NTNU
Det skapende universitet

Inkrementell vekst av genomet for evolusjon av genotype representasjoner for kunstige cellulære organismer.

Andreas Giskeødegård

Master i datateknikk
Innlevert: juni 2013
Hovedveileder: Gunnar Tufte, IDI
Medveileder: Stefano Nichele, IDI

Norges teknisk-naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

Norwegian University of Science and Technology

Masters Thesis

Incremental Genome Growth for the Evolution of Genotype Representations of Artificial Cellular Organisms

Author:
Andreas Giskeødegård

Supervisor:
Gunnar Tufte
Co-supervisor:
Stefano Nichele

A thesis submitted in fulfilment of the requirements for the degree of Master of Computer Science

in the

The Computer Architecture and Design Group
Department of Computer and Information Science

June 2013

"There's so much possibility in their minds, but they are still, after all, individually stupid and small minded and half-blind and half-mad"

-The Hive Queen.

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Abstract

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Master of Computer Science

Incremental Genome Growth for the Evolution of Genotype Representations of Artificial Cellular Organisms

by Andreas Giskeødegård

In this thesis, we explore the possibilities of evolving cellular automaton attractors of different sizes with an algorithm which slowly expands the genotype of the individuals. Attractors of different sizes are grown in cellular automatons with different settings, to ensure that potential success is not strictly limited to the size and settings of the automaton. The properties of the attractors which are produced by this novel algorithm are compared to attractors evolved with static genotypes which provide similar and vastly larger search spaces. Comparing and analysing the results show that a slow expansion of search space during evolution produce results with compact and effective representations, favouring small transients, and the behaviour is persistent with changing cellular automata parameters. A large search space provided attractors with arbitrary transient lengths, and an excessive and ineffective usage of available chromosomes in the genotype. The attractors developed on a static genotype, but fixed to a size in which the growing genotype evolution was able to find solutions, gave similar results as the growing genotype. Both produced a small set of attractors, which repeated similar patterns and often yielded identical solutions. The set of different solutions produced by the fixed genotype is found to be slightly larger than that of a growing genotype, because of the growing genotypes favouring of short transients. This type of automatic generation of representation, is thought to be good alternative too a manual or bloated representation, and the reduced search space exploration, and the incremental process of improving fitness can be helpful both with runtimes, and with reducing the chances of getting stuck at local optimise. This is simply a proof that a evolution with a expanding search space is possible, and creates good results and representations, but needs further testing to be better adjusted and integrated to useful problem solving.

Acknowledgements

I would like to thank my supervisor Gunnar Tufte for having me as a masters degree student, and for advice to settle a confused mind early in my career at NTNU.
Further I would commend and thank my co-supervisor on an excellent job, and a stable and reliable source of inspiration in a large and confusing research-field.
Lastly I would like to thank my family and friends for helping in any way they could, giving me good moral support, and some well meant temptations and breaks...

Contents

List of Figures

List of Tables

Abbreviations

CA **C**ellular **A**Automata

EA **E**volutionary **A**lgorithm

GA **G**enetic **A**lgorithm

Chapter 1

Introduction

Within the field of complex systems, and cellular computation, the limitations of computation is very different than in todays mainly sequential computation paradigm. The sequential computation is pushing the boundaries of the physical capabilities of the material it is produced in, while the theory of cellular computation is a untapped source, and can be potentially used as a vastly parallel, and scalable computation tool[1]. Human beings normally process thoughts and problems in a sequential pattern, making a vastly parallel structure difficult to program and utilise. To bridge the gap where humans lack in ability to program for large parallel systems, we have taken to the use of adaptive programming[2]. Adaptive programming is were algorithms are used to create programs or systems which take advantage of and creates solutions in complex systems like cellular computation. These adaptive programming techniques involve the usage of methods like evolutionary algorithms. But evolutionary algorithms suffer from an inherent scalability problem[3], which makes the construction of large and complex cellular computation systems difficult. For instance is the representation of a problem and solution often designed in a 1 to 1 relationship, making the problem as large as the solution. To increase the scalability of the evolutionary algorithms, researchers have for instance gotten inspiration from nature, and started using development along side evolution[3], so that the solutions which are created are much smaller than the problem, and thus increasing the scalability.

There are still multiple difficulties with the usage of adaptive programming for problem solving, like the obstacle of finding a representation for a solution which is large

enough to contain a real solution. This is a difficult task and is done with large amounts of heuristics[4]. Solutions which use development have many other emergent properties, which are not as obvious as a smaller representation. There are effects such as scalability[5], and robustness[6]. Nature is a prime example on robust and scalable development of organisms, so taking further inspiration from her(Mother nature) seems wise.

By taking inspiration from complexification, growing representation, and nature, we created an algorithm to see if this kind of function would translate to the evolution development systems. Further exploring how this could automatically create a representation size without the usage of difficult heuristics, and have the sizes be compact and effective. The gradual increase in search space, could theoretically reduce the likelihood of a evolution getting stuck at a local optima, and would need to explore less of the state space than a normal fixed representation does. Cellular automata are used as a representation model of development from a genotype to a phenotype, which also can be translated to cellular computation. A gradual growth in search space should result in smaller more compact solutions, and seeing how a smaller representation affects behaviour in the resulting organism should be interesting.

A framework to run the algorithm was designed and implemented, and then configured. The framework is designed for running three different kinds of evolution functions; Growing, Restricted, and Full. in the Growing, the representation, i.e. genotype, grows until it is a size where a solution is found. The Full has a genotype size equal to the rule space, which gives it a chromosome for every configuration the neighbourhood of a site can achieve. As for the Restricted, it also has a fixed genotype size, but it is significantly less than the genotype size for the Full. The framework uses attractors of different lengths as measures of complexity and problems to evolve. To check if the algorithm was able to create valid results, at a reasonable rate, and was able to create "better" representations than the Full, three experiments were designed and run. The experiments would find attractors of different lengths, with the three types of evolution, and further check to see if the results are persistent when changing the state space with an expansion in geometry and state size for the cellular automaton.

The thesis is structured by having a background/introduction to the research field in Chapter 2. This is followed by Chapter 3, where the main hypothesis is explained, and

what parts of this we will try to explore in the experiments. In Chapter 4 there is a thorough explanation of how the framework functions, as to give a context to how the results were created, and the means to replicate a system which creates similar or same results. Getting results without the ability to recreate or show where they came from will essentially make the results hollow, and questionably imaginary. Chapter 5 explained the two main parts of the experimentation, which is first, the configuration of the framework, needed to make it run and be able to produce results. Second, the 3 above explained experiments to see if the system is able to create solutions, and have a persistent behaviour when changing parameters in the cellular automaton. Chapter 6 will present the results in two main section, being configuration and experimentation. Then Chapter 7 will discuss findings and properties of the results, and suggestions on how this can be further explored, and how the results may contribute to the field. Chapter 9 will be a conclusive chapter with a summary of the thesis, results and final thoughts.

Chapter 2

Background information and theory

2.1 Cellular automata

2.1.1 History

In the early 1950's John von Neumann explored the idea of creating a machine who could create an exact copy of itself, a self-replicating machine[7]. Construction or simulation of such a machine was very difficult back then, so instead von Neumann studied the logical structure such a machine would need. In his study, von Neumann created a framework which he called "cellular spaces". The framework is a dynamical system with a discrete understanding of time and space[8]. Multiple sites are connected to other sites, and each site has multiple states. The state for the next discrete time step would be based of the states of all the connected sites. His design worked as a self-replicating machine, and has in later years been simplified multiple times by the likes of Codd[9], Banks[10], Conway[11][12] to name some. "Cellular spaces" has also changed name, and is now called Cellular automata(CA). The original purpose was to study and model biological systems, but has later been reintroduced with a variety of purposes under multiple names like; tessellation automata, homogeneous structures, cellular structures, tessellation structures, and iterative arrays[13].

There is a theory called the theory of everything, which is basically the idea that everything in the universe is built up of the same fundamental particle. The idea of a fundamental particle has been around for a very long time, and could easily have been an influence on the CA framework. CA sites can be seen as a fundamental particle, and thus the rules for a CA can simulate the rules for the fundamental particles in the theory of everything[14].

2.1.2 Characteristics

There are some terms which are needed when talking about CA:

Development step:
The movement from one discrete time step to the next within the geometry of the CA. During this step each site checked for matching rules in the CA genotype. If a matching rule is found, the site changes state to the next state determined by the matched rule.

Attractor:
A set of development steps which ends up with the geometry of the first step being equal to the geometry of the last step. Essentially creating a loop in the geometry which will continue as long as the development is run.

Point attractor:
An attractor whose set consists of a single development step which makes no change to its geometry.

Transient:
A set of development steps which leads from the very first development step, to the first development step of an attractor.

Trajectory:
A set of development steps consisting of the set of a transient and its belonging attractor.

As for describing the construction and functionality of the CA, it can be split into 4 parts[15].

2.1.2.1 Geometry of the Cellular Automaton

The geometry of a CA describes the area containing all the sites, and describes their placement in space. This can be defined in different ways, but the normal definition is on a n-dimensional grid where n is the number of dimensions requested. An example of a more unconventional way of definition can be done through group graphs[15], which is just an example and will not be further used. When constructing the geometry of an n-dimensional grid and the size of n has been determined, the functionality of the geometry should be decided as either infinite or finite. An infinite geometry will grow with the structure, no matter how large it gets, while a finite geometry will have a fixed size in all n dimensions. When a growing structure reaches the boundary of a finite geometry, it needs a boundary condition. Boundary conditions are how to apply local transition rules to a site when parts of its neighbourhood is outside the geometry space[15][16].

Null boundary condition makes the CA believe that all sites outside geometry space is in the null/quiescent state.

Fixed boundary condition makes the CA believe that all sites outside geometry space is in a static predefined state.

Periodic boundary condition is when the geometry space is considered folded in its dimension. Meaning if a site has a neighbour outside of geometry space, the dimension would fold and the neighbour would be the extreme site on the other side of the dimension, as seen in Figure 4.1. This can be represented with a piece of string (1-dimension) which gets tied in a loop, i.e. folded.

Mixed boundary condition is when there can be different condition on different boundaries.

2.1.2.2 States of the sites

A site is an entity which can hold, and change state. The different kinds of states the site can hold is a finite set of states which can be called a state set. Normally all sites in a CA have the same state set to chose from, this is called a monogeneous CA[16]. If sites in a CA have different state sets, it is called a polygeneous CA [15]. One of the states in a state set is defined as a null/quiescent state. This quiescent state is the default off state for the CA, and the state all sites except the initial configuration has at development step 0. For a state to turn from the quiescent state to another state

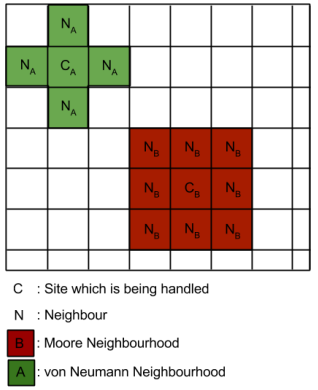


FIGURE 2.1: Image displaying the von Neumann, and the Moore neighbourhoods for cellular automata .

there should normally be a site with a different state than the quiescent state in its neighbourhood.

2.1.2.3 Neighbourhood of sites

For an arbitrary site, there is a set of sites which it is connected to that is called a neighbourhood. In some cases this neighbourhood is decided by the geometry of the CA like in group graphs, but in n-dimensional CA the neighbourhood can be designed. The size and relative placement of a neighbourhood is uniform across the geometry for CA, changing from uniform to non uniform will change the classification to random boolean networks, or neural networks. In a CA with 2-dimensional geography there are two common neighbourhood structure, the von Neumann, and the Moore neighbourhood[17]. In Figure 2.1, we can observe that in a Moore neighbourhood the relative placement of the neighbourhood is always the 8 sites closest to the current site in addition to the current site, while in the von Neumann neighbourhood it is only the perpendicular sites of the current site in addition to the current site.

