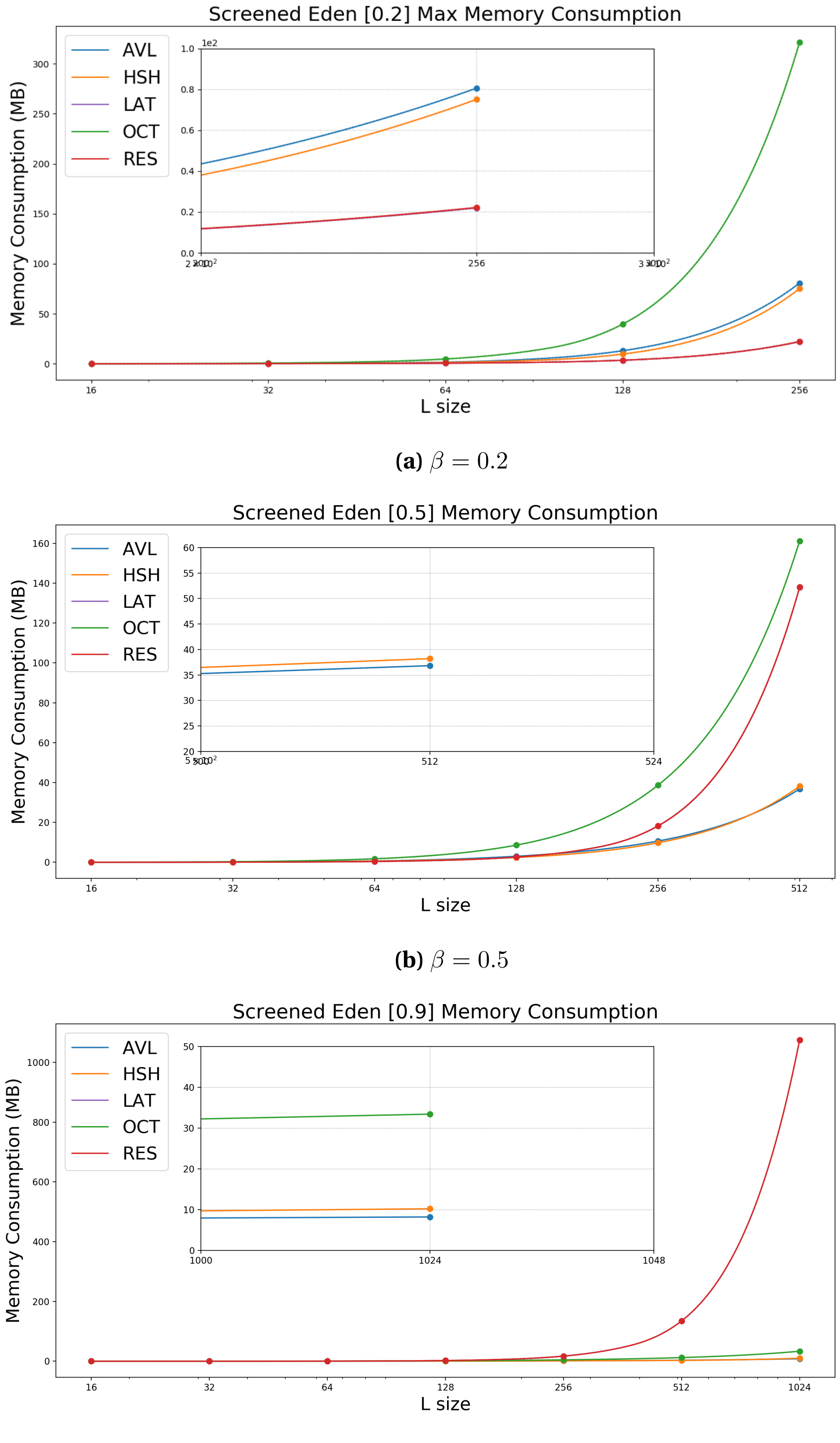
Figure 5.9: Maximum Memory Consumption for the 3 standard Eden models in MegaBytes. The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

times is the easiest way to ensure that each cell has the correct probability for being selected. Because of this, the Memory difference between the EdenA and EdenC model is very minimal with the EdenC model consuming the most memory, which is to be expected because it has a high fill percentage which can be seen in Section 3.4. The EdenB model is different, though acrounding to the fill percentage the memory consumtion of this model should be similar to that of the EdenA model it is signigicantly larger, this is to be expected however when the method for the handling of the growth sites is taken into account.

The general trend is the same for all three of the different Eden models with the RES and LAT version of the models consuming the least amount, and identical amounts of memory for all three versions of the model. The reason for this is that they both make use of the same variable for the storage of a value, the char, which is the smallest of the data type used. The filled percentage of the model also meant that the storage of the empty cells was not as big of an impact in caparision to increased memory consumption for a single infected cell of the other structure making them the least memory hungry implementations of the model. The reason that the LAT and RES use the same amount of memory is that of the stopping condition and the resizing method that was used making the Resizing domain method eventually generate the same number of cells as the Lattice version by the end. When the timing is taken into account this means that for the Eden model there is no benefit is the selection of any other data structure especially the OCT as this at an L size of 512 in the case of the EdenA model consumes 4 times the amount of memory with only a decrease in the total run time.

5.2.2 Screened Eden Growth Model

This section will discuss the memory consumption for the Screened Eden growth model for the beta values 0.2, 0.5, and 0.9 for a maximum L size of 256, 512, and 1024 respectively due to the runtime of this model it was not possible to get up to the L size



(a) $\beta = 0.2$

(b) $\beta = 0.5$

(c) $\beta = 0.9$

Figure 5.10: Maximum Memory Consumption for the Screened Eden model in MB. for beta values 0.2, 0.5, and 0.9. The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

of 1024 for the lower beta value models.

The max memory consumption for the Screened Eden model with a beta value of 0.2 can be seen in Figure 5.10a and as stated earlier this value is the closest to that of the EdenA model and the memory consumption of this model demonstrates this with the two least memory hungry method being the LAT and RES which due to the RES method used here with them consuming the same amount of memory. Though this is not always the case as the beta value used in the simulation increases.

The memory consumption for the 0.5 setting of the Screened Eden model can be seen in the Figure 5.10b. In the case of runtime, this setting has the RES, LAT, and AVL methods being highly tied regarding which one of the methods was the fastest; this means that in this case, the memory consumption of the model could be a big decision maker in the selection of the appropriate method. The two models that consumed the least amount of memory where the HSH and the AVL methods with the RES and LAT versions consuming only slightly less memory than the largest method which in this case is the OCT method. What this means is that because there is very little difference between the HSH and the AVL method, it is possible that either of these methods would be a good choice for the simulation of the model if only the memory consumption were taken into account until the data in Tables C.13, C.14, and C.15 are looked at. Here it can be seen that there is a small amount of significance for the AVL over the HSH this would mean that even though it would be only slight, there would be a benefit to the usage of the AVL method in this case and when the timing is also considered this shows that the AVL tree is the best choice for minimising the both of these aspects.

The memory consumption for the Screened Eden model with a beta value of 0.9 can be seen in Figure 5.10c. As the beta value increases the amount of memory consumed but the more modern data structure slowly decreases until it reaches this point where all of the structures consume considerably less memory then the LAT and RES methods at an L size of 1024 this shows the benefit of the use of these structures on

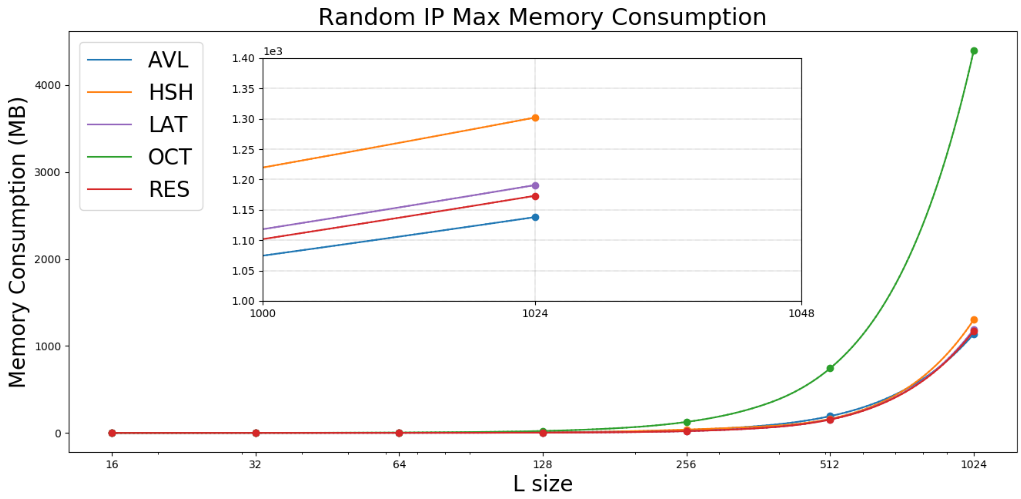


Figure 5.11: Maximum Memory Consumption for the Random Invasion Percolation model in MB. The floating box in the centre of each of the graph shows a zoomed in area sharing the same units as the main graph for both the x and y axis

a sparsely filled model. In this case the method that consumes the least amount of memory is that of the AVL method and considering that fact that in terms of runtime the AVL method is only slightly slower than the RES and LAT method the use of this method could be a large benefit as it could allow for considerably more runs of the model to be performed simultaneously on the same computer with only a slight increase in the median runtime of each of the separate runs of the model. The results of the statistical tests run on this data can be seen in Tables C.16, C.17, and C.18.

5.2.3 Random Invasion Percolation

The IP model is a useful model to show the possible memory benefits that can be gained by utilising one of the alternative methods for the storage of the cluster that are discussed within this work. Figure 5.11 shows the maximum memory consumption for the IPR model for an increasing L size. In this system, it quickly becomes apparent that the use of the OCT method for the storage of the IPR cluster would be a highly ineffective method because the memory that the method consumes is massive in comparison to the others. Though there might still be a specific situation where this additional memory consumption would be an acceptable hit, such as in the case were

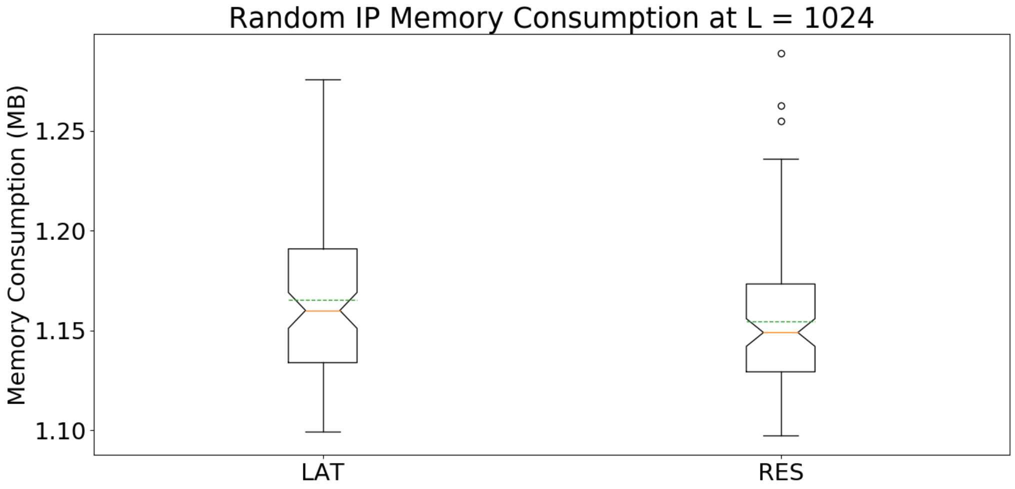


Figure 5.12: Random Invasion Percolation Memory Consumption Boxplot for the LAT and RES methods shown the benefit of the RES method in the max memory usage of the system.

only a single cluster is being grown for rendering and a method such as ray-tracing is the desired method to perform this rendering. However, if the aim is to run as many of these models as possible in series or parallel, especially in parallel, the results shown here would suggest that the use of the OCT method would be a highly inadvisable one, especially when the runtime of this model is taken into account as can be seen in Figure 5.6.

The difference between the memory consumption of the LAT and the RES model is quite an informing difference in the growth of the cluster in the domain, this shows that even in the case of an IPR model the domain doesn't actually require the full domain as the RES method doesn't reach a full $(1024 \times 1024 \times 1024)$ domain size which is obvious considering that the only difference between the LAT and the RES is the actual resizing of the domain and the memory consumption of a single cell is identical. The difference in the memory consumption between these two methods can be seen in Figure 5.12. The statistical test showing that this is a significant difference and that these boxplots are from different distributions can be seen in Tables C.19, C.20, and C.21.

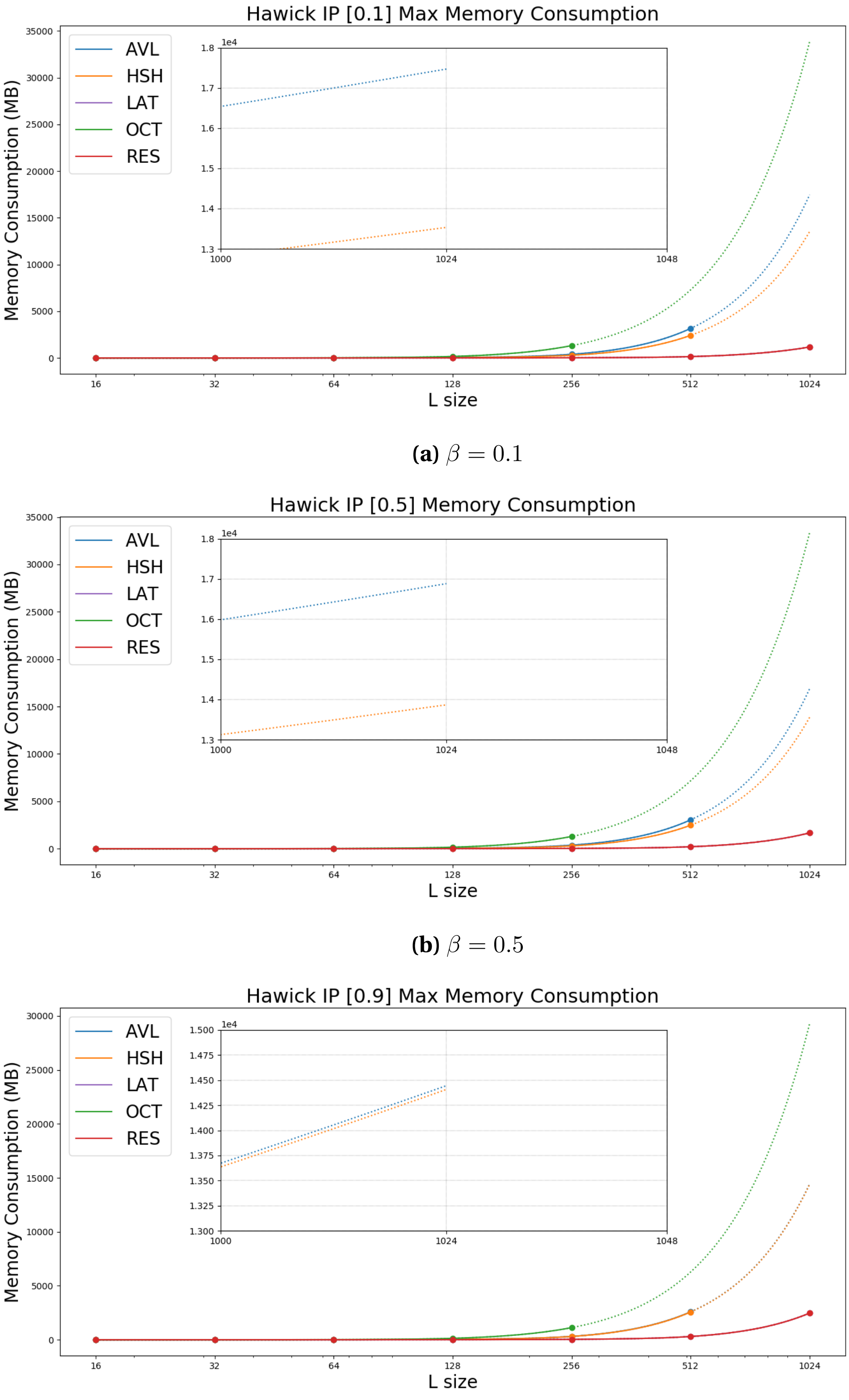
The method that consumes the least amount of memory in the case of the IPR model turns out to be the AVL tree which is the slowest of the five different structures; this is an excellent example of how when attempting to optimise a program sometimes a trade-off between the runtime of the simulation and its memory consumption must be made. It would be possible in specific situations for the saving in memory, to be able to compensate for the increase in runtime by allowing for more runs of the model to be simulated at the same time on a computer, however, it would require thousands of runs to be able to compensate for the massive increase in runtime especially if the aim was to run a simulation with an L size of more than that of 1024.

5.2.4 Hawick Invasion Percolation

The HIPs version of the Invasion Percolation model out of all the different models that have been used here is the one that fills the domain the most and could in no way be considered to be a sparse model and therefore it is expected that methods such as the AVL and HSH will consume considerably more memory in comparison to the LAT or RES method, the maximum memory consumption for the model with beta values of 0.1, 0.5, and 0.9 can be seen in Figure 5.13. The method which consumes the most amount of memory, in this case, is that of the OCT. It was only possible to be able to obtain the maximum memory consumption of this method up to an L size of 256 because of an out of memory exception.

The fact that the only two of the five structure consumed a small enough amount of memory that they were able to be run to an L size of 1024, the LAT and the RES, shows the efficiency of these methods. They also happen to consume the same maximum memory which because of how the model blankets the domain before it begins to move down is to be expected. When the runtime is taken into account, this shows that in the case of this model there would be no reason to make use of the other data structures for its simulation.

As the beta value increased the number of runs that encountered out of memory



(a) $\beta = 0.1$

(b) $\beta = 0.5$

(c) $\beta = 0.9$

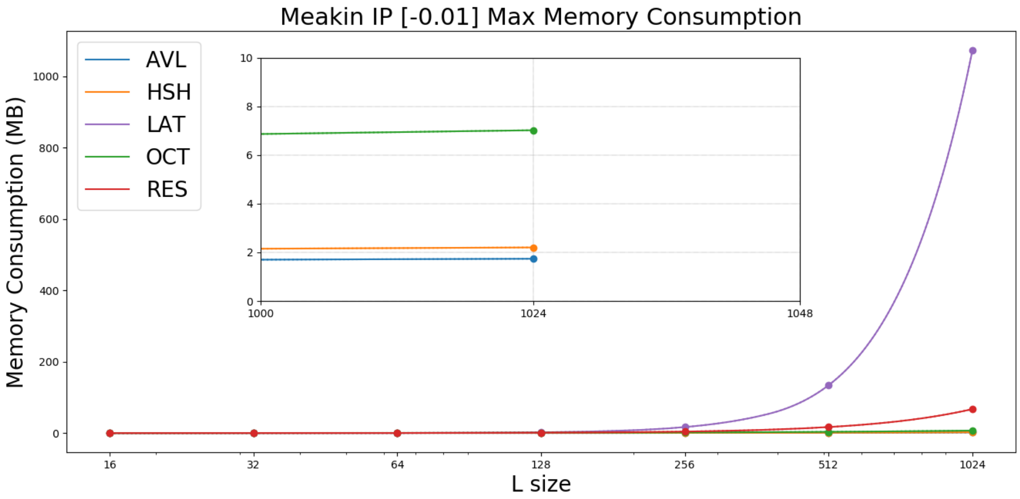
Figure 5.13: Maximum Memory Consumption for the Hawick Invasion Percolation model in MB. for beta values 0.1, 0.5, and 0.9. The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

exception increased. The reason for this is somewhat counterintuitive when the fact that the higher betas value are more sparsely filled is taken into account. However, it is a straightforward matter to explain and boils down to the fact that in order to be able to ensure that a cell is not added to the growth sites list more than once it is added to the structure in a neighbouring state, not an infected state. This along the much higher surface areas of the 0.9 version means that the number of cells that are stored in both the growth sites list and the cluster structure would be much higher and significantly affect the memory consumption of the model. This increased memory usage is enough to cause program crashes at these larger domain sizes especially with the OCT structure which consumes the most memory for a single out of all the structures tested here. This is an exciting feature and shows how the different methods used in the handling of certain features of a model can have unexpected and odd results.

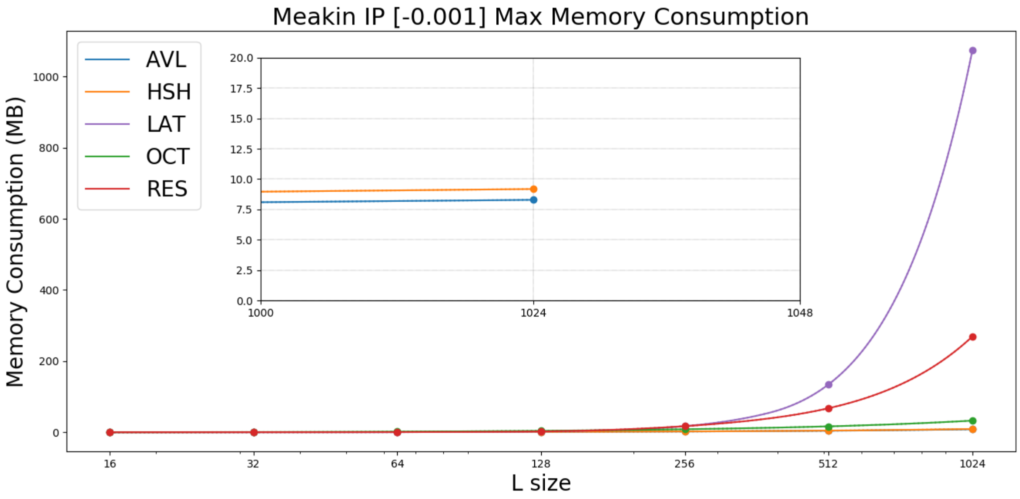
5.2.5 Meakin Invasion Percolation

The MIPs model in the runtime section was one of the models where the more modern data structure was faster. In the case of this model, this is also the case for the memory consumption where some more modern data structure that was tested this can be seen in Figure 5.14. In this model, the OCT, AVL, and HSH methods consume considerably less memory than the RES and LAT structures. The model that consumes the least memory in the AVL method up to the point that $(g = -0.0001)$ at this point the structure that is the most efficient in term of the memory consumed becomes the HSH method.

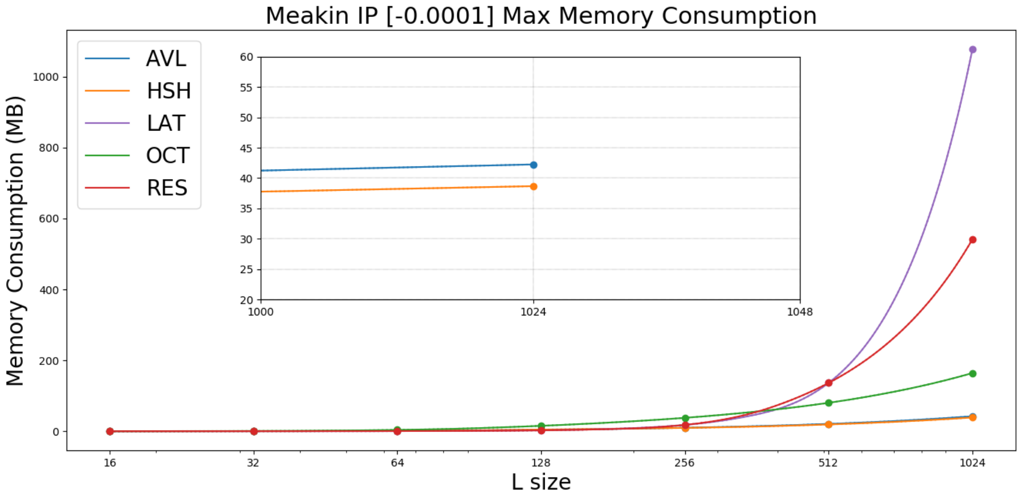
With a g value of -0.01, the LAT model consumes 617 times more memory than the AVL tree at an L size of 1024 which consumes the least amount of memory. The next best option in the HSH function, however, this may have an only slightly higher median the spread of the memory is considerably larger and the data in Table C.22 shows the AVL method is at an L size of 1024 highly significantly better than the HSH method in terms of the memory consumption. As the g value tends towards zero the



(a) $g = -0.01$



(b) $g = -0.001$



(c) $g = -0.0001$

Figure 5.14: Maximum Memory Consumption for the Meakin Invasion Percolation model in MB. for beta values -0.01, -0.001, and -0.0001. The floating box in the centre of each of the graphs shows a zoomed in area sharing the same units as the main graph for both the x and y axis

spread between the AVL and the HSH memory consumption increases function causing the AVL method to be more and more effective until when $(g = 0)$ and the model become the IPR model and the spread between the two methods is at its maximum.

The MIPs method is a clear example of how highly sparse models (with a fill of less than 1% for all the g values tested) can gain significant benefits from the application of more modern methods. When the timing data is also taken into account, this shows that for this model the worst method that could be used is that of the LAT method which is the current method that is used. Moreover, the two fastest of the methods being the HSH and the OCT method which are in turn the methods that used the least and the 3rd least amount of memory showing considerable benefits and with the Octrees benefits in ray tracing this could very quickly be a highly powerful application.

5.3 Summary

At the start of this chapter three expected outcomes where listed **E1**, **E2**, and **E3**. this summary will refer back to the outcomes and will point out an example of where the expectation can be found within this chapter to aid in the searching of the chapter.

It has demonstrated that in certain situations such as with the MIPs model with the HSH structure and the Screened Eden model with the RES at specific beta values where fill of the domain is very low, the standard method used for the simulation of growth models the LAT method is not the best option, and more modern data structures are able to offer benefits in both the cases of the runtime of the model and or the memory consumption of the model. This was listed as expected outcome **E1** and example of this can be seen in Figure 5.8

When the model in question is highly sparse, and the domain size it is being simulated within is unusually large, possible savings both in terms of the memory that is consumed and the run times of the model are able to be obtained through the use of a more modern data structure, this can be due to the considerably large construction time of the different structures. However, this very quickly stops being the case as the cluster fills more and more of the domain, which can be seen in the case of the Standard Eden models and the HIPs model, this was an expected result and was listed as **E2** one example of where this can be seen is in Figure 5.7 where the LAT and RES method are consitantly significantly faster the the otehr structures tested.

Expected outcome **E3** was to do with the increase in the effectiveness of the modern data strcutres as the sparsity of the model increases due to the increase in the domain size this is demonstrate and in most easily visable in the timing graphs for the Screened Eden Growth model, which can be seen in Figure 5.5. This shows that it is the sparsity of the domain that is the critical aspect for the decision between structres and not the size of the cluster itself.

The Screened Eden model is an excellent example of how sometimes the runtime of these systems is not as crucial to the selection of the structure as the memory consumption is, with the AVL and RES methods having a very similar runtime to that of the LAT method, however the AVL method has a significant memory saving. Another structure that offers a considerably memory saving in this case is that of the HSH method even though it was a slower structure. This shows it is not simple matter to decide whether one structure is the best for a particular model as it can depend on a number of factors and in a case such as here where the model makes use of screening values the ideal structure can change along with the screening value.

Additionally the Screened Eden model has also shown the need for a RES method that is able to resize multiple edges simultaneously without having to resize of all the sides based off of the position of the cell that is the reason that the domain needs to be resized. The reason for this is that it could allow for a method that behaves more like that of the one used for the MIPs model, this could allow for great saving in memory and in turn improve the runtime of the model.

So far this thesis has only considered possible optimisations that can be gained through the software-based version of growth models. However, it is commonly known that hardware-based implementations of a simulation can often offer substantial benefits to the runtime of the simulation. In the following chapters of this thesis discuss a possible implementation of the Eden growth model that makes use of memristor's, the fourth fundamental circuit element, for the storage of the state of the cells. Additionally new additional methods for the analysis of these models to confirm the effectiveness of the memristive system that has been designed will be investigated.

If you wish to forget anything on
the spot, make a note that this
thing is to be remembered.

Edgar Allan Poe

CHAPTER 6

Memristors for Growth Models

THE memristor was first discussed in 1971 by Chua (Chua, 1971, 2011, 2014). In his first paper (Chua, 1971), Chua talked about a gap in the then-trio of fundamental circuit elements and how a relationship between the magnetic flux and the charge passing through a device was missing. He stated that a circuit element could close this gap that he named the memristor which would be the fourth fundamental circuit element alongside the Capacitor (discovered in 1745), the Resistor (discovered in 1827), and finally the inductor (discovered in 1831). Its resistance unlike a resistor is based on the history of the current that had flowed through it and would be fixed when a current was not flowing through it (Chua, 2011, Kavehei et al., 2009, Trefzer, 2017); this was later generalised in 1976 (Chua and Kang, 1976).

Each of the different fundamental circuit elements has different specific uses. The first of these the Inductor is responsible for the storage of energy within a magnetic field. Capacitors store energy within an electrical field and resistors dissipate electri-

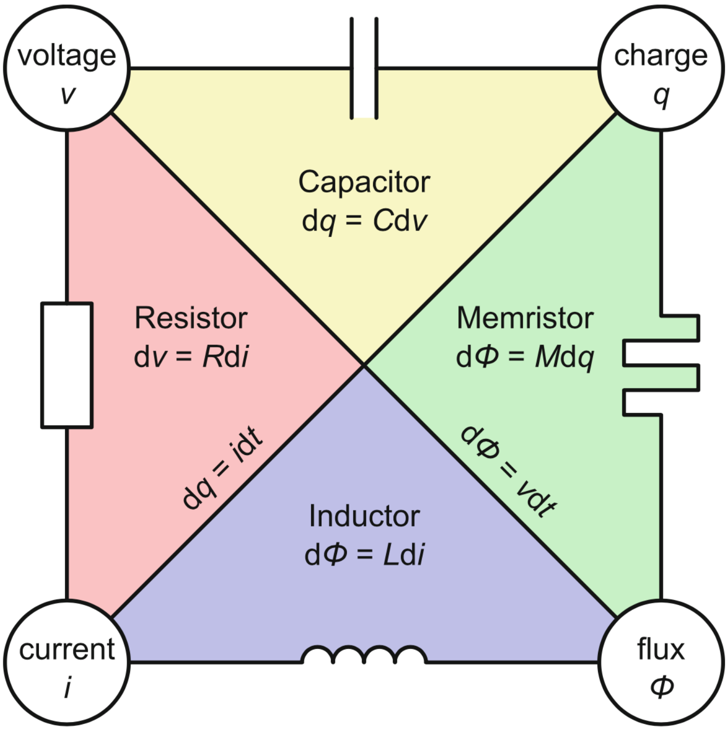


Figure 6.1: The four fundamental circuit elements (Trefzer, 2017).

cal energy. The memristor completes the conceptual symmetry of the fundamental elements a diagram of this can be seen in Figure 6.1.