The neighbourhood is defined with and input and an output neighbourhood. An input neighbourhood is the sites on which the current sites next state is dependant. While an

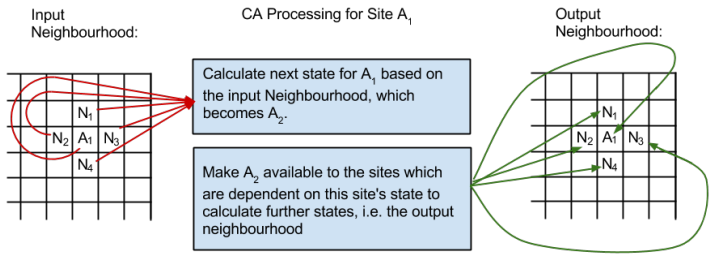


FIGURE 2.2: Showing how input and output neighbourhoods work.

output neighbourhood is the sites which need the state of the current site to calculate their next state[16]. This is displayed in Figure 2.2. The current site does not need to be part of neither its input or its output neighbourhood. If the input and output neighbourhood is the exact same sites, it is called input output symmetric neighbourhood. While if they are not the exact same, but the same size, it is called input output balanced neighbourhood[15][16].

2.1.2.4 Local transition rule for the sites

The local transition rule is a function or a table which a site uses to extract its state for the next discrete time step. For CA there are a multitude of different permutations of rules that can be used, and the number of possible rules depend on the number of states and the size of the neighbourhood. The number of possible transition functions for a CA is defined by equation 2.1, which grows at an exponential speed[18].

$$K^{K^N} \tag{2.1}$$

These functions are usually deterministic, but some non deterministic functions have been studied in connection with language theory[15]. In the usage of transition functions on a CA the application can differentiate:

Uniform CA is when all the sites have the exact same transition functions[19].

Non uniform CA is when cells can have different transition functions[19].

Hybrid CA is when all the cells in a CA has different transition functions[16].

The above CA types define the spread and differentiation in transition functions in sites. But there are also variations of these which are able to change the used transition functions between discrete time steps, these are called Programmable CA[15].

2.1.3 Cellular automaton classification

Stephen Wolfram, who has dedicated many years of his life to the study of CA and complex systems, has classified the behaviour of CA into different classes[20]. The classes was firstly based on the different appearance and behaviour of the visualisation of different rules for simple 1-dimensional CA[20]. Later it has be analysed and and also shown as a good classification for higher dimension CA such as 2, 3, and possibly further[21].

2.1.3.1 Class 1, Homogeneous state

In this class, almost all initial configurations develop into a unique homogeneous state. This ensures the complete destruction of all information contained in the initial state after few development steps. Class 1 CS have either multiple attractors which all lead to a single point attractor, or one wide point attractor, making it impossible to induce the initial configuration from the finished[20].

2.1.3.2 Class 2, Simple stable or periodic structures

In this class, after a few development steps the initial configuration develops into separate stable and/or periodic structures.[20] This class is made up of CA with different attractors which do not necessarily act as transient structures and end up in one large attractor. The attractors can be both point and cyclic, but the cyclic attractors have a short cycle. These CA have been thought of as filters on the initial condition, where the transition rules decide which of the initial conditions are allowed to propagate and live and, and which will die/sent to the quiescent state[20]. In class 2, changes in the initial condition have a finite range of influence since after reaching its attractors, a class 2 CA loses all propagation of information.

2.1.3.3 Class 3, Chaotic pattern

In this class, almost any initial condition will lead to an aperiodic/chaotic pattern. This aperiodic chaos can create structures or patterns, which emerge with frequencies and placements ranging from irregular to regular[20]. Within this chaos there is a trend in 1-dimensional CA that the density of non quiescent state cells tend toward a fixed number. This number is often $\frac{1}{K}$ where K is the size of the state set[20]. The average minimum propagation speed for a class 3 CA is always above 0, so it will never disappear. Because of this there will never be a cycle in an infinite class 3 CA, such as there is in both class 2 and 1. This means that every sites value is dependent on an ever increasing number of initial states. In a finite CA there will always be a cycle, since there is a finite number of possible formations of states that can be formed in the finite geometry. State space defines the number of possible formations and is given by equation 2.2 For class 3 CA the average cycle length normally grows quite slowly with increase in the neighbourhood size. As for the transient, it is normally short for class 3 CA with regular patterns, while irregular pattern class 3 CA seem to often create very long transients with a growth in neighbourhood size.

2.1.3.4 Class 4, Long-lived complex structures

In class for there is a some elements from all the other classes. There are some initial conditions which end up as class 2, with stable or periodic structures. Much of the initial conditions act as class 1, in that it returns to the quiescent state. But some parts function almost like a mix of class 2 and class 3, in that there is borderline chaotic and aperiodic patterns, but holds some of the more structured elements of class 2. Approximately 93% of the initial configurations comes to a halt (like class 1), while 7% generate persistent structures (like the class 2/3)[20]. It is proven that many of the CA located in this class is capable of universal computation, or at least computation[18]. The emergence of class 4 CA in 1-dimensional geometry has a minimum requirement of $k > 2$ or$r > 1$, where k is the number of states and r is the number of neighbours on each side of the current cell. This shows that the threshold to develop complexity is quite low[22]. Class 4 CA are probably the most interesting CA, but also the rarest. In the possible CA for a state and neighbourhood setup, the occurrence of class 4 CA is only a few percent, even with high neighbourhoods and state size[22]. A fascinating statistical occurrence is that class

class 4 CA with different rules seem to often yield qualitatively similar behaviour and similar persistent structures[20]. With regards to the statistical properties of class 1-3, they exhibit definite properties in the "infinite volume", this is because fluctuations gets smaller as larger systems are inspected. This is not the case for class 4 CA[20], since in larger cases there can arise structures which propagate over vast distances, or create structures which produce "signals", like glider guns [23]. An arbitrary change in the initial condition can create an arbitrary change in the development process which can influence the entire CA.

2.1.4 Current usage of cellular automata

The CA is a completely different way of doing calculations than in computers. There is a vast parallelism and scalability, with a decentralised control of the system. This makes CA ideal for simulating complex biological and physical systems. Normal computers can also do this simulation, but a CA would do better since the workings of CA are closer to natural systems than a computer is. There are multiple areas where CA are being used in todays society like: gas behaviour [24], ferromagnetism [24], forest fire propagation [25], urban development [26], turbulence in fluids [24], economy [27], spread of criminality [28], traffic flow [29], immunology and biological ageing [24], the flow of electricity in a power grid [24], and crystallization [24].

Other than modelling and simulation, the chaotic and irrational nature of class 3 CA make CA a good source of random numbers. Random numbers have applications in multiple fields like statistical sampling, computer simulation, cryptography, gambling, and more. Wolfram's Mathematica uses the 1-dimensional CA with rule 30 as a source for its random numbers[21].

2.1.5 Future possibilities

The way CA can represent complex systems makes it a good platform for potential new computing paradigms. By using simple elements with small computational functionality, the construction of the cells can be become easy. With a distributed control system, all the cells need not be connected directly to a controller which scales much better than the current central controller in respect to building difficulty. By scaling up the parallelism

on a normal computing system with a central controller, not only is there an issue with the central controllers capacity, but also with the geometry, design, and placement of all the wires. A decentralised CA would be able to scale infinite without an increase in construction complexity in comparison[1]. This can be a possible way to create vastly parallel computers or computer components to help us handle the data deluge. If the future holds machines using cellular computation and CA, they would require a whole new way of looking at programming. But not only would one acquire vastly parallel machines to handle huge data tasks, the programs would also be robust. They might be able to perform self repair if parts of the system fails, or at least perform graceful terminations[6]. Another idea would be in the aspect of artificial life. If our world is constructed on the same principles as systemic computation[30], with fundamental particles being the building blocks[14] [31], there is the possibility that using an artificial fundamental particle(could be hypothetical or digital) with another or the same set of rules could result in artificial life.

2.2 Computation

Computation is defined as the process of arriving at an output state, which happens when applying a set of rules to an initial state[32]. Some simple examples would be doing calculus with an abacus, or running a computer program. In a computer, the hardware [14] is what makes computation possible, it is the logic circuits in which rules can be made. The hardware has a set of rules that are valid rules, i.e. instruction sets. When the valid rules are put together in combinations, we create programs. Giving these programs an initial state, and then setting the computer to work, computation is being done. In todays computer development we have reached some barriers, like the memory and powerwall, and having CPUs which contain so many transistors that powering all of them would make cooling impossible. To continue the growth in processing capability, the road has lead to implementing more processing units which work together, i.e. parallel processing. Within parallel processing the computation works the same way as in sequential computers, only that some parts can be performed at the same time. With cellular computation, we change the amount of computation a single processing unit can do to a low amount compared to todays processing units but we let them work together in vast parallel amounts. The main idea about computation does not change,

but in cellular computation, the computation of a single processing unit is simple, but when looking at a vast amount of them they will be able to compute arbitrary complex computations. There have been built many universal computation machines within cellular space, but they replicate sequential behaviour, which gives a low exploitation of the potential in vast parallelism. For cellular computation to become effective, it must be applied to areas where its strengths are exploited, or it must be a basis of a whole new computer/computation paradigm[1].

2.2.1 Universality

Universality is the idea of something being able to perform all tasks without any sort of rebuilding or reconstruction, only change the programming[20], which for a CA would be the initial condition. This idea does not consider time as an issue, so this universal construction can be infinitely slower than a specialised construction at performing a specified task. The term universality can be adopted into categories, for example the the category of computation. In computation, universal computation is a term for a construction that can do all computational tasks by simply editing the programming. A computer is in most cases considered a universal computation machine[18], but in theory a program can be infinite in size which is impossible with our current technology. The specialised version of universality is normally what is used in research. Proving that something is universal in its category, like computation, can be extremely difficult and technical. There are systems which have been proven to exhibit universal computation. So by proving that a new system can simulate systems which already have been proven to exhibit universal computation, one would prove that the new system also exhibits universal computation. If a system with different possibilities can be programmed to act as a proven universal system, the system is universal[33].

2.2.1.1 Universality in Cellular automata

A universal cellular automata is a CA which can simulate all other CA with equal or lower state size with only changing the initial condition. A CA capable of universal computation is a CA which can do any computation with only changing the initial condition. Both of these types have been found[34]. There are many CA capable of universal computation, ranging from von Neumann's huge complex construction to Wolframs one

dimensional rule 110 [35]. The simplest way to prove a CA is capable of universal computation is to do like Banks, Codd and many others[10][18], let the CA act as a logical universe wherein the construction of a computer is possible. Create patterns and structures which can perform three fundamental tasks[18]. First, Storage of information over arbitrarily long times. Second, transmission of information over arbitrarily long distances. Third, the interaction between information which has been stored and transmitted, then potentially modified by the interaction. By defining these parts, they were able to prove that their CA could do anything that a hardwired machine or a turing machine could do[10]. A turing machine is capable of universal computation, while hardwired computer is potentially capable of universal computation. Potentially is here used because for it to be capable of universal computation, it needs infinite storage space [36]. Because of the intricate behaviour and complexity needed for universal computation, the CA which can be able to exhibit universal computation is likely in class 4 CA[20]. This does not mean that all class 4 CA exhibit universal computation, only that these kinds of CA are more likely to. Wolfram stated in [22] "In general the behaviour of a universal computer cannot be predicted and can be determined only by a procedure equivalent to observing the universal computer itself.", and for class 4 CA a specific site state may depend on many initial site state, and may apparently only be determined by an algorithm equivalent in complexity to explicit simulation of the CA development.

2.3 Emergence

2.3.1 Emergent properties

Emergent properties is an old philosophical term, which George Henry Lewes wrote about in 1879 in his book called "Problems of Life and Mind". The idea was based on chemistry and biology. Roughly summarised the idea is that when fundamental entities act or react together, there are entities which arise/emerge (such as patterns, substances, properties). These emergent entities are considered novel or irreducible with respect to the fundamental entities[37] [38]. For example it can be said that the consciousness is an emergent property of our brain. The idea of emergence in our world can be brought to universal scale, where one could claim that our universe and everything in it is only layers

of emergent properties of simple fundamental entities. This school of thought is called emergentism. Emergentism lies between reductionism and vitalism. While emergentism is the idea that everything are emergent properties of fundamental particles which are irreducible to its micro state, reductionism means that everything can be reduced to the sum of its parts. Vitalism on the other hand is the idea that for all living things, there must be something else which constructs the consciousness(or soul)[37]. According to Yaneer Bar-Yam[39] there are two types of emergent properties which can appear:

Local emergence: When the emergent property is not uniquely connected to this system, a tiny sub system of the entire system has the same emergent property as the whole. An example of this would be the behaviour of gas, were pressure and temperature are emergent properties. Temperature and pressure is equal for the entire gas, analyse a small part of it and the properties are the same as the whole system[39].

Global emergence: When the emergent property is strictly related to the behaviour of every single part of the entire system. Analyse only a small part of a system with global emergence, and the emergent properties would not be the same as for the whole system. Here each part of the whole system plays a specific role, and the role is tightly dependant on the behaviour of all the other parts of the system to achieve the global emergence. Yaneer Bar-Yam has an example using Hopfield networks[39].

2.3.2 Emergent properties in cellular automata

A CA has both local and global emergent properties. Local emergent properties can easily be seen in class 1 CA, and can also be seen in class 3 CA (for instance as the density of states). Class 4 CA hold the global emergence were single structures can propagate and influence any part of the system, but viewed alone it would not give any hint to the emergent property of the global state. For most research on emergent properties in CA, there is a focus on computational properties, how to make the system perform complex calculations [40]. For most of these calculations the usage of complex systems are employed, and a CA is an excellent tool for simulating complex systems, since it has been showed that CA have the possibility of emergent complexity[41][42][43].

2.3.3 Complexity

Emergent properties can create complex systems. The defining factors of a complex system is that the building blocks act together interdependently to create a system. Taking a part of the system out of its context would remove all its functionality, and render any study or statistics found useless in regards to the whole system. Removing a part would also alter the functionality of the entire system[39][44]. As quoted from Bay-Yam[39] "It is because we cannot describe the whole without describing each part, and because each part must be described in relation to other parts, that complex systems are difficult to understand" The classification of a complex system makes it the same as the global emergence of emergent properties. This means that global emergence is the emergence of complexity. When visualising a complex system it is good to note that the building blocks of the systems can be systems of their own. An example of this can be a society, which is a complex system, were the building blocks are human beings, which can be viewed as systems. This wrapping of systems within systems scale well from a micro to a macro view, and is called systemic computation[30]. Here the building blocks of a system do not have to be simple CA states, they can be systems ranging from complex to simple. The concept of simple systems emerging from complex building blocks is called emergent simplicity, while complex systems emerging from simple building blocks is called emergent complexity[39].

2.3.3.1 Measuring complexity

Understanding what a complex system is, makes it harder to define how complex it is with regards to other complex system. A procedure for extracting a number for this complexity which is generally accepted is not yet available. To actually have such a procedure would help to globally evaluate hypothesis. Without such a procedure any hypothesis based on complexity would not be refutable, and thus proof it delivered can not be a basis of anything[45]. Since it has been argued by Grassberger[46] that no single quantity is sufficient to measure complexity, because it depends on how meaning is assigned to the term, the logical way would be to create some generally accepted ways of measuring complexity based of different ways of observing the complex system. Resulting measures should be able to express complexity in multiple areas, like structure, genotype, phenotype, function[45]. There already exists multiple ways of measuring complexity,

but they are not generally accepted as the correct way to measure or on exactly what kind of complexity they measure. Research on the subject has been done[45], but have not resulted in any specific generally accepted measurements. Some of the different kinds of measurements which exists are kolmogorov complexity, Shannon entropy, Functional complexity, Bennet's logical depth, and physical complexity. Taras Kowaliw studied complexity measurement and found 3 types which he thought would be best suited for artificial embryogeny, which is what development in a CA is. These measurement types were phenotypic, genotypic and functional complexity, but he embroided them a little[45].

For CA, Langton[18] defined a variable which indicated what class a CA was likely to be. The lambda variable(λ) is used to find probabilities of a CA being in either ordered (Class 1 and 2), chaotic(class 3), or in between(class 4). This parameter is also a clear indication on average attractor and transient length. There are also other types of parameter which can be used to analyse CA, were there are clear coherence of the variable value and the attractor and transient length[44].

2.4 Representation

In nature, we can think of living organisms as complex systems. The construction of living organisms is done with cells reacting to nearby proteins by either action or further synthesis of proteins. The DNA which contains the genes is not a blueprint on how an organism will turn out, there is a process were the genes develop, from genes to a functional organism[47]. A Danish scientist called Wilhelm Johannsen came up with the idea of genotype and phenotype[48], which is used to separate the building rules and the distinct features of a fully grown organism. The genotype is the collection of genes located in the DNA. While the phenotype is the observable attributes of the organism when the growth has subsided / partially stabilised. Examples of a phenotype could be eye colour, amount of hair on the head of a person, or the colour or smell of flower. When trying to simulate natural systems, with the use of either evolution or development, a representation of the item we want to evolve or develop is needed. When using computers this representation is often done with binary strings, were an example could be that the first four bits represent one gene, and the next four represent another gene etc. But to be able to interpret the bit-genes to a phenotype, there needs

to be some kind of mapping functionality. There are many different ways of doing this mapping, but some common ones are described next.

2.4.1 Direct binary mapping

In binary mapping, a gene can be either on or off, and is represented by a single bit. This is a one to one mapping, meaning that a single gene in the genotype maps directly to a single trait in the phenotype [49][50]. For example having "gene 1" of the genotype represent the eye colour for the organism. Changing "gene 1" would directly change the eye colour of the organism. With this kind of mapping there is a 1 to 1 relationship between geno- and phenotype, which represents a model without development but with strict control that makes it good for manual design.

2.4.2 Voting mapping

This is a many to one mapping where multiple bits in a genotype represent one phenotypic trait. The phenotypical trait is dependant on the majority of the bits in the genotype, like each of the bits in the genotype can cast a vote on what phenotype to express[49][50]. Voting mappings resulting effect on evolutionary algorithms is that they generally need bigger mutations on a genotype level for the result to show in the phenotype. This makes changes more gradually, and follows the idea that a small change in genotype should result in a small change in phenotype, which both are beneficial for evolution. A negative side of this kind of mapping is the scalability, were the development of specific phenotypes would require much larger genotypes. There are another type of voting mapping which using the same type of many to one mapping, lets single bits in the genotype cast its "vote" on multiple phenotypic traits. This reduces the scaling problem, and helps the solution from getting stuck on false fitness optimise in the fitness landscape[51]. Voting mapping is not a model that uses development.

2.4.3 Cursor based mapping

Cursor based mapping is a mapping which uses development. The idea is to have a cursor which can move around the phenotype and change states/traits of it. For this mapping the genotype is a sequence of commands which the cursor can perform, while

the phenotype is the result after all the genotype commands have been executed. Here a gene in a genotype is not directly mapped to a phenotypical trait, but mapped to an arbitrary change in the phenotype[49][50]. Using this kind of mapping, one could construct many different phenotypes with the same rules, but the genotype would grow very large depending on the complexity of the phenotype.

2.4.4 Cellular automata mapping

A CA can be interpreted as an autonomous developmental geno- to phenotype mapping. The way it behaves can be compared to natures way of growing organisms, where the DNA would be storage of the genotype that can contain any permutation of the transition functions allowed by the size of the neighbourhood and the number of states. For an organism the genotype would be a specific permutation or set of rules, and the phenotype would be the structure of the CA after it has stabilised in some attractor, or when an acceptable number of iteration of rules have been applied[49][50]. Using this mapping the size of the genotype can be smaller than the phenotype, which is good when scaling to bigger organisms. But this construction is dependant on an initial condition, which is not found in the genotype like some of the other mappings. There are issues with using CA mapping in evolutionary algorithms, since the effect of small changes in genotype can result in massive phenotype changes, it is computationally hard to program the organisms.

2.5 Embryogeny or Development

In nature the growth from genotype to phenotype is not a direct mapping. There is no single gene for eye colour, or for each strand of hair on your head. If there was a gene for every cell in your body, the genotype would be much larger than it is. The way natures uses the genotype to phenotype mapping is through Embryogeny, which is normally called Development[47]. Natures genotype consists mainly of genes for proteins, and the synthesis of proteins. Proteins concentration in the environment makes cells react in different ways[47], and these reactions of the cells will over time grow, change, and develop an organism. If visualising that the organism being developed is a table, the genotype would be how to create the different materials, like metal and wood, and ways

of shaping these materials, like long planks, nails, screws etc. We see that the table genotype could be used to construct many different tables, but the environment and initial condition is what decides how it will end up. In our analogy a person deciding how to shape the metal and wood, and how to put it together would be an example of an initial condition. The concept of computational development, is the abstraction of development as observed in nature, but within computational systems. To efficiently use the idea of computational development, it is often combined with other ideas. Computational development is very often used in relation to genotype-phenotype mappings, and evolutionary algorithms. Genotype-phenotype mappings with regards to development is quite self explanatory, while the connection to evolution might be somewhat obscure. Evolution is used because the number of permutations of possible developed structures are large, and the task of manually finding the optimal structure would be insurmountable. So using evolution as a search algorithm works well. Computational development can also be viewed as a process for CA. Here the development process can be viewed as the transient. Which is the multiple time steps with applications of rules to an initial condition resulting in a stable pattern or structure. Using development in computational work has both great benefits and difficulties. In his article Kumar [47] lists both the disadvantages;

- Difficult to evolve by computer, since small changes can result in big consequences which is dissimilar to evolutions normal procedure.
- Difficult to analyse.
- Difficult to create by hand because of the vast amount of possibilities and the complex structures which can emerge.
- Computationally expensive.

He also lists the potential advantages of using computational development techniques for solutions to problems such as;

- Easier to simulate biological hypothesis and experiments without doing wet-experiments.
- Will reduce the genotype in respect to the developed phenotype.
- Phenotypes which can regenerate and repair or die gracefully.

- Emergence of complexity will happen automatically instead of being programmed into the genotype.
- The regulation of development which is how implicit embryogenies works, and helps effects in development to develop gradually instead of instantly.
- Systems with great adaptability. Either to initial conditions, or to environment or to changes etc.
- Anticipate that it will enhance the capabilities for evolutionary computation and artificial life.

Kumar also categorises computational development into 3 types of embryogenies [52]. They are external, explicit, and implicit embryogenies.

2.5.1 External Embryogenies

External Embryogenies is a system where the meaning of the genes in a genotype is stored in a separate external area, which is not mutated during evolution. As an example one could have a genotype were gene-21 would always be the colour of the skin of an organism. Any mutation occurring within gene-21 would result in a different skin colour, but no amount of mutation outside of gene-21 would be able to change skin colour. External embryogenies makes the development conform to stricter rules, which gives the user more control but less diversity. This does not mean that the finished organism is worse at performing its function compared to other embryogenies.

2.5.2 Explicit Embryogenies

Explicit Embryogenies is when each step of the development process is explicitly defined and stored in the genotype along with other potential information. This entails that evolution would be able to affect any single part of the development process. Hand designing these processes are much more difficult than with the external embryogenies, but it is possible. The normal approach is to use some evolutionary algorithm to optimise the process.

2.5.3 Explicit Embryogenies

Implicit Embryogenies is when the development process is not defined, but is a process which emerges from elements reacting together according to rules defined in the genotype. The big difference between implicit and explicit embryogenies is; in explicit every step of the development process is explicitly defined in the genotype, and the development process rules and the functional rules are separate rules even if they might be in the same genotype. While implicit embryogenies does not have a specific set of rules for the development process. The rules for the development process is the same as for other behaviour, and a phenotype-state at a time step determines which rules to apply for the process to continue to the next state. Implicit embryogenies is the development process closest to natural systems.

2.6 Evolution

2.6.1 What is evolution

Charles Darwin is knows as the inventor of our understanding of evolution, and did the scientific work to back it up. He published a book called "On the origin of species" in 1859 [53]. The main idea in behind natural selection, which leads to evolution, is that the individuals who have the best abilities to survive, and/or to attract a mate (these are not necessarily the same, just look at the peacock), are more probable to have multiple offspring. Thus their genetic material will have a larger impact on the next generation than an individual with less apt abilities for survival and mate attraction. This genetic material will change slowly by mutation and crossover, which can make it better or worse. The better will again have an advantage over the less apt genes. This leads to slowly changing the genes in a population, toward the genetics of the fittest individuals. Evolution is the process were accumulations of self-replicating systems change their default genotype and phenotype over time by mutations in the genome, or by sexual reproduction, or both. The changes happen gradually over long time periods(in regard to the life of the reproducing elements), which involved multiple generation switches. This random changing and mixing of genes produce multiple traits, some helpful and others not. Traits that happen to be better adapted to the environment have better

chances of survival and reproduction and their traits will be more likely to continue to be propagated down the generations. Kumar says that[47]; evolution is like a river flowing down a rugged landscape, it always finds the easiest path. And by saying this he points out that evolution might be lazy, but i see evolution as thorough and pragmatic, it checks most of the solutions and possibilities available within the start potential, and uses the one which improves the organism best.