There has been a fair bit of discussion over whether the memristors made by people such as the researchers at HP Labs are actually a real memristor (Meuffels and Soni, 2012, Stanley Williams, 2008, Vongehr and Meng, 2015). One of the arguments against this being a genuine memristor states that what has really been made is a chemical capacitor, and it will only be functionally correct until the chemical inhomogeneity in the TiO_2 is balanced out (Meuffels and Soni, 2012). Another argument (Stanley Williams, 2008) boils down to there having been two specific misunderstandings made by HP Labs. The first being that they have overlooked the extra design space that arises when working with non-linear circuit elements and the second one she states is "more profound" this being that they misunderstand the mathematical definition of the memristor. This work here will not answer this question and will act as though the devices are real memristors. The simulation used here is to simulate devices such as though made by HP labs and so it does not matter if the device does

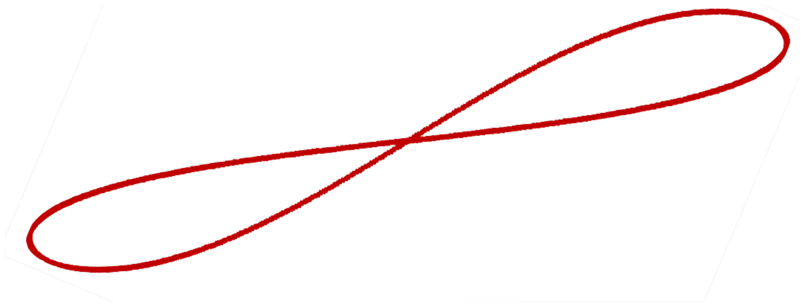


Figure 6.2: A pinched hysteresis loop. This is the iconic aspect of the memristor which occurs when you plot the voltage against current with sinusoidal voltage wave.

not wholly fit the original definition of the memristor which Chua himself argued for a change to in 2011 (Chua, 2011).

Chua famously can be often quoted saying, “If it’s pinched it’s a memristor” this was also the title (Chua, 2014) of one of his papers which focused on the iconic aspect of the memristor, the hysteric loop. This is what is formed when a sinusoidal voltage is applied to a memristor and the subsequent current against the voltage. A hysteretic loop looks somewhat like a slanted infinity symbol, an example of this can be seen in Figure 6.2. Exven before Chua’s paper, there are a considerable number of papers talking about anomalous loops on IV graphs (Dearnaley et al., 1970); this could mean that for years people were accidentally making a circuit with memristive properties within them but were unable to identify them.

An important question that should be answered is why the memristor has been selected for use here. The reason for this is that memristors offer highly interesting behaviour that where as is not currently taken advantage here in future work could lead to a more generalisable agent architecture for the simulation of growth models and that is their non-linear behaviour (Wang et al., 2015) as seen in the iconic pinched hysteresis loop which is discussed later on within this chapter. Even though is this specific application the memristor is used more like a counter this inclusion of this device offer substantial room for increasing the complexity of the models simulated.

The structure of this chapter will be as follows; first, there will be a discussion on how the memristor works explaining how it is possible for the device not only the change it resistance but also to be non-volatile with this change. This will then be followed with a discussion on some current uses of memristors in a few different areas including neuromorphic computing in order to give an idea of the wide-reaching benefits that this device can offer. This will then be followed by a discussion on the implementation of the memristor simulation including the differential equation that was used to simulate what happens to a memristor when a voltage positive or negative is applied to the device. The final part of this chapters will involve a description of the hardware agent that has been designed for the simulation of the Eden growth model as well as how the simulation works and the different variables that are used within the simulation that allows for different behaviour to be gain from the model. In order to be able to distinguish between the standard software implementation of the Eden model and this memristive based hardware implementation, the hardware method will be referred to a the MEden model which stands for Memristive Eden model.

6.1 How do Memristors work

The memristor can be written in terms of voltage and current without the need for magnetic flux; this can be seen in Equations 6.1 and 6.2. This is a highly unusual aspect of memristor and shows that the change in the resistance of the device is not due to magnetic flux. The change is instead an electrochemical process that affects the actual structure of the material used in the device (in the case of HP Lab's memristor that would be TiO_2(Strukov et al., 2008)) which is only possible due to the size.

$$v = R(w)i \tag{6.1}$$

$$\frac{dw}{dt} = f(i) \tag{6.2}$$

where:

R represents the generalized resistance

w represents the state variable of the memristor

i represent the current

v represents that voltage

dt represents the change in time

The easiest way of thinking of how the memristor works is to think of a pipe with water flowing through it (Trefzer, 2017). If the water flows in one direction through the pipe the diameter its increases which in turn reduces the resistance in the flow of water through the pipe, however, if the water flows in the opposite direction the pipe decreases in diameter which increases the resistance against the flow of water. So long as there is no water flowing through the pipe, its diameter will remain fixed. In this analogy, the memristor is represented by the pipe and the water flowing through the pipe is representative the flow of charge through the memristor and the diameter of the pipe in analogous of the resistance of the memristor changing based on the direction of flow and staying fixed when there is no flow.

The first time that a memristor was intentionally created was in 2008 by HP labs (Strukov et al., 2008). This memristor consisted of between 3nm and 30nm of titanium oxide which can be considered to be a wide band gap semiconductor which is auto doped; this is sandwiched between two platinum wires. When a current flows through the memristors it alters the chemical composition of the titanium oxide spreading the doped portion which is now TiO_{2-x} where $x = 0.05$ this means that about 0.5% of the oxygen is now missing causing there to be bubbles of oxygen vacancies to be present in one side of the memristor this is referred to as the doped side of the memristor.

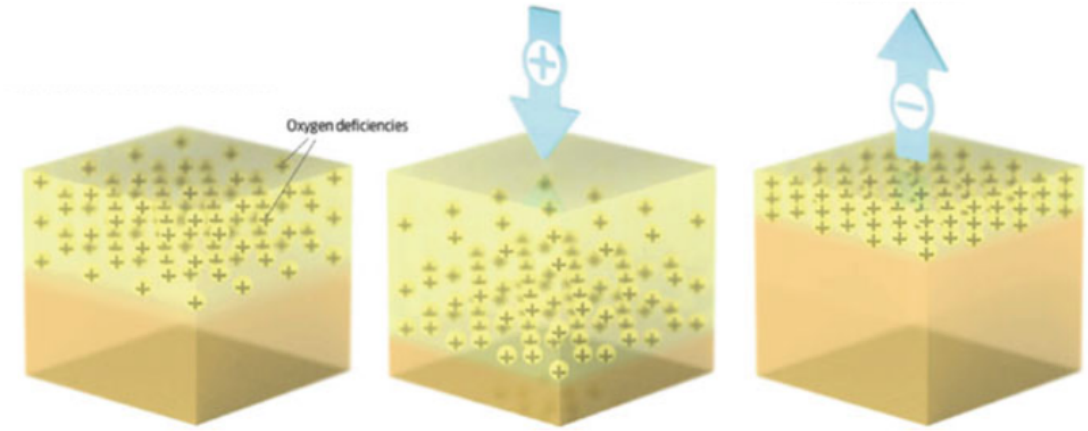


Figure 6.3: Visualization of how to oxygen deficiencies flow through a memristor based on the direction of the flow of current. (Tetzla, 2013, Trefzer, 2017).

The other side of the titanium is considered the undoped side of the switch which is the standard "perfect" titanium oxide or TiO_2 this can be thought of as the insulating side. When a current flows from the doped side of the memristor to the undoped side the oxygen vacancies within the memristor are spread further into the memristor this, in turn, lowers the resistance of the memristor. When it flows in the opposite direction the oxygen deficiencies move back and the resistance of the memristor increases; this works because these bubbles within the TiO_{2-x} carry a positive charge. Just like with the water pipe analogy when the flow stops so does the movement of the oxygen giving the device its non-volatile nature as an input of energy is required for the vacancies to be moved. A graphical example of this can be seen in Figure 6.3.

The scale of these switches goes a long way in explaining why it took so long for the memristor to be found and purposely fabricated as they need to be made in the nanometre scale to be efficient enough to be practical. This is because the effects of memristance follow that of an inverse square law, meaning that as the component gets smaller the more significant the effect of memristance with it becoming almost wholly unobservable in the millimetre scale as Stanley states (Stanley Williams, 2008) "memristance is a million times as important at the nanometre scale as it is at the micrometre scale, and its essentially unobservable at the millimetre scale and larger".

6.2 Current Uses of Memristors

Memristors are a relatively new device but even still there is an extensive body of work that makes use of them these include areas such as Cybersecurity, Big data, and they have even been used in order to create reconfigurable logic circuits (Mazumder et al., 2012). This section will go into a brief amount of detail on some of the areas in which memristors are currently being used, this will be by no means an exhaustive list but is intended to give an idea of the range areas in which memristors are finding a foothold.

The first of these is that of ReRAM, sometimes referred to as RRAM. This stands for resistive random access memory which is a non-volatile form of memory and is the area that will be discussed here that could have the widest reaching positive effect to the world with it offering benefits to all computing with is small size low energy cost and non-volatile nature. It shares many similarities to phase change memory (PCM) in that they both have a level of non-volatility, and it is possible for them to be arranged in similar architectures such as the crossbar (El-Hassan et al., 2016, Hamdioui et al., 2015, Vontobel et al., 2009, Yu et al., 2017) and it is not yet sure which of these two technologies will take over the market.

There has been much discussion about whether RRAM could be the replacement for flash memory which currently dominates the semiconductor memory market or DRAM which is the current method for the handling of random access memory in computers. Flash memory functions through the use of an array of memory cells for the storage of data; these cells are made up of large numbers of floating-gate transistors. Each is capable of storing one bit of data can require three different connection to function, these connections being a bit line, a word line and ground. This makes the amount of space required larger than that of the memristor the reason for this is the architecture that the memristor would use, the crossbar which will be discussed later on within this section. The way in which these gates function is by applying either a large positive or negative voltage to the gate. The large voltage means that the

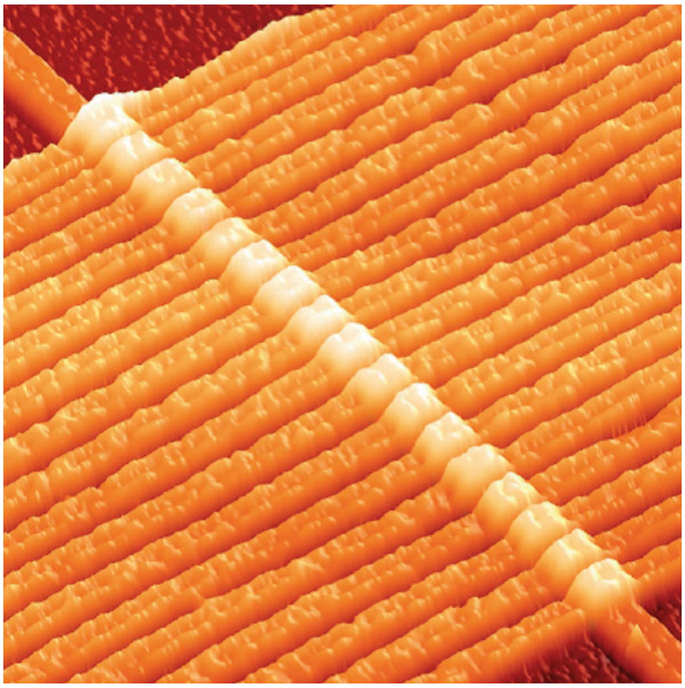


Figure 6.4: HP memristor in a crossbar architecture. image obtained through the use of an atomic force microscope. A total of 17 memristors are shown in this image.

transistors in the memory have a relatively short life with them only being capable of 100,000 P/E cycles before wear and tear being to deteriorate the storage device making it increasingly inconsistent and unreliable. This is where memristors come in to save the day so to speak. The most common architectures that are used for the design of RRAM systems is that of a simple crossbar system. This system consists of a matrix of connected memristors which can be quickly indexed even in parallel depending on the implementation. This design also allows for the stacking of layers for a 3D crossbar which allows for a very high level of data destiny within the device. An example of the crossbar can be seen in Figure 6.4. By used memristors to store this data instead of capacitors as in DRAM is it no longer necessary to have to refresh the memory continually so that it does not lose the stored data significantly decreasing the energy consumption as they memristors only required energy to be read from or written to and from. This memristive memory could also be used to replace flash memory as it is much more data-dense meaning that more data can be stored in the same size drive.

There is a wide range of different terms that are used by different manufacturers of

Non-volatile RAM systems (Trefzer, 2017) these include RRAM, CBRAM, PRAM. There is also Intel's Optane which makes use of a 3D crossbar system which they have called crosspoint. Optane is capable of read speeds of up to 1,450MB/s and write speeds of up to 640MB/s. They have not as of the time of writing this stated what the system is they have only said that it is not a phase-change memory which means it could very well be the first example of a memristive based memory on the market that is available to the public.

The second area is the one in which memristors have gained much traction, and this is in the field of Neuromorphic computing which makes use of analogue circuits in an attempt to mimic biological architectures such as the human brain. It was first conceived of by Carver Mead in the 1980s. Since the discovery of the memristor, the neuromorphic computing community has developed many systems making use of this element (Kvatinsky et al., 2014a, Uppala, 2015, Yakopcic et al., 2011). The main reason that these circuit elements are used in these types of systems is that of their ability to easily store floating-point values in a non-volatile manner. A lot of the neuromorphic systems are used to implement neural nets (Bala et al., 2016, Trefzer, 2017, Yu et al., 2017) for a number of purposes including image recognition and natural language processing. The non-volatile nature of the system means that when not in use the network remains trained acting as its own storage system.

An additional reason that memristors are used in systems such as neural nets is that they are capable of performing logic operations they can be also be implemented in such a way that makes then reconfigurable (Borghetti et al., 2010, Hamdioui et al., 2017, Lehtonen and Laiho, 2009) allowing storage to become compute. It has been shown that it is possible to use them for the implementation of full logic gates (Kvatinsky et al., 2014b) and even a complete adder system (Gale, 2015).

Finally, the main application that the memristor has been put to that is of concern within this thesis, and the final one that will be discussed is how they are used in the simulation of cellular automata. There has already been a substantial volume of work undertaken to utilise memristors within agents of a cellular automaton for the storage of the state of the cell. One being a minimalistic excitable cellular automaton that works within a 2D environment (Adamatzky and Chua, 2011). In the model, each cell can exist in one of three states resting, excited, and refractory. Each cell in the model will update itself depending on the state of the neighbouring cells that it is connected to, the connections between the cells can be altered to change the behaviour of the model. This is not the only example of memristors being used for cellular automata; there is a wide range of different current models that used this type of approach (Itoh and Chua, 2009, Wang et al., 2015).

One example of a different implementation of a CA to solve a specific problem using memristors is a shortest path solver (Pershin and Di Ventra, 2011, Stathis et al., 2014, Ye et al., 2013), with these CAs a memristive network is used for the solving of a shortest path algorithm such as what a Dijkstra's or A-star algorithm does. One of these pieces of work, in particular, was one of the key inspirations in the creation of the MEden model that is the work done by Stathis et.al (Stathis et al., 2014). This inspiration can be seen in the similarities in the design of the two agents, the agent designed for the MEden model can be seen Figure 6.9.

6.3 Simulation of Memristors

Due to the high cost and the low yield of memristors currently, it was necessary to build a simulation of a memristor in code in order to test the possible effectiveness of this method for the simulation of Eden-like clusters. To this end, a bespoke C++ program was created that simulates the effects of a memristive network made up of agents like the one shown in Figure 6.9, the design of this agent will be discussed in more detail in Section 6.4. There where other method that could have been used for the simulation of the model be the C++ simulation was selected as it allowed for an easier method to control the output of the system allowing it to easily write out the clusters that are produced by the model. The first step in the development of this simulation was to wirte a code based simulation of a memristor in that acts as closely as possible to a real memristor. One thing that will not be within the scope of the simulation is to simulated any possible flaws that a real memristor amy have and as such the memristor that will be simulated will be a perfect memristor.

In order to achieve this, it is essential to understand the mathematical equations that describe the processes going on within the memristor (Gomaa et al., 2016) these where first proposed by Strukov et al. (Karamani et al., 2017, Strukov et al., 2008). The first equation that is required to gain an understanding of the memristor is that of Ohm's law, this will allow for the calculation of the current that passes through the memristor, based off of the devices current resistance and the voltage that is being supplied. This can be seen in Equation 6.3 and is probably the most recognisable equation in electronics.

Unlike with a standard resistor where the resistance is fixed, with the memristor, the resistance of the component will continuously be changing as the current passes through it. To this end, it is necessary to be able to calculate this change in resistance. In order to be able to do this the first step is to understand what caused this change in resistance which as stated earlier is the movement of the oxygen vacancies. Therefore, the specific resistance of the memristor at a given time is proportional to the spread of the oxygen ions throughout the memristor; Equation 6.4 describes this process.

The next step is to understand how the x in Equation 6.4 will change over time; this value represents the rate of spread of the doped section of the memristor. This is calculated through the use of Equation 6.5. This rate will depend on two different things, the first being the dopant mobility and the second being the amount of charge that is flowing through the memristor. Dopant mobility characterises the speed at which the dopant, in this case, the oxygen vacancies move through the material. The Dopant mobility is described with the units $m^2s^{-1}V^{-1}$. In this case its value is $10^{-15}m^2s^{-1}V^{-1}$ which will be a constant applied in Equation 6.5.

$$V(t) = M(x)I(T) \tag{6.3}$$

where:

V is the voltage
M is the internal resistance of the memristor
I is the current

$$M(x) = R_{ON}x + R_{OFF}(1-x) \tag{6.4}$$

where:

$x = \frac{w}{D} \in (0,1)$
w is the width of the dopant spread in the memristor
D is the total length of the memristor
R_{ON} is the minimum possible resistance
R_{OFF} is the maximum possible resistance

$$\frac{dx}{dt} = \frac{\mu_v R_{on}}{D^2} i(x) f(x) \tag{6.5}$$

where:

$f(x)$ represents the window function
$\mu_v = 10^{-15} m^2 s^{-1} V^{-1}$ this is the average dopant mobility

$$f(x) = 1 - (2x-1)^{2p} \tag{6.6}$$

where:

$x = \dfrac{w}{D} \in (0,1)$
p is a positive integer

$$f(x) = 1 - (x - stp(-i))^{2p} \tag{6.7}$$

$$stp(i) = \begin{cases} 1, i \geq 0 \\ 0, i < 0 \end{cases} \tag{6.8}$$

where:

$x = \dfrac{w}{D} \in (0,1)$
p is a positive integer

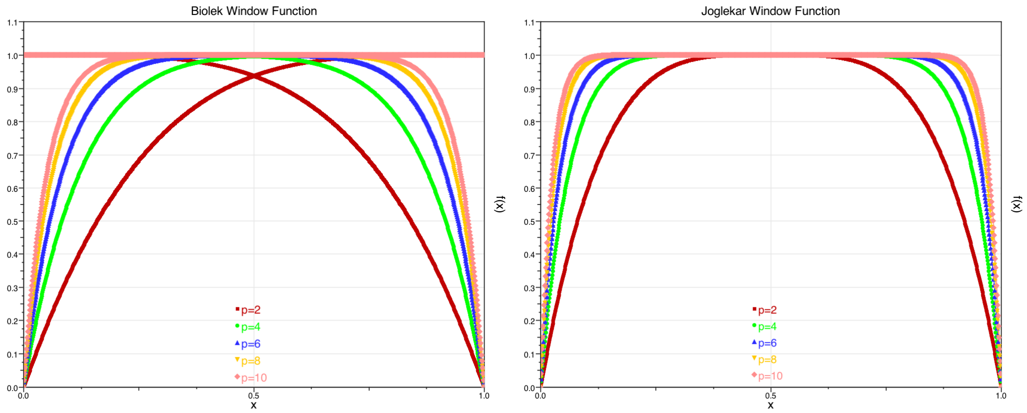


Figure 6.5: Results for the window functions Biolek on the left and Joglekar on the right where the x axis show the state variable in the range of 0 and 1 and the y axis represent the return value from the window function. The Biolek model displays the value for positive and negative I values for each of the p-values. The p-values that are shown are $p = 2$, $p = 4$, $p = 6$, $p = 8$, and $p = 10$. These functions can be seen in Equation 6.7 for the Biolek function and Equation 6.6 for the Joglekar function. The y axis has no units as this value is just a constant.

Equation 6.5 make mention of a window function, the job of this window function is to simulate the non-linear nature of the dopant drift. Two window functions are commonly used for this purpose both of which were tested. The first of these is the Joglekar function (Joglekar and Wolf, 2008) which can be seen in Equation 6.6. This method does have one major issue though this is known as a boundary lock; this is where no external force can change the resistance of the memristive device when the boundary is reaches meaning that with the Joglekar function once R_{ON} or R_{OFF} is reached the memristor would be fixed at the resistance and just become a resistor. That is where the second equation comes in the Biolek function which can be seen in Equation 6.7.

This equation was created by Biolek(Biolek et al., 2009); it was found that it was better as instead of only relying on a single value that is the state value or x the function makes use of a second value to describe the flow. So he added a dependency with the current i. By depending on the current, this solves the boundary lock issue by allowing the currents' direction to pull the device out of this state. The results of both of these equations for a variety of p-values can be seen in Figure 6.5.

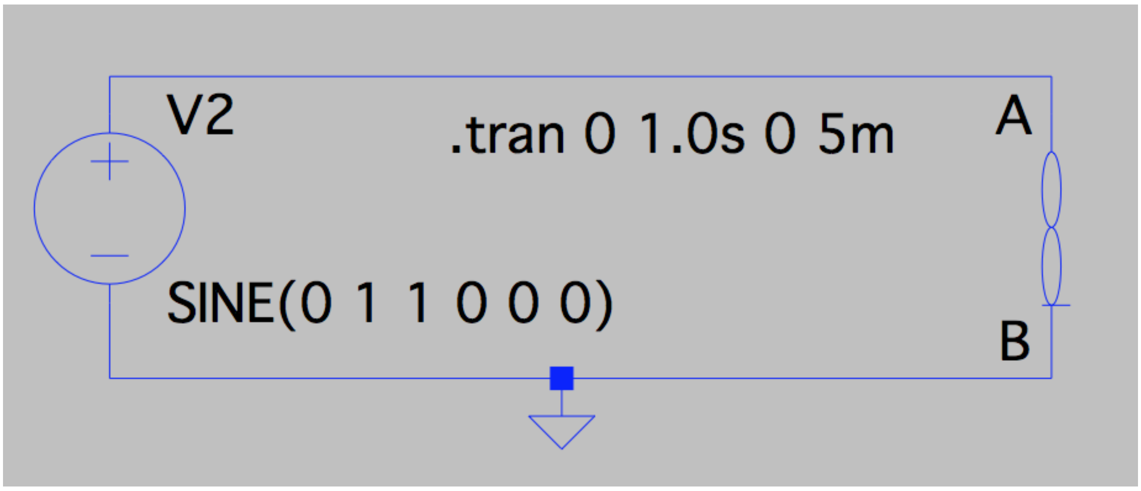
In order to test that the bespoke memristor simulation which was written in C++

and based on the Equations 6.7, 6.3, 6.4, 6.7, and finally 6.5 that has been developed functions correctly, it had to be tested against other accepted results for the behaviour of this circuit element. To this end a SPICE[1] simulation (Batas and Fiedler, 2011, Vourkas and Sirakoulis, 2015) was written that allowed for confirmation that the mathematical simulation in the C++ code was acting correctly. SPICE is known to be able to handle the simulation of circuits with a high level precision and is widely used in the world of electronics. SPICE is an open-source analogue circuit simulator, it allows for the construction of a circuit on a computer with a schematic capture system, the results of which can be seen in Figure 6.6a.

The SPICE model that has been used is a simplified version of a sub-circuit written by Ykaopcic et al. (Yakopcic et al., 2011, Yakopic et al., 2012) and Biolek (Biolek et al., 2009) this can be seen in Figure 6.6b and the circuit diagram can be seen in Figure 6.6a. The main difference between this version of the code and the one presented in the paper is simply the summation of $(uv*Ron/pow(D,2))$ with the variable K and the removal of some additional calculations which were not needed such as the calculation of the flux. The variables that are used to define the memristor are the same in both the C++ code and the SPICE model these being; $R_{ON} = 100\Omega$, $R_{OFF} = 16,000\Omega$, $D = 10\,\mathrm{nm}$, $\mu_v = 1^{-14}\mathrm{m}^2\,\mathrm{s}^{-1}\,\mathrm{V}^{-1}$, and $p = 7$.

The results for the SPICE version of the memristor simulation can be seen in Figure 6.7. The voltage was a pure 1Hz sine wave measuring a 1-volt peak the simulation was run for a total of 3 seconds. It gave a high peak amperage of $129.17\mu A$, and a low peak of $-129\mu A$, the wave has a slight slant forward it is this slant that is responsible for the pinched hysteresis loop that is shown at the top of Figure 6.7. The resistance fluctuated smoothly between $10.99K\Omega$ and around $4.48K\Omega$ after the first sine wave it never wholly reached the initial starting resistance again but came very close. With these results, it is possible to check whether the memristor in the C++ program is accurate enough to give a realistic response when used in the MEden model.

[1] Simulation Program with Integrated Circuit Emphasis



(a) The circuit diagram for the circuit that is being tested with the memristor represented by the pinched hysteresis loop on the right and the voltage source on the left.

```
.SUBCKT memristor Plus Minus PARAMS:
+ Ron=100 Roff=16K Rinit=11k D=10N uv=10F p=7
+K=(uv * Ron / pow(D,2))
Gx 0 x value={I(Emem) * K * f(V(x),I(Emem),p)}
Cx x 0 1 IC={(Roff - Rinit) / (Roff - Ron)}
Emem plus aux value={-I(Emem) * V(x) * (Roff - Ron)}
Roff aux minus {Roff}
.func f(x,i,p)={1 - pow(x - stp(-i), 2 * p)}
.ENDS memristor
```

(b) SPICE sub-circuit used for the simulation of a memristor within SPICE (Biolek et al., 2009). This version of the memristor makes use of the Biolek window function. This code is based on that shown in a paper (Yakopic et al., 2012)

Figure 6.6: The custom SPICE code and the SPICE circuit diagram for the simulation of the memristor used to gather effects of a sinusoidal voltage that is applied to a memristor so the C++ code version of the memristor can be tested.

When the results C++ implementation of the memristor is compared to the results of the SPICE implementation, it can be seen that when they have the same sine wave voltage applied to it in the simulations, that the SPICE simulation produces the results this can be seen in Figure 6.8. The C++ version calculates the current to fluctuate in the same manner as the SPICE version but between $129.47\mu A$ and $-129.35\mu A$ and it calculates the resistance to move between $11K\Omega$ and $4.46K\Omega$. This means that the C++ code is off from the SPICE code by $0.47\mu A$ on the high peak current and $-0.35\mu A$ on the low peak. The resistance is only slightly off as well with the high peak being $0.01K\Omega$ off and the low peak being $0.02K\Omega$ off. Having said that the general trend

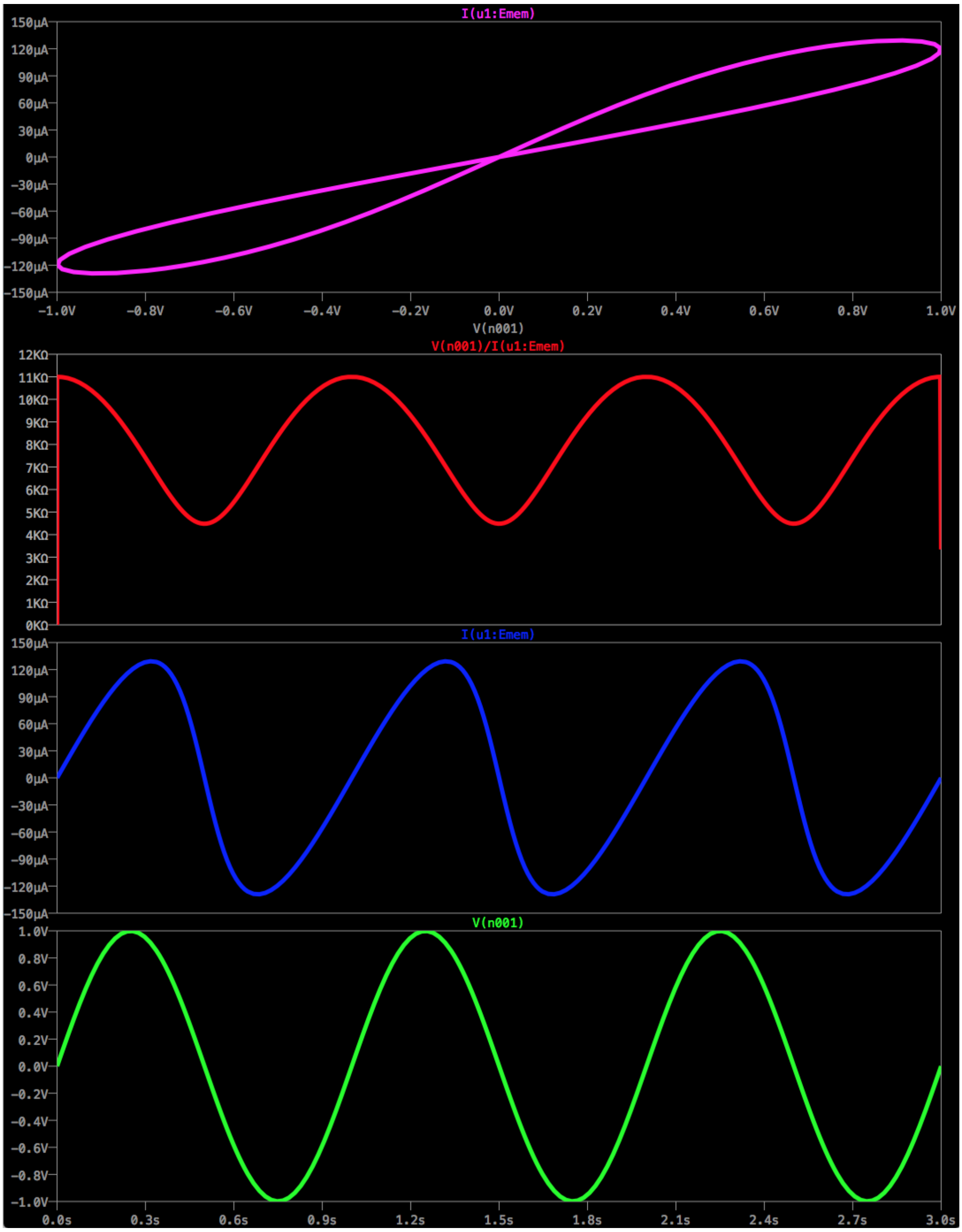


Figure 6.7: Data from the SPICE simulation of the memristor the top graph shows the IV graph showing the pinched hysteresis loop the second graph with the red line show the change in the resistance of the memristor over time and the final graph show the voltage in green and the current in blues over time.