2.6.2 Using evolution for problem solving

Evolution, like many other biological systems have been adapted by technology, and we have tried to simulate it to help us solve problems. The way evolution is adapted to the digital world of computation is through Evolutionary Algorithms(EA). There are different kinds of EA[47]; Genetic Algorithm, Genetic Programming, Evolutionary Programming, Evolutionary Strategies. The main purpose for any EA is to find the optimal solution to some kind of problem, but all possible outcomes are known and located in the possible search space for this problem. By looking at multiple random placed outcomes in the search space, and gently changing them over time to look at the difference in regards to a best possible solution, the algorithms will "search" its way to a hopefully optimal solution. So EA are a crossing between a random generator, and a search algorithm that is tuned so they work good together. They are used for instance to optimise the design of systems, and solving of multi-dimensional problems better than software produced by human designers[54]. For EA there is two precondition, one which implies that there must be a goal, some functionality which is desirable. The other is a way of measuring how close an outcome is to this goal, or how well it performs its functionality. This vicinity to a goal, or performance is called fitness, and the procedure which measures it is called a fitness function.

2.6.2.1 GA as an example

All EA have some kind of representation of the outcome and an interpretation of results. Of the above mentioned EA, it was only Genetic Algorithm who originally made use of a representation with a split genotype and phenotype. Which is why we will use GA as an example on how EA work. For notations sake; it is completely possible to change other EA to use a separate geno-phenotype representation. Figure 2.3 show the normal flow for

a GA. The initial steps of configuring the GA, and initialising the population has been left out, and Figure 2.3 shows only the basic parts of how a GA works. After initialisation of the population, the main procedure to find an optimal individual starts. If an optimal individual is found, the termination condition is reached. The loop which consists of selection, crossover and mutation is used to iterate through the generations, and create new generations based of the previous generations best individuals. In the selection process, two or more individuals for the current generation are randomly selected with weighting based on the fitness of the individuals. The selected individuals will be placed into the next generation. But on its way to the next generation it can be changed in different ways. The ways which can change selected individuals is crossover of genes within the currently selected individuals, or random mutations. When the first selected individuals have been placed in the next generation, with or without change, the new generation is still almost empty. Filling the next generation is done by continuously repeating the selection, crossover and mutation part of the algorithm. When this is done, the individuals are checked to see if a optimal individual has been found, if yes the GA is finished, if not the selection, crossover, and mutation will have to be repeated again. This entire process continues until an optimal individual is found. This is the workings of the basic GA.

2.7 Scalability

2.7.1 Defining scalability

Scalability is the quality of being scalable, which is to be able to be changed in size or scale. When discussing scalability with regards to systems, it is the measure of its ability to retain functionality when changing the size of the problem which the system is applied to. In this definition the term system can be applied to anything, from morphogenesis, to communication infrastructure, to algorithms. As an example for a system lacking in scalability, one can observe a bubble sort algorithm sorting a large data set. And as an example of good scalability we can look at morphogenesis, where the system of building organisms from a zygote using rules, better known as chromosomes, described in a genome, is used by all known animals. Some have small genomes, some have small bodies, but they use the same system, which gives it good scalability.

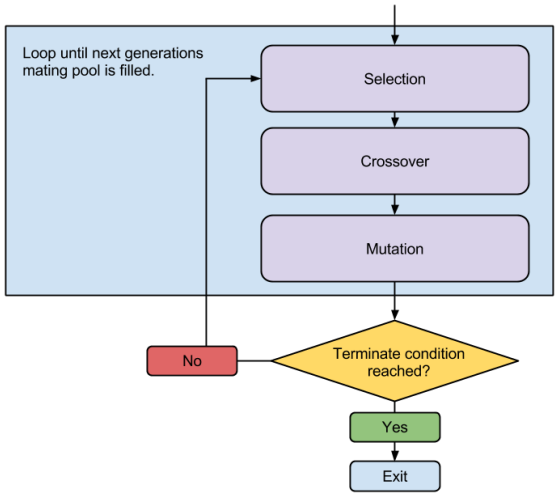


FIGURE 2.3: Normal functionality of a default genetic algorithm.

2.7.2 Scalability within context

Scalability within the boundaries of EA and CA strongly relate to some definitions of spaces. These spaces are interconnected, which makes the scalability aspects also interconnected.

2.7.2.1 Spaces

When thinking of these spaces they are defined according to an EA which is used to evolve an organism with the use of development in the form of a CA. This means that the ordinary thought of search space and state space being the same is not correct, since the solution is in the genotype, while the check to find fitness is in the phenotype.

State space:
This is a set containing the combinations of states the geometry, where an organism develops, can obtain. The formula for calculating the state space is the equation 2.2.

This also defines the theoretical limit for attractor a CA within a finite geometry.

$$k^{Gs} \tag{2.2}$$

Where k is the number of states pr. site, and Gs is the number of sites in the geometry.

Rule space:
This is a set containing the permutation of the states of the sites in a neighbourhood, and is calculated according to equation 2.3.

$$k^{n} \tag{2.3}$$

Where k is the number of states pr. site, and n is the number of sites in a neighbourhood.

Chromosome space:
This is a set containing the permutation of the rule space and a next state, and is calculated according to equation 2.4. Every single mutation within a chromosome in a CA will reside within this space.

$$k^{Rs} \tag{2.4}$$

Where k is the number of states pr. site, and Rs is the rule space.

Search space:
This is a set containing the combinations of a set of size x and the chromosome space, where x is the amount of chromosomes in a genotype. A set of all the genotypes that can be created given a size x. The formula for calculating the search space is the equation 2.5. This is essentially a set of all the different CA that can be constructed with a genotype of size x.

$$\frac{Cs!}{x!(Cs-x)!} \tag{2.5}$$

Where Cs is the chromosome space, and $x$is the number of chromosomes in a genotype.

The scalability of the EA and CA are bound to each other, and to the spaces above. There are mainly 3 aspects which i call scaling in resources, scaling in effectiveness, and scaling in geometry.

2.7.2.2 Scaling in resources

This is a frameworks ability to create different scale solutions with a reasonable size within a reasonable timeframe. Here, using EA to create CA which act as a solution within reasonable time. EA are traditionally resource greedy and suffer in scalability when increasing the search space[55][56]. One cause of the lacking scalability is that there is a traditional 1 to 1 mapping[47] which can give too large to be practical solutions with regards to the size of the genotype. To solve this it is possible to do like nature, and choose to implement development into the EA to decrease the genotype size. It is shown that development can increase the scalability of an EA[52]. By implementing a fitness function which is reliant on the outcome of the development/growth of the CA is a way of solving the 1 to 1 problem.

2.7.2.3 Scaling in effectiveness

Effectiveness with regards to an EA is the measure of the amount of chromosomes used in an evolved solution compared to the minimum amount of chromosomes needed for the solution to work. Also the amount of chromosomes used in an evolved solution compared to the amount of chromosomes available for each individual in a population during the EA progress. With regards to a CA the effectiveness is the measure of used rules compared to the rule space created by the neighbourhood and the amount of states. By having a EA be more thorough in its exploration of the search space, it is able to use less chromosomes to have the same outcome as a more relaxed EA could. This is good for the overall scalability since reducing the genotype size greatly decreases the search space, and should in theory require less from the EA pr. generation.

2.7.2.4 Scaling in geometry

This scaling is the measure of how well an evolved solution will handle development in smaller or larger geometry than it was evolved in. Increasing the geometry of a EA developing CA makes the state space grow exponentially. On a finite automata the state space is defined as k^s where k is number of states and s is the amount of sites in total. This means increasing a 2 state 2 dimensional CA from a 4x4 to a 5x5 geometry, increases the state space from 2^{16} to 2^{25}. Because of this exponential growth, it would

be preferable to evolve a CA on smaller geometries, while having it perform the same or similar functions when scaling the geometry up and down. This scalability in geometry has been observed in Tufte's article[5], and is a behaviour which would be interesting to further understand and be able to recreate during the evolution of most organisms.

2.7.3 Why scaling

In todays society, many of the problems we use computers to solve are large. Very large problems with a huge amount of variable parameters can be solved with an EA, but even so it requires much computational power and often produces large solutions. Having a good understanding of scalability can help both the usage of computational power, the size of the solutions, and might even be able to improve the size of the problem. To increase the EA scalability to large problems, development has been introduced, which creates small solutions to problems which normally require large solutions. This again makes the solutions manageable, making scaling a valuable tool. The evolution development (EvoDevo) systems are easily applied in a world of cellular development, for instance CA, which makes the CA already a system which scales well. Creating a better understanding of what creates scalable solutions, and how to create scalable systems within the confines of EvoDevo systems will be beneficial in the creation of large and complex cellular systems. Some ways which scalability can be increased is by reducing the state or search space for a wanted solution, which essentially reduces the size of the fitness landscape, making systems scale to solve larger problems. Knowing what makes algorithms create solutions which are scalable in geometry can reduce the problem, making the resource consumption manageable. Having a good understanding of scalability gives a better utilisation of resources, and makes it possible to create solutions to problems which one normally do not have enough resources to solve, or makes runtimes of currently solvable problems faster. More bang for the buck as it is called. There are of course multiple other ways of reducing the real runtime of a EA, for instance massive distributed or parallelised algorithms. But these will also benefit from a smaller search space and state space which can be achieved by scaling in efficiency. By understanding how solutions scale in geometry, it can become possible to evolve small solutions which scale up to larger geometries with linearly increase, or without the loss of functionality.

Chapter 3

Hypothesis

3.1 Main idea behind hypothesis.

The foundation of this thesis is based on the idea that when evolving cellular automata as a EvoDevo-system[57] the size of the genotype, both available and used, has an impact on the resulting evolved system and its emergent properties. Exactly what these properties might be is hard to predict, but we are hoping to see some results towards improvement in the scalability in geometry, or restrictions on the complexity based on efficiency of the genome. There are experiments where only a handful of the available rules are activated through the development process[55], but still achieving good results. Because a typical EA works with a full combination set of chromosomes, the solutions usually have poor effectiveness in the usage of chromosomes. By forcing a EA to make better usage of the chromosomes at hand, the evolved solution would have a more compact and effective genotype. This reduces the search space for an EA, which is helpful when considering the evolving of large organisms. In large organisms, there is usually a wish for a large amount of specific behaviour. For the organism to be able to perform all the requested behaviours, it often needs high amount of states pr. cell, and large neighbourhoods. When a EA evolves this organism, it needs to work on a set of chromosomes which is both large enough to perform the task, and do exaptations[58] along the way. This size is normally set to the size of the rule space, but with a high amount of neighbours and state count this number is very large. When the genotype size grows, the search space grows exponentially. It is difficult to predict exactly how large the genotype size needs to

be for a specific organism, so having it grow until it is large enough would be beneficial with regards to the search space.

As an example of the reduction in search space, a CA with 2 states pr. site, and a neighbourhood of 3:

Rule space = $Rs = 2^3 = 8$

Chromosome space = $Cs = 2^{Rs} = 2^8 = 256$

Normal search space = $\frac{Cs!}{Rs!(Cs-Rs)!} = 409663695276000$

While a slight reduction the genotype size available, for instance from 8 to 6, the Search space would be:

Reduced search space = $\frac{Cs!}{6!(Cs-6)!} = 368532802176$

By reducing the genotype size by 25% we reduced the search space with about 99%. This is a very small and simple CA, so the numbers are not astronomical. But when the neighbourhood or states pr. site increases a little, the payoff for reducing the genotype size, or let it grow until reaches a reasonable size, could be great.

With EA who are more conservative over the genotype size, there might be other benefits, and there might be drawbacks regarding both the evolution process and the emergent behaviour of the evolved organism. By further studying the usage of rules throughout development, and comparing against organisms evolved without restraints, one would be able to look for patterns or differences which can be useful for the design of future solutions and EA.

3.2 What we would like to look at in this report

For this thesis we will look closer on the correlations between the results and evolutionary methods. We will try to evolve structures using a large, small, or growing genotype. By comparing the methods against each other, we can get an understanding of how restrains affect the evolved organism. For results to have more weight, we will have to look at the genome usage for evolved solutions on different complexities. If finding similar structures or patterns while changing complexity but keeping restriction on genotype, it will mean

that restrictions are most likely the cause of the findings. Further comparison and examining of eventual results will be done to look for emergent properties or changes in emergent properties within evolved organisms. Some of the properties which will searched for is increased scalability with reduced genotype size. We will try to reach some answers to questions like: Will we find that growing a genotype produces compact and effective genotypes, or will the results be similar to those of an organism evolved with a large genotype? How will this type of evolution scale for increased state space? Could a EvoDevo-system which evolves organisms with a growing genotype exhibit emergent or implicit behaviour not found in organisms evolved with a static large genotype? Will this kind of evolution actually be able to produce results at all ?