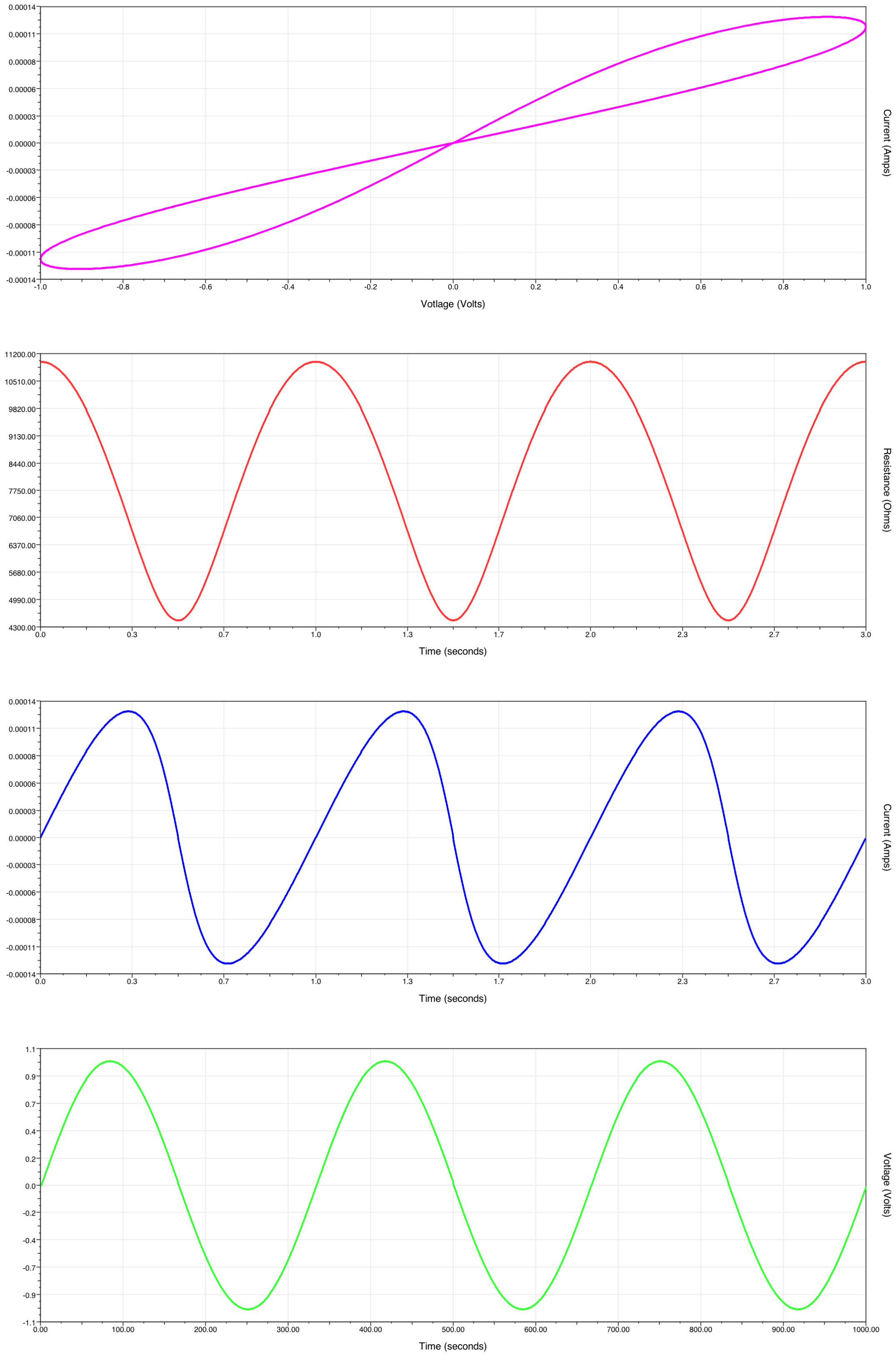


Figure 6.8: Graphs show the Current and the Resistance of the C++ simulated memristor as a 1Hz wave voltage with a peak of 1 volt flows through it.

between the two is the same with the same patterns being from and the differences are only minimal; this means is that the C++ method can be used for this purpose.

6.4 MEden Agent

The most important aspect of the MEden model simulation is the actual agent itself, this section will discuss the final MEden agent that is used for the majority of the experiments which can be seen in Figure 6.9 as well as a previous design for the model that was found to be unfit. This final design of the MEden agent can be seen in Figure 6.9, it has a single memristor within the state machine of the agent. This state machine is responsible for storing the state of the agent and determining whether the agent is infected and as such should be attempting to infect its neighbouring agents.

The memristor is set up with a R_{ON} value of 100Ω and a R_{OFF} of $16,000\Omega$. The agent is connected to its four neighbouring agents (north, east, south, and west) as it is the 2D version of the MEden model that will be focused on here, it is possible to make use of this same agent design for 3D by adding additional connections to the front and back agents, however. Each of these neighbouring connections has its own randomly pulsing voltage output, one for each of the neighbours. When the resistance of memristor drops below the activation threshold, the state machine closes the switches connecting the pulse generators to its neighbouring cells completing the circuit allowing the agent to infect its neighbouring agents.

The first design for the MEden agent was set-up with all the memristors within the network at the same initial resistance, and the pulse generators would be fixed and pulsing constantly. The issue with this version of the MEden model was that instead of resulting in the rough clusters it would generate clusters similar to that of a noise-reduced Eden growth model. As shown in Figure 6.9 each of the neighbouring cells is attached to a different pulse generator this was not initially the case in the initial design of the agent there was only one pulse generator. This change was made

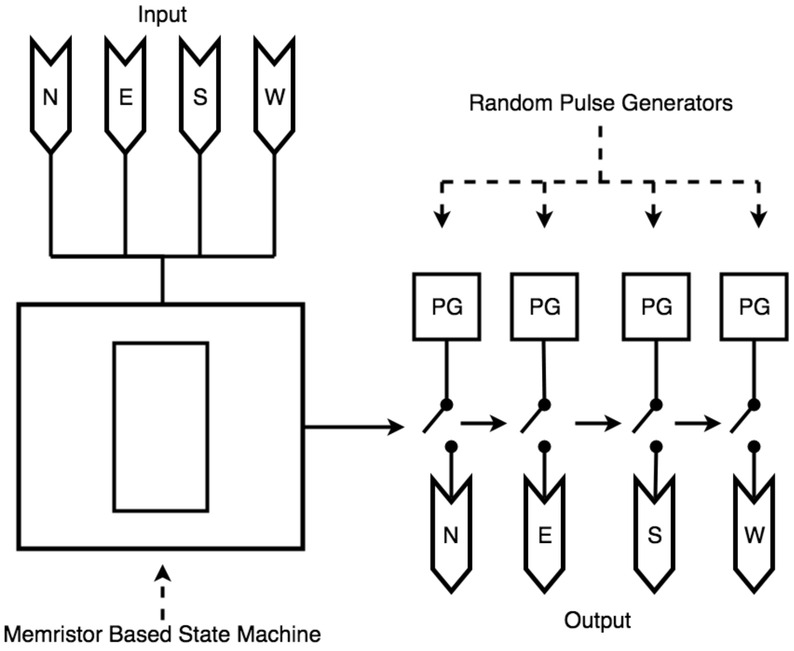


Figure 6.9: Diagram of the 2D memristor based cellular automata agent that is used in the MEden simulation. The initial resistance for each memristor is set within a random range based on the P_{max} of the run and the Random Pulse Generators will pulse the neighbouring cells with a change between $(0, P_{chance}]$. After the internal resistance of the state machine reaches the activation threshold the switches connecting the random pulse generators to the neighbouring cells are closed.

for two reasons, this first of these reasons was to ensure that when a pulse is sent the same amount of voltage is supplied to each cell whereas if there was only one generator connected to all four of the cells more the voltage would be split between the four neighbouring cells in proportion to the internal resistance of the memristor. This would mean that an agent with a lower resistance would receive a higher share of the total voltage and as such, it would have its internal resistance lowered at a much higher rate cuasing a biasing to the growth of the cluster. This could be used to allow for a possible screening effect to be included in future work. The second reason has to do with the Pulse Chance that is assigned to each of the generators, having four generators means that when one of the generators pulses its neighbour, it is possible for the other neighbour cell not to be sent a pulse, this increased the stochastic nature of the model.

Two variables can be tuned, in the MEden model allowing for the type of cluster that is produced to be more specifically selected to meet the user's requirements.

$$P = X_a - (\theta \times P_{max}) \tag{6.9}$$

$$P = X_a - [\theta(U \times P_{max})] \tag{6.10}$$

where:

P is the initial starting setting for the memristor at point (x, y)
X_a is the activation threshold for the memristor in the range [0,1]
U is a random value in the range [0,1]
P_{max} is the maximum pulse count to activate the memristor
θ it the normalized quantity of resistance that a memristor will alter by form a single pulse

$$C = P_{chance} \tag{6.11}$$

$$C = U \times P_{chance} \tag{6.12}$$

where:

C pulse chance for the pulse generator (x, y)
U is a random value in the range [0,1]
P_{chance} is the maximum chance a pulse generator has to send a pulse

The first of these variables P_{chance} relates to the random pulse generators that send the voltage pulse to the neighbouring agents. This variable governs the maximum chance that each generator has to send a pulse to one of the neighbours. There are two different equations that will be tested to generate the pulse chance for each of the random pulse generators; these can be seen in Equations 6.11 and 6.12. A version of the MEden model that makes use of Equation 6.12 is denoted with the prefix RC and in the case of when Equation 6.11 is being used it is denoted with the prefix FC.

The second of these two variables P_{max} concerns itself more with the initial resistance of each of the memristor within the domain. Initially, the cell is set up with a random memristance which is found through the use of one of two simple equations which can be seen in Equations 6.9 and 6.10. Equation 6.10 has a random factor included in it this means that the number of pulses required to activate each memristor can vary between 1 and P_{max} adding a level of randomness to the simulation. A version of the MEden model that makes use of Equation 6.10 is denoted with the prefix RM and in the case of when Equation 6.9 is being used it is denoted with the prefix

Name	Acronym	Equations
Random P_{max} and Random P_{chance}	RMRC	6.10 and 6.12
Fixed P_{max} and Fixed P_{chance}	FMFC	6.9 and 6.11
Random P_{max} and Fixed P_{chance}	RMFC	6.10 and 6.11
Fixed P_{max} and Random P_{chance}	FMRC	6.9 and 6.12

Table 6.1: The name of the four MEden models and the equations configurations that each of them uses

FM.

The combination of these four equations means that there are four possible combinations of MEden that with be investigated in Chapter 7. These versions are FMFC[2] which will make use of Equation 6.9 for initial resistance calculation and Equation 6.11 for the individual pulse generators chances, FMRC[3] will use Equations 6.9 and 6.10, RMFC[4] will use Equations 6.10 and 6.9, and finally RMRC[5] will use Equations 6.10 and 6.10. All of these are also described in Table 6.1.

Algorithm 5 describes the simulation that has been written to test the MEden model. The simulation will begin by initializing all the agents within the system's pulse generators and their initial resistance based on one of the set of equation in Equations 6.11, 6.12, 6.9, and 6.10. Once all the cells have been initialised a seed is "planted" meaning it is set as infected which closes the connections to its neighbouring cells. In this case, the seed cell is the centre most cell in the domain, though it is entirely possible for this seed cell to be any of the cells and for them to even be more than one. The system is designed in such a manner that it can handle any seeding that it is set up with, even in the case of multiple clusters that will end up interacting with one another.

After the seed has been set it will begin to pulse the neighbouring cells, this is where the main body of the simulation begins and where the software implementation it differs the most from the theoretical hardware based implementation. The

[2] Fixed P_{max} Fixed P_{chance} MEden Model

[3] Fixed P_{max} Random P_{chance} MEden Model

[4] Random P_{max} Fixed P_{chance} MEden Model

[5] Random P_{max} Random P_{chance} MEden Model

Algorithm 5 Eden Growth model pseudo code for the memristor simulation

```
initialise the grid of memristors
set the initial memristance for the memristors
assign the random pulse chance in the range (1,MAX] to each of the pulse generators
set seed cluster memristor to on
while current time step is < max time step do
    for all cells do
        if cell is on then
            pulse neighbouring cells
        end if
    end for
    for all cells do
        test memristance and update cell state to either on or off
    end for
end while
```

cluster growth portion of the simulation can be split up into two phases the pulse phase and the state update phase. The software implementation of the pulse phase as a simplification, each cell is iterated over and checked for whether it is infected. If it is then the agent will send a pulse to each of it connected neighbours this will continue until all the cells within the domain have gone through this process. Once this is done the state update phase will begin, this phase will also iterate over all the cell, and if they are not infected it will test the resistance of the memristor, and if it is below the activation threshold it will set it to infected which will mean in the next pulse phase that cell will begin to pulse its neighbouring cells attempting to infect them.

A theoretical hardware implementation would be designed so that the initial steps of the process would be identical to that of the software simulation; the difference is in the growth portion. In the simulation, this portion of the model is done serially in order to simplify the code and to allow for as many versions of the model to be run side by side on an HPC. Whereas in the hardware simulation all the cells will pulse their neighbouring cells simultaneously and then enter the state check phase and perform this test at the same time as well.

6.5 Summary

This chapter started off with a brief history of the initial conception of the memristor in 1971 and its fabrication in 2008. The memristor is a relatively new circuit element and as such is a thriving area of research in both industrial and academic realms. This new element could lead to countless benefits in the realm of computing and beyond.

This was followed by a discussion of some of the current uses for memristors in several areas including neuromorphic computing where they are being used in an attempt to simulate how the human brain functions for the computation of neural networks. Also, the shortest path solving system where a memristor-based CA is simulated for the computation of the shortest path within a maze with both a single and multiple exit points.

In order to be able to simulate the memristor, it is essential to understand the mathematical equations that describe the function of this unique circuit element these equations are discussed in detail in Section 6.3. This section also goes into the SPICE simulation which is a commonly accepted one and compares it to the ke C++ code that was used for the simulation of the MEden agent and the SPICE model showing that what differences there are between them are minimal and the purpose was to copy the general behaviour, not the exact resistance drops.

The final section within this chapter was a discussion of the design of the MEden agent focusing on where the inspiration for this agent came from and the changes that have been made to make sure that this agent functions in the desired manner for the running of the model. This section also goes into the design a previous version of the MEden agent a discusses why it was unfit for the purpose it was intended for. This is then followed by a description of both the algorithm for the running of the simulation and how the actual hardware would function if fabricated.

It is all well and good discussing the creation, and use of memristive-based agents for the simulation of growth models and the clusters that are produced may to the naked eye even look similar to the actual cluster. However, growth models exhibit a variety of complex physical characteristics that can be hard to identify by eye. As such the following chapter will analyse the four different versions of the MEden model with both the most commonly used method, the fractal dimension, and two new methods for analysis; components labelling and convolutional neural networks.

All you have to decide is what to do with the time that is given to you.

J.R.R. Tolkien,
The Fellowship of the Ring

CHAPTER 7

MEden Results and New Analytical Methods

WHEN attempting to develop a new algorithm or as in this case a hardware-based implementation for the simulation of a cellular automaton based growth model it is essential to compare the behaviour and results of the simulation against the currently used methods, in order to make sure that any differences can be quantified and understood. It is important to avoid the trap of just because they look the same to assume that they are the same.

There may be certain situations where this approach is good enough, for example if attempting to simulate the flight of birds for a film and the two methods are visually similar but the new method is much faster and easier to work with. However, as in the case here where these models are studied for particular behaviours that they may exhibit it is essential that the new method acts as closely to the original as possible and all differences are understood.

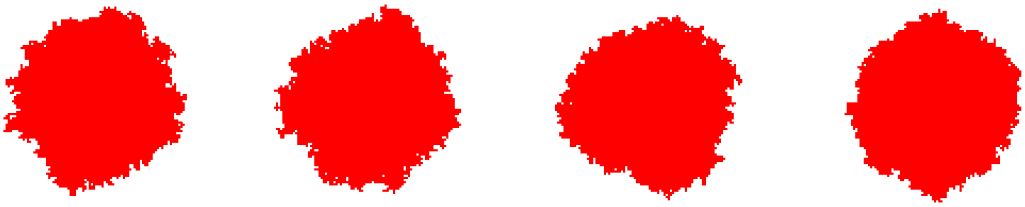


Figure 7.1: Examples of the cluster generated by the MEden model specifically the RMRC version of the MEden model where the stochastic processes are applied to both the P_{max} and P_{chance}. These images have been gather from the images that are fed into the image processing method disucssed in Section 7.3

This chapter will attempt to show that the MEden method for the simulation of Eden like clusters, if it was to be fabricated it would be theoretically able to simulate Eden clusters at not only a much faster rate then the current computational simulations used, but also that it is possible for this method to move between the different classes of the Eden model. This Chapter will also aim to find any limitations of this method such as in the types of clusters that can be produced, seeing if it can consistently produce EdenA, EdenB, and EdenC cluster or if it is only capable of producing one or two of these cluster types.

This chapter will discuss the experiments that have been run to classify the MEden growth model and compare it to the Eden growth model. It will begin with what can be considered the common method used to show that a new method is producing mathematically similar clusters, that being the fractal dimension. This section will begin with an explanation of the fractal dimension along with how it is actually computed. This will then go into the results obtained for both the standard Eden methods in addition to the MEden methods. Summing up the section will be a discussion on the limitation of this method showing the reason for the two alternate methods that have been developed for this task. Finally, there will be a discussion of the two proposed methods, these being the component labelling and the image classification. These sections will follow a similar structure to that of the fractal dimensions section beginning with an explanation of the implementation of the method followed by a discussion of results harvested from the different methods.

The MEden model can produce clusters that are very visually similar to that of other methods for the simulation of Eden clusters. Some examples of the clusters produced can be seen in Figure 7.1. But the question that is attempted to be answered here is whether these clusters are representative of pre-existing Eden clusters and if so how can we quantify this as well as any differences between them. This chapter will have a very specific macro structure to it as it attempts to do two things. First show that at least one of the four different MEden methods is capable of producing Eden like clusters as well as demonstrate that two proposed analytical methods work for the identification of Eden cluster from a single cluster. As such the four versions of the MEden model (RMRC, FMFC, RMFC, and FMRC the difinitions for these can be found in Table 6.1) will all start off being compared when a version of the model is considered to be inadequate it will then be discarded and will not flow into the next sections of the chapter.

7.1 Fractal Dimension

In order to classify these types of models, a measure known as the fractal dimension of the object is often used (Ivanenko et al., 1999, Lahiri et al., 2015, Sauer and Schroer, 1987). This measure tends only to be used to classify algorithms for the simulation of Eden models as Eden-Like but not to distinguish between the three version of the Eden model. The term fractal itself often has some misconception surrounding it; commonly it is thought that a fractal must be perfectly self-similar. This would mean that the object can be subdivided into smaller parts that are the same as the whole for example if you take the Sierpinski triangle and split it into three smaller Sierpinski triangles each one of these is identical to the original and even to each other. This could be considered to be a perfect fractal. However, this self-similarity is also present in shapes that are not fractals such as a cube which can be subdivided into 8 smaller identical looking cubes. This is where the fractal dimension of the cluster comes into play with it an object can be thought of to be a fractal by the manner in which it scales

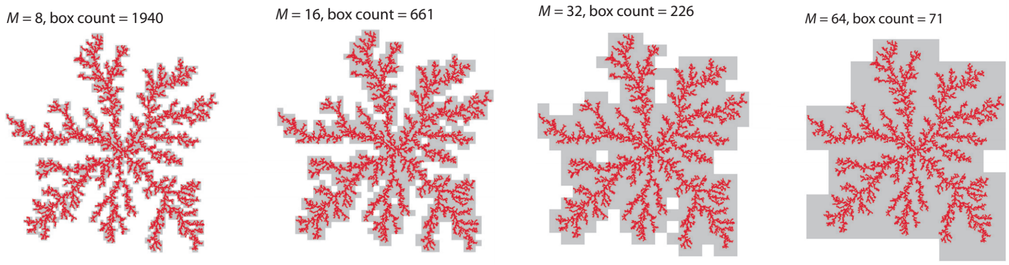

M = 8, box count = 1940
M = 16, box count = 661
M = 32, box count = 226
M = 64, box count = 71

Figure 7.2: Four successive steps of the first part of the box counting method for the calculation of the fractal dimension of a cluster. Where the grey boxes in the images represent the boxes that are classed as having an infected cell within it when the domain it iterated over checking areas of increasing size and the red cell are representative of cluster that is being analysed (Charbonneau, 2017).

and the fractal dimension of the object is the measure of this.

What is the fractal dimension and what does it tell us, this can be a somewhat abstract concept and is as such difficult to explain. One method to explain this is to think of a square, if you were to scale the size of the square by a factor of k then the width and height of the square would each scale by a factor of k which would mean that the area of the square would scale by a factor of k^2. If we took a 3D cubes and again scaled this by a factor of k its height, width, and depth would all be scaled by a factor of k each, and the volume of the cube would scale by a factor of k^3. For regular shapes such as the square and the cube, it is straightforward to work out the surface area or volume as each scale by their dimension to the power of k or k^D. This is not always the case though, if we take a 2D representation of the UK for example if we were to scale the UK by a factor of k how would the area of the UK change this is a much more complicated question as the width and height of the UK are not uniform and as such scale differently.

This is the problem that the fractal dimension aims to answer, for complex shapes such as the UK when they are scaled what D is. Interestingly for the UK $D \approx 1.21$, this means that when a 2D image of the UK is scaled by a factor of k its area will increase by a factor of $k^{1.21}$. Therefore, the fractal dimension of an object can give an idea to the effect that scaling may have on that object as well as the roughness of the object. It is for this reason that the fractal dimension is used in the comparison of growth models such as the Eden growth model as it can show that the roughness and scalability of the clusters are similar.

The method that has been used to calculate the fractal dimension of the Eden clusters comes from Natural Complexity (Charbonneau, 2017). The method is called box-counting and has been developed for use on a grid-based system; it is based on the mass-radius method. The method can be split up into two different main parts, the first of these parts can be thought of as the counting section this is where the method gets its name from. This involves placing squares of increasing size across the domain in a non-overlapping manner and if the box contains an infected cell inside of it then the method increments the box count by one a graphical example of the box-counting method can be seen in Figure 7.2 where the grey blocks represent boxes that where found to have at least one infected cell within it. Once all the different sized boxes have been checked its then time for the second part of the method.

The second part of this method is to calculate the slope of the line of best fit for the point of M or the radius of the box which in this case will be treated as the height and width of the box against the box count for each of the M sizes on a log-log graph. Though if the data was to be put on a log-log graph, it would give a negative slope the easiest way to fix this is instead of just calculating the log of the box size, to calculate the log of the inverse or $\log(\frac{1}{scale})$. This slope value is the fractal dimension for the cluster. The code used to obtain the fractal dimension can be seen in Figure 7.3.

7.1.1 Results

The first thing that must be looked at when analysing the fractal dimension of the MEden model to see if it can produce fractally similar clusters to and of the three main version of the Eden growth model, that is the fractal dimension for the Eden Growth models themselves. In order to get this information the EdenA, EdenB, and EdenC model where each run for $500,000$ times for 5000 iterations and the fractal dimension

```
void fractal_calculator::calculate(const int *crossbar, const int d, double *frac_dim){
 int L = d, n_scales = 1;
  while (pow(2, n_scales) < L){
  n_scales++;
 }

 double *scale = new double[n_scales];
 double *n_box = new double[n_scales];

 for (int iscale = 0; iscale < n_scales; ++iscale){
  n_box[iscale] = 0;
  cosnt int blocksize = (int)pow(2, iscale + 1);
  const int n_block = L / blocksize;
  for (int i = 0; i < n_block; ++i){
   int i1 = blocksize * i;
   int i2 = blocksize * (i + 1);
   for (int j = 0; j < n_block; ++j){
    int j1 = blocksize * j;
    int j2 = blocksize * (j + 1);
    if (contains_cell(crossbar, i1, i2, j1, j2))
    n_box[iscale]++;
   }
  }
  scale[iscale] = blocksize;
 }

 for (int i = 0; i < n_scales; ++i){
  scale[i] = log(1 / scale[i]);
  n_box[i] = log(n_box[i]);
 }

 double sumx=0.0,sumx2=0.0,sumxy=0.0,sumy=0.0,sumy2=0.0;
 for (int i = 0; i < n; i++){
  sumx += x[i];
  sumx2 += pow(x[i], 2);
  sumxy += x[i] * y[i];
  sumy += y[i];
  sumy2 += pow(y[i], 2);
 }
 frac_dim = (n*sumxy-sumx*sumy)/(n*sumx2-pow(sumx,2));

 delete[] scale; delete[] n_box;
}
```

Figure 7.3: This is the C++ code that is used in this work for the calculation of the fractal dimension of both the MEden and the Eden clusters this code is adapted from code presented in Natural Complexity (Charbonneau, 2017) and is specifically designed for calculating the fractal dimension for growth models that have been grown upon a square lattice grid.

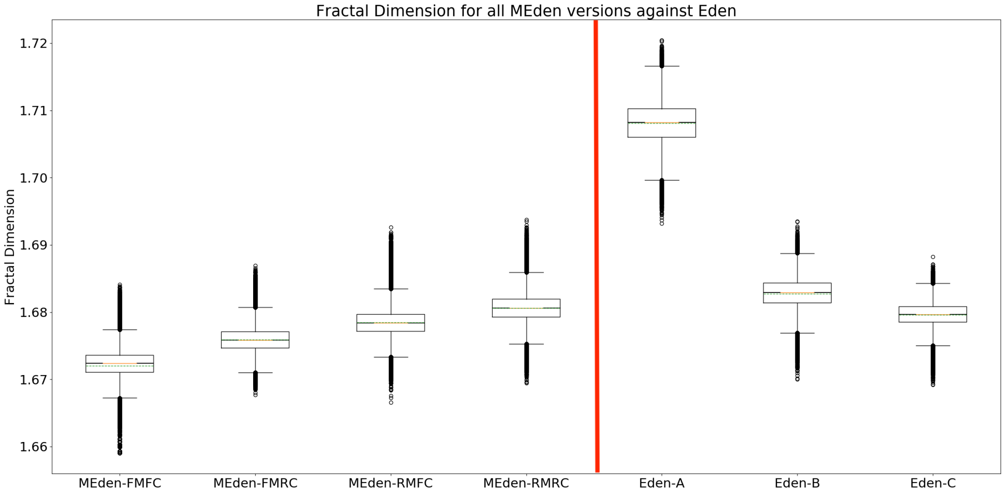


Figure 7.4: Box-plots showing the range for the fractal dimension produced by all the Eden and MEden models. The MEden models show the data from a range of different P_{max} in the range of 1 to 50 in increments of 1 and P_{chance} in the range of 1 to 100 in increments of 1. Each plot is a collection of 500,000 data points meaning that each of the mean graphs has 100 runs for each of the variables.

was calculated at the end of these iterations for each of the models. $500,000$ may seem like a vast number of runs, and this quantity of data is a lot over what may have been needed to obtain an accurate understanding of the range of the models. The reason that it was run so many times was to be able to make a fair comparison with the MEden model. The MEden model has two different variables that can be manipulated in order to give different results the P_{chance} and the P_{max} when testing the MEden model the variables would be used in the range of 1 to 100 for the P_{chance} and 1 to 50 for the P_max each for 100 times to allow for a distribution to be obtained as such it was only fair to run the three different standard versions Eden models the same number of times.

The range of the fractal dimension for the EdenA, EdenB, and EdenC models can be seen in a box-plot diagram in Figure 7.4 on the right-hand side of the red line, some additional data that gives a more detailed understanding of these plots can be seen in Table 7.1. This shows that the EdenB and EdenC models have a large overlap in their possible fractal dimension ranges with all the EdenC points being contained within the range of the EdenB this would make categorising a single cluster from either of

Model Type	lower IQR	upper IQR	mean	median	skew
EdenA	1.706	1.710	1.708	1.708	-0.201
EdenB	1.681	1.684	1.683	1.683	-0.412
EdenC	1.679	1.681	1.680	1.680	-0.323
FMFC	1.671	1.675	1.672	1.672	-1.233
FMRC	1.675	1.677	1.676	1.687	0.311
RMFCn	1.677	1.679	1.678	1.678	0.675
RMRC	1.679	1.682	1.681	1.681	0.019

Table 7.1: the data for the fractal dimension for the three Eden variants (Eden A, Eden B, and Eden C) and the MEden growth for P_{max} from 1 to 50 in increments of 1 and P_{chance} from 1% to 100% in increments of 1%. All have been done in 2D for the data shown here.

these methods based solely on this method next to impossible. However, in the case of the EdenA model, this is not the case, with it have a significantly higher fractal dimension than the other two methods and very minimal overlap.

Now that we know the possible ranges for the three versions of the Eden model the next step is to run the same test for the four versions of the MEden model (FMFC, FMRC, RMFC, RMRC) and see if their range of fractal dimension overlap in any way with the EdenA, EdenB, and EdenC model. This will show that the MEden model is not only able to produce clusters that are visually similar to that of the Eden growth model but that these clusters are also fractally similar to the Eden models and as such can be considered to be producing Eden-Like clusters. As stated earlier the MEden model will be run for a range of variables and the results from the run can be seen in Figure 7.4 and as with the Eden models there is more data to help describe the box-plots in Table 7.1.

The first thing that is noticeable when looking at the fractal range for the four MEden models and the three Eden models in Figure 7.4 is that none of the four MEden models come even close to being able to match that of the EdenA model. What this means is that these methods will likely not be able to produce EdenA like clusters, this will be a useful fact when attempting to clarify the results in the following sections. However, this is not the case with the EdenB and EdenC models with there being a decent amount of overlap between these models and the four MEden models though it is clear that two of these models are a better representation than the others.

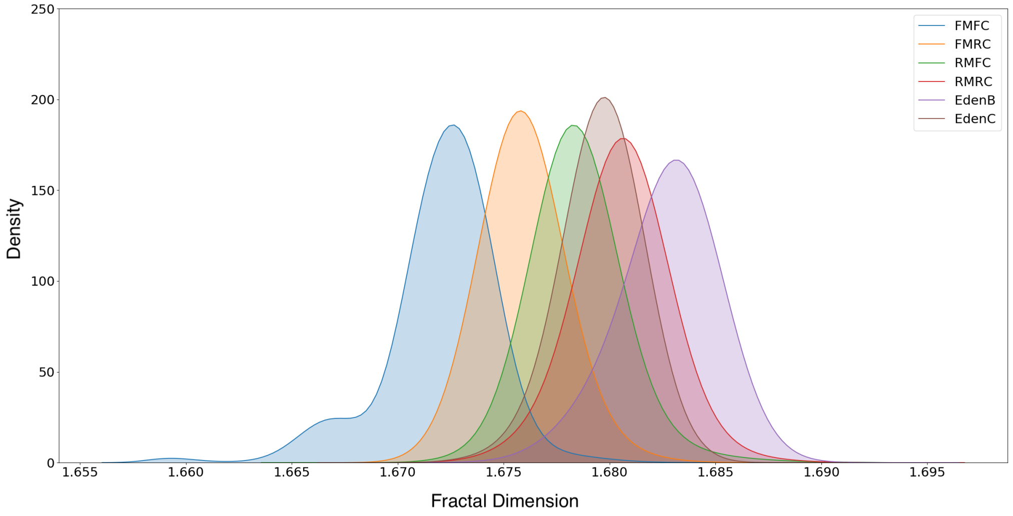


Figure 7.5: KDE graphs showing the range for the fractal dimension produced by all the Eden and MEden models. The MEden models show the data from a range of different P_{max} in the range of 1 to 50 in increments of 1 and P_{chance} in the range of 1 to 100 in increments of 1. Each plot is a collection of 500,000 data points meaning that each of the mean graphs has 100 runs for each of the variables. The EdenA model has been removed as it didn't overlap with any of the MEden models. This shows which of the models are able reproduced the EdenB and EdenC models more consistently. The versions that are the RMRC and the FMRC. These KDE plots where made using matplotlib with default settings.