Chapter 4

Framework

An explanation on the framework used during experimentation, so that results can be regarded in respect to the rules and functionality of the framework. This explanation will make results possible to recreate and further exploration on a similar setup possible.

4.1 Random function

For a EA to work, there has to be a random function so as to be able to create a chance of events happening. In C, which this framework is written in, it is normal to use the *rand*() function to obtain a random number. The default procedure to obtain a random number in a range [0, n] where n is RAND_MAX, is by the use of integer division/modulo:

Random number = $rand()\%n$

This procedure will return a random number within the correct range, but if the RAND MAX does not evenly divide to the range, the distribution will be skewed and with it the correctness of the random function. Since most of the random numbers for the experiment framework operate in a range which does not evenly divide with RAND_MAX, an implementation of the Mersenne twister algorithm[59] has been added. This ensure a better random generator with a larger cycle, and does not suffer from another of *rand*()'s problems, which is short cyclical pattern in the low bits, or bit dependencies. Using Mersenne twister and making all used ranges, range from 0 - 1 we are content

with the uniform and randomness of our random function. The random seed is an array of 4 unsigned long values. For the seed values to be distributed, they are read from the */dev/random* file. This will ensure a certain amount of randomness to them, which is enough for a seed to function properly.

4.2 Cellular automata framework

4.2.1 Neighbourhood and geometry

The framework for the CA supports both 1 dimensional and 2 dimensional CA. For 2 dimensional there is only support for a square geometry. The limitations of the size is bound to an integer, but using a 2 dimensional geometry larger than 4 greatly decreases the speed of the framework. For the boundary conditions, we decided upon periodic boundaries. Meaning a loop from north to south, and east to west on a 2 dimensional geometry, which can be observed in Figure 4.1. There are no boundaries from one corner to another corner. This is because the Moore neighbourhood is not a part of the framework. For a 1 dimensional CA the neighbourhood is a default one, as defined by Stephen Wolfram, but with a $r = 2$ [20], while the 2 dimensional CA uses a von Neumann neighbourhood[17].

4.2.2 Representation

The genotype for the CA is an array of 32-bit fields, where n-bits of each field are reserved for a neighbour site based on the amount of states pr. site. Example 2 states will require 1 bit, 3 states requires 2 bits, and 4 states also require 2 bits. This means that not all of the bits in the field are used, and that sometime not all the permutations within the n-bits are used either. The n-bits goes from least significant toward most significant bit in the 32-bit field in the sequence: Next state, East state, Centre state, West state, South state, North state. North and South are not part of the representation for a 1 dimensional CA. The phenotype for the CA is the organism which is developed according to the genotype. This can be an organism after x number of development steps, or until a specific shape or behaviour has emerged, or as in our case when we achieve an attractor. Since the framework functions within a finite geometry, an attractor will always appear.

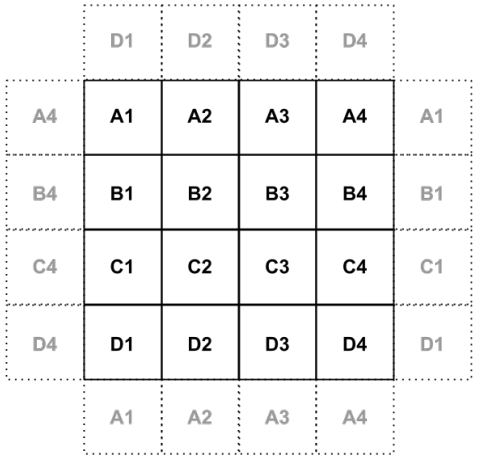


FIGURE 4.1: Image displaying how periodic boundaries function in a 2-dimensional space.

There is also a mechanism which stops the development after x-steps, since there might be attractors which are very large (potentially up to the size of the state space), and that would slow the framework to a halt.

4.2.3 Development and attractor locating

In the framework there is a partially realtime search for attractors, where after x development steps the framework stops developing and checks if any attractors have been developed. If no attractor is found, the development steps currently in memory are stored to file, before continuing x development steps and initiating another check for attractors. Each attractor check is performed against development steps on file, and in memory. Because of the partially real time search, the runtime of the development process becomes slower for more development steps, or longer attractors. The number of checks needed to locate an attractor within a set of n elements increases in polynomial

time, and is given by equation 4.1.

$$\frac{n(n+1)}{2} \tag{4.1}$$

A single development step requires for each site in the geometry that we find neighbourhood sites, then search the entire genotype for all the rules which are triggered. There is a need to search the entire genotype because there is no restriction on duplicate chromosomes. Duplicates can appear through adding, mutation, or crossover since there is no mechanism to explicitly prohibit it. As a regulation mechanism for similar chromosome, or chromosomes which have the same state on its input neighbourhood, modulo is used. The mechanism function so that for each chromosome which can be applied, the sites next value is summed in a variable. When all the chromosomes in the genotype has been tried, the summed next value is moduloed against the maximum number of states pr. site, thus ensuring a deterministic outcome.

4.2.4 Testing

To make sure the frameworks CA behaviour is satisfactory to the requirements of a CA, tests were constructed. Testing was done for both 1 and 2 dimensional geometry. For 1 dimensional testing Stephen Wolframs rules for different CA settings[13] were used, some examples are rule 30, rule 110 etc. The framework was run for enough development steps for the CA to make use of the border conditions and some interaction after the use of border condition. When sufficient development steps had been performed, the development process would be manually checked against the behaviour given by mathworld[60]. This testing confirmed that the basic behaviour of the CA worked. Further testing was done on the 2 dimensional CA, here specific rule sets were devised which would test border conditions in both dimensions, and also that rules were properly applied. These rules would create blinking patterns, or snakes that moved in multiple directions but across boundaries. The confirming of the testing was done manually for each development step with the help of a visualization module on the framework. With the testing of the CA done, we were sure that the framework would be able to develop according default CA behaviour, with discrete time steps and synchronous application of rules on the geometry.

FIGURE 4.2: Displaying how mutations can cause states which are void.

4.3 Genetic algorithm framework

4.3.1 Mutation

4.3.1.1 Void mutation

Mutation is handled with a possibility of each bit in a chromosome (one of the active/used bits in the 32-bit field) has a chance of mutating. This means that any combination of mutations are possible. All the mutation is done through the flipping of bits, which makes problems if the number of states pr. site is not in within the set defined by 2^n. If the states pr. site is outside this set, a random flip within active bits can make the state mutate to a state which is void. Figure 4.2 shows how this can occur. The solution to this problem is done by using modulo by the max number of states pr. site.

4.3.1.2 Mutation rate and type

The mutation rate for the framework had many configuration steps to gain a mutation rate which was thought suitable to use. Early a mutation rate of 0.05 (5%) was used, but when the genotype grew in size, the suitability of the mutation rate changes with the growth. This lead to a mutation rate pr. individual vs static mutation rate. Mutation rate pr. individual is calculated like this: $Mg =$ mutations pr. generation.

$Ps =$ population size.

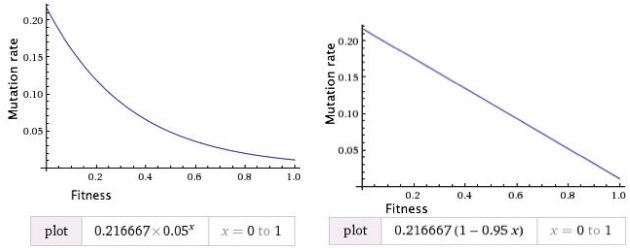
$mr =$ mutation rate.

$bg =$ number of bits used to represent an individuals genotype.

$Mg = ps * bg * mr.$

$Mi = Mg/ps$

Using mutation rate pr. individual gives a better results for the growing genome, but it also greatly reduced the chance of mutation as the genome grows. A way of patching this problem was to greatly increase the mutation rate, but this lead to a poorly adjusted rate. By implementing an adaptive mutation rate as Kitano did[61], we tried to create a rate which would be large enough that it would work on larger genotypes, and would still hold for smaller genotypes. Two versions of the adaptive mutation rate pr. individual were implemented, one exponential, and one linear, which can be seen in Figure 4.3. Both types would regulate the mutation rate from set mutation rate to 5% of set mutation rate. Further experimentation made it clear that the adaptive linear mutation rate pr. individual was suitable for use in the framework.



(a) Exponential adaptive mutation

(b) Linear adaptive mutation

FIGURE 4.3: Graphs showing how the mutation rate shrinks when fitness increases for two different adaptive mutation strategies. These graphs were created using data for a mutation rate of 2.6 pr. individual, with a genotype size of 1. This gives a starting mutation chance of almost 22%, which shrinks as the fitness increases from 0 to 1.

4.3.2 Fitness

The fitness is a real number between 0 and 1, and is found by calling a fitness function. There were two main ideas for the fitness functions, linear and exponential. For the exponential function the formula found in equation 4.2 was used.

$$\frac{1}{e^{\frac{di}{va}}} \tag{4.2}$$

Here di is the difference in attractor against the requested attractor, and va is a hard coded value to which changed based on the length of the requested attractor. For the linear fitness function there would also be a need for a number to divide by, like val for exponential. But this would give unwanted properties as soon as a longer attractor than the requested attractor appeared. To use such a system would require a rework of the fitness to be any real number equal and over 0, and the requested attractor would have fitness of 1.0. To remedy this the linear fitness is regulated for each generation by using the individual which an attractor with the biggest difference to the requested attractor as a denominator, shown in equation 4.3.

$$\frac{max\_di - di}{max\_di} \tag{4.3}$$

Here max_di is the biggest difference to the requested attractor in a generation and di is an individual's difference in attractor to requested attractor. This formula works but has a flaw, which is when a big percentage of the population have the same attractor length and this attractor length is the biggest difference, this part will have a fitness of 0. In the framework a fitness of 0 is much worse than a very low fitness, for instance 0.0001. The simple solution to this problem was adding a 1 to the max_di value, ensuring that no attractor could get a 0 fitness score. Further testing and comparing showed that the linear function performed better than the exponential one.

4.3.3 Crossover, and crossover with different sizes.

Crossover is another functionality of the framework, it supports crossing of equal and different size genotypes. After a weighted selection of two individuals which will have their genetic material transferred to the next generation, there is a probability of crossover. When applying crossover, we first establish which of the individuals has the shortest genotype. Then we get a random value in the range of the size of the shortest genotype, which tells us how large the part to be crossed over will be. Now the size and position of the crossover section is defined as from smallest genotype size - random value, to smallest

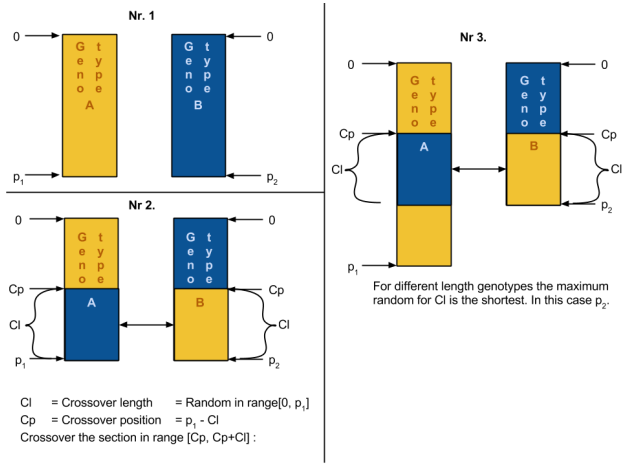


FIGURE 4.4: This explains how crossover happens in the framework. A selection Cl whole chromosomes from chromosome at position Cp. When different lengths genotypes are selected, the shortest genotype is used to produce Cl and Cp making the crossover happen without a change in the length of any of the selected genotypes, as displayed in Nr.3.

genotype size. Figure 4.4 shows how this crossover works in this framework, and how it does not interfere with the sizes of the different genotypes, but swaps chromosomes.