The easiest way to compare the MEden models to the Eden models and see which versions can represent the EdenB and EdenC models more accurately is through the use of a KDE[1] which is similar to a histogram this can be seen in Figure 7.5. This graph makes identifying which of the MEden model are more consistent than the others more accessible; it also shows that all of the methods can produce Eden like cluster though not all as well as each other

Up until now all the MEden models data has been summed up for all the different variables though this does give a decent understanding of the full range of the model it doesn't however give an understanding of at what specific P_{chance} and P_{max} settings the model generates a cluster with a certain fractal dimension. Figures 7.6 and 7.7 show all the data points generated by the simulation and the median fractal dimension at that specific variable set-up respectively. The different colours of the points represent which cluster they are most like Blue means that it falls below the

[1] Kernel Density Estimation

EdenB and C's fractal dimension. Green is representative of the fact that the point falls in the IQR[2] of the EdenC clusters fractal dimension, red is used to represent that same for EdenB. Finally, Grey is used for EdenA though no point reached this high, no point reaches past the brown section of the colour-bar which is used to show points in-between the whiskers of the EdenA and EdenB models. The same colouring has been used for both Figures 7.6 and 7.7.

Figures 7.6 and 7.7 give further information on the applicability of the four version of the model. The FMFC version of the model is the most different from that of the others, even when looking at all the data point that where generated it becomes clear that this method is too limited in production Eden-Like clusters. It often produces clusters that would be too smooth for anything other than the lower end of the EdenB and C clusters and below except for extreme case with a very low P_{max} with clusters that fall even lower on the fractal dimension being produced at high P_{chance} settings. The median clusters that are produced by this method are not able to reach into the EdenC IQR except at a single point with a P_{chance} of 100 and a P_max of 1.

The remaining three versions of the MEden model are much less limited in the production of Eden like clusters with the RMRC and the RMFC being the best of the four this becomes very clear when looking at the data shown in Figure 7.7 with the average fractal dimension produced. This shows that whereas the FMRC can produce clusters with a high fractal dimension similar to that of the IQR of the EdenB and EdenC it is much more likely to produce clusters that have a fractal dimension less than that of the EdenC IQR. However, the RMRC and the RMFC are considerably more effective in the generation of the EdenB and EdenC like clusters, with the RMRC never dropping lower than the EdenC cluster fractal dimension. This means that both of the RMRC and the RMFC can be considered to be the most effective of the four methods both of them have benefits over the other. For example, the RMRC can give a better fractal representation of two of the Eden model and the RMFC being faster as such these will be the two version of the model that will be discussed in

[2] Interquartile Range

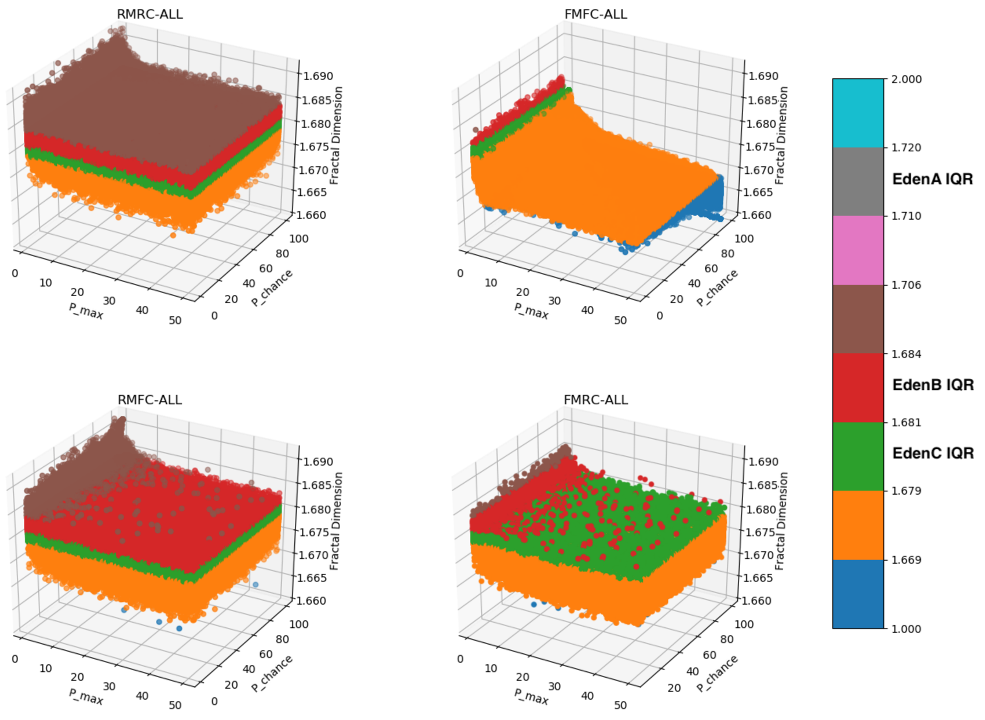


Figure 7.6: These graphs show all the data points for the fractal dimension of four version of the MEden model at each of the different P_{chance} and P_{max} values tested.

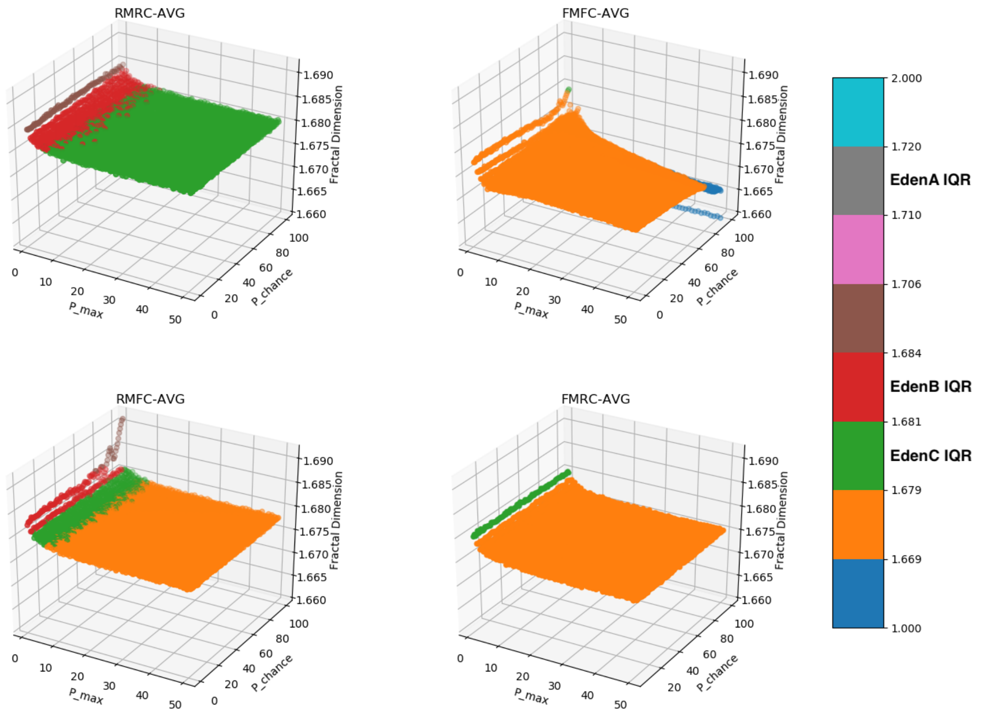


Figure 7.7: These graphs show the average fractal dimension for the four version of the MEden model at each of the differen P_{chance} and P_{max} values tested. Each point is an average of 100 runs of the simulation. This graph shows that the **FCFC!**[a] and FMRC models where as it is possible for them to produce Eden C like clusters it would be highly inconsistent making them unreliable.

[a] **FCFC!**

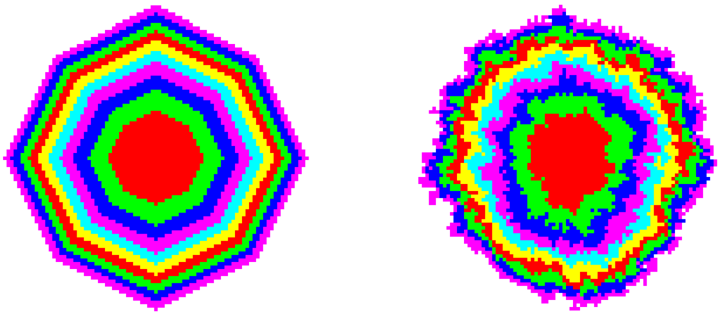


Figure 7.8: The clusters shown here demonstrate one of the issues with purely trusting the fractal dimension of a geometrical object to show that any possible new algorithm produces similar clusters. Both of these clusters have a fractal dimension of approximately 1.6792. However, it is obvious that the cluster on the left is not an example of an Eden like cluster whereas the one on the right it to be precise the cluster on the right is generated through the use of the Eden C method.

the following section. The FMRC and the FMFC will not be discussed in the following section as the results here show that their fractal range is too limited to be considered a possible replacement simulation for the Eden growth model.

7.1.2 Issues with the Fractal Dimension

This section will go into a few of the reasons that lead to the creation of an additional method for the analysis of a growth model. The fractal dimension of an object is a useful tool that can allow for a detailed understanding of an object and is of great benefit in the classification and analysis of an alternate method for the simulation of a growth model. However, it does have some drawbacks, the first of these is that is it not a scalable method especially in the analysis of a method such as the MEden growth model. In the MEden growth model, two main variables can be tailored to produce different styles of clusters. The issue comes in when it is needed to verify whether the clusters produced conform to what would be produced by a more commonly accepted method for the simulation of these models.

Through the use of the FMFC method, it is possible to produce clusters that have a fractal dimension that falls within the range of the EdenB and EdenC models even though is not comparable to an Eden cluster. The growth clusters that are produced have a fractal dimension of approximately 1.6792. Which lies at the lower end of the interquartile range for the EdenC model but instead of looking like an Eden model with a rough, jagged perimeter with a rounded shape, it instead forms a perfect hexagon; this can be seen in Figure 7.8. In order for this to be identified it would mean that the cluster would have to be visualised and then manually identified by a human for the range of possible variable this could be a very time-consuming thing to do. It would mean that it would be impossible to automate this process in order to make sure that the clusters that are produced are accurately representative of an Eden cluster.

The second problem with this method comes into play with the range of the fractal dimension produced by the EdenB and EdenC models. The range of the fractal dimension for both of the models has an A measure of 0.856 which was gained with the use Vargha-Delaney's method and a p-value of $6.8e{-15}$ which was gain with the Mann-Whitney U test. What this means is that there is a large significant difference between the two distributions which might lead one to think that it would be easy to identify which of the two models are being used to generate a cluster. However, due to the overlapping between the two models which can be seen in Figure 7.4 and Table 7.1, in order to be able to distinguish between the two models it would require multiple runs of the model so that a method such as the Vargha-Dellany or the Mann-Whitney U test can be used to determine the chance that one or the other has been produced and it would only be able to tell if the new method produces the same range of clusters. All of this means that whereas the fractal dimension is a very useful tool and does have the ability to classify how close the results of the new method is to the original it is useless in the classification of individual clusters when two methods have a significant overlap such as is presented here or if a method produces a different range of clusters.

Because of this, two new methods have been developed for the analysis and classification of clusters the following two sections of this chapter will go into these two methods. The first of these methods can identify a quantifiable difference between the EdenB and EdenC clusters. The other method that will be discussed whereas it does not come up with a quantifiable difference it still able to tell the difference between the Eden B and C methods of the clusters. The hope for these methods is that they will aid in the classification of not just Eden model but growth models in general in order to show how these methods would work the MEden model will be used as a case study to demonstrate the methods.

7.2 Connected Component Labelling

The first of the two new methods that will be discussed makes use of CCL[3] an often used method in the field of graph theory for understanding the structure of complex graphs. The reason making use of this method here for this is that by labelling the cell in-order as they are infected and then using this method, it would be possible to gain a better understanding of how the cluster grows as the way that the cell fits together. This method was designed as a better way to distinguish between clusters such as the EdenB and EdenC as where with the fractal dimension there would need to be multiple runs of the model run into to tell which it would be and then it would only be with a high probability if a cluster that falls outside of the range of the EdenC was produced indicating that it was an EdenB method meaning that it would be classifying the method, not the clusters. With this method, it is possible to categorise individual clusters.

CCL aims to find subsets of connected components, these components are uniquely labelled and are based on a given heuristic. One common area other than that of graph theory in which this class of algorithm is used is in computer vision (Davies, 2012, Dillencourt et al., 1992). In combination with image filters, it can allow a computer to be able to pick out individual parts of an image and depending on the filter that is used the type of object that will be picked out will change. This means that a

[3] Connected Component Labeling

program attempting to find red doors could use a filter to turn all pixels with a red value over a certain level white and all over pixels black, allowing for a much simpler image to be processed. Then the individual components can be labelled and subsequently be checked for a match to a door with the use of a convolutional neural net or other such methods.

Algorithm 6 Two Pass Component Labelling

```
equivalences, labeledGrid, nextLabel=1
for x: row ∈ grid do
    for y: columns ∈ grid do
        if grid[x,y] is infected then
            Neighbours = cells within mask whose state match current cells state
            if Neighbours is empty then
                equivalences[nextLabel]= set(nextLabel)
                labeledGrid[x,y] = nextLabel++
            else if tree is right heavy then
                $s$ = smallest label ∈ Neighbours
                for L ∈ Neighbours do
                    equivalences[L] = union linked[L] and L
                end for
            end if
        end if
    end for
end for
for x: row ∈ grid do
    for y: columns ∈ grid do
        if grid[x,y] is Infected then
            labeledGrid[x,y] = find(labeledGrid[x,y])
        end if
    end for
end for
return labeledGrid
```

There are a large range of methods that can be used for CCL, a lot of the work on these algorithms as of late has been to make use of the massive amounts of parallel power that can be gained from the use of a GPU through the application of languages such as CUDA and OpenCL (Hawick et al., 2010, Kalentev et al., 2011, Playne and Hawick, 2018, Wu et al., 2005, 2009). For this work however the algorithms used all ran on the CPU and in serial as it was not necessary that the component labelling be performed in real time. Three main methods are commonly used for the computation of CCL the one component at a time method, the two-pass method, and finally the

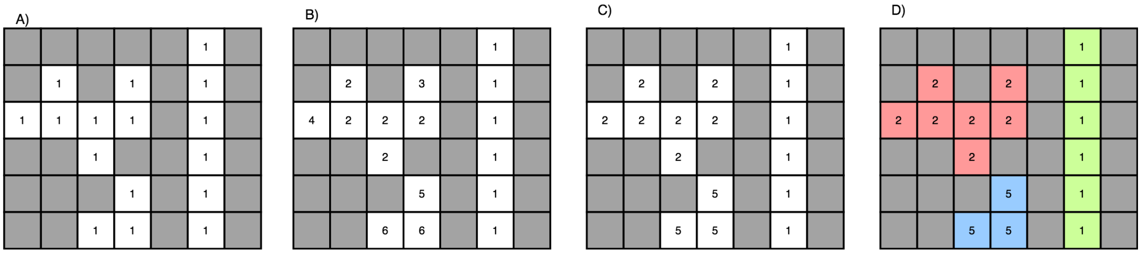


Figure 7.9: Two pass component labelling image A shows the grid before the first pass image B shows the state of the grid after the first pass and image C shows the state of the grid after the second and final pass. The final image, image D shows a coloured version of the finished grid to more easily identify the different components.

one-pass method. Whereas the one component at a time method is a very easy to implement and understand variant of CCL it is very slow in processing the data as it requires numerous passed through the data. Considering the size of the grid that is being dealt computation speed is still somewhat relevant to be able to average out as many results as possible in a reasonable time frame. The other two methods the two-pass and the one-pass are both considerably faster especially in a case like the one here where multiple different states can be touching each other that should be classed as different components.

The two-pass method was the one that was selected for use with this work, this was selected because even though the one pass method can run faster than the two pass when an image consists of a high number of small components the speed increase with the one pass method is negligible in comparison to the implementation complexity of the algorithm for this specific use case. The first pass of the algorithm assigns temporary labels and records the equivalences of these labels; this is done through the use of a union-find type structure. On the second pass, each label is replaced with the smallest label within the table of equivalences. An example of the steps of the component labelling can be seen in Figure 7.9, where image A represents the initial state of the grid image B is the image after the first pass, image C is the grid after the second pass, and the final image, image D is a coloured version of the components to make it easier to see the different components. Algorithm 6 shows the algorithm for the two pass CCL that has been used in this work.

Masks are an important aspect of the CCL algorithm as they let the algorithms know which of the neighbouring cell should be looked at for a specific cell when looking for component labels to assign. There are two main types of masks use, one for the Moore neighbourhood and one for the more common Von Neumann neighbourhood on a 2D grid; these are often called 8-connection and 4-connection respectively. As the Eden model itself makes use of a Von Neumann neighbourhood when it grows, that is the mask style that will be used. This mask looks at the cell north of the current cell as well as the cell west of it in the 3D version it also looks at the cell behind the current cell; the Moore version also takes into account the cells at the north-east and north-west of the current cell. A graphical representation of the Von Neumann and Moores mask can be seen in Figure 7.10.

$$s = \left\lfloor \frac{i_s}{(\frac{t}{S})} + 1 \right\rfloor \tag{7.1}$$

where:

s is the state of the cell

i_s is the infection step of the cell

t is the total cells in the cluster

S is the number of states desired

In order to run the component labelling algorithm, the cells in the cluster must be split into different groups with each group having a distinct state. The cells state will be set equal to the time step that it was infected on in the case of the Eden model and with the number of cells in the cluster at the time of infection in the case of the MEden model. The issue arises here that if left untouched each cell would have a different state and therefore would be labelled as having a component count equal to the number of cells in the domain which would be entirely uninformative. The answer to this can be seen in Equation 7.1, the total cells within the cluster are divided by the total number of states to give a step value. If the state of the cell is greater than zero it is then divided by this step value and has one added to it; this would mean that in

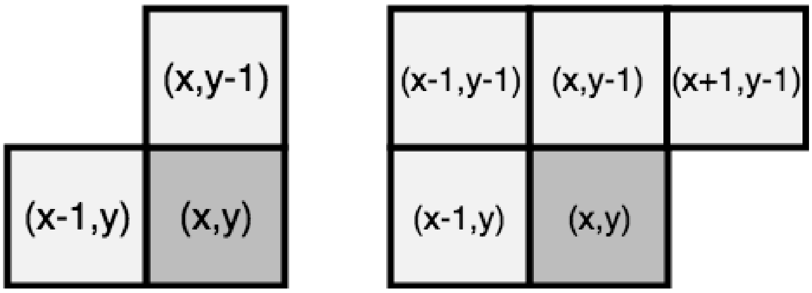


Figure 7.10: This diagram is representative of the Von-Neumann neighbourhood mask used for the component labelling on the left and the Moore neighbourhood mask on the right. Because of the type of neighbourhood used with the Eden model the mask that will be used for this work will be the Von-Neumann mask on the left.

a cluster of 1,000 cells with 5 states would have a step value of 200 and the ending state of a cell that was infected on time step 300 would be given a state of 2 because $\lfloor \frac{300}{200} + 1 \rfloor = \lfloor 2.5 \rfloor = 2$. The flooring of the value is dealt with by storing it as an integer value, which will truncate the float causing it to be floored. The plus one is included in the equation to deal with cells that were infected in a time step that is less than the step value making sure that the minimum state value is one otherwise it would become an uninfected cell.

One issue does arise with the labelling of the states of the cells in the MEden model, and this is because of the way that the update state system iterates through all the cells in order. This means that if two or more cells are infected on the same time step (which happens very often) a cell with a larger k index would always be considered to be infected after cells that precede it in the domain; this would mean that it would not be a fair comparison to the software Eden models and would alter the results of the component labelling. To deal with this issue when multiple cells are infected on the same time step at the end of the update function, all the cells that where infected would have been stored in a list, and each of the states would be shuffled and randomly assigned to each of the cells. This means that it would be possible for cells further on within the domain to have a smaller infection step then cells preceding it.

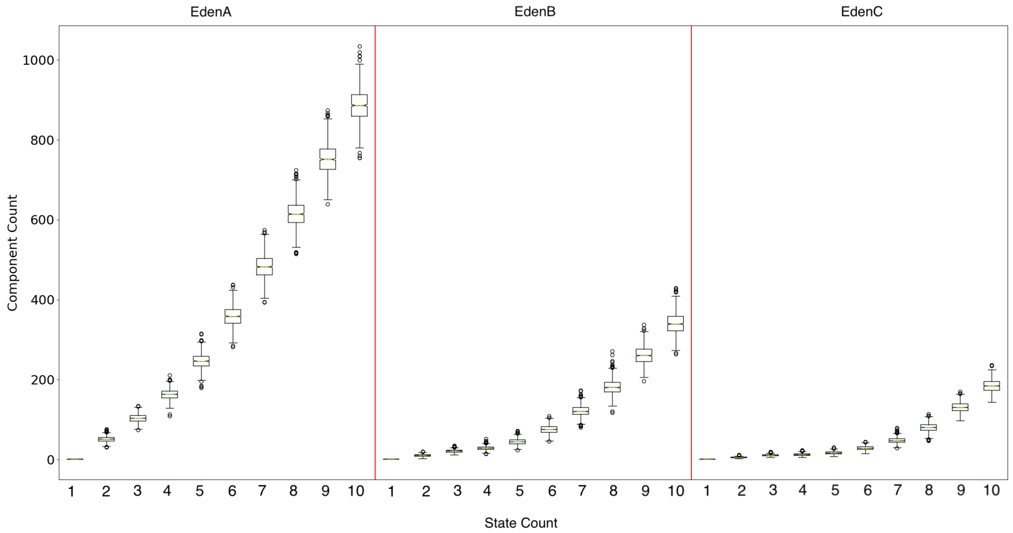


Figure 7.11: Boxplots showing the component count range for the EdenA EdenB, and EdenC models for state count in the range of 1 to 10. Th state count defines the nunbe of distinct states the the celles are divided into based on the timestep the where infect on, the equation used for setting this state is defined in Equation 7.1

7.2.1 Results

The first step of testing is to get the component count for the three standard versions of the Eden model for a range of different maximum states and to see if there is a point at which the ranges of the component produced do not overlap so that it will be possible to identify one cluster from another. This is a critical part of the method as if it is not possible to get a distinct group for each of the models it will not be possible to use this method to identify a single cluster and as such, it would be no better than the fractal dimension is the analysis of clusters. The first decision to make is the range of the state used. If the cluster was labelled with a single state then it would always have a component count of 1, which is not useful so the experiment was run with a state count from 2 to 10 and performing the component labelling on the model with this range to see if the separation point between the component count can be found the results for the EdenA, EdenB, and EdenC standard methods can be seen in Figure 7.11.

As the number of states within the cluster increase the EdenB and EdenC component count ranges slowly start to separate until it reaches 7 states at which point

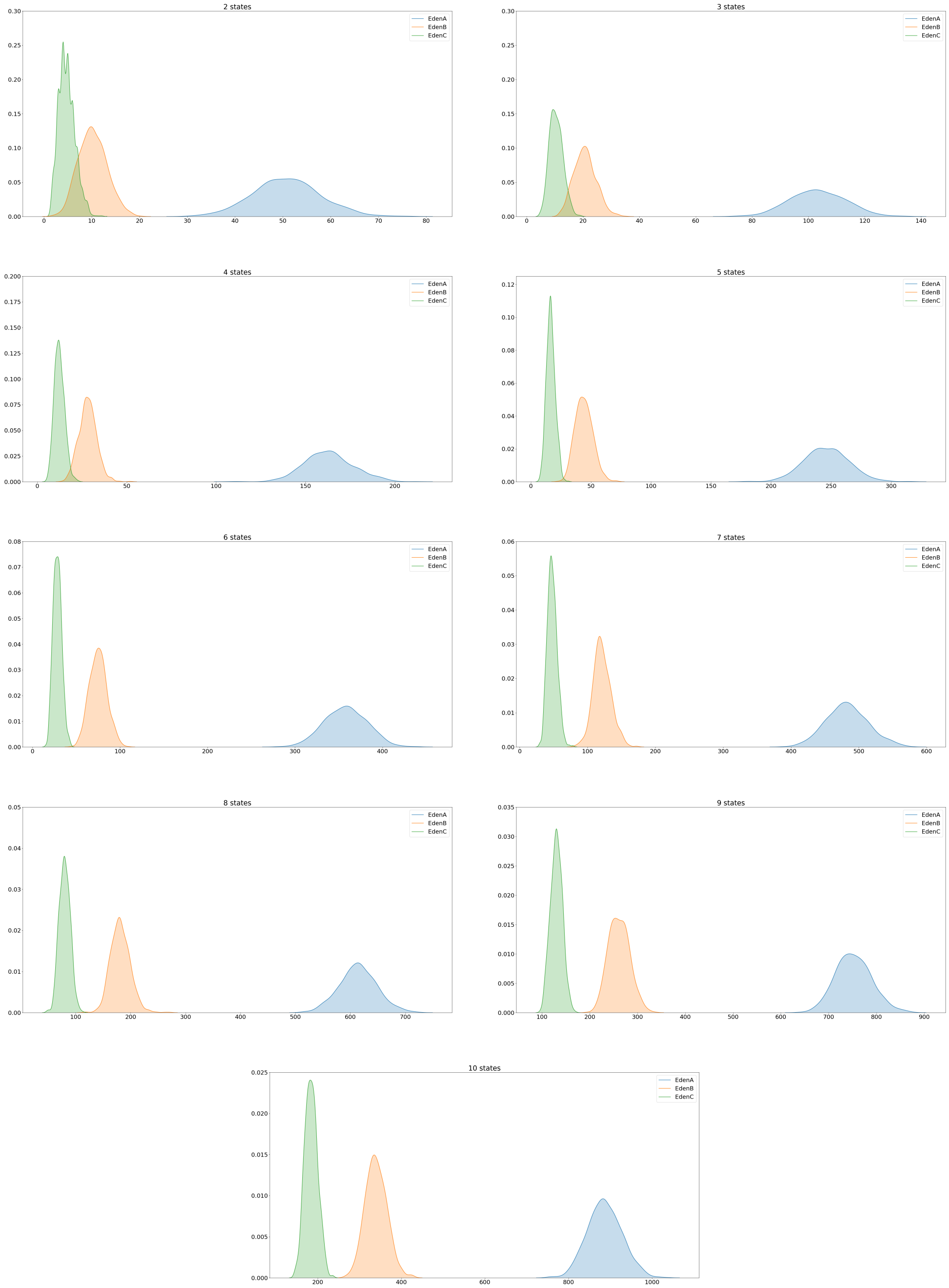


Figure 7.12: KDE graphs for the component count of the EdenA (blue), EdenB (orange), and EdenC(green) for state rangeing from 2 to 10. This graphs show that for a cluster of 5000 cluster it is possible to identify distinct section for the EdenB and EdenC cluster past 7 states. KDE graphs were made using matplotlib pyplot with default settings.

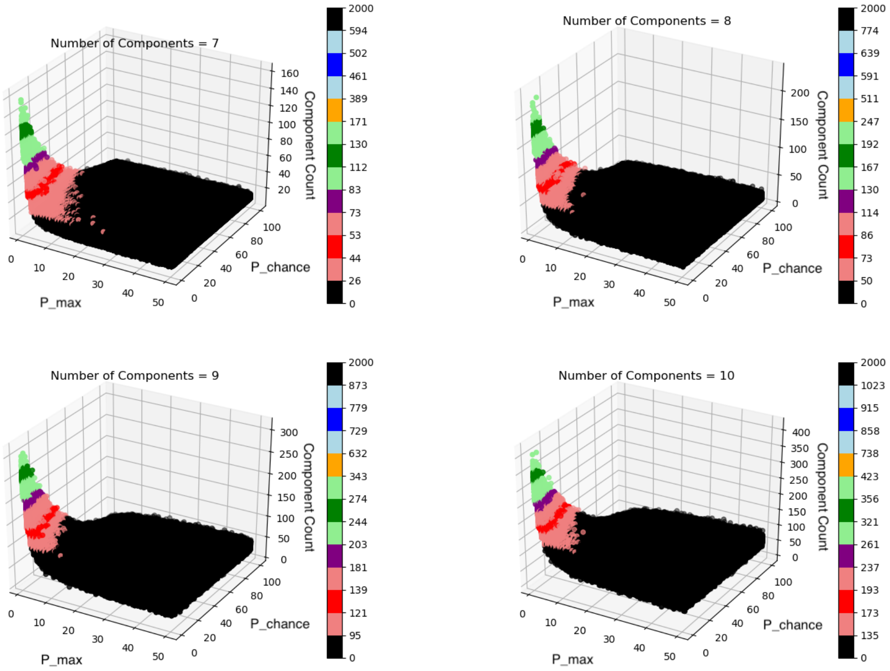


Figure 7.13: RMFC components counts for P_{chance} in the range of 1 to 100 and P_{max} in the range of 1 to 50 showing all the data points gathered for cluster with 7, 8, 9, and 10 state counts. The colours represent the boxplot for the EdenC (red), EdenB (green), and EdenA (blues) where the darker colour represents the interquartile range and the lighter colour represents the whiskers.

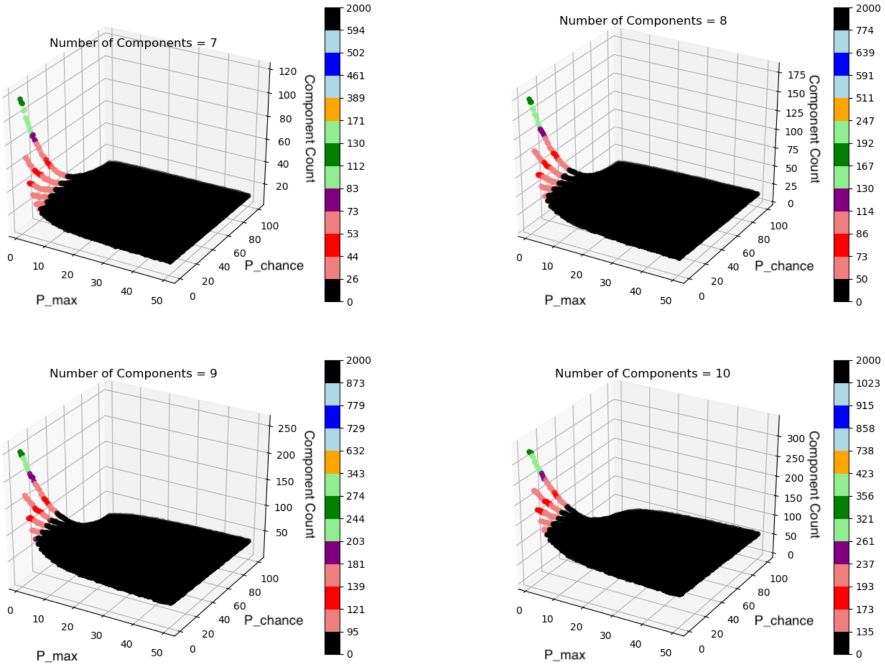


Figure 7.14: RMFC components counts for P_{chance} in the range of 1 to 100 and P_{max} in the range of 1 to 50 showing the median for each point for cluster with 7, 8, 9, and 10 state counts. The colours represent the boxplot for the EdenC (red), EdenB (green), and EdenA (blues) where the darker colour represents the interquartile range and the lighter colour represents the whiskers.

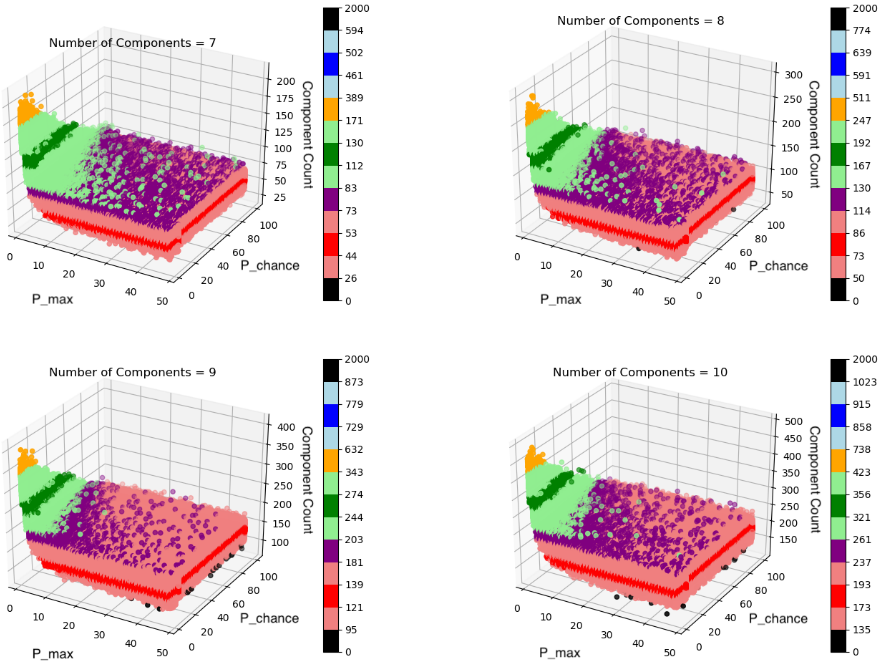


Figure 7.15: RMRC components counts for P_{chance} in the range of 1 to 100 and P_{max} in the range of 1 to 50 showing all the data points gathered for cluster with 7, 8, 9, and 10 state counts. The colours represent the boxplot for the EdenC (red), EdenB (green), and EdenA (blues) where the darker colour represents the interquartile range and the lighter colour represents the whiskers.

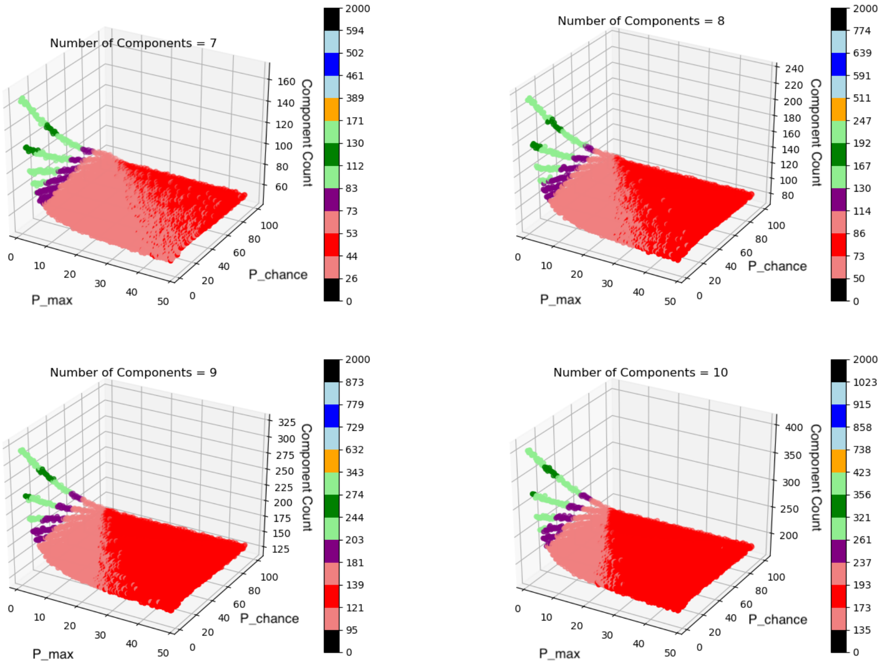


Figure 7.16: RMRC components counts for P_{chance} in the range of 1 to 100 and P_{max} in the range of 1 to 50 showing the median for each point for cluster with 7, 8, 9, and 10 state counts. The colours represent the boxplot for the EdenC (red), EdenB (green), and EdenA (blues) where the darker colour represents the interquartile range and the lighter colour represents the whiskers.

two distinct sections emerge one for each of the clusters with only tiny cross over one the extremes of the tails, points that exist outside of a 99% confidence interval. As the number of states continues past this point to increase the separation continues to increase. The EdenA cluster from the point of 2 states onwards was always distinctly separate from the other two versions making it very easily identifiable with this method. A KDE showing the spread of the component counts for 7 states can be seen in Figure 7.12 along with the graphs for from 2 to 10. This shows that is it possible to make use of component labelling for the classification of individual clusters without the need for multiple runs. This means that if the distribution of the fractal dimension of a new method doesn't match that of the current methods, it would be possible to see what type of clusters it can produce and classify the algorithm in addition to single clusters.

The RMRC and the RMFC version of the MEden model were the two that most closely matched the range of the fractal dimension of the standard Eden model, and as such, they will be the two versions of the MEden model that will be discussed here. The first of these two that will be discussed will be the RMFC Figure 7.13 shows a 3D scatter plot with all the data points gathered for state counts 7, 8, 9, and 10 in the run the model for the same P_{chance} and P_{max} values that where used in the fractal dimension test this being 1-100 and 1-50 respectively with 100 runs for each point and Figure 7.14 shows the median of these 100 runs for each point. The colours that are used on the graphs are to represent the boxplot of the three different Eden models component counts with the darker colour representing the interquartile range and the light colour the whiskers of the boxplot these boxplots are shown in Figure 7.11. The red is for the EdenC, green is for the EdenB, and finally blue is for the EdenA.

The RMFC version of the MEden model whereas in the fractal dimension test appeared to be producing clusters that are very close to that of the EdenB and C model, when the component labelling data is looked at this is not the case there is only a very small range of variables that are able to produce something close to what could be considered to be an Eden model what this means is that the way in which the clusters that grow are not similar to that of the Eden model, and they grow too uniformly to be classed as Eden Models. This would have been a very time-consuming task to be able to identify without the use of this method and might not have even been possible. The RMRC version, however, is a different story.

Figures 7.15 and 7.16 show the same data as Figure 7.13 and 7.14 do but for the RMRC version of the MEden model instead, with all the colouring representing the same things. The RMRC version of the MEden model is much more able to recreate the same component ranges of the Eden model with even in some case it being able to generate clusters that could even be considered to be on the upper end of the EdenB and lower even of the EdenA this can be seen in Figure 7.15 represented with the yellow data points that pop up when the P_{chance} and P_{max} are set close to 1. Unlike with the RMFC version of the MEden model as these two variables increase it is much more able to produce EdenC clusters.

7.3 Image Classification

The final method for the classification of these model will be discussed is that of image classification or image recognition. There is a range of applications that use this type of image classification, they are used to different levels of specificity, for example, it is now possible for these methods to be used to generate a sentence that describes what is going on within an image (Karpathy and Fei-Fei, 2017). Though the method that will be used here is much simpler than this and will involve the classification of growth clusters into one of three categories EdenA, EdenB, and finally EdenC not into sentences.

There are a number of different methods within topic image classification (Nath et al., 2014) that are used, the specific one that has been selected for use here is called a CNN[4] which is an example of a SIANN[5]. CNN's have managed to prove themselves to be a highly effective solution for this problem (Alpaydin, 2016, Krizhevsky et al., 2012, Simard et al., 2003). A CNN is an example of what is called a deep feed-forward neural network this means that the connections with the network do not form any cycles, they also fall under the umbrella of Deep Learning (Bengio, 2012, Lecun et al., 2015).

Deep learning is a class of machine learning that has become highly popular in recent years, it is used to do things including but not limited to automated driving, medical research (detection of cancer cells (Sirinukunwattana et al., 2016) and even in drug discovery (Chen et al., 2018)), and even in the speech recognition such as that used by devices such as the Amazon Echo and the Google Home. The deep in deep learning refers to the number of hidden layers in the network, in deep learning, it is common for a network to have as many as 100 layers as opposed to 2 or three which is more common in traditional neural networks. As such deep learning not only requires a large amount of data for it to be trained on, but it also requires a large amount of computing power which is why it has only gathered so much traction in recent years. Additionally, because of tools such as TensorFlow (Abadi et al., 2015) and PyTorch (Paszke et al., 2017), it is now straightforward for people to write and make use of these complex systems.

CNN's were selected specifically for the image classification here due to there ability to classify complex images such as being able to identify shop fronts in images of cities with ease, this is due to the fact that a CNN can learn specific patterns that certain class of images have and learn filters that can identify these patterns this allows the network to generalize from the past set of images that the network saw during the training process of the network, this allows the network to identify images that is has never before seen. Additionally, there is the minimal amount of preprocessing that the method requires. This section will begin with a brief description of what a CNN

[4] Convolutional Neural Network

[5] Space Invariant Artificial Neural Network

is and how it is structured, in addition to how they function this will then be followed with a discussion on the tools used to implement the CNN along with the code for the specific CNN used. CNN's are a form of machine learning which is a potent tool as Alypaydin says:

> *"we use machine learning whe we believe that there is a relationship between observations of interest, but we do not know exactly what"* (Alpaydin, 2016)

This is exactly the situation that presents itself here with three classes of Eden clusters which we can identify as being different even though they have a high level of visual similarity. Without a decent level of experience with these models which can take years to obtain, it can be difficult to tell the difference between an EdenB or C cluster even when they are side-by-side and nearly impossible when they are shown individually. However, it is possible to train a neural network to make expert level decision in a fraction of the time a human might need to learn.

The first step in understanding what a CNN is in the gain an understanding of what an NN or neural network is. A neural network often referred to as an artificial neural network is a machine learning technique that is inspired by the biological network with a brain and how it can change in order to learn and adapt these systems are used to solve problems in the same way that the human brain would. A NN is made up of a collection of nodes commonly referred to as a neuron, these neurons are arranged into sets of layers with connections between them that allow one neuron to communicate to another these connections are often referred to as synapses a diagram describing this can be seen in Figure 7.17.

The main part of the neural networks functioning goes on within what is called the hidden layers; this can be seen represented by the purple node in Figure 7.17. The purpose of these hidden layers is to receive a piece of data most commonly this is in the form of a real number then performs some form of calculation on all the values

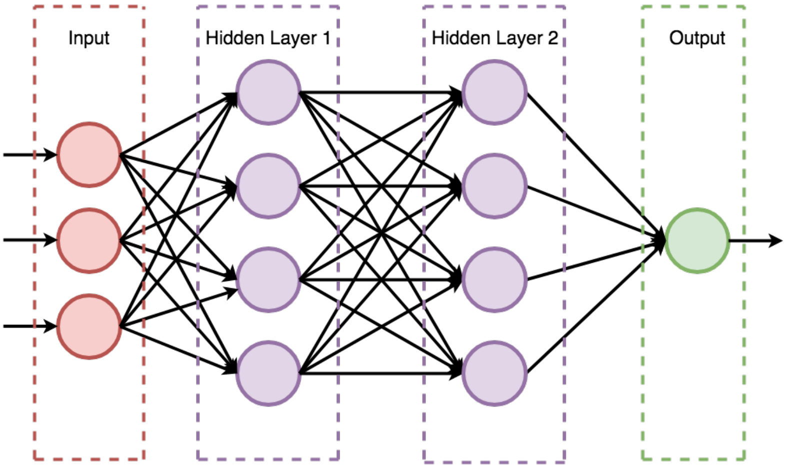


Figure 7.17: This diagram shows an example of a neural network with 3 input neurons, two hidden layers each containing four neurons, and finally a single output. This is an example of a multilayer perceptron.

that are fed into it, and then an activation function is applied to be that through the use of some non-linear. The product of this calculation the continues to flow through the network until it finally ends up at the output layer which will perform the final calculation and output a result. The number of output neurons would not be a fixed value for example if you had a neural network whose job was to identify numbers in the range 1 to 9 that have been handwritten and fed into the network. There could be nine output neurons, and each neuron would spit out the chance that the input was that specific value and the one with the highest probability would be the network's selection.

The name hidden layer comes from the fact that the output of these neurons is unknown. This is because it is connected to the input of a consecutive hidden neuron or to the output neuron which itself will perform some calculation that obscures the initial input. Now this is not to say that it is impossible to find out what is going on with the network, if the program that is running the neural network is bespoke code all handwritten from scratch it is easily possible to insert logging layers into the network so that the values passed between neurons in these hidden layers can be obtained. However, due to the amount of time and effort that this would take to code it all from scratch most researchers and developers are opting to use API tools such as Keras (Keras, 2018), as we have here, to allow them to build a neural network as quickly as possible. It becomes a highly difficult task to looking into these hidden layers, as API like Keras act somewhat like a black box.

The API used for the implementation of the CNN used here was Keras. It is able to have one of three different backends; these are TensorFlow, Theano, and CNTK. For this work TensorFlow was selected as it appeared to be the most commonly used one of the three and as such have the widest range of information that was easily accessible, making the implementation a much faster process. Though using Keras does greatly reduce the amount of code needed to be written for the implementation of the CNN this is at the sacrifice of the ability to fine tune all parameters offer by Tensorflow.

CNN's are very similar to the standard NN in that they take an input, in this case, an image and pass this through multiple hidden layers until it finally reaches the output layer and a result is spat out. Though they differ from NN's with the addition of one key aspect, convolutional layers. These layers within exist the hidden layers section of the network and are where CNN's get there name from. It is these convolutional layers within the network that gives the CNN its advanced pattern detection capabilities, which make them perfect for image classification.

The convolutional layers act in a very similar way to the previously described hidden neurons, with them receiving an input transforming it and then outputting the transformed data the difference is in what type of transformation goes on with the neurons of the layer. Convolution layers get their names from this different type of transformation that goes on within them, a convolution. When the data is passed to the convolutional layer, each node within this layer will run a different filter over this image to transform it in some manner. The purpose of these filters is to identify specific features within the data, each of these filters will be looking for different features such as edges, squares, or circles to name just a few. As the data descends deeper into the network and the amount of data has been greatly reduced it is possible for filters to be able to identify more complex aspects of the data, in the case of a picture of a

$$relu(x) = \begin{cases} x, i \geq 0 \\ 0, i < 0 \end{cases} \tag{7.2}$$

$$leakyrelu(x) = \begin{cases} x, i \geq 0 \\ (\alpha \times x), i < 0 \end{cases} \tag{7.3}$$

human this could be an ear for example. With CNN's one image becomes many filtered images equal to the number of nodes in the layer making them very memory intensive, this gives an idea of why they have only become commonplace in recent years.

In order to understand the CNN that is used here, there are four features in addition to the convolutional layers that should be discussed, these are activation layers, pooling layers, dropout layers, and fully connect layers. The first of these is the Fully connected layers. These are the simplest of the features to understand as these are standard neural network layers where every layer where every neuron connect to one in the consecutive layers.

Next feature that will be discussed is the activation layer; they layer applies an activation function to the data that is past to it which in this case is the ReLU[6] function or Rectified Linear Units function. An activation function is a form of normalization and is used here for a straightforward purpose, to stop the maths going on within the network from exploding. Only non-linear activation functions can allow the network to compute complex nontrivial problems with minimal numbers of nodes, example of other activation functions are TanH, Logistic Sigmoid, and Softmax (which is used at the end of the network to extract the selection of the network) through the most common one in use is the ReLU function. ReLU works by setting all values less the zero to be zero this helps speed up the training of the CNN. It is similar to the stp function discussed in the previous chapter and can be seen in a more mathematical

[6] Rectified Linear Unit

form in Equation 7.2.

ReLU does have one problem that has been called the dying ReLU problem this is when the output of the ReLU function is 0 which happens when the function has a large negative bias; this means that the gradient will always be 0 and this will cancel out the error value that is back-propagated. What this means is that earlier layers in the network can't work out the error; this has been attempted to be fixed by a method called the Leaky ReLU (Xu et al., 2015). In Leaky ReLU a small gradient allowed for the values that previously would have a zero gradient, this can be seen in Equation 7.3. The α value is very commonly set as 0.01 though it is possible for this value to be different and even to be trainable by the network when this is the case it is often referred to a Parametrized ReLU or Trainable Parametrized ReLU respectively. This issue wasn't encountered here and as such the Leaky ReLU method was not needed.

Thirdly, a dropout layer is use in order to help deal with overfitting and only comes into effect while training the network. Overfitting is when a function is fit too close to the training data making so even though in training the network has high accuracy when the network is presented with unseen data is it unable to achieve comparable accuracy at classifying it. There is also such a thing as underfitting, but this is discussed much less frequently then overfitting as underfitting is where the network gets a very low accuracy with the training data, and this means that the design of the network is most likely inappropriate for the job it is being used for. Dropout functions aid in the reduction of overfitting by randomly selecting a fraction of the input units as setting them to 0 each update during the training of the network. This stops the next learning a specific feature of your training set too closely forcing it to look at different aspects of the image in order to classify it making the network more general within the domain.

Finally, there is Pooling; this is a significant aspect of CNN's as it is how the network can deal the vast amounts of data that can be generated from the output of convolutional layers as well as to help stop the network becoming overfit. The functioning of these layers is as such, if you have images with 128x128 as is the case here, if 500 features have been learned over 5x5 inputs each convolution results in an output size of $(128-5+1)^2 = 15376$ and with the 500 features this mean $124^2 \times 500 = 7,688,000$ which can lead to a number of issues. This is where pooling comes in, pooling works by taking a window of a certain size and scanning over the image, this scanning is based on a stride value which should never be greater than any of the dimensions of the window to ensure that nothing is missed.

There are two main forms of pooling mean pooling (sum pooling can also be used with a very similar effect to the mean pooling) and max pooling, in mean pooling all the values in the window are averaged, this new value is used to represent all the cells in that area so with a 3×3 window 9 values would become 1. Max pooling is very similar only instead of taking the average the method extracts the maximum value within the window. Max pooling is better suited to the extraction of more features such as edges but does end up discarding a large portion of the data whereas with average pooling all the data within the window is taken into account and it produces a smoother representation, one is not better than the other they just better suited to specific situations.

7.3.1 Network Design

The network that was implemented to perform the experiments here can be seen in Figure 7.18 and a summary of the model including a parameter brake down can be seen in Figure 7.19. The network consists of 3 main sections where a convolutional layer making use of the ReLU function is followed by a pooling layer making use of the max pooling method; this is finally followed by a dropout layer with a dropout percentage of 25. There are three of these stacks the first containing 32 filters and the second and third having 64 filters each. The final part of this network is a single fully connected layer with 64 neurons within it, the dropout level used in this final part of the network is 50%. The softmax function was used in the final layer for the

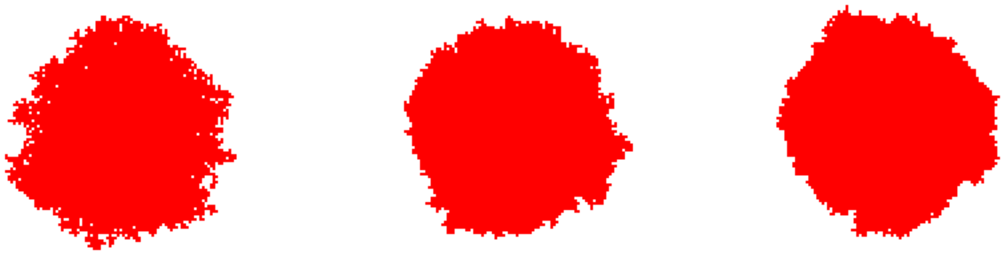
```
model = Sequential()
#===============================================================
model.add(Conv2D(filters=32, kernel_size=(3, 3), input_shape=input_shape, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
#===============================================================
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
#===============================================================
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
#===============================================================
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(numClasses, activation='softmax'))
```

Figure 7.18: This is the keras code that was used to build the Convolutional Neural Network that was used for the classification of the Eden and MEden clusters.

Layer (type)	Output Shape	Param Count
$conv2d_1$ (Conv2D)	(None, 128, 128, 32)	896
$maxpooling2d_1$ (MaxPooling2D)	(None, 64, 64, 32)	0
$dropout_1$ (Dropout)	(None, 64, 64, 32)	0
$conv2d_2$ (Conv2D)	(None, 64, 64, 64)	18496
$maxpooling2d_2$ (MaxPooling2D)	(None, 32, 32, 64)	0
$dropout_2$ (Dropout)	(None, 32, 32, 64)	0
$conv2d_3$ (Conv2D)	(None, 32, 32, 64)	36928
$maxpooling2d_3$ (MaxPooling2D)	(None, 16, 16, 64)	0
$dropout_3$ (Dropout)	(None, 16, 16, 64)	0
$flatten_1$ (Flatten)	(None, 16384)	0
$dense_1$ (Dense)	(None, 64)	1048640
$dropout_4$ (Dropout)	(None, 32)	0
$dense_2$ (Dense)	(None, 3)	195

Total params	1,105,155
Trainable params	1,105,155
Non-trainable params	0

Figure 7.19: This is the results of the summary function built into keras this shows the number of parameters in the network as well as the structure of the network its self. The None value appearers often throughout the list this value this is to represent the batch size of used in the network but this is set to None as this means that the batch size is not fixed and makes the training of the model simpler.



(a) Eden-A **(b)** Eden-B **(c)** Eden-C

Figure 7.20: Examples of the images used in the training, validation, and testing of the convolutional neural network used for the classification of Eden clusters. The cluster a one solid colour as this means that the CNN doesnt have the infection step infection. The idea behind this is to see if the shape of the clusters themselves are distinct enough to be identified via a CNN. As can be seen in these images the difference between these clusters is minimal especially in the case of the EdenB and the EdenC model where it can be difficult for even the human eye to be able to tell the difference between them even when they are correctly labelled as they are here.

classification. This thesis does not claim that this is the optimal design of a network for this task the purpose here is to prove that CNN can be used in a highly effective manner for this purpose.

Training is an integral part of the process this requires three things a training set, a validation set, and a test set. The training set and validation set is used in the training process the training set it what the neural net is trained against, the validation set is used to attempt to test how generalisable the network is and help in the avoidance of overfitting. After training is completed, it is a good idea to test the network against a test set which will be a collection of data that the neural network has never seen to ensure that it is still effective at identifying these images as has not become overfit, this will show in the network has become overfit to the training and validation data. If the network has a high accuracy on the test data as well as in training, then it can be assumed that the network is effective. The training set used here contained $30,000$ images with $10,000$ images per class and then $9,000$ images for the validation set with $3,000$ images per class.

Examples of the data used in the training validation and testing of the CNN can be seen in Figure 7.20. These examples of the Eden clusters also go to show how difficult it can be for even for a human to classify them without reference to other clusters at hand. Unlike with previous 2D Eden cluster shown these clusters consist of one colour, this was done to minimize the number of features that the network would have to tell the difference, as it has already been demonstrated that it is possible to be able to identify Eden clusters with the additional state information with methods such as component labelling.

7.3.2 Results

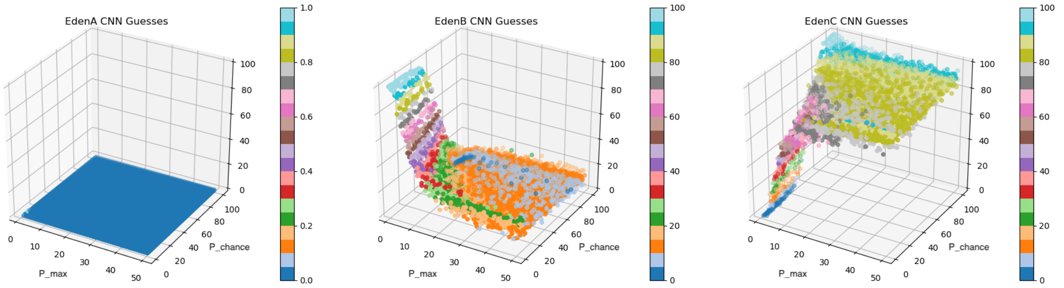
The first step in the testing of the neural network was to make sure that it hadn't become overfit during the training. Most of the time when dealing with neural networks the most difficult part of the whole process is the obtaining of the data for training validation and for testing, because of this it's very common to see that people will take all of their data and split it into three for training, validation, and testing. A common split is a $6/2/2$ split meaning, 60% of the data be used for training, 20% be used for validation with the final 20% being held in reserve for testing the network. However, as the data used here was computationally generated, this split was not needed as it was possible to generate more data as and when it was needed. The validation set used was just under a third of the size of the training set. For the final testing $150,000$ where used to give the best possible idea of the network's effectiveness with $50,000$ images coming from each of the three different versions of the Eden growth model.

In order to represent the effectiveness of the network, a useful tool is a confusion matrix, the confusion matrix for the trained network used here can be seen in Table 7.2. This matrix shows that the network is highly capable of identifying the difference between the three different clusters with it being best at identifying EdenA like clusters, this makes sense as the EdenA cluster is very different from the other two clusters with a significant difference especially when looking at it fractally. When it does make a mistake on this, however, it is only classified as an EdenB this also makes sense as when the fractal dimension of the clusters was tested there was a small level

n = 150,000	Predicted: EdenA	Predicted: EdenB	Predicted: EdenC
Actual: EdenA	49999	1	0
Actual: EdenB	0	49185	815
Actual: EdenC	0	23	49977

Overall Acurracy = 99.44%

Table 7.2: Confusion matrix for the CNN used for the classification of the Eden A, B, and C model. This shows the nerual network tested against 150,000 completely unseen EdenA, EdenB, and EdenC images with the being 50,000 images for each of the classes. This show that a highly effective network but has more trouble with the classification of EdenB cluster then it does with the EdenA and EdenC clusters.



(**a**) EdenA guesses (**b**) EdenB guesses (**c**) EdenC guesses

Figure 7.21: Graphs showing the percentage of clusters formed by the RMRC model that are classified as EdenA, EdenB, and EdenC for a range of different P_{max} in the range 1 to 50 and P_{chance} in the range of 1 to 100, both in increments of one.

of overlap. The network has the most trouble with the identification of EdenB clusters, but this still gives a 98.3%. This means that for every decision that the network makes there is a 99.44% chance that it is the right choice which is a 66.11% better chance then if it was guessing at random, this also give a Cohen Kappa of 0.992 with a 95% interval of 0.991 to 0.992. This shows that the network is highly effective in the classification of Eden-like clusters and confirms that there is a significant visual difference between the three types of clusters. This also shows that this network can be effectively used in the classification of the clusters generated through the use of the MEden method.

Now that the network itself has been tested and shown to be highly effective in the classification of unseen data it is possible to use it to classify cluster the where generated by the MEden method of Eden simulation. Just as with the previous sections in the chapter the MEden model will be run for a P_{chance} from 1 to 100 and a P_{max} from 1 to 50 both of which will increment in step of 1, this will allow for a surface analysis the results of the model. Each of the variable settings was run 100 times, and the results of the runs classified by the neural net. Figure 7.21 shows the percentage of each of the runs that resulted in either an A, B, or C classification these graphs are the results for the RMRC variant of the MEden model. These graphs conform to the general pattern produced by both the analysis of the fractal dimension and the analysis of the component labelling. This shows that as both or either of the P_{max} or P_{chance} increase in values the closer towards the EdenC the system tends towards to. With a P_{max} of 1 the system will produce clusters that the network would classify as EdenB, up until the point that P_{chance} gets too large. This method doesn't tell whether a cluster is on the lower end of the EdenB model it is only able to tell that it fits in the range of the EdenB cluster.

7.4 Timing

Timing is not the focus of this work, as it is commonly known that a hardware-based implementation will be inherently faster than a software-based approach. However, it is essential to discuss the run time of the model as if the speedup could be considered negligible then it might not be worth going to the expense of fabricating such a system. This section, therefore, aims to demonstrate that this particular method does grant a substantial speed up making it worthwhile. There will be two different forms of timing that will be discussed here, that of the theoretical runtime for the model if it were to be fabricated which will be referred to as the hardware runtime. The second is the run time for the simulation used for testing the MEden model and its architecture; this will be referred to as the software timing.

As in the other sections in this chapter, the first step is to gather the data for

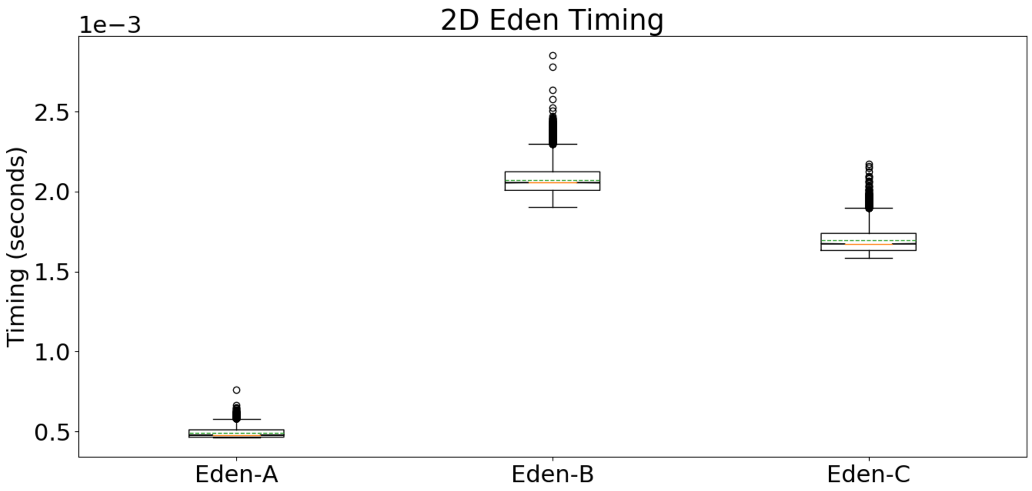


Figure 7.22: Graph showing the results for the run time of the standard software implementation of the EdenA, EdenB, and EdenC models.

the runtime of the three standard Eden growth models so that this can be compared against the MEden method. The results of these experiments can be seen in Figure 7.22, here you can see that the EdenA is the fastest with a full range of between 4.61×10^{-4} and 7.68×10^{-4} seconds but this is with a very considerable positive skew and a median of 4.47×10^{-4} giving a skew of 1.298, this means that it is much more likely that a run of the Eden model of fall in the lower end of this runtime. The EdenB model is the slowest of the three models with a minimum run time of 1.903×10^{-3} and a maximum runtime of 2.85×10^{-3}, but it too has positive skew with a median of 2.321×10^{-3} meaning a skew of 1.18. EdenC is the final of the three models and is the second fastest of the three models with a range of 1.55×10^{-3} and 2.17×10^{-3} seconds and a median of 1.67×10^{-3}. The focus here will be on the EdenB and C model as it has been demonstrated in the previous sections of this chapter these are the only two versions of the Eden model that the MEden method can replicate consistently with it only being able to get the lowest of the EdenA cluster with a very low P_{max} and P_{chance} very rarely.