4.3.4 Adding

The idea about growing the genotype, or trying to diminish the amount of unused genotype space is not new. Inspiration for a growing genotype comes from ideas and articles like complexification[4] and selective genome growth[62]. Adding of chromosomes to genotypes happen right after potential elitism, but before mutation, crossover, and filling of the next generation. There are some limitations on the adding functionality, which ensures that the genome growth does not get out of control. Essentially there are 4 regulation mechanisms which control the growth of the genome. First, there is a maximum limit of how many chromosomes can be added, which is the size of the rule space. Second, there is only a percentage chance pr. individual that a growth in a genotype will happen. Third, there is a counter and a threshold which specifies that a

growth can only occur after x generations without an increase in an overall best fitness. Meaning that as soon as an increase in fitness occurs, the counter resets and starts to count toward x again, ensuring that growth does not happen every generation. Last, there is the mechanism of elitism. Testing showed that using elitism of 1, achieved better results towards end goals, but also helped controlling the growth of the genotype. This added effect is because the elite will have a good fitness compared to the population, and adding chromosomes have a tendency to decrease fitness until it is optimised. Continued adding of chromosomes will further decrease fitness until it is optimised, meaning that when filling new generations, the elite will have a larger chance of selection. For a added chromosome to "stick" it needs to not decrease the fitness to such an extent that a selection for the next generation is improbable. The adding process has a number of possibilities to add a chromosome equal to the size of the population. For each check, there is a percentage change an add will occur. A hit on the chance will result in a weighted selection from the population to get an individual who will have a growth in its genotype. The growth takes a random chromosome from the current genotype, and adds it at the end. A copy is used since this is partially based of occurrences which happen in nature[4], and the appearance of a completely random gene is unlikely. There is a single restriction on the adding process, which limits the amount of added chromosomes for an individual to 1 pr. generation. As mentioned earlier there is a gene regulation mechanism when chromosomes with the same preconditions (variation of states for neighbours which results in a rule without an answer/next state) exist in the same genotype. This mechanism is the modulo, where the next state is added together, and the moduloed on the number of states pr. site. The gene regulation mechanism is in place to solve problems that occur with functionality like the copying of a chromosome.

4.3.5 Weighting

In the process of selecting individuals for the next generation or for adding of chromosomes, a weighted selection happens. The weighting is a value constructed from three parts. First part is the fitness of the individual, which is calculated by the fitness function. Second part is a weight based on the number of activated rules, which is calculated:

$$\frac{\text{Number of used chromosomes in the development process}}{\text{Size of genotype}}$$

This helps create a more efficient genotype by awarding an individual who uses a large percentage of available rules. A side effect of this weighting is an additional regulation to the adding of chromosomes. By calculating the weight based on the size of a genotype, a individual who has added a chromosome which becomes inactive, will have a lower chance to be picked, than if the added gene was not there. Third part of the weighting is a weight based simply on the size of the genotype. This was implemented to help the genome grow, because with the inherent regulation in the weighting on active rules, and elitism, and the fact that adding a chromosome to a fit individual will likely decrease its fitness[4], made it difficult to have the added chromosomes not disappear through natural selection. The three different weights were normalised the same way, by summarising the specific weight for all the individuals in the generation, then each individual's weight was divided by the sum. These weights can again be adjusted so they weight a percentage compared to each other, for instance 60% for fitness, 20% for active rules and 20% for genome weight.

4.4 Genotypes

There are three different ways of running the GA; Full, Restricted, or Growing. For a Full run on the GA the initial population consists of CA with a full genotype (genotype size is the size of rule space), and a chromosome set to each of the states in the rule space. For all the chromosomes in a full CA the next value is set to 0, or the quiescent state. A Restricted run is equal to the Full run, except when it comes to the size of the genotype, and initial state of the chromosomes. In a Restricted run the size of the genotype is defined as a value $\in \{1, 2, .. ,$ rules pace size$\}$. All chromosomes are initially blank in the Restricted run, to make it a fair comparison to the growing genotype. By blank, it is meant that all values are left in the quiescent state. The Growing run starts with a single blank chromosome in its genotype. Every x generation without growth in fitness, some individuals have a probability of adding a clone of one of its current chromosomes. Other than this it works under the same rule and configurations as the other runs.

4.5 Growing genotype algorithm

The main difference between a normal GA and this growing algorithm is, that during the evolution process the amount of available rules for each individual can grow. The size of the available rules, i.e. the genotype, starts at 1 which is for most problems too small, but during evolution grows to a size where a solution is able to be produced. When the size of the genotype grows, the hope is that the solution found uses a much smaller genotype than what a manually constructed genotype will use. Another difference is the fitness function, which is based of the development of an organism, making this system an EvoDevo-system. In Figure 4.5 one can see the general flow of how the algorithm works. The initial configuration and initialisation of a population and such is not included, since it is implicit. Starting at the development of all individuals, which in theory is a loop, which develops each individual, calculates the fitness based on the phenotype, finds the best fitness, and see if there has been a fitness growth compared to the best recorded fitness so far. Most of these parts of the development is explained earlier, but the checking to see if there has been a growth in fitness also iterates or resets the add counter, which determines the theoretically minimum optimisation time for each added rule to the growing genotype. Further in Figure 4.5 we come to the "Should add chromosome" section, which is judges based on the add counter if the individuals should have a chance of adding a chromosome. If should add flows to yes, the "add chromosome" section automatically resets the add counter, and there is a fixed percent chance of adding a chromosome for each individual, as explained earlier. The sections "Perform Mutations" and "Perform Crossover" is also only a chance of doing this, most individuals or genes get through the sections without any noticeable effect.

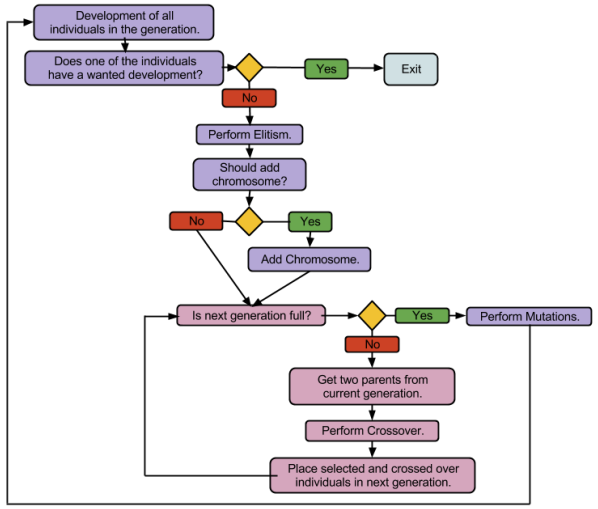


FIGURE 4.5: Flow chart showing how the main sequence of processes happen in the constructed algorithm.

Chapter 5

Experiment

5.1 Areas to explore

The idea of copying or imitating nature to evolve solutions to problems has brought forth the EvoDevo process. By using development EvoDevo systems achieve better scalability in problem size. An aspect that is not used much in EvoDevo systems is the growth of genotype size, which occurs in natural evolution[63][64][65]. There are devised algorithms which take inspiration from the natural growth of genotype[62][4], which have gotten encouraging results. The implemented algorithm is inspired by these, but unlike Lee Altenberg's implementation, the growth of genotype size is kept in the generations by natural selection, instead of brute forcing an increase in fitness. Altenberg's algorithm the process of adding a chromosome is based of a repetitive structure which tries to add a chromosome until it successfully increases the fitness of the individual, before the the individuals are further evolved. In our algorithm we take advantage of evolutions natural selection, which only allows added chromosomes which have positive or approximately neutral impact on the fitness. We want to test if our implementation of a EvoDevo system with growing genotype is able to evolve organism of different complexity, and how these results compare to results for other representations, i.e. size of genotype. These sizes will be what we call a Restricted and a Full size genotype, were the Full has a genotype size equal to the rule space, ensuring that every possible neighbourhood configuration for a site has a related chromosome to begin with. In the Restricted, the genotype size is less than the rule space, but higher than 1, which means it is manually configured to a size which most likely contains a solution, but will not have a chromosome

for each possible neighbourhood configuration. The complexity will be measured as the attractor length, which can be interpreted as a kolmogorov complexity[66] based on the developed phenotype. The algorithm will further be tested by changing the number of states pr. site, and the size of the geometry. By using multiple geometries and number of states we will be able to see how the algorithm reacts, and be able to draw conclusions. The results of the different experiments will be further examined to look for efficiency in the usage of rule space, and for other emergent properties of the evolutionary method, like scalability in geometry, or scaling in resources.

5.2 Configuration

The difference of formulating an algorithm and actually implementing a working example is large. For this algorithm there are many variable parameters whose changes affect the effectiveness of the GA from, working to unusable. To be able to perform our experiments with expectations of a system able to achieve results, these parameters needs to be configured. A large problem with the configuration of GA is that different configurations for the combinations of the parameters needs to be compared. To be able to compare them, runs of the GA needs to be performed, which takes time. And for the runs to be somewhat measurable, there needs to be more than a single run for each setting, so an average can be used. In our framework, the amount of parameters grew as problems arose, and is currently at 8 variable parameters and 3 binary parameters. If defining 3 values for each of the 8 variable parameters, there is a total of 52488 runs to obtain the best configuration of the system. This number is accurate if we do not create a finer granularity for the variable parameter than 3, and only run each setting once, not thinking of average values. In the framework a single run can take from 30 seconds to multiple minutes, which would make the configuration run from 18 days, to couple of months. This is too long for the time frame of this thesis, so to solve this issue different values were fixed without configuration. Further cutting the configuration time was done by incrementally finding "optimal" values for parameters instead of checking the permutations of combinations available. This was done by first finding the best of 3 values for one of the parameters, while the other parameters were hard coded to suitable levels. The hard coded variables were swapped with the configured variables as soon

as they were discovered through earlier runs, which essentially changes the amount of configuration needed from permutations, to just number of parameters.

Early configuration yielded other results than later configurations, which is because the framework got reworked quite often to achieve wanted behaviour. This rework is what constructed many of the variable parameters, which in the earliest configurations were only 5; mutation, elitism, crossover, add threshold and add chromosome. The other parameters are; weighting of active rules, weighting of genotype size, linear/exponential fitness function, adaptive/non-adaptive mutation, linear/exponential adaptation rate, and adaptation rate. For all these parameters to function, multiple configuration runs had to be run, and rerun, and reworked to only be rerun again. Frameworks that are not identical, will need to be configured independently, which means that if someone will try to imitate or copy this framework, they will essentially need to configure it themselves. Elitism is the number of individuals which go unchanged to the next generation, untouched by mutations adding of chromosome and crossover. When filling the next generation with individuals, there is a selection of two individuals from the current generation, and they are transferred to the next generation. The crossover is during the transfer from current to next generation, parts of the chromosomes are swapped between the two selected individuals. This only happens some times, and how often this happens is determined by the crossover rate, which is a percentage chance. The selection two individuals to transfer to the next generation is not completely random, it is based on how fit each individual is. A more fit individual has a higher chance of being selected. The weighting of active rules is implemented in this part of the algorithm. A high usage of the current genotype is seen as a positive attribute for an individual, and because of this a weighting is applied based on how good utilisation of the genotype an individual has. This weighting influences the selection chance, along with the fitness. Calculation of fitness can be done in many ways, here we separate between linear and exponential fitness, where a linear will give an increase in performance the same amount of increase in fitness at all times. An exponential function will give the same increase in performance different increase in fitness depending on how far away from the wanted result the performance is. After elitism, but before filling the next generation, there is a chance of adding chromosomes, as can be seen in Figure 4.5. Both the threshold and the add chromosome rate affects this part. The threshold is the amount of generations without an increase in best recorded fitness which has to pass before a chromosome can

be added. Add chromosome rate is the percent chance of adding a chromosome to an individual, with limitations of 1 added chromosome pr. individual pr. generation. When the next generation is filled with individuals, there occurs mutations. The amount of mutations is based on the mutation rate, which in this case is adaptive. An adaptive mutation rate changes based on how fit an individual is, since when an individual is far from perfect, there is a higher need for change than when the individual is close to perfect. This adaptive rate can, similarly to the fitness, be linear or exponential (converging). Which means the chance for a mutation drops either with a steady pace, or fast to begin with, and slower the closer it gets to perfection.