The first set of timings concerning the RMRC model to be discussed will be the software timing. It is not important that this is slower than the software version of the standard Eden model as it is the timing for a software simulation of the hardware-

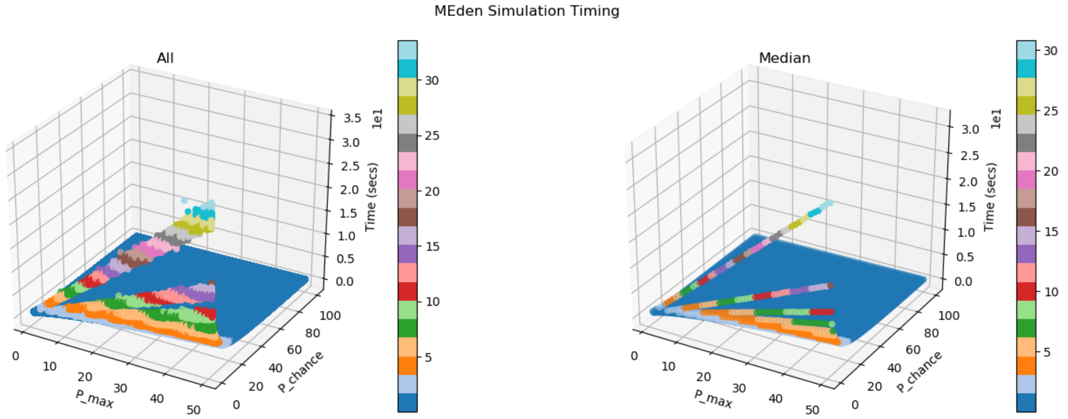


Figure 7.23: These graphs show the full range of the simulation run time for the MEden model. The left graph shows all the 100 runs for each of the data points and the graph on the right shows Median run for each of the data point.

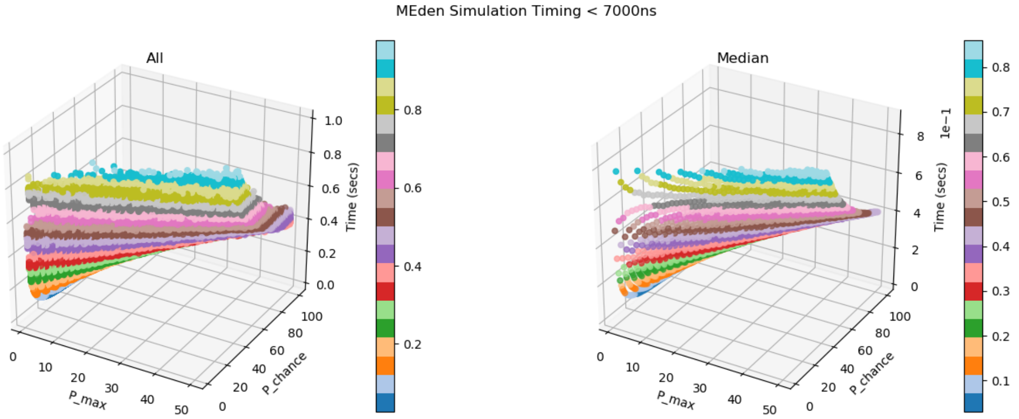


Figure 7.24: These graphs show all the data points of the software simulation MEden model where the theoretical runtime is less than $7\mu s$ this is based on the number of iterations that are required to grow the model in the simulation. The left graph shows all the 100 runs for each of the data points, and the graph on the right shows Median run for each of the data points.

based system and as such is expected to be slower. The results for this can be seen in Figure 7.23 where the full range of data is shown and Figure 7.24 where all the points whose theoretical timing is less the 7×10^{-6} seconds. This ranges from a maximum runtime of 0.977 seconds to a minimum of 0.0267 seconds; this is considerably slower than even the slowest of the standard software Eden model which took 2.85×10^{-3} seconds. This shows the general effect that the different P_{max} and P_{chance} setting can have on the runtime of the simulation with it increase as the P_{max} and increases and the P_{chance} decreases, which is to be expected.

Next is the hardware timing of the RMRC model, this will be compared to the Eden models timings to give an idea of the possible theoretical speed up that this method could offer. The first step is to understand how long it takes the model to perform a single update; there are two different states that the MEden model can exist within. These are called the Pulse and Update states both of them are designed to follow one another repeating until the stopping condition has been met, this could be one of many things such as touching the sides of the domain or infecting a certain number of cells as with the standard Eden model. Both the Pulse and the Update states last for 1 nanosecond; this time will allow for the updating of the cells to be performed easily as it is based on timings used for RRAM (Sakib et al., 2016). This means that a single time step of the MEden model should take around two nanoseconds to complete theoretically.

Figure 7.25 showing the results for the full range of variable test and Figure 7.26 showing the run that have a runtime of less than 7×10^{-6} seconds. The was done because most of the points on the graph fall under this boundary 79.86% to be accurate and it can be difficult to tell if there is any difference between these points in Figure 7.25 with most of them being the dark blue points. Interestingly enough it is possible to the RMRC model to produce both EdenB and C model within the less than 7×10^{-6} seconds range.

With the full range of data, the slowest run took a total of 6.998×10^{-6} seconds this was with $(P_{max} = 50)$ and $(P_{chance} = 1)$, the cluster that was produced could be classified as an EdenC cluster. This works out to be approximately 221 times faster than the fastest run of the EdenC model as shown in Figure 7.22. The fastest run of the RMRC model also produced an EdenC like cluster and took only 1.22×10^{-7} seconds which is a massive $12,704$ times faster the fastest of the software EdenC models. These are even faster than the runtime for the fastest of the Eden models the EdenA which has a fastest run time of 4.61×10^{-4}, the slowest of the RMRC model is approximately 66 times faster, and the fastest is approximately $3,779$ times faster than this EdenA

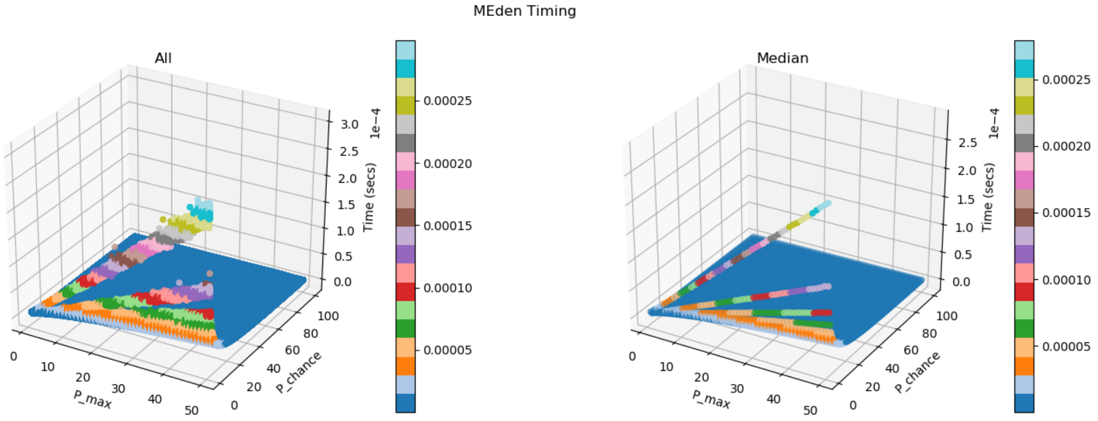


Figure 7.25: These graphs show the full range of the theoretical hardware run time for the MEden model based on the number of iterations that are required to grow the model in the simulation. The left graph shows all the 100 runs for each of the data points and the graph on the right shows Median run for each of the data points.

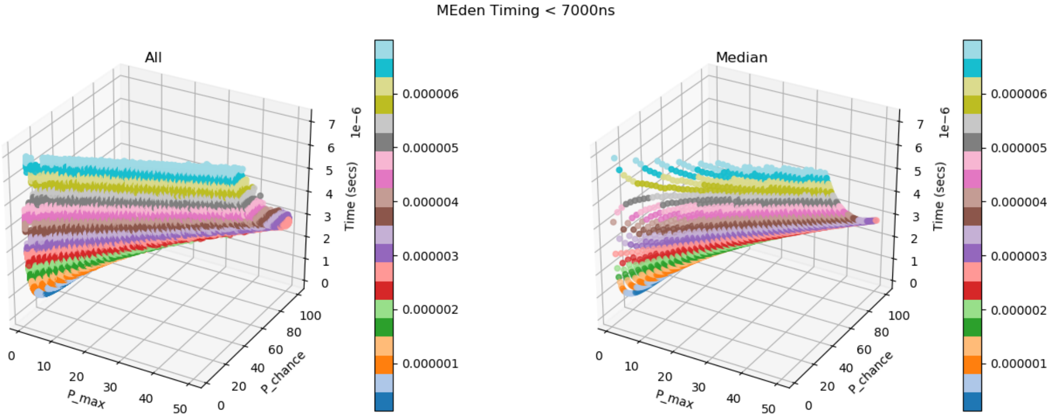


Figure 7.26: These graphs show all the data points of the theoretical hardware MEden model where the runtime is less than $7\mu s$ this is based on the number of iterations that are required to grow the model in the simulation. The left graph shows all the 100 runs for each of the data points, and the graph on the right shows Median run for each of the data points.

run.

As shown in the previous sections of this chapter it is possible to produce EdenB like cluster with the RMRC mode with a P_{max} of 1 and P_{chance} of $\%$. This would produce the highest end of the EdenB model and would be the slowest settings to produce this cluster. At this specific variable set-up run times range from a maximum time of 9.554×10^{-6} seconds to a minimum time of 7.718×10^{-6} seconds with a median run time of 8.415×10^{-6} seconds. This means that even with the slowest run this method for the simulation of the EdenB model is nearly 200 times faster than the

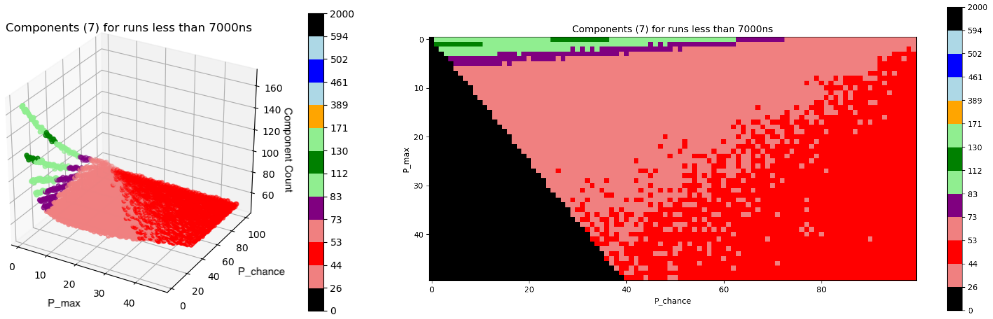


Figure 7.27: These graphs show the component range for the 7 state RMRC model for all the run that took less than 7×10^{-6} seconds. This show that is it possible to get the full range of possible cluster produced by this method with a massive speed increase between 221 times and 12704 times faster than the software version of the EdenB and EdenC models. The image on the right shows a top-down view where the black cells are the cells that have a run time greater than 7000ns.

fastest version of the software EdenB model as shown in Figure 7.22.

With 79.86% of the runs taking less than 7×10^{-6} seconds and with its ability to produce the full range of the MEden models cluster type production within this range which can be seen in Figure 7.22. This shows that there would be no need the step outside of this region in order to obtain the full possibilities from the MEden model this would lead to a minimum increase in speed from the fastest of the EdenC model by 221 times and for the EdenB model of 271 times faster. This method would make the running of the EdenB and EdenC model considerably faster than even that of the EdenA model which as a minimum run time of 4.47×10^{-4}.

7.5 Summary

This chapter has demonstrated that not only is the MEden model capable of producing Eden like clusters that are comparable to the EdenB and EdenC clusters in a fraction of the time that the standard software-based approach is able to do. But is it also able to allow for a more fine-grained selection of the cluster produced through the manipulation of the P_{chance} and P_{max} variables within the system.

It has also demonstrated that whereas the fractal dimension is a useful tool in the analysis and classification of these types of clusters, it is not able to identify clusters with the precision that would be necessary especially in a situation that is present here where the range of the fractal dimensions overlap one another so heavily. To this end, two additional tools for the analysis and classification of growth model clusters have been presented and demonstrated to be effective in the classification of these clusters even with the high level of fractal similarity. These methods being the use of a graph theory tool component labelling and the use of Convolutional Neural Networks.

The final section of the work looked at the possible speed increase (an expected occurrence when implementing a software-based algorithm in hardware) that this model could theoretically offer to the system, as it would be pointless to develop this type of system only for it to end up being slower. With a maximum increase of approximately 12,704 times for the EdenC and a maximum runtime of 7×10^{-6} seconds for both the EdenC and EdenB models. It has also been demonstrated that the full range of clusters producible by the method can be obtained with this limitation of a maximum runtime of 7×10^{-6} this means a theoretical speedup of between 221 and $12,704$ which even on the lower end is a considerable speed-up.

There are no endings, and never will be endings, to the turning of the Wheel of Time.

But is was *an* ending

Robert Jordan
& Brandon Sanderson,
A Memory of Light,
Book 14 of The Wheel of Time

CHAPTER 8

Conclusion

GROWTH models are a highly useful and widely used form of simulation that offers many benefits to humanity as they enable the simulation of highly unpredictable systems such as cancer which can lead to a better understanding of them. The work in this thesis has explicitly focused on two growth models, the Eden Growth Model and the Invasion Percolation Model, which are used in the simulation of cancer as well as for other purposes. Here they are the primary models studied to investigate methods for the optimisation of the runtime and memory consumption of a software-based implementation of these models through the use of alternate data structures. It also investigates a hardware-based implementation of the Eden growth model which is capable of simulating both the EdenB and EdenC variants with a maximum possible speed reduction of $12,704$ times. Whilst testing the validity of the hardware implementation of the Eden model it became apparent that the current method for the analysis and classification of growth model clusters was not enough to accurately define the cluster that was produced by the hardware implementation, as such two additional methods where developed which leveraged techniques from two different areas graph theory and image classification too help with this issue.

This chapter is the conclusion of the work that has been discussed in the previous chapters. The first section of this chapter will be a summary of the work contained here, which will be broken down into the individual objectives described in Chapter 1 and will mention when in the thesis they can be found. This will then be followed by a discussed on the contributions of this work and then a discussion of some possible future work which could lead on from this thesis.

8.1 Summary of Thesis

The section will discuss the different hypotheses of this thesis and will go into where they were proven or disproven along with details of the evidence that is supplied. The three different Hypotheses which were first mentioned in Section 1.1, are as follows:

H_1) Modern data structures such as the AVL and Hash Table can offer both memory and runtime benefits to growth models.

H_2) A hardware implementation of the Eden model using memristors can simulate the Eden growth model faster than the software-based version, and the same architecture can simulate different versions of the model.

H_3) The use of connected component labelling and convolutional neural network improves that ability to analyse and classify individual clusters of the Eden growth model into the three main classes more accurately than the fractal dimension.

In order to test the validity of these hypotheses, they were broken down into a set of different objectives which could help in guiding the would and give manageable milestones to achieve along the was. These objectives are first shown in Section 1.2, but the will also be listed here before a discussion on them.

8.1.1 H_1

The objectives of this hypothesis are as follows:

O_1) Investigate the domain fill for the various models as the size of the cluster increases.

O_2) Develop an optimal method for the implementation of each of the data structures for the model in clusters the handling of the growth sites.

O_3) Investigate the effect on the runtime when making use of different data structures.

O_4) Investigate the effect on the memory consumption when making use of different data structures.

This set of objectives were aimed at testing the validity of Hypothesis 1. Objective 1 was fulfilled in Section 3.4, this showed that the effects of the domain size on these models could have a variety of effects including in some cases such as with the MIP model increasing the sparsity of the model as a whole. which could lead to a possible significant saving in the memory consumption or runtime.

The fulfilment of Objective 2 can be seen in Chapter 4, the developed algorithms for the data structure are discussed and in Chapter 3 where the methods for the handling of the growth sites list is discussed. Model-specific changes that were made in order to ensure that each of the models were running in a structure tailor-made for it as also discussed one such example of this would be the negative mask for the AVL tree in the case of the EdenA model in order to stop a cell being added to the growth sites list more than once.

Objectives 3 and 4 are discussed within Chapter 5, this showed that the answer for the best data structure is not a simple one. For models such as the MIP where the runtime is relatively short, with the slowest cases the median result is approximately 4 seconds, and the domain fill is very low, that a structure such as the hash table or the AVL tree not only offer benefits to the runtime of the system, but they also offer benefits to the memory consumption of the simulation especially at larger domain sizes.

However, when the runtime of the simulation increases such as with the Screened Eden model the difference in the runtime for higher betas values of the AVL tree does not significantly differ from the Lattice and the Resizing method but the amount of memory consumption of the AVL method is considerable showing that it would be the better choice. This is not always the case as the amount of the domain that is filled by the cluster increases, as is the case in the standard Eden and HIP models which have between approximately 40% and 96% fill, the Lattice method is still the fastest and consumes the least amount of memory. This demonstrates that there are benefits that are available to both the runtime and memory consumption of highly sparse growth models, especially with larger domain sizes and that these difference can be significant especially in the case of the amount of memory that can be conserved.

8.1.2 H_2

The objectives of this hypothesis are as follows:

O_5) Propose a design for a memristive based agent for use in a cellular automaton for the simulation of the Eden model.

O_6) Analyse the fractal dimension of the Standard Eden growth model and compare to that produced by the memristive based Eden growth model.

This set of objectives were aimed at testing the validity of Hypothesis 2. The fulfilment of Objective 5 can be seen in Chapter 6. This Chapter detailed what a memristor is, along with the specific implementation that was used for the simulation of the memristive architecture that was developed. This chapter compares the C++ implementation to a SPICE implementation showing that the C++ memristor can reproduce the effect of a memristor effectively even if not 100% perfectly. Also within this chapter is contained a discussion on the design of the agent that was developed as well as the equations that can be used for the tailoring of the clusters that can be produced by the model.

Objective 6 was to make use of the current method for the testing of a new method for the simulation of the Eden growth model, that being the fractal dimension and to see if it was possible for the MEden method of the simulation of the Eden model was able to produce fractally similar cluster the that of the standard Eden varients. Chapter 7 shows that the range of the fractal dimension that is produced by the RMRC and the RMFC versions of the MEden model can produce clusters that are fractally similar to that of the EdenB and EdenC methods; they are however not able to produce EdenA clusters; which is a limitation of this particular method. This demonstrated that this method was able to produce clusters that by the current method would be classed as Eden clusters and that it was able, through the manipulation of the two variables P_{max} and P_{chance}, to tailor the specific type of cluster that was produced from the high end of the EdenB model to the bottom end of the EdenC model giving flexibility to the system that is not present in the current algorithms, widening the styles of clusters that can be produced by a single system.

Unfortunately due to a lack of access to physical memristors the testing of this method had to be done in silico. Though within Section 7.4 it was possible to demonstrate that the theoretical speed up that could be obtained from the use of this method ranged from 221 to $12,704$ times faster than the simulated versions of the Eden, growth model. This speed-up is only theoretical and in order to be able to state for sure that this is the range of speed increase that can be obtained through the use of this method this architecture would have to be fabricated. Though this shows that there is a sizeable possible benefit in the simulation of Eden clusters through this method and that the fabrication of this system could be a worthwhile endeavour.

8.1.3 H_3

The objectives of this hypothesis are as follows:

O_7) Design connected component labelling for the analysis of the Eden growth model.

O_8) Test the validity of the connected component labelling for the analysis of the Eden growth model.

O_9) Design convolutional neural network for the analysis of the Eden growth model.

O_{10}) Test the validity of the convolutional neural network for the analysis of the Eden growth model.

This set of objectives were aimed at testing the validity of Hypothesis 3. They are discussed in Chapter 7. Objective 7 was to develop a CCL algorithm that would work with the Eden growth model, setting the state of the cell based on the infection step; this is discussed in Section 7.2. The first step of Objective 8 was to get the component count for the three standard Eden growth models for a range of different state counts in order to see if it was possible for them to separated, so it was possible to identify individual clusters within these ranges. It was found that from a state count of 7 and up the ranges of the component counts for the EdenB and EdenC where distinct. This meant that is possible to identify individual clusters produced by these methods. This was then compared to the component counts of the RMRC and RMFC version of the MEden model showing that the clusters produced by the RMRC version of the MEden model where able to match this component count range however not the RMFC showing that just because they are fractally similar doesn't mean they are the same.

The first step in Objectives 9 and 10 was to develop a CNN that was able to distinguish between a different cluster of the Eden model that was produced by the software method Section 7.3 shows that the CNN developed was able to do this with an accuracy of 99.44%; this showed that this method was highly useful in the identification of these clusters even on an individual level unlike that of the fractal dimension. When this method was used to test the clusters produced by the MEden method it showed that the same trend that was present in the connected component labelling was also present here with the lower P_{max} and P_{chance} values producing more EdenB cluster and with the higher P_{max} and P_{chance} producing more EdenC like clusters.

8.2 Future Work

The work in this thesis is not the end of the work that can be performed in the areas discussed; many possible projects can emerge from the work that is shown here. The first of these pieces of future work is to develop an asymmetric domain resizing method for the Screened Eden model as it could offer significant benefits to runs of the model especially those that make use of larger beta values. It would then be possible to develop a method that can switch between the two different resizing methods depending on the beta value that is input into the simulation allowing for the memory consumed by the system to be minimised as well of the runtime of the system. This would allow for a single computer to be able to run more instances of a single model in parallel making better use of the resources that are available and speeding up future research that makes use of models such as this.

Next is the development of a hybrid data structure management system that can allow for in the case of a model such as the MIPs method to start off making use of a data structure such as the AVL or HSH method which is considerably quicker to construct, while generating the lattice structure in the background. Once the lattice is constructed and while the simulation is running the system would copy all the data from one to the other and then switch to using the lattice or whichever structure was decided to be best at that point. It is possible that a method such as this could be developed in such a way that it could balance out memory consumption with runtime minimising any signification impact one may have on the other. This method could have significant benefits to the Screened Eden growth model especially in cases where the beta value used is high as the amount of the domain filled by the cluster in these cases is very low and this would mean a significant reduction in the amount of memory that would be needed as well as possibly improving the runtime of the model but before this type of claim can be made this would be needed to develop and test the method.

The next set of future works revolves around the MEden method. The first and most evident of these is to fabricate the architecture. Unfortunately, due to budget constraints, it was not possible to do this for this work, and so the simulation discussed in Chapter 6 was developed to test the hypothesis that this system could simulate the Eden model. In order to take this forward, it would be needed first to be able to gain access to enough memristors to be able to build the system on a small scale, for example, on a 4x4 grid and then compare the result of this smaller grid to the simulation and identify any differences between the two. This can then be used to improve the simulation to give a better idea of the effectiveness of this method without the need for a massive monetary investment. The development of this system as shown here could offer significant benefits to the simulation of the Eden model both in terms of the runtime of the simulation but also in terms of the versatility of the model with a single system being able to produce a broader range of clusters than a single algorithm which in the case of a Monte-Carlo style run allows for the simulation to cover more ground.

Next is to investigate whether the Meden model with its current design is capable of being used for the generation of 3D Eden clusters and if it differs in the type of clusters that are produced by this method from when it is used to produce 2D Eden clusters. It is possible that the RMRC version of the MEden model which is the best of the four for the simulation of Eden models in 2D would not be the best for the 3D MEden model. It would also be interesting to investigate the effect that this scaling has on the runtime of the system, as it is feasible that this increase in the dimension count would speed up the simulation unlike with the 3D software version in which it slows down considerably due to the increase in searches; this would be due to the inherent parallelism of this system.

The final set of plans revolves around the analytical tools that have been developed and tested within Chapter 7 for the analysis of the Eden model. This first would be to test the connected component labelling and the convolutional neural network methods with a broader range of models such as the Invasion Percolation model or the models such as the Kawasaki model as there might be limitations to the types of models that these methods can be used for, and it would be essential to find rules that can aid in the decision as to which method to use and when. This would allow for a better understanding of the different methods that are used for the simulation of these systems and allow for them to be more consistent and predictable and could even lead to the identification of different classes of current models thereby expanding our knowledge of these complex systems.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mane, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viegas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems (White Paper). *Prelim. White Pap.*, 2015. ISSN 0270-6474. doi: 10.1038/nn.3331.

Andrew Adamatzky and Leon Chua. Memristive excitable cellular automata. *Int J Bifurc. Chaos*, 21(11):3083, 2011. ISSN 0218-1274.

G. M. Adelson-Velskii and E. M. Landis. An Algorithm for the Organization of Information. *Dokl. Akad. Nauk USSR*, 146(2):263–266, 1962.

Ahmed Al-futaisi and Tadeusz W Patzek. Extension of Hoshen – Kopelman algorithm to non-lattice environments. 321:665–678,

Ethem Alpaydin. *Machine learning: the new AI.* The MIT Press, 2016. ISBN 9780262529518.

Robert Axelrod. The Dissemination of Culture. *J. Conflict Resolut.*, 41(2):203–226, 1997. ISSN 00220027.

Robert Axelrod and William D Hamilton. The Evolution of Cooperation. *Evolution (N. Y).*, 211(March):1390–1396, 2008. ISSN 00368075.

James W Baish and Rakesh K Jain. Fractals and Cancer. *Perspect. Cancer Res. Fractals*, 60(7):3683–3688, 2000. ISSN 0008-5472.

Anu Bala, Adedotun Adeyemo, Xiaohan Yang, and Abusaleh Jabir. High Level Abstraction of Memristor Model for Neural Network Simulation. *Embed. Comput. Syst. Des. (ISED), 2016 Sixth Int. Symp.*,

Daniel Batas and Horst Fiedler. A memristor SPICE implementation and a new approach for magnetic flux-controlled memristor modeling. *IEEE Trans. Nanotechnol.*, 10(2):250–255, 2011. ISSN 1536125X.

Jeroen Bédorf, Evghenii Gaburov, and Simon Portegies Zwart. A sparse octree gravitational N-body code that runs entirely on the GPU processor. *J. Comput. Phys.*, 231 (7):2825–2839, 2012. ISSN 00219991.

F Bellifemine and A Poggi. An object-oriented framework to realize agent systems. *Reports(Centro Stud. e*, 6:6, 2000.

Fabio Bellifemine, Agostino Poggi, and Giovanni Rimassa. JADE–A FIPA-compliant agent framework. *Proc. PAAM*, pages 97–108, 1999.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, 7700 LECTU:437–478, 2012. ISSN 03029743.

C H Bennett. Dissipation, Information, Computational Complexity and the Definition of Organization, 1985.

Charles H. Bennett. Logical Depth and Physical Complexity. pages 207–235, 1988. ISSN 0946-9613.

Charles H. Bennett. Universal computation and physical dynamics. *Phys. D Nonlinear Phenom.*, 86(1-2):268–273, 1995. ISSN 01672789.

Charles H Bennett. *How to define complexity in physics, and why.* Oxford University Press: Oxford, UK, 2003.

Zdeněk Biolek, Dalibor Biolek, and Viera Biolková. SPICE model of memristor with nonlinear dopant drift. *Radioengineering*, 18(2):210–214, 2009.

Eric Bonabeau. Agent-based modeling: methods and techniques for simulating human systems. *Proc. Natl. Acad. Sci.*, 99(suppl. 3):7280–7287. ISSN 0027-8424. doi: 10.1073/pnas.082080899.

Julien Borghetti, Gregory S. Snider, Philip J. Kuekes, J. Joshua Yang, Duncan R. Stewart, and R. Stanley Williams. 'Memristive' switches enable 'stateful' logic operations via material implication. *Nature*, 464(7290):873–876, 2010. ISSN 0028-0836. doi: 10.1038/nature08940.

Donald S Burke, Joshua M. Epstein, Derek A T Cummings, Jon I Parker, Kenneth C Cline, Ramesh M Singa, and Shubha Chakravarty. Individual-based Computational Modeling of Smallpox Epidemic Control Strategies. *Acad. Emerg. Med.*, 13(11):1142–1149, 2006. ISSN 1553-2712.

Tullio Ceccherini-Silberstein and Michel Coornaert. The Garden of Eden theorem: old and new. pages 1–43,

Paul Charbonneau. *Natural Complexity: A Modeling Handbook*. Princeton University Press, Princeton, 1 edition, 2017. ISBN 978-0-691-17684-0.

Hongming Chen, Ola Engkvist, Yinhai Wang, Marcus Olivecrona, and Thomas Blaschke. The rise of deep learning in drug discovery. *Drug Discov. Today*, 23(6): 1241–1250, 2018. ISSN 18785832.

Leon Chua. Memristor-The Missing Circuit Element. *IEEE Trans. Circuit Theory*, 18,

Leon Chua. Resistance switching memories are memristors. *Appl. Phys. A Mater. Sci. Process.*, 102(4):765–783, 2011. ISSN 09478396.

Leon Chua. If it's pinched it's a memristor. *Memristors Memristive Syst.*, 9781461490: 17–90, 2014. ISSN 13616641.

Leon O. Chua and Sung Mo Kang. Memristive Devices and Systems. *Proc. IEEE*, 64 (2):209–223, 1976. ISSN 15582256.

Cppreference. std::vector, 2017. `http://en.cppreference.com/w/cpp/container/vector`. Last Accessed On: 2017-08-02.

Cyril Crassin, Fabrice Neyret, Sylvain Lefebvre, Elmar Eisemann, and Inria Sophia-antipolis. GigaVoxels : Ray-Guided Streaming for Efficient and Detailed Voxel Rendering. *ACM SIGGRAPH Symp. Interact. 3D Graph. Games*, 1(212):15–22, 2009.

Charles Darwin. *The Origin of Species*. P. F. Collier & Son, New York, 1909. ISBN 9780874216561. doi: 10.1007/s13398-014-0173-7.2.

E. R. Davies. Computer and Machine Vision. In *Comput. Mach. Vis.*, pages 229–265. Academic Press, 2012. ISBN 9780123869081. doi: 10.1016/C2010-0-66926-4.

G. Dearnaley, A.M. Stoneham, and D.V. Morgan. Electrical phenomena in amorphous oxide films. *Reports Prog. Phys.*, 33(1129),

Michael B. Dillencourt, Hannan Samet, and Markku Tamminen. A general approach to connected-component labeling for arbitrary image representations. *J. ACM*, 39 (2):253–280, 1992. ISSN 00045411. doi: 10.1145/128749.128750.

Radu Dobrescu and Loretta Ichim. Two dimensional modeling and fractal characterization of tumor vascular network. *Proc. - 2009 Int. Jt. Conf. Bioinformatics, Syst. Biol. Intell. Comput. IJCBS 2009*, pages 483–486, 2009.

Yadolah Dodge. Kolmogorov-Smirnov Test. In *283–287*, pages 283—-287. Springer New York, New York, 2008.

Renato P. Dos Santos. On the Philosophy of Bitcoin/Blockchain Technology: Is It a Chaotic, Complex System? *Metaphilosophy*, 48(4-5), 2017. ISSN 1467-9973.

Christian A Duncan, Michael T Goodrich, and Stephen Kobourov. Balanced aspect ratio trees: combining the advantages of k-d trees and octrees. *J. Algorithms*, 38(1): 303–333, 2001. ISSN 0196-6774.

Fatemeh Ebrahimi. Invasion percolation: A computational algorithm for complex phenomena. *Comput. Sci. Eng.*, 12(2):84–93, 2010. ISSN 15219615.

Murray Eden. A Two-Dimensional Growth Process. *Proc. Fourth Berkeley Symp. Math. Stat. Probab.*, pages 223–239,

Editorial. No man is an island, 2009. `https://www.ncbi.nlm.nih.gov/pubmed/1038652`. Last Accessed On: 2019-04-14.

Nemat H. El-Hassan, M. R.Ahamed Shahad, T. Nandha Kumar, and Haider Abbas F. Almurib. AC sense circuit for Phase Change Memory based crossbar memory. *2015 IEEE Student Conf. Res. Dev. SCOReD 2015*, 1:412–414, 2016.

Joshua M. Epstein. Modeling civil violence: an agent-based computational approach. *Proc. Natl. Acad. Sci. U. S. A.*, 99(3):7243–7250, 2002. ISSN 00278424.

Joshua M. Epstein and Robert Axtell. *Growing Artificial Societies: Social Science from the Bottom Up.* Brookings Institution Press, 1996. ISBN 0262050536.

R. Foote. Mathematics and Complex Systems. *Science (80-.).*, 318(5849):410–412, 2007. ISSN 0036-8075. doi: 10.1126/science.1141754.

Stan Franklin and Art Graesser. Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents. *Intell. agents III agent Theor. Archit. Lang.*, pages 21–35, 1997. ISSN 1752458X.

P Freche, D Stauffer, and H E Stanley. Surface structure and anisotropy of Eden clusters. *J. Phys. A*, 18:1163–1168,

Francis Fukuyama. Game theory, agent-based modeling, and the evolution of social behavior. *Foreign Aff.*, 77(2):142–143, 1998. ISSN 00157120.

Ella Gale. Single Memristor Logic Gates: From NOT to a Full Adder. pages 1–27,

Angel García, Universidad Valle, Joan Vila, Alfons Crespo, and Sergio Sáez. A Binary-Tree Architecture for Scheduling Real-Time Systems with Hard and Soft Tasks Angel. (572),

M Gardner. Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Sci. Am.*, 223(October):120–123., 1970. ISSN 0036-8733.

M Gardner. *Wheels, life, and other mathematical amusements.* W.H. Freeman, 1983.

N. Goldenfeld. Simple Lessons from Complexity. *Science (80-.).*, 284(5411):87–89, 1999. ISSN 00368075. doi: 10.1126/science.284.5411.87.

Ahmed Gomaa, N Khaled, A Mohammed, Radwan Ag, Zidan Ma, Salama Kn, A G Radwan, M Affan Zidan, and K N Salama. Repository On the mathematical modeling of memristors On the Mathematical Modeling of Memristors. 2016.

Antonin Guttman. R-trees. *Proc. 1984 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '84*, page 47, 1984. ISSN 01635808. doi: 10.1145/602259.602266.

Said Hamdioui, Lei Xie, Hoang Anh Du Nguyen, Mottaqiallah Taouil, Koen Bertels, Henk Corporaal, Hailong Jiao, Francky Catthoor, Dirk Wouters, Linn Eike, and Jan van Lunteren. Memristor Based Computation-in-memory Architecture for Data-intensive Applications. *Proc. 2015 Des. Autom. Test Eur. Conf. Exhib.*, pages 1718–1725, 2015.

Said Hamdioui, Siddharth Joshi, Shahar Kvatinsky, Gert Cauwenberghs, Lei Xie, Nimrod Wald, Koen Bertels, Hesham Mostafa Elsayed, and Henk Corporaal. Memristor For Computing : Myth or Reality ? pages 722–731,

K A Hawick. Multiple Species Phase Transitions in Agent-Based Simulations of the Schelling Segregation Model. Technical report, Department of Computer Science, University of Hull, Hull, dec 2013.

K A Hawick. Modelling Flood Incursion and Coastal Erosion Using Cellular Automata Simulations, 2014.

K A Hawick. Simulations of Structural Asymmetries and Dimensional Dependence of Scaling in the Screened Eden Model. In *Proc Int. Conf Model. Identif. Control*, pages 205–212. ACTA Press, 2016.

K A Hawick and C J Scogings. Dynamical Runaway Growth and Simulation of Cancer amongst Spatial Animat Agents. In *IASTED Int. Conf. Appl. Simul. Model. Palma Mallorca, Spain.*, pages 142–147, Palma de Mallorca, Spain, 2009.

K. A. Hawick, A. Leist, and D. P. Playne. Parallel graph component labelling with GPUs and CUDA. *Parallel Comput.*, 36(12):655–678, 2010. ISSN 01678191. doi: 10.1016/j.parco.2010.07.002.

K A Hawick, L R F Odiam, and L A D Stockwell. 3D Printing for Visualisation of the Complex Physical Structures of Agent-Based Simulation Models on Lattices. In *Proc. Int. Conf. Model. Simul. Vis. Methods*, pages 43–49. WorldComp, 2016a.

K A Hawick, L A D Stockwell, and L R F Odiam. 3D Print Technology for Cellular Agent-Based Growth Models. In *Model. Identification, Control*, Innsbruck, 2016b.

K.A. Hawick. *Domain Growth In Alloys.* PhD thesis, Edinburgh University, 1991.

K.A Hawick. Gravitational and Barrier Effects in d-Dimensional Invasion Percolation Reservoir Models. In *Proc. Int. Conf. Power Energy Syst. Appl. (PESA 2011)*, pages 259–266. IASTED, 2011.

Brian Hayes. Nature's Algorithms. *Am. Sci.*, 82:206–210, 1979.

H. J. Herrmann. Geometrical cluster growth models and kinetic gelation, 1986. ISSN 03701573.

J Hoshen. Percolation and cluster structure parameters : The enhanced Hoshen-Kopelman algorithm. 56(2):1455–1460,

Alwyn V. Husselmann. *Data-parallel Structural Optimisation in Agent-based Modelling*. Phd thesis, Massey University, 2014.

AV Husselmann and KA Hawick. Spatial Agent-based Modelling and Simulations - A Review. Technical report, Computer Science, Massey University, Auckland, 2011.

Makoto Itoh and Leon O. Chua. Memristor Cellular Automata and Memristor Discrete-Time Cellular Neural Networks. *Int. J. Bifurc. Chaos*, 19(11):3605–3656, 2009. ISSN 0218-1274. doi: 10.1142/S0218127409025031.

Y V Ivanenko, N I Lebovka, and N V Vygornitskii. Eden growth model for aggregation of charged particles. *Eur. Phys. J. B*, 11(3):469–480 (arXiv:cond–mat/9806345), 1999.

Selinger J. Ising Model for Ferromagnetism. In *Introd. to Theory Soft Matter From Ideal Gases to Liq. Cryst.*, page Chapter 2. Springer, 2016. ISBN 9783319210544. doi: 10.1007/978-3-319-21054-4.

Yu Jiang and Hu Gang. New growth model: The screened Eden model. *Phys. Rev. B*, 39(7):4572–4576,

P Jime, F Thomas, and C Torras. 3D collision detection : a survey. 25, 2001.

Yogesh N. Joglekar and Stephen J. Wolf. The elusive memristor: properties of basic electrical circuits. *Eur. J. Phys.*, 30(Ml):1–24, 2008. ISSN 0143-0807.

R Jullien and R Botet. Scaling Properties of the Surface of the Eden Model in D = 2, 3, 4. *J. Phys. a-Mathematical Gen.*, 18(12):2279–2287,

Oleksandr Kalentev, Abha Rai, Stefan Kemnitz, and Ralf Schneider. Connected component labeling on a 2D grid using CUDA. *J. Parallel Distrib. Comput.*, 71(4):615–620, 2011. ISSN 07437315. doi: 10.1016/j.jpdc.2010.10.012.

Rafailia-eleni Karamani, Vasileios Ntinas, Ioannis Vourkas, and Georgios Ch Sirakoulis. 1-D Memristor-based Cellular Automaton for pseudo-Random Number Generation.

Andrej Karpathy and Li Fei-Fei. Deep Visual-Semantic Alignments for Generating Image Descriptions. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):664–676, 2017. ISSN 01628828.

Hayles N. Katherine. *Chaos Bound: Orderly Disorder in Contemporary Literature and Science*. Cornell University Press, Ithaca, 1 edition, 1990.

Omid Kavehei, Yeong-Seuk Kim, Azhar Iqbal, Kamran Eshraghian, Said F. Al-Sarawi, and Derek Abbott. The fourth element: Insights into the memristor. *2009 Int. Conf. Commun. Circuits Syst.*, pages 921–927, 2009. doi: 10.1109/ICCCAS.2009.5250370.

Kyozi Kawasaki. Diffusion constants near the critical point for time-dependent Ising models. *Phys. Rev.*, 145(1):224–230, 1966. ISSN 0031899X.

Joseph Kehoe. The Specification of Sugarscape. pages 1–80, 2015.

Keras. Keras Documentation, 2018. `https://keras.io/`. Last Accessed On: 2019-04-14.

Aaron Knoll. A Survey of Octree Volume Rendering Methods. *First IRTG Work.*, page 9,

D. Knuth. AVL trees Knuth. In *Art Comput. Program. Vol. 3. Sorting Search.*, pages 459–477. Addison Wesley Longman Publishing Co., third edition, 1998a.

D. Knuth. Red-Black trees knuth. In *Art Comput. Program. Vol. 3. Sorting Search.*, chapter Chapter si, page 477. Addison Wesley, third edition, 1998b.

Donald E. Knuth. *The Art of Computer Programming Volume 3. Sorting and Searching*, volume 3. Addison Wesley, third edition, 1998c. ISBN 0201896850. doi: 10.2307/2005383.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.*, pages 1–9, 2012. ISSN 10495258.

Kiriakos N. Kutulakos and Steven M. Seitz. Theory of shape by space carving. *Int. J. Comput. Vis.*, 38(3):199–218, 2000. ISSN 09205691.

Shahar Kvatinsky, Advisors Eby Friedman, Avinoam Kolodny, and Uri Weiser. Memory Intensive Computing. (January),

Shahar Kvatinsky, Guy Satat, Nimrod Wald, Eby G. Friedman, Avinoam Kolodny, and Uri C. Weiser. Memristor-based material implication (IMPLY) logic: Design principles and methodologies. *IEEE Trans. Very Large Scale Integr. Syst.*, 22(10):2054–2066, 2014b. ISSN 10638210.

James Ladyman, James Lambert, and Karoline Wiesner. What is a complex system? *Eur. J. Philos. Sci.*, 3(1):33–67, 2012. ISSN 1879-4912. doi: 10.1007/s13194-012-0056-8.

Tapobrata Lahiri, Rajeev Kumar, and D.V Rai. Analysis of Cancer Growth using Modified EDEN Model. *Int. J. New Innov. Eng. Technol. Anal.*, 3(2):18–23,

Samuli Laine. Efficient Sparse Voxel Octrees. *IEEE Trans. Vis. Comput. Graph.*, 17 (212):55–64, 2011. doi: http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.240.

S M Lea. Nonlinear Relationships. Technical Report January, University of Notre Dame, Notre Dame, 2015.

Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. ISSN 14764687.

Eero Lehtonen and Mika Laiho. Stateful implication logic with memristors. *2009 IEEE/ACM Int. Symp. Nanoscale Archit. NANOARCH 2009*, pages 33–36, 2009.

Ce Charles E Leiserson, Rl Ronald L Rivest, Clifford Stein, and Thomas H Cormen. *Introduction to Algorithms, Third Edition*, volume 7. MIT Press, 2009. ISBN 0262033844. doi: 10.2307/2583667.

George Henry Lewes. *Problems of Life and Mind*. The Riverside Press Cambridge, Boston, 1875. ISBN 1425555780.

Sean Luke. MASON: A Multiagent Simulation Environment. *Simulation*, 81:517–527, 2005. ISSN 0037-5497.

Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. Mason: A new multi-agent simulation toolkit. *Proc. 2004 SwarmFest Work.*, 8(2):316–327, 2004. ISSN 0037-5497.

Charles Macal and Michael North. Tutorial on Agent-Based Modeling and Simulation PART 2: How to Model with Agents. *Proc. 2006 Winter Simul. Conf.*, (November):73–83, 2006. doi: 10.1109/WSC.2006.323040.

Antonio Machi and Fillippo Mignosl. Garden of Eden Configurations for Cellular Automata on Cayley Graphs of Groups. *Discret. Math*, 6(1):44–56,

Jonathan Machta and Raymond Greenlaw. The parallel complexity of growth models. *J. Stat. Phys.*, 77(3-4):755–781, 1994. ISSN 00224715.

C L Magee and O L de Weck. Complex System Classification. *Incose*, page 18, 2004. ISSN 23345837.

H.B. Mann and D.R. Whitney. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *Ann. Math. Stat.*, 18(3):688–697, 1946. ISSN 00034851.

Yder Masson. A fast two-step algorithm for invasion percolation with trapping. *Comput. Geosci.*, 90(October 2013):41–48, 2016. ISSN 00983004. doi: 10.1016/j.cageo.2016.02.003.

Yder Masson and Steven R Pride. A Fast Algorithm for Invasion Percolation. *Transp. Porous Media*, 102(October 2013):301–312, 2014. ISSN 0169-3913. doi: 10.1007/s11242-014-0277-8.

Pinaki Mazumder, Sung Mo Kang, and Rainer Waser. Memristors: Devices, models, and applications. *Proc. IEEE*, 100(6):1911–1919, 2012. ISSN 00189219.

Donald JR Meagher. *Octree encoding: A new technique for the representation, manipulation and display of arbitrary 3-d objects by computer.* Electrical and Systems Engineering Department Rensseiaer Polytechnic Institute Image Processing Laboratory, 1980.

Donald J.R. Meagher. The Octree Encoding Method for Efficient Solid Modeling. page 164, 1982.

Paul Meakin. Noise-reduced and anlsotropy-enhanced Eden and screened-growth models. 38(1),

Paul Meakin, H Eugene Stanley, Antonio Coniglio, and Thomas A Mitten. Surfaces, interfaces, and screening af fractal structures. 32(4):2364–2369,

Paul Meakin, Jens Feder, Vidar Frette, and Torstein Jossang. Invasion Percolation in a destabalizing gradient. *Phys. Rev. A*, 46(6),

Paul Meuffels and Rohit Soni. Fundamental Issues and Problems in the Realization of Memristors. jul 2012.

M Mitchell. Genetic algorithms: An overview. *Complexity*, 1(1):31–39, 1995. ISSN 10762787. doi: 10.1002/cplx.6130010108.

Melanie Mitchell. An Introduction to Genetic Algorithms (Complex Adaptive Systems). *MIT Press*, page 221, 1998. ISSN 08981221.

Siddhartha Sankar Nath, Girish Mishra, Jajnyaseni Kar, Sayan Chakraborty, and Nilanjan Dey. A survey of image classification methods and techniques. *2014 Int. Conf. Control. Instrumentation, Commun. Comput. Technol.*, pages 554–557, 2014. doi: 10.1109/ICCICCT.2014.6993023.

Bruce F. Naylor. A tutorial on binary space partitioning trees. *Comput. Games Dev. Conf. Proc.*, pages 433—-457,

Muaz Niazi and Amir Hussain. Agent-based computing from multi-agent systems to agent-based models: a visual survey. *Scientometrics*, 89(2):479–499, 2011. ISSN 0138-9130. doi: 10.1007/s11192-011-0468-9.

J Quinn Norris, Donald L Turcotte, and John B Rundle. Loopless non-trapping invasion percolation model for fracking. *Phys. Rev. E - Cover. Stat. nonlinear, Biol. soft matter Phys.*, 89(2):19, 2014.

Adam Paszke, Gregory Chanan, Zeming Lin, Sam Gross, Edward Yang, Luca Antiga, and Zachary Devito. Automatic differentiation in PyTorch. *Adv. Neural Inf. Process. Syst. 30*, (Nips):1–4,

Yuriy V. Pershin and Massimiliano Di Ventra. Solving mazes with memristors: A massively parallel approach. *Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys.*, 84(4):1–6, 2011. ISSN 15393755.

Daniel Peter Playne. *Generative Programming Methods for Parallel Partial Differential Field Equation Solvers Doctor of Philosophy*. Phd thesis, Massey UNiversity, 2011.

Daniel Peter Playne and Ken Hawick. A New Algorithm for Parallel Connected-Component Labelling on GPUs. *IEEE Trans. Parallel Distrib. Syst.*, 9219(c):1–14, 2018. ISSN 10459219.

R. B. Potts. Some generalized order-disorder transformations. *Math. Proc. Cambridge Philos. Soc.*, 48(1):106–109, 1952.

Craig Reynolds. Boids Background and Update, 1995. `http://www.red3d.com/cwr/boids/`. Last Accessed On: 2015-11-11.

Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Comput. Graph.*, 21(4):25–34, 1987. ISSN 00978930.

Mohammad Nazmus Sakib, Rakibul Hassan, and Satyen Biswas. A Memristor-based 6T1M Hybrid memory cell without state drift during successive read. *Proc. 9th Int. Conf. Electr. Comput. Eng. ICECE 2016*, 0(1):202–205, 2016.

Benedkit Sauer and Alexander Schroer. Fractal Growth. *Sci. Am.*, 256(1):94–100, 1987. ISSN 0036-8733.

Hiroki Sayama. Introduction to the Modeling and Analysis of Complex Systems. page 477, 2015.

Thomas C Schelling. Dynamic models of segregation. *J. Math. Sociol.*, 1(2):143–186, 1971. ISSN 0022-250X. doi: 10.1080/0022250X.1971.9989794.

P Y Simard, D Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. *Doc. Anal. Recognition, 2003. Proceedings. Seventh Int. Conf.*, pages 958–963, 2003. ISSN 15205363.

Korsuk Sirinukunwattana, Shan E.Ahmed Raza, Yee Wah Tsang, David R.J. Snead, Ian A. Cree, and Nasir M. Rajpoot. Locality Sensitive Deep Learning for Detection and Classification of Nuclei in Routine Colon Cancer Histology Images. *IEEE Trans. Med. Imaging*, 35(5):1196–1206, 2016. ISSN 1558254X.

J. Maynard Smith and G. R. Price. The Logic of Animal Conflict. *Nature*, 246:15–18, 1973.

R. Stanley Williams. How We Found The Missing Memristor. *IEEE Spectr.*, 45(12): 28–35,

Dimitrios Stathis, Ioannis Vourkas, and Georgios Ch Sirakoulis. Shortest Path Computing Using Memristor-Based Circuits and Cellular Automata. *2014 Conf. Cell. Autom. Res. Ind.*, 8751:398–407, 2014. doi: 10.1007/978-3-319-11520-7.

Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–3, 2008. ISSN 1476-4687. doi: 10.1038/nature06932.

András Szabó and Roeland M H Merks. Cellular Potts modeling of tumor growth , tumor invasion , and tumor evolution. 3(April):1–12, 2013.

Miklos N. Szilagyi. Investigation of N-person games by agent-based modeling. *Complex Syst.*, 21(3):201–243, 2012.

Q K Telesford, S L Simpson, J H Burdette, S Hayasaka, and P J Laurienti. The brain as a complex system: using network science as a tool for understanding the brain. *Brain Connect*, 1(4):295–308, 2011. ISSN 2158-0014.

Ronald Tetzla. Memristors and Memristive Systems. pages 1–409, 2013. doi: 10.1007/978-1-4614-9068-5.

DW Thompson. *On growth and form.* Cambridge University Press, 1942. doi: 10.2307/2019330.

Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. ... *Conf. Complex Syst.*, pages 16–21, 2004a. ISSN 10639667. doi: 10.1109/ICVD.2004.1261037.

Seth Tisue and Uri Wilensky. NetLogo: Design and implementation of a multi-agent modeling environment. In *Proc. Agent 2004 Conf. Soc. Dyn. Interact. Reflexivity Emerg.*, 2004b. ISBN 9780769550886. doi: 10.1109/ICHCI-IEEE.2013.6887799.

Martin A. Trefzer. Memristor in a Nutshell. In *Guid. to Unconv. Comput. Music*, pages 159–180. Springer, 2017. ISBN 978-3-319-49881-2. doi: 10.1007/978-3-319-49881-2_6.

Roshni Uppala. Simulating Large Scale Memristor Based Crossbar for Neuromorphic Applications.

A. Vargha and H.D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *J. Educ. Behav. Stat.*, 25(2):101–132,

John Von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Champaign, IL, USA, 1966.

Sascha Vongehr and Xiangkang Meng. The Missing Memristor has Not been Found. *Sci. Rep.*, 5:11657, 2015. ISSN 2045-2322.

Pascal O Vontobel, Warren Robinett, Philip J Kuekes, Duncan R Stewart, Joseph Straznicky, and R Stanley Williams. Writing to and reading from a nano-scale crossbar memory based on memristors. *Nanotechnology*, 20(42):425204, 2009. ISSN 0957-4484.

Ioannis Vourkas and Georgios Ch Sirakoulis. Memristor Modeling. In *Memristor-Based Nanoelectron. Comput. Circuits Archit.*, pages 9–28. Springer, 2015. ISBN 9783319226477. doi: 10.1007/978-3-319-22647-7.

Bartlomiej Waclaw, Ivana Bozic, Meredith E Pittman, Ralph H Hruban, Bert Vogelstein, and Martin a Nowak. Spatial model predicts dispersal and cell turnover cause reduced intra-tumor heterogeneity. *Nature*, 525:261–264, 2015. ISSN 0028-0836. doi: 10.1038.

C. Y. Wang and J. B. Bassingthwaighte. Biological growth on a surface. *Math. Biosci.*, 142(2):91–106, 1997. ISSN 00255564.

Lidan Wang, Xiaodong Wang, Shukai Duan, and Huifang Li. A spintronic memristor bridge synapse circuit and the application in memrisitive cellular automata. *Neurocomputing*, 167:346–351, 2015. ISSN 18728286. doi: 10.1016/j.neucom.2015.04.061.

B.T Werner. Complexity in Natural Landform Patterns. *Science (80-.).*, 284(5411): 102–104,

Wikipedia. Octree, 2018. `https://en.wikipedia.org/wiki/Octree`. Last Accessed On: 2019-04-13.

David Wilkinson and Jorge F Willemsen. Invasion percolation: a new form of percolation theory. *J. Phys. A Math. Gen*, 16(May):3365–3376, 1983. ISSN 0305-4470.

T.A. Witten and Sander L.M. Diffusion-Limited Aggregation, a Kinetice Critical Phenomenon. *Phys. Rev. Lett.*, 47(19):4, 1981.

Michael Wooldridge. *An Introduction to Multiagent Systems.* John Wiley & Sons, LTD, Sussex, 1 edition, 2002. ISBN 0-471-49691-X.

Kesheng Wu, Ekow Otoo, and Arie Shoshani. Optimizing connected component labeling algorithms. page 1965, 2005. ISSN 0277786X. doi: 10.1117/12.596105.

Kesheng Wu, Ekow Otoo, and Kenji Suzuki. Optimizing two-pass connected-component labeling algorithms. *Pattern Anal. Appl.*, 12(2):117–135, 2009. ISSN 14337541.

Yan Bo Xie, Yu Jian Li, and Bing Hong Wang. Generalized Eden model with a screening effect. *Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys.*, 83(5):1–3, 2011. ISSN 15393755.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolutional Network. 2015.

Chris Yakopcic, Tarek M. Taha, Guru Subramanyam, Robinson E. Pino, and Stanley Rogers. A memristor device model. *IEEE Electron Device Lett.*, 32(10):1436–1438, 2011. ISSN 07413106.

Chris Yakopic, Tarek M. Taha, Guru Subramantam, and Robinson E. Pino. Memristor SPICE Modeling. In *Adv. Neuromorphic Memristor Sci. Appl.*, pages 1–320. Springer, 2012. ISBN 9789400744912. doi: 10.1007/978-94-007-4491-2.

Zhanyou Ye, Shi Hong Marcus Wu, and Themistoklis Prodromakis. Computing shortest paths in 2D and 3D memristive networks. pages 2–12, 2013.

Hao Yu, Leibin Ni, and Hantao Huang. Distributed in-memory computing on binary memristor-crossbar for machine learning. *Stud. Comput. Intell.*, 701(3):275–304, 2017. ISSN 1860949X.

APPENDIX A

Statistical Tests Explained

Statistical test are very useful tools in the analysis of large quantities of data to gain an understanding of the data especially in cases when a sample of data is drawn from a population and used as a representation of the population. These test allow use to be able to identify whether the sample is a good representation of the original population as well as whether any changes to the population have an effect that could be considered to be positive or negative to the overall population.

A.1 Non-parametric tests

There are three different statistical tests that are used in this thesis help inform on the results shown in Chapter 5 these being the Mann-Whitney U test which is used to calculate the probability that two samples are drawn from populations with the same median. The Kolmogorov-Smirnoff test which is used to calculate the probability that two different sample are from populations with the same distributions. Finally, there is the “A” measure which is used to measure the importance of scientific significance or importance. Both the Mann-Whitney U and the Kolmogorov-Smirnoff test are non-parametric tests which means that the tests make no assumptions on the form of the distribution. This means that these tests work as well for non-normally distributed data as they do for normally distributed data .

The Mann-Whitney U test makes use of a null hypothesis that can be rejected with a confidence level of 95% when the returned result of the test is less then 0.05, however just because the result is greater then 0.05 that is not a confirmation of the hypothesis. The hypothesis for this test can be seen below:

H_0 : samples **X** and **Y** have the same medians

The Kolmogorov-Smirnoff test makes use of a null hypothesis that can be rejected with a confidence level of 95% when the result of the test returns a value of less than 0.05 and as with the Mann-Whitney U test just because the result is greater then 0.05 that is not a confirmation of the hypothesis. The hypothesis for this test can be seen below:

H_0 : samples **X** and **Y** have the same distributions

The “A” measure returns a values between 0.5 and 1, the higher the value return the higher the level of significance of the result. The boundary for the level of significance can be seen below:

a* == *0.50 means no effect (distributions have the same medians)

a* >= *0.56 means a small effect

a* >= *0.64 means a medium effect

a* >= *0.71 means a large effect

Both the Mann-Whitney U and Kolmogorov-Smirnoff tests and very easy to implement with statistics library's such as SciPy, a library for python, have built in functions for the computation of these test. In the case of pyplot these functions are $mannwhitneyu$ and ks_2samp. However, this is not the case for the "A" measure and as such this method had to implemented specifically and the code used to calculates the the "A" measure can be seen in Figure A.1.

```
def A_measure(X, Y):
  equal = 0.0
  greater = 0.0

  for x in X:
    for y in Y:
      if x == y:
        equal += 1
      elif x > y:
        greater += 1
      nm = len(X) * len(Y)

  data = (greater / nm) + ((0.5 * equal) / nm)

  if(data < 0.5):
    return (1.0 - data)
  else:
  return data
```

Figure A.1: Code for the calculation of the "A" measure for two different distributions, X and Y. A test used for the calculation of scientific significances. This implementation of the "A" measure is able to deal with distribution of different sizes though all distributions in this work are of the same size.

A.2 Boxplot

Figure A.2 shows an example of a boxplot which is a useful graphing tool that is used through this work there are a number of features of this plot the show be explained in order to full understand what and boxplots shown in the main body of this work mean.

The first is this to mention are the two lines running through the box. The orange line shows the median for the distribution and the green line shows the mean for the distribution. It is helpful to be able to see both as this can be a good indication of the

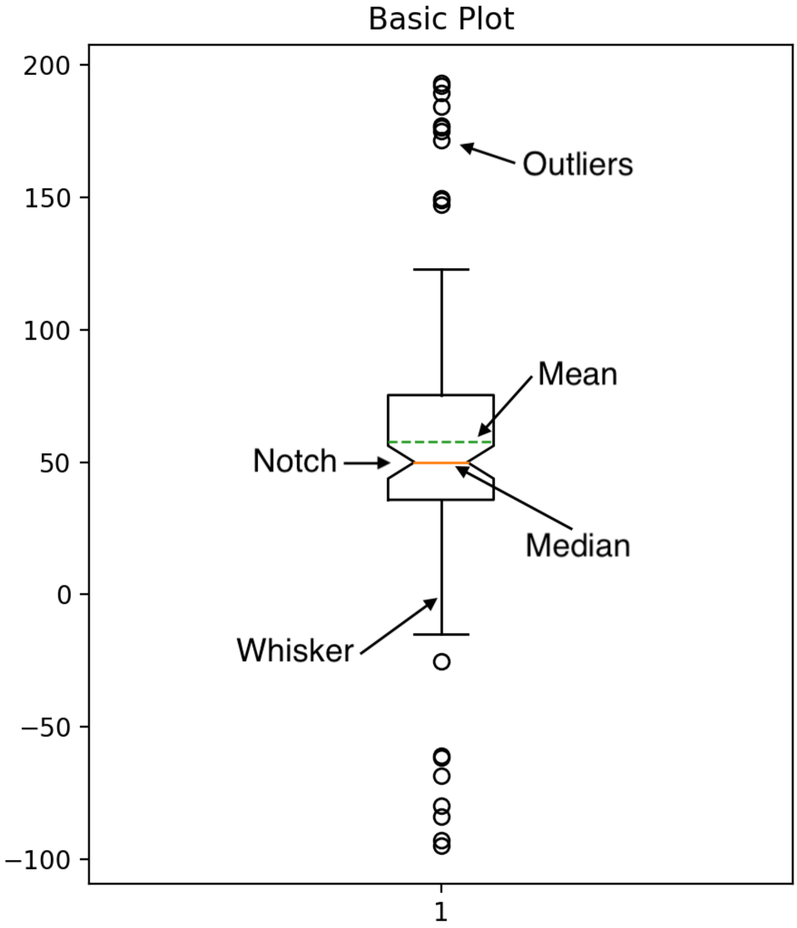


Figure A.2: Example boxplot use to help explain the different features of this type of diagram

skew of the data the close they are too one another the more normally distributed the data will be.

The Notch in the box is another important aspect of the diagram this show the confidence range for the median of the distribution as being in this work all of the data will be a sample from a population and the the whole population it is possible for there to be some error on the median. The narrower this notch is the better.

The main parts of the boxplot plot are the box and the whiskers. The box is used to represent the Interquartile Range (IQR) this shows where the data that falls between the 75th commonly called $Q3$and the 25th percentiles commonly called $Q1$ lies the range between these is the IQR. The whiskers of the diagram are calculate with $Q1 - (1.5 \times IQR)$ for the lower and $Q3 - (1.5 \times IQR)$ for the upper. The outliers these are shown through the use a hollow circles and these are all the points that fall out of the the range of the whiskers of the diagram. All of these feature are labelled in on the graph in Figure A.2.