5.3 Main experiment

5.3.1 Settings

The variable parameter settings which is used during the running of the main experiments is found during the configuration of the framework. Most of the parameters are overlapping settings for the three different evolutionary methods, full, restricted, and growing. Exactly which parameters are used is found at the end of configuration results. Some settings are not present in the configuration description, but still needs to be described. When running the GA for a specific attractor, a single run can have very different results which often depends on random mutations. For the result to be comparable, there needs to be an average value based off multiple runs, so that random cases which can get lucky, or unlucky only have a slight impact on the measurements. The number of runs pr. attractor is decided to be 20, which with a big enough population gives a margin of error which is seen as acceptable. Small population sizes are faster, but are more likely to get stuck at local optima, so a larger one is preferable [67]. Our population size is set at 100 individuals, giving an adequate genetic diversity and fitness landscape exploration within a generation. For the smallest site space we use, with a geometry of 4x4 and a states pr. site of 3, this still gives a large amount of different states for the geometry(3^{16}). This gives a theoretical maximum trajectory of 3^{16}, which is very unlikely. But there is the potential of trajectories longer than 1000 or 10 000 emerging from random mutations, which can contain reasonable attractors. Because of the cumulative resource usage of real time attractor checking, the emergence

of long trajectories causes a large uncertainty in the runtime of the experiments. To avoid excessive time usage on the development of randomly large trajectories, there was added a limit to the maximum amount of development steps. Since there are some limitations on the attractor length the framework is able to grow in a time frame suitable for the thesis, because of the reduction in growth speed in the genotype, the maximum attractor length to be evolved was set to 400, while maximum development steps was set to 2000. Having a maximum development step gave the framework an approximate maximum runtime which was reasonable, and the possible minimum transient of 1600, which was thought of as long enough since the wanted results are attractors and not transients. The settings for the main experiments are the variable parameter settings found in the configuration, located in configuration results, and;

- Population of 100.
- Maximum development step of 2000.
- Number of reruns pr. wanted attractor of 20.

5.3.2 Comparison of methods

Is imitating nature and growing and evolving a genetic representation and an organism a viable function for different complexities in EvoDevo-systems? This experiment done to compare how the growing evolution function compares to the full or restricted evolution function. Comparing the results will show if the growing evolution is better or equal to the full evolution function with regards to genotype exploitation, or emergent properties of successful solution. The usage of a full genotype of 243 rules, which is the rule space for a von Neumann neighbourhood with 3 states pr. site, is a trivial size compared to todays computational power. But with vastly complex problems, this full genotype size becomes so large it is unreasonable to define the entire rule space, and to find a fixed size for this problem becomes a major obstacle[4]. In these cases a function where the representation changes and grows until the size is large enough to contain a solution will be highly helpful. First there will be multiple different attractors which will be evolved using the full and the growing evolution function. When the results for these runs have completed, we are able to use the genotype usage of the growing function to determine plausible genotype sizes for the restricted mutation function. The restricted will be ran

for 3 different genotype sizes for each of the wanted attractors, which are larger, smaller, and the same as the average genotype usage for the successful runs on the growing function. As for the amount of larger and smaller than average, is determined on the size of the genotype form the growing, giving a larger genotype a greater difference in genotype size.

The attractors which will be evolved are: 7, 10, 20, 40, 80, 160, 200, and 400. It is clear that evolving an attractor of 40 will be done faster than evolving an attractor of 200, so for each attractor there was a decision on the maximum amount of generations the evolution would be allowed to use. Looking at Figure 6.11 we can get a rough view on the amount of generations needed to achieve the different attractor lengths with the default settings. Using these values as a reference the number of generations were decided and ca be viewed in Table 5.1.

TABLE 5.1: Attractor lengths and number of generations available to achieve said attractor

Attractors	7	10	20	40	80	160	200	400
Generations	4000	4000	4000	6000	6000	8000	8000	10000

5.3.3 Functionality where states increase

This experiment is to evolve multiple attractors with a growing evolution function with different amount of states pr. site. For systems that should evolve large solutions, there is a need for variable and intricate behaviour, which is unable to be achieved by a simple 5 neighbourhood 3 states CA. A simple way of making CA capable of very complex behaviour is to increase the amount of states pr. site, which in turn increases the rule space, the search space, and the state space exponentially. If the framework is able to achieve results with the same setting while changing the number of states pr. site, it would mean that the framework could support a growing evolution in both search and state space, which means it would in theory be able to evolve arbitrary complex solutions. That being said, this framework is only designed for a growth in genotype size, so to be able to have a good growth in state space as well it would probably mean the implementation of new regulation- and conditional regulation-mechanisms. These experiments are mainly to see if the growth evolution function is able to create solutions

with more states pr. site, and if any of the properties from the comparison experiment are found in these experiments as well. For these runs the attractors to evolve is: 7, 10, 20, and 40. The evolutionary function will only include growing evolution, because with more states pr. site the possibility of random very long trajectories increases, which would lead to very long runtimes. A growing evolution will hopefully avoid this by being an incremental process which builds a solution from a single chromosome. For each of the attractors there will be runs with 5, 6, and 7 states pr. site, and we already have the results for 3 states pr. site from the above experiment. The number of generations is the same as for the above experiment as well, with 4000 generations for 7, 10, and 20 in attractor length and 6000 generations for 40 in attractor length.

5.3.4 Functionality where geometry increase

This experiment is to evolve attractors with the growing evolution function with different size geometries, which will see if the geometry regulates the process, and how changing the geometry affects the solutions. Does behaviour of organism of a specific complexity change if it is evolved in a different size geometry ? We will study the usage of rules, and types of attractors to see if any obvious emergent behaviour changes. Further, the solutions will be redeveloped in geometries other than the geometry used for its evolution, to see if there is any inherent scalability when evolving in smaller or larger sized geometries. Here we will look for both scaling and equal complexity of the evolved solution. By increasing the geometry, the state space expands enormously and the probability of very long trajectories increase dramatically. Simple rule combinations which achieve short attractors for small geometries, can end up giving very long trajectories when an increase in geometry is done. This is why only very small attractors will be run in this test. The attractors to be evolved are 5, 7, and 10, where all of them will be able to use 8000 generations. These attractors will be evolved on 2 dimensional geometries of 4x4, 6x6, 8x8, and 16x16, and the redevelopment will be done in these geometries excluding the geometry where it was originally developed. Most of the settings for this experiment was equal to the default configuration, but there were some differences, three to be exact. First, the weighting for genome size was set to 0.05 instead of 0.2. Second, the weighting for active rules were set to 0.15 instead of 0.00. Both of these were to encourage a more efficient genotype usage. Thirdly, a new weighting was implemented which was a weighting on how much usage the development of the organism depended

on the border condition. The weighting on border condition is a negative weight, since we wanted to see if the framework would be able to create organisms that used the condition as little as possible, since using a border condition creates collisions. This weighting was able to deduct up to 25% of the compound weight for the selection of individual process.

Chapter 6

Result

6.1 Configuration results

Some of the earliest configuration was done to both the framework and the parameters was the adding of a weight for active rules, and the separation of an exponential and linear fitness function. This was followed by runs on multiple different settings, trying to find what would be a good mutation-rate, weighting, threshold, and if one should use exponential or linear fitness function. An early configuration run was done where multiple settings for weighing of active rules, mutation rate, and thresholds were done for both linear and exponential fitness function.

During the evolution, the best fitness is found and recorded, creating a variable with the all-time best fitness. The threshold which tells how often an individual can add a chromosome to its genotype is based off how many generations pass without an increase in the all-time best fitness. A value of 20 for the threshold gave results which would have a growth rate in the genotype that would not be too fast, but still slow enough to be able to do some optimisation on the added chromosomes before further chromosomes are added. A quick look at the graph in Figure 6.1 makes it clear that it made sense to stop using the exponential fitness function, which was the original one, and start using the linear function. All of the runs in for the graphs in Figure 6.1 were made without elitism and with a mutation-rate pr. individual instead of a global one. A mutation-rate pr. individual makes the chance for mutations in the chromosomes of individuals shrink as the genotype grows, and not using elitism ensures that all individuals can be

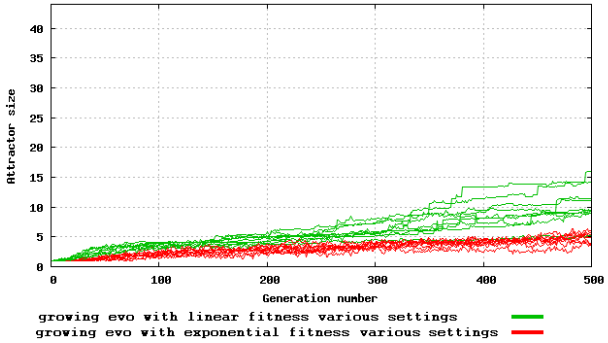
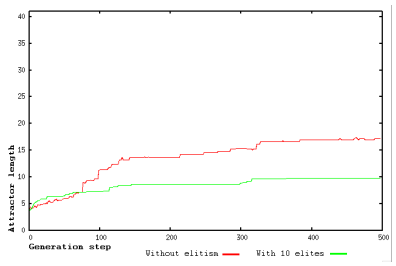


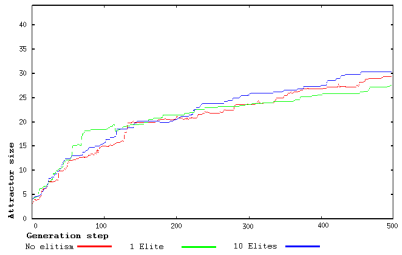
FIGURE 6.1: This graph shows the result of evolving an attractor of length 40 with multiple different setting. The colour separates the runs with an exponential fitness function(red), from the runs with a linear fitness(green) function.

grown, mutated, and crossed over. Not using elitism was supported by early testing where elitism had a very negative effect on the evolution process which can be observed in the graph in Figure 6.2 a, but this effect changed as the framework got reworked and elitism became a viable parameter seen in the graph in Figure 6.2 b. Elitism ended up being a major genotype size regulation mechanism in the latest framework builds. This can be observed in the graph in Figure 6.3. In the graph the blue line is the gradual evolution of the attractor, while the fat green line is the span of genotype size within the generation, where the lowest point in the green line is the minimum genotype size, while the highest point is the maximum genotype size in the generation. Looking at the figure, we see that it has a flat minimum line, while the maximum is growing on top of it. The growing of the genotype stagnates when the spread from the minimum genotype size grows too far. There are of course both negative, like increased possibility of getting stuck at local fitness optima, and positive effects of using elitism, like a faster runtime of a GA and not losing the best individual due to a randomly bad natural selection.

After reading about Kitano[61] and his use of an adaptive mutation function which increased his performance, this was implemented in two types, exponential and linear. Configuration runs were used to decide which would be used within the framework. Runs were done for evolving attractors of length 40 with the adaptation ranging from 5 - 100 % of the mutation rate, where they had similar results that can be observed in



(a) Elitism is bad



(b) Elitism is good

FIGURE 6.2: These are graphs from runs at different times during the construction and improvement of the framework. On graph a, which is the oldest, we see that disabling elitism(red) outperforms elitism enabled(green). While on graph b, which was later, elitism became better with no elitism (red), elitism of 1 individual (blue), and elitism of 10 individuals (green).

the graph in Figure 6.4. Both exponential adaptive and linear adaptive mutation rate performs better than non-adaptive mutation rate, and they both perform similarly good. Even if they have similar results, the linear rate gives on average a quicker growth in attractor than the exponential, and achieves a slightly better success rate as well. The choice is not as clear as when deciding whether or not to use the linear or exponential fitness function, but it is clear enough to choose the linear adaptive mutation rate.