APPENDIX B

Statistical Test Results for Timing

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.705	0.502	0.997	0.736	0.707	0.997	0.878	0.997	0.734	0.999
L=32	0.704	0.998	0.998	0.708	0.997	0.997	0.861	0.713	0.998	0.998
L=64	0.703	1.0	0.998	0.732	1.0	0.995	0.861	1.0	1.0	0.998
L=128	0.704	1.0	0.998	0.751	1.0	0.996	0.879	1.0	1.0	0.998
L=256	0.681	1.0	0.998	0.736	1.0	0.997	0.863	1.0	1.0	0.997
L=512	0.73	1.0	0.998	0.759	1.0	0.997	0.899	1.0	1.0	0.998
L=1024	0.694	1.0	0.998	0.765	1.0	0.998	0.888	1.0	1.0	0.998

Table B.1: Construction Time A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.46	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.2: Construction Time Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.999	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.3: Construction Time Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.522	1.0	0.77	1.0	1.0	0.802	1.0	1.0	0.548	1.0
L=32	0.833	1.0	0.614	1.0	1.0	0.904	1.0	1.0	0.575	1.0
L=64	0.985	1.0	0.78	1.0	1.0	0.967	1.0	1.0	0.662	1.0
L=128	1.0	1.0	1.0	1.0	1.0	0.952	1.0	1.0	0.749	1.0
L=256	1.0	1.0	1.0	1.0	1.0	0.915	1.0	1.0	0.912	1.0
L=512	1.0	1.0	1.0	1.0	1.0	0.906	1.0	1.0	0.873	1.0
L=1024	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.986	1.0

Table B.4: EdenA A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.641	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.296	0.0
L=32	0.0	0.0	0.013	0.0	0.0	0.0	0.0	0.0	0.105	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.5: EdenA Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.791	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.401	0.0
L=32	0.0	0.0	0.068	0.0	0.0	0.0	0.0	0.0	0.214	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.6: EdenA Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.528	0.571	0.581	0.573	0.595	0.56	0.597	0.651	0.503	0.655
L=32	0.512	0.531	0.519	0.532	0.526	0.525	0.526	0.546	0.502	0.547
L=64	0.51	0.517	0.502	0.517	0.509	0.511	0.509	0.517	0.5	0.517
L=128	0.512	0.517	0.505	0.519	0.505	0.509	0.506	0.514	0.5	0.515
L=256	0.507	0.511	0.502	0.513	0.505	0.51	0.508	0.514	0.502	0.515

Table B.7: EdenB A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.501	0.084	0.048	0.075	0.021	0.146	0.018	0.0	0.944	0.0
L=32	0.777	0.446	0.643	0.442	0.534	0.537	0.534	0.261	0.968	0.257
L=64	0.806	0.682	0.966	0.688	0.831	0.783	0.831	0.679	0.992	0.679
L=128	0.766	0.679	0.895	0.65	0.913	0.835	0.88	0.743	1.0	0.712
L=256	0.872	0.795	0.953	0.745	0.897	0.806	0.854	0.727	0.956	0.719

Table B.8: EdenB Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.67	0.188	0.136	0.188	0.045	0.188	0.03	0.002	1.0	0.001
L=32	0.999	0.959	0.999	0.89	0.959	0.959	0.959	0.549	1.0	0.549
L=64	1.0	0.992	1.0	0.999	1.0	1.0	1.0	0.992	1.0	0.999
L=128	0.999	0.992	1.0	0.992	1.0	1.0	1.0	0.999	1.0	0.992
L=256	0.999	0.999	1.0	0.959	0.999	0.999	0.999	0.999	1.0	0.999

Table B.9: EdenB Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.604	0.969	0.707	0.97	0.988	0.802	0.988	0.985	0.527	0.984
L=32	0.956	0.999	0.62	0.999	0.924	0.977	0.931	1.0	0.518	1.0
L=64	0.998	1.0	0.515	1.0	0.803	0.998	0.806	1.0	0.513	1.0
L=128	1.0	1.0	0.542	1.0	0.646	1.0	0.658	1.0	0.521	1.0
L=256	1.0	1.0	0.551	1.0	0.612	1.0	0.645	1.0	0.539	1.0
L=512	1.0	1.0	0.557	1.0	0.562	1.0	0.631	1.0	0.571	1.0

Table B.10: EdenC A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.012	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.518	0.0
L=32	0.0	0.0	0.004	0.0	0.0	0.0	0.0	0.0	0.661	0.0
L=64	0.0	0.0	0.725	0.0	0.0	0.0	0.0	0.0	0.751	0.0
L=128	0.0	0.0	0.306	0.0	0.0	0.0	0.0	0.0	0.611	0.0
L=256	0.0	0.0	0.218	0.0	0.006	0.0	0.0	0.0	0.341	0.0
L=512	0.0	0.0	0.167	0.0	0.129	0.0	0.001	0.0	0.084	0.0

Table B.11: EdenC Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.67	0.0
L=32	0.0	0.0	0.045	0.0	0.0	0.0	0.0	0.0	0.89	0.0
L=64	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.89	0.0
L=128	0.0	0.0	0.67	0.0	0.003	0.0	0.0	0.0	0.89	0.0
L=256	0.0	0.0	0.67	0.0	0.03	0.0	0.012	0.0	0.549	0.0
L=512	0.0	0.0	0.437	0.0	0.255	0.0	0.012	0.0	0.188	0.0

Table B.12: EdenC Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.508	0.555	0.519	0.554	0.567	0.51	0.566	0.568	0.504	0.568
L=32	0.515	0.503	0.508	0.502	0.516	0.523	0.515	0.504	0.505	0.505
L=64	0.738	0.505	0.581	0.502	0.741	0.68	0.741	0.587	0.506	0.587
L=128	0.611	0.573	0.582	0.571	0.542	0.547	0.535	0.51	0.51	0.516
L=256	0.539	0.551	0.775	0.53	0.578	0.733	0.552	0.814	0.518	0.775

Table B.13: Screened Eden 0.2 A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.85	0.183	0.648	0.194	0.103	0.81	0.11	0.099	0.931	0.101
L=32	0.71	0.937	0.841	0.966	0.69	0.577	0.714	0.929	0.905	0.901
L=64	0.0	0.913	0.05	0.955	0.0	0.0	0.0	0.035	0.878	0.034
L=128	0.007	0.077	0.047	0.085	0.306	0.253	0.389	0.808	0.802	0.704
L=256	0.342	0.216	0.0	0.46	0.059	0.0	0.21	0.0	0.666	0.0

Table B.14: Screened Eden 0.2 Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.789	0.255	0.789	0.255	0.067	0.992	0.096	0.188	1.0	0.188
L=32	0.959	0.999	0.992	0.992	0.789	0.67	0.789	0.999	1.0	0.999
L=64	0.0	0.67	0.096	0.89	0.0	0.0	0.0	0.067	0.992	0.02
L=128	0.03	0.096	0.03	0.067	0.437	0.045	0.338	0.549	0.437	0.067
L=256	0.067	0.549	0.0	0.67	0.002	0.0	0.012	0.0	0.89	0.0

Table B.15: Screened Eden 0.2 Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.571	0.543	0.586	0.541	0.617	0.514	0.616	0.632	0.503	0.631
L=32	0.765	0.569	0.652	0.567	0.809	0.656	0.807	0.707	0.507	0.705
L=64	0.798	0.513	0.671	0.513	0.832	0.701	0.83	0.708	0.504	0.705
L=128	0.838	0.525	0.624	0.546	0.851	0.786	0.863	0.648	0.52	0.672
L=256	0.522	0.537	0.669	0.552	0.533	0.589	0.551	0.638	0.527	0.596
L=512	0.539	0.527	0.89	0.546	0.509	0.841	0.514	0.863	0.522	0.822

Table B.16: Screened Eden 0.5 A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.085	0.296	0.037	0.316	0.004	0.732	0.005	0.001	0.937	0.001
L=32	0.0	0.094	0.0	0.105	0.0	0.0	0.0	0.0	0.874	0.0
L=64	0.0	0.751	0.0	0.76	0.0	0.0	0.0	0.0	0.917	0.0
L=128	0.0	0.548	0.003	0.259	0.0	0.0	0.0	0.0	0.632	0.0
L=256	0.599	0.363	0.0	0.202	0.424	0.031	0.212	0.001	0.511	0.02
L=512	0.35	0.508	0.0	0.269	0.829	0.0	0.738	0.0	0.59	0.0

Table B.17: Screened Eden 0.5 Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.096	0.437	0.045	0.437	0.02	0.789	0.03	0.005	1.0	0.005
L=32	0.0	0.136	0.0	0.188	0.0	0.0	0.0	0.0	1.0	0.0
L=64	0.0	0.789	0.0	0.89	0.0	0.0	0.0	0.0	1.0	0.0
L=128	0.0	0.959	0.008	0.67	0.0	0.0	0.0	0.001	0.959	0.0
L=256	0.0	0.188	0.0	0.255	0.0	0.0	0.001	0.002	0.549	0.005
L=512	0.001	0.67	0.0	0.136	0.003	0.0	0.008	0.0	0.67	0.0

Table B.18: Screened Eden 0.5 Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.573	0.501	0.683	0.501	0.58	0.596	0.579	0.671	0.5	0.672
L=32	0.763	0.545	0.709	0.544	0.797	0.575	0.797	0.742	0.504	0.742
L=64	0.886	0.548	0.743	0.547	0.848	0.742	0.847	0.688	0.5	0.688
L=128	0.909	0.537	0.657	0.541	0.881	0.878	0.885	0.591	0.501	0.593
L=256	0.839	0.534	0.623	0.513	0.822	0.89	0.833	0.657	0.518	0.637
L=512	0.741	0.578	0.782	0.556	0.702	0.867	0.713	0.847	0.525	0.842
L=1024	0.577	0.561	0.966	0.579	0.587	0.753	0.606	0.923	0.507	0.947

Table B.19: Screened Eden 0.9 A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.075	0.991	0.0	0.976	0.051	0.019	0.054	0.0	0.994	0.0
L=32	0.0	0.276	0.0	0.282	0.0	0.067	0.0	0.0	0.929	0.0
L=64	0.0	0.242	0.0	0.252	0.0	0.0	0.0	0.0	0.996	0.0
L=128	0.0	0.375	0.0	0.324	0.0	0.0	0.0	0.027	0.984	0.024
L=256	0.0	0.409	0.003	0.751	0.0	0.0	0.0	0.0	0.67	0.001
L=512	0.0	0.057	0.0	0.172	0.0	0.0	0.0	0.0	0.547	0.0
L=1024	0.062	0.137	0.0	0.054	0.034	0.0	0.01	0.0	0.856	0.0

Table B.20: Screened Eden 0.9 Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.096	0.338	0.0	0.338	0.136	0.008	0.136	0.0	1.0	0.0
L=32	0.0	0.437	0.0	0.437	0.0	0.045	0.0	0.0	1.0	0.0
L=64	0.0	0.549	0.0	0.549	0.0	0.0	0.0	0.0	1.0	0.0
L=128	0.0	0.255	0.0	0.437	0.0	0.0	0.0	0.001	1.0	0.008
L=256	0.0	0.67	0.0	0.999	0.0	0.0	0.0	0.0	0.959	0.0
L=512	0.0	0.255	0.0	0.255	0.0	0.0	0.0	0.0	0.789	0.0
L=1024	0.001	0.136	0.0	0.096	0.001	0.0	0.0	0.0	0.89	0.0

Table B.21: Screened Eden 0.9 Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.563	0.943	0.632	0.93	0.942	0.699	0.924	0.98	0.524	0.972
L=32	0.604	0.935	0.567	0.891	0.894	0.671	0.824	0.961	0.664	0.93
L=64	0.697	0.968	0.507	0.937	0.914	0.701	0.859	0.976	0.608	0.948
L=128	0.717	0.96	0.573	0.928	0.885	0.663	0.808	0.952	0.662	0.91
L=256	0.763	0.968	0.667	0.948	0.895	0.626	0.832	0.94	0.638	0.903
L=512	0.829	0.986	0.755	0.975	0.869	0.608	0.795	0.925	0.66	0.883
L=1024	0.955	0.999	0.921	0.998	0.898	0.619	0.88	0.947	0.568	0.936

Table B.22: IP Random A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.123	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.561	0.0
L=32	0.011	0.0	0.1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.859	0.0	0.0	0.0	0.0	0.0	0.008	0.0
L=128	0.0	0.0	0.076	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.002	0.0	0.0	0.001	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.008	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.004	0.0	0.0	0.099	0.0

Table B.23: IP Random Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.14	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.443	0.0
L=32	0.02	0.0	0.069	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.894	0.0	0.0	0.0	0.0	0.0	0.099	0.0
L=128	0.0	0.0	0.099	0.0	0.0	0.001	0.0	0.0	0.002	0.0
L=256	0.0	0.0	0.001	0.0	0.0	0.005	0.0	0.0	0.005	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.069	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.005	0.0	0.0	0.261	0.0

Table B.24: IP Random Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.521	0.926	0.68	0.883	0.943	0.675	0.905	0.979	0.564	0.958
L=32	0.579	0.978	0.664	0.875	0.973	0.743	0.838	0.998	0.843	0.969
L=64	0.69	0.99	0.724	0.737	0.97	0.875	0.561	1.0	0.883	0.901
L=128	0.775	0.57	0.709	0.527	0.917	0.925	0.683	0.714	0.523	0.668
L=256	0.921	1.0	0.699	0.672	1.0	0.989	0.873	1.0	1.0	0.581
L=512	0.983	1.0	0.686	0.872	1.0	0.998	0.997	1.0	1.0	0.892
L=1024	1.0	1.0	0.642	0.966	1.0	1.0	1.0	1.0	1.0	0.988

Table B.25: IP Meakin [g=-0.01] A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.606	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.119	0.0
L=32	0.055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.139	0.0	0.0	0.0
L=128	0.0	0.089	0.0	0.513	0.0	0.0	0.0	0.0	0.582	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.047
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.26: IP Meakin [g=-0.01] Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.794	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.047	0.0
L=32	0.099	0.0	0.002	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.344	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.02	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.27: IP Meakin [g=-0.01] Kolmogorov–Smirnov Results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.522	0.936	0.65	0.919	0.942	0.682	0.923	0.983	0.516	0.972
L=32	0.607	0.936	0.58	0.878	0.91	0.681	0.825	0.963	0.689	0.923
L=64	0.682	0.966	0.545	0.926	0.926	0.726	0.838	0.981	0.76	0.953
L=128	0.776	0.993	0.501	0.895	0.929	0.791	0.669	0.997	0.902	0.919
L=256	0.896	0.882	0.589	0.699	0.606	0.886	0.689	0.883	0.632	0.636
L=512	0.953	0.98	0.704	0.526	1.0	0.935	0.944	1.0	0.98	0.745
L=1024	0.997	1.0	0.83	0.786	1.0	0.994	1.0	1.0	1.0	0.909

Table B.28: IP Meakin [g=-0.001] A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.597	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0
L=32	0.009	0.0	0.051	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.268	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.974	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.029	0.0	0.01	0.0	0.0	0.0	0.001	0.001
L=512	0.0	0.0	0.0	0.528	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.29: IP Meakin [g=-0.001] Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.794	0.0	0.008	0.0	0.0	0.001	0.0	0.0	0.677	0.0
L=32	0.003	0.0	0.069	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.677	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.961	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.013	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.008	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.30: IP Meakin [g=-0.001] Kolmogorov–Smirnov

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.548	0.937	0.629	0.916	0.932	0.681	0.909	0.978	0.542	0.965
L=32	0.61	0.931	0.559	0.885	0.893	0.673	0.82	0.959	0.681	0.927
L=64	0.684	0.954	0.505	0.913	0.9	0.684	0.819	0.96	0.662	0.922
L=128	0.735	0.964	0.574	0.953	0.88	0.682	0.861	0.956	0.59	0.943
L=256	0.811	0.989	0.657	0.949	0.9	0.698	0.724	0.977	0.836	0.902
L=512	0.94	0.992	0.816	0.889	0.72	0.769	0.679	0.929	0.818	0.626
L=1024	0.991	0.521	0.945	0.705	1.0	0.856	0.9	1.0	0.787	0.69

Table B.31: IP Meakin [g=-0.0001] A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.244	0.0	0.002	0.0	0.0	0.0	0.0	0.0	0.307	0.0
L=32	0.007	0.0	0.148	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.894	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.069	0.0	0.0	0.0	0.0	0.0	0.027	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.002
L=1024	0.0	0.61	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.32: IP Meakin [g=-0.0001] Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.443	0.0	0.02	0.0	0.0	0.0	0.0	0.0	0.794	0.0
L=32	0.031	0.0	0.344	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.961	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.193	0.0	0.0	0.0	0.0	0.0	0.069	0.0
L=256	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.002
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.33: IP Meakin [g=-0.0001] Kolmogorov–Smirnov

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	1.0	1.0	0.977	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=32	1.0	1.0	0.991	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=64	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=128	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=256	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=512	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table B.34: IP Hawick [beta=0.1] A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.35: IP Hawick [beta=0.1] Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.36: IP Hawick [beta=0.1] Kolmogorov–Smirnov

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	1.0	1.0	0.971	1.0	1.0	1.0	1.0	1.0	0.997	1.0
L=32	1.0	1.0	0.95	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=64	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=128	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=256	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=512	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Table B.37: IP Hawick [beta=0.5] A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.38: IP Hawick [beta=0.5] Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table B.39: K
olmogorov–Smirnov]IP Hawick [beta=0.5] Kolmogorov–Smirnov

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.963	1.0	0.769	1.0	1.0	0.998	1.0	1.0	0.76	1.0
L=32	0.997	1.0	0.755	1.0	1.0	0.993	1.0	1.0	0.966	1.0
L=64	1.0	1.0	0.997	1.0	1.0	1.0	1.0	1.0	0.997	1.0
L=128	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=256	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=512	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
L=1024	1.0	1.0	–	1.0	1.0	–	1.0	–	1.0	–

Table B.40: IP Hawick [beta=0.9] A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	–	0.0	0.0	–	0.0	–	0.0	–

Table B.41: IP Hawick [beta=0.9] Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	–	0.0	0.0	–	0.0	–	0.0	–

Table B.42: IP Hawick [beta=0.9] Kolmogorov–Smirnov

APPENDIX C

Statistical Test Results for Memory

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.764	0.989	0.543	0.989	0.99	0.994	0.99	1.0	0.5	1.0
L=32	0.769	1.0	0.556	1.0	1.0	1.0	1.0	1.0	0.5	1.0
L=64	0.584	1.0	0.586	1.0	1.0	1.0	1.0	1.0	0.5	1.0
L=128	0.582	1.0	0.657	1.0	1.0	1.0	1.0	1.0	0.5	1.0
L=256	0.96	1.0	0.71	1.0	1.0	1.0	1.0	1.0	0.5	1.0
L=512	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0

Table C.1: EdenA A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.298	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=32	0.0	0.0	0.177	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=64	0.041	0.0	0.037	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=128	0.047	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0

Table C.2: EdenA Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

Table C.3: EdenA Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.554	0.881	0.988	0.881	0.889	0.99	0.889	1.0	0.5	1.0
L=32	0.585	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0
L=64	0.539	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0
L=128	0.748	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0
L=256	0.939	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	1.0

Table C.4: EdenB A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.19	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=32	0.04	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=64	0.341	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0

Table C.5: EdenB Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0

Table C.6: EdenB Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.598	0.93	0.986	0.929	0.98	0.983	0.98	1.0	0.517	1.0
L=32	0.59	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.522	1.0
L=64	0.95	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.517	1.0
L=128	0.758	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.509	1.0
L=256	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.526	1.0
L=512	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.532	1.0

Table C.7: EdenC A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.017	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.672	0.0
L=32	0.03	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.593	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.682	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.819	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.521	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.437	0.0

Table C.8: EdenC Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.959	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.549	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.789	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.789	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.789	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.789	0.0

Table C.9: EdenC Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.505	0.878	0.986	0.878	0.857	0.976	0.857	1.0	0.5	1.0
L=32	0.577	0.984	0.994	0.984	0.949	0.996	0.949	1.0	0.5	1.0
L=64	0.708	1.0	0.998	1.0	1.0	1.0	1.0	1.0	0.502	1.0
L=128	0.668	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.505	1.0
L=256	0.551	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.52	1.0

Table C.10: Screened Eden 0.2 A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.898	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=32	0.06	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.969	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.895	0.0
L=256	0.213	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.63	0.0

Table C.11: Screened Eden 0.2 Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=32	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.549	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89	0.0

Table C.12: Screened Eden 0.2 Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.567	0.812	0.974	0.812	0.752	0.992	0.752	1.0	0.5	1.0
L=32	0.749	0.848	0.986	0.848	0.623	0.992	0.623	0.996	0.5	0.996
L=64	0.78	0.803	0.994	0.802	0.724	1.0	0.725	1.0	0.501	1.0
L=128	0.763	0.555	1.0	0.559	0.926	1.0	0.935	1.0	0.524	1.0
L=256	0.557	0.991	1.0	0.991	0.93	1.0	0.93	1.0	0.529	1.0
L=512	0.564	1.0	1.0	1.0	1.0	1.0	1.0	0.814	0.528	0.811

Table C.13: Screened Eden 0.5 A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.105	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=32	0.0	0.0	0.0	0.0	0.003	0.0	0.003	0.0	0.999	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.973	0.0
L=128	0.0	0.181	0.0	0.155	0.0	0.0	0.0	0.0	0.562	0.0
L=256	0.163	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.484	0.0
L=512	0.118	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.492	0.0

Table C.14: Screened Eden 0.5 Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.959	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.549	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89	0.0

Table C.15: Screened Eden 0.5 Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.629	0.749	0.974	0.749	0.626	0.991	0.626	0.999	0.5	0.999
L=32	0.742	0.629	0.972	0.629	0.954	0.994	0.954	0.999	0.5	0.999
L=64	0.821	0.972	0.989	0.973	1.0	1.0	1.0	0.863	0.503	0.862
L=128	0.912	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.505	1.0
L=256	0.659	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.516	1.0
L=512	0.771	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.526	1.0
L=1024	0.633	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.511	1.0

Table C.16: Screened Eden 0.9 A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.002	0.0	0.0	0.0	0.002	0.0	0.002	0.0	0.999	0.0
L=32	0.0	0.002	0.0	0.002	0.0	0.0	0.0	0.0	0.999	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.944	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.91	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.702	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.531	0.0
L=1024	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.785	0.0

Table C.17: Screened Eden 0.9 Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.999	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.959	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.789	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89	0.0

Table C.18: Screened Eden 0.9 Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.525	0.997	0.992	0.997	0.989	0.985	0.987	1.0	0.508	1.0
L=32	0.538	0.967	0.977	0.965	0.967	0.979	0.969	1.0	0.538	1.0
L=64	0.509	0.932	0.988	0.933	0.879	0.993	0.88	1.0	0.529	1.0
L=128	0.517	0.794	0.974	0.79	0.631	0.971	0.626	1.0	0.534	1.0
L=256	0.523	0.637	0.967	0.634	0.563	0.966	0.565	0.99	0.509	0.99
L=512	0.533	0.616	0.97	0.615	0.52	0.973	0.51	1.0	0.537	1.0
L=1024	0.524	0.762	0.983	0.758	0.657	0.986	0.657	0.99	0.578	0.989

Table C.19: Random Invasion Percolation A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.543	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.849	0.0
L=32	0.538	0.967	0.977	0.965	0.967	0.979	0.969	1.0	0.538	1.0
L=64	0.821	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.477	0.0
L=128	0.673	0.0	0.0	0.0	0.001	0.0	0.002	0.0	0.409	0.0
L=256	0.572	0.001	0.0	0.001	0.125	0.0	0.116	0.0	0.825	0.0
L=512	0.426	0.005	0.0	0.005	0.627	0.0	0.818	0.0	0.367	0.0
L=1024	0.553	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.058	0.0

Table C.20: Random Invasion Percolation Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.67	0.0
L=32	0.005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.255	0.0
L=64	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.549	0.0
L=128	0.012	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.549	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89	0.0
L=512	0.008	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.096	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.136	0.0

Table C.21: Random Invasion Percolation Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.521	0.926	0.68	0.883	0.943	0.675	0.905	0.979	0.564	0.958
L=32	0.579	0.978	0.664	0.875	0.973	0.743	0.838	0.998	0.843	0.969
L=64	0.69	0.99	0.724	0.737	0.97	0.875	0.561	1.0	0.883	0.901
L=128	0.775	0.57	0.709	0.527	0.917	0.925	0.683	0.714	0.523	0.668
L=256	0.921	1.0	0.699	0.672	1.0	0.989	0.873	1.0	1.0	0.581
L=512	0.983	1.0	0.686	0.872	1.0	0.998	0.997	1.0	1.0	0.892
L=1024	1.0	1.0	0.642	0.966	1.0	1.0	1.0	1.0	1.0	0.988

Table C.22: Meakin Invasion Percolation (g=-0.01) A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.606	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.119	0.0
L=32	0.055	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.139	0.0	0.0	0.0
L=128	0.0	0.089	0.0	0.513	0.0	0.0	0.0	0.0	0.582	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.047
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table C.23: Meakin Invasion Percolation (g=-0.01) Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.794	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.047	0.0
L=32	0.099	0.0	0.002	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.344	0.0	0.0	0.0
L=128	0.0	0.0	0.0	0.02	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table C.24: Meakin Invasion Percolation (g=-0.01) Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.522	0.936	0.65	0.919	0.942	0.682	0.923	0.983	0.516	0.972
L=32	0.607	0.936	0.58	0.878	0.91	0.681	0.825	0.963	0.689	0.923
L=64	0.682	0.966	0.545	0.926	0.926	0.726	0.838	0.981	0.76	0.953
L=128	0.776	0.993	0.501	0.895	0.929	0.791	0.669	0.997	0.902	0.919
L=256	0.896	0.882	0.589	0.699	0.606	0.886	0.689	0.883	0.632	0.636
L=512	0.953	0.98	0.704	0.526	1.0	0.935	0.944	1.0	0.98	0.745
L=1024	0.997	1.0	0.83	0.786	1.0	0.994	1.0	1.0	1.0	0.909

Table C.25: Meakin Invasion Percolation (g=-0.001) A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.597	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.7	0.0
L=32	0.009	0.0	0.051	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.268	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.974	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.029	0.0	0.01	0.0	0.0	0.0	0.001	0.001
L=512	0.0	0.0	0.0	0.528	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table C.26: Meakin Invasion Percolation (g=-0.001) Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.794	0.0	0.008	0.0	0.0	0.001	0.0	0.0	0.677	0.0
L=32	0.003	0.0	0.069	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.677	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.961	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=256	0.0	0.0	0.013	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.008	0.0	0.0	0.0	0.0	0.0	0.0
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table C.27: Meakin Invasion Percolation (g=-0.001) Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.548	0.937	0.629	0.916	0.932	0.681	0.909	0.978	0.542	0.965
L=32	0.61	0.931	0.559	0.885	0.893	0.673	0.82	0.959	0.681	0.927
L=64	0.684	0.954	0.505	0.913	0.9	0.684	0.819	0.96	0.662	0.922
L=128	0.735	0.964	0.574	0.953	0.88	0.682	0.861	0.956	0.59	0.943
L=256	0.811	0.989	0.657	0.949	0.9	0.698	0.724	0.977	0.836	0.902
L=512	0.94	0.992	0.816	0.889	0.72	0.769	0.679	0.929	0.818	0.626
L=1024	0.991	0.521	0.945	0.705	1.0	0.856	0.9	1.0	0.787	0.69

Table C.28: Meakin Invasion Percolation (g=-0.0001) A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.244	0.0	0.002	0.0	0.0	0.0	0.0	0.0	0.307	0.0
L=32	0.007	0.0	0.148	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.894	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.069	0.0	0.0	0.0	0.0	0.0	0.027	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.002
L=1024	0.0	0.61	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table C.29: Meakin Invasion Percolation (g=-0.0001) Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.443	0.0	0.02	0.0	0.0	0.0	0.0	0.0	0.794	0.0
L=32	0.031	0.0	0.344	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=64	0.0	0.0	0.961	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=128	0.0	0.0	0.193	0.0	0.0	0.0	0.0	0.0	0.069	0.0
L=256	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.0	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.002
L=1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Table C.30: Meakin Invasion Percolation (g=-0.0001) Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.577	1.0
L=32	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.544	1.0
L=64	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.525	1.0
L=128	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.515	1.0
L=256	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.529	1.0
L=512	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.504	1.0
L=1024	–	–	–	–	1.0	–	1.0	–	0.517	–

Table C.31: Hawick Invasion Percolation (beta=0.1) A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.06	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.284	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.552	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.721	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.487	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.923	0.0
L=1024	–	–	–	–	0.0	–	0.0	–	0.683	–

Table C.32: Hawick Invasion Percolation (beta=0.1) Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.096	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.67	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.959	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.789	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.437	0.0
L=512	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.89	0.0
L=1024	–	–	–	–	0.0	–	0.0	–	0.89	–

Table C.33: Hawick Invasion Percolation (beta=0.1) Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.514	1.0
L=32	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.521	1.0
L=64	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.51	1.0
L=128	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.546	1.0
L=256	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.562	1.0
L=512	1.0	1.0	–	1.0	1.0	–	1.0	–	0.516	–
L=1024	–	–	–	–		–		–	0	–

Table C.34: Hawick Invasion Percolation (beta=0.5) A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.734	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.61	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.81	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.265	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.132	0.0
L=512	0.0	0.0	–	0.0	0.0	–	0.0	–	0.693	–
L=1024	–	–	–	–		–		–		–

Table C.35: Hawick Invasion Percolation (beta=0.5) Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.959	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.67	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.959	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.096	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.136	0.0
L=512	0.0	0.0	–	0.0	0.0	–	0.0	–	0.549	–
L=1024	–	–	–	–		–		–		–

Table C.36: Hawick Invasion Percolation (beta=0.5) Kolmogorov–Smirnov results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.565	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.536	1.0
L=32	0.514	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.532	1.0
L=64	0.52	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.582	1.0
L=128	0.545	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.546	1.0
L=256	0.649	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.581	1.0
L=512	0.599	1.0	–	1.0	1.0	–	1.0	–	0.514	–
L=1024	–	–	–	–	–	–	–	–	0.517	–

Table C.37: Hawick Invasion Percolation (beta=0.9) A measure results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.116	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.389	0.0
L=32	0.735	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.432	0.0
L=64	0.636	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.045	0.0
L=128	0.273	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.268	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.049	0.0
L=512	0.017	0.0	–	0.0	0.0	–	0.0	–	0.736	–
L=1024	–	–	–	–	–	–	–	–	0.684	–

Table C.38: Hawick Invasion Percolation (beta=0.9) Mann-Whitney U results

	AVL-HSH	AVL-LAT	AVL-OCT	AVL-RES	HSH-LAT	HSH-OCT	HSH-RES	LAT-OCT	LAT-RES	OCT-RES
L=16	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.188	0.0
L=32	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.549	0.0
L=64	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.096	0.0
L=128	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.045	0.0
L=256	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.067	0.0
L=512	0.0	0.0	–	0.0	0.0	–	0.0	–	0.67	–
L=1024	–	–	–	–	–	–	–	–	0.89	–

Table C.39: Hawick Invasion Percolation (beta=0.9) Kolmogorov–Smirnov results