This framework should run multiple types of evolutionary methods, i.e. growing, restricted, and full. For comparisons sake it was decided that all these should use the

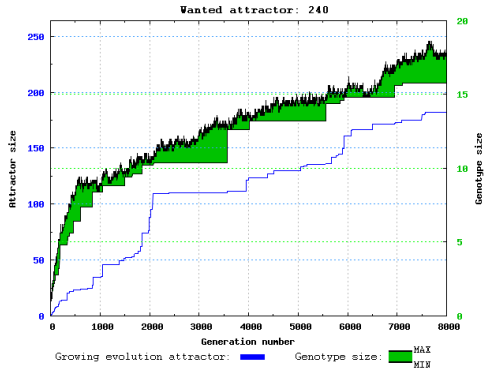


FIGURE 6.3: This graph shows results for the evolution of a growing genotype with elitism 1 in the later versions of the framework. The blue line is growth in attractor, and the green line is growth in genotype size. The change in width for the green line is the difference in minimum to maximum genotype size for the individuals in the generation. The periodically flat bottom structure for the genotype growth, keeps the genotype from growing too much, making the elitism help regulate the growth in genotype

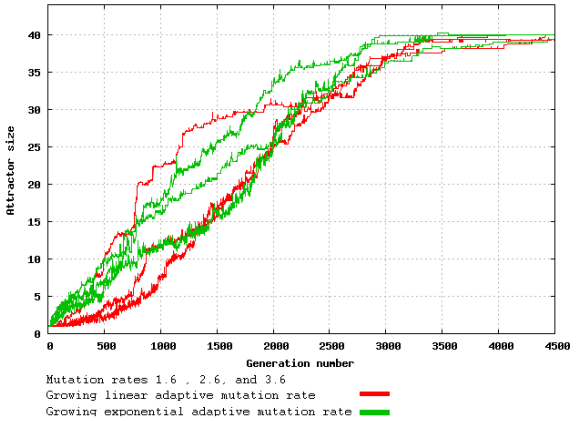
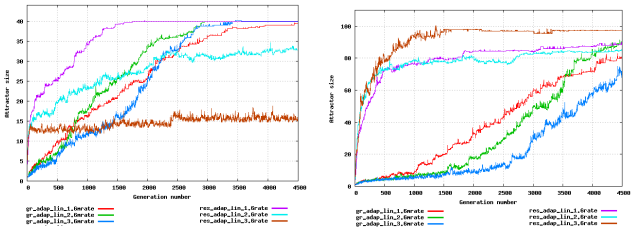


FIGURE 6.4: Each of the plots in this graph the average data of 20 evolutions. Each plot was evolved with a growing evolution to achieve an attractor of length 40, but varied in mutation rate, and in type of adaptive mutation function. Green plots represents adaptive linear mutation, while red represent adaptive exponential mutation.

same settings for overlapping parameters. When configuring the adaptive linear mutation rate the values 1.6, 2.6, and 3.6 was used on attractors of 40 and 100 which can be viewed in the graphs in Figure 6.5 a and b respectively. This mutation rate was pr. individual, which basically means that after the transfer of individuals from the current generation to the next generation, each individual will on average have 1.6 , 2.6 or 3.6 mutations. The number of mutations pr. individual is unaffected by the size of the genotype, making the chance for a mutation in a chromosome lower as the genotype grows. Studying the graph for attractor length 40 and simply taking the growing evolution in mind, a mutation rate of 1.6 is the fastest to achieve an average attractor close to the wanted attractor, but fails to when going from close to perfect. 2.6 and 3.6 as mutation rate is a little different, while they use a longer time to achieve an average attractor close to the wanted attractor, they both get 20 successful attractors out of 20 runs. But when taking the restricted evolution into account, both 2.6 and 3.6 fall in performance, while 1.6 performs close to the growing evolution. This should have made 1.6 the best fit for mutation rate configuration, but configuration runs with a wanted attractor of 100 gave results which made the overall performance for 1.6 decrease. In the results for attractor 100 the 3.6 rate for restricted evolution gives the best results, while 2.6 and 1.6 achieve similar results. For the growing evolution 3.6 gives the worst result, while 2.6 achieves the best, with 1.6 being in between. 2.6 was selected because it had the best result for the growing evolution with both attractor lengths, and decent result for the restricted evolution. There was also a higher weighting on the performance of the growing than the restricted, since we are trying to achieve a growing evolution.

An issue appeared where the amount of chromosomes in a growing genotype stagnated, which was believed to be caused by a low mutation rate as the genome grew, and the adaptive mutation rate further shrank the mutation rate. As a remedy to this problem a return to a static but adaptive mutation rate was tried, a mutation rate pr. chromosome instead of pr. individual. The basic idea was mutation pr. individual tells how many mutations there will on average be in an individual, no matter how big the genotype is. Making a large genotype which is represented by for instance 1200 bits, have a actual mutation chance of 0.21% pr. bit with the current 2.6 mutations pr. individual. A mutation pr. chromosome will have a 21% chance pr. bit since there are 12 bits pr. chromosome in this example. Some configuration runs were made to see if this configuration had any merit. These runs gave results which encouraged and discouraged



(a) Attractor length of 40

(b) Attractor length of 100

FIGURE 6.5: Graphs comparing results for attractor of length 40 and 100, evolved with a Growing and a Restricted evolution function. Each line shows the average value of 20 runs with the same setting, and each setting is a different adaptive linear mutation rate. Growing evolution with mutation rate 1.6 is Red, 2.6 is Green, and 3.6 is Blue. While for the Restricted evolution, mutation rate 1.6 is purple, 2.6 is cyan, and 3.6 is brown.

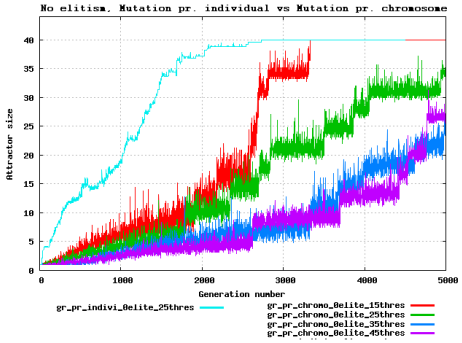


FIGURE 6.6: Graph showing the evolution of attractors with the Growing evolution method without elitism. The Cyan plot is a pr. individual mutation rate, while the others are pr. chromosome mutation rates.

the usage of this mutation model, and discovered a flaw in the system.

When viewing the graphs in Figure 6.6 and 6.7 it can be observed that the growth process is not very stable, but when enabling elites the growth calms down and gives good results on the wanted attractor, but not superior to the mutation pr. individual method. At this point the discouraging factor of mutation pr. chromosome is seen in

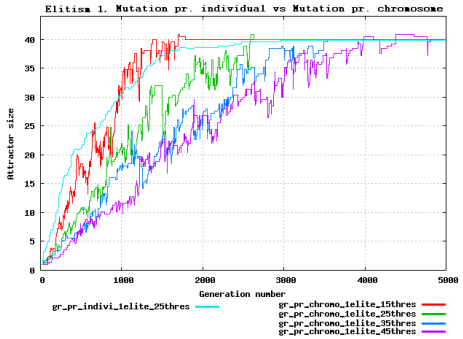
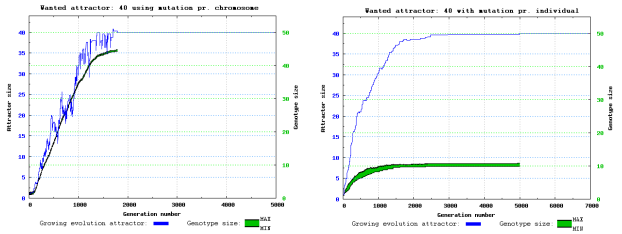


FIGURE 6.7: Graph showing the evolution of attractors with the Growing evolution method with elitism of 1 individual. The Cyan plot is a pr. individual mutation rate, while the others are pr. chromosome mutation rates. Having 1 elite greatly increased the success rate.

the graphs in Figure 6.8 a and b, where the growth of rules are very high, which fits poorly to the idea of effective genotype usage.

Another thing which was discovered with these runs was a flaw in the elitism design. When observing Figure 6.7, the graphs have multiple fall in attractor length, which converts to fall in fitness, which again should have been eliminated by the usage of elitism (this is an average of 20 runs, so it could appear but should not be frequent). This frequent drop in fitness, was caused by the function which increases genotype size, since the function was exempted from the elitism and could affect the elites, eventually ending up reducing the fitness of the elite. Changing this so the GA behaviour for elitism followed the main idea of elitism, that the best individual is transferred unchanged to the next generation, had a dramatic impact on the mutation pr. chromosome. Growth in genotype became slow, and it achieved zero to few satisfactory results, observable in the graph in Figure 6.9, which discouraged the idea of further usage of mutation pr. chromosome.

One of the problems of continued reworking of the framework, is reconfiguring parameters which have already configured, based on the newly added framework elements. The weighting of rule activation was redone when the above changes and calibrations had been done, running for attractors of 40 and 100 with the active rule weighting of 0.15, 0.25, 0.35, and 0.45. This weighting is taken into account when selecting individuals



(a) Mutation pr. chromosome (b) Mutation pr. individual

FIGURE 6.8: Graphs showing the different results for mutation pr. individual and mutation pr. chromosome with an elitism of 1 individual. The blue line is growth in attractor, and the green line is growth in genotype size. The change in width for the green line is the difference in minimum to maximum genotype size for the individuals in the generation. There is a very high growth in genotype size which discouraged the continued usage of pr. chromosome mutation.

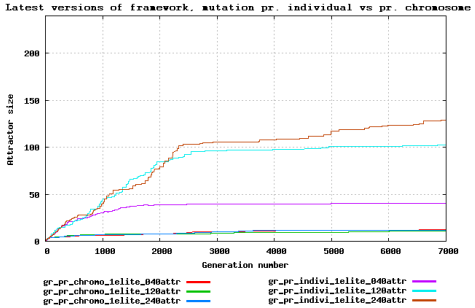
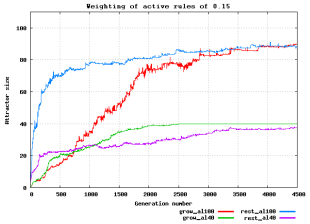


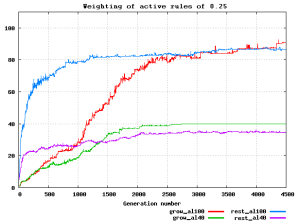
FIGURE 6.9: Graphs showing the evolution of different attractor lengths for the Growing evolution function, after a feature in the elitism functionality was changed. Compares the mutation pr. individual vs mutation pr. chromosome. This shows that mutation pr. chromosome achieves horrible results, while mutation pr. individual gets good results for each of the different attractors. The plots are based on the mean value of 20 runs.

for either adding of chromosomes or for parents for individuals for the next generation. Having a 0.15 in active rule weighting means that the value for active rule weighting is scaled down to 15% of its value, while the fitness value is scaled down to the remaining percentage out of 100%, i.e. 1 minus 0.15 which is 0.85. These runs lead to graphs which did not evenly reflect the results. On the graphs in Figure 6.10 a, b, c, and d it seems

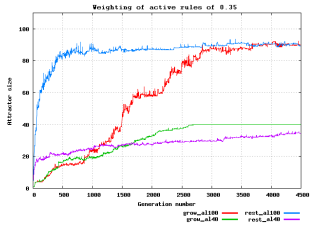
like a weight of 0.15 is the best choice when thinking of both growing and restricted mutation function, since it has similar average values which are quite high for both the attractors.



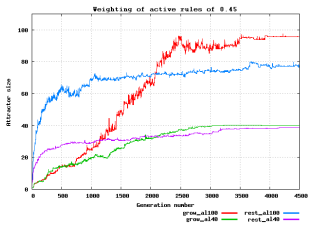
(a) Weight 0.15



(b) Weight 0.25



(c) Weight 0.35



(d) Weight 0.45

FIGURE 6.10: Multiple runs to configure the weighting of active rules, for both the Growing and the Restricted evolution function. The plots are based on the mean value of 20 runs. Restricted evolution of attractor 40(Purple), attractor 100(Blue). Growing evolution of attractor 40(Green), attractor 100(Red).

But when looking at Table 6.1 and 6.2, and taking the different attributes into account, the 0.15 weight is no longer the best. 0.25 is equal or better in all but 1 attribute, which is total rules for attractor 40, making a weighting of 0.25 for the active rules the selected value.

Originally this would have concluded the configuration of the framework, but some dry runs for experiments of attractors longer than 100, showed a significant decrease in success rate and growth in genotype. The main reason for this was thought of as the weighting of active rules, since this was implemented with the prospect of keeping an

TABLE 6.1: Attractor length 40.

Weight	Restricted Success	Growing Success	Used Rules	Total Rules
0.15	13/20	20/20	8.3	13.35
0.25	13/20	20/20	8.2	13.45
0.35	12/20	20/20	8.8	13.85
0.45	18/20	20/20	9.45	14.75

TABLE 6.2: Attractor length 100

Weight	Restricted Success	Growing Success	Used Rules	Total Rules
0.15	11/20	14/20	12.21	18.64
0.25	14/20	16/20	11.37	17.25
0.35	15/20	15/20	11.46	17.27
0.45	9/20	18/20	11.45	17.95

effective and small genotype size. Some simple configuration runs were made which used a high active rule weight, and one with no active rule weight, which resulted in a better attractor growth and surprisingly not an significant increase in genotype size because the elitism held the growth back. This is the discovery of the regulatory mechanism of the elitism, which lead to the removal of active rule weight, a decrease in the threshold for adding chromosomes, and the implementation of a weight which favoured a larger genotype. The new weight was simply added to encourage further growth in genotype to achieve larger attractors, and did not cause an overly large growth in genotype size. As seen in the graphs if Figure 6.11 a, the weighting did hold down the growth in attractor, and by encouraging the growth of genotype, the growth in attractor was further increased. Looking at graphs in Figure 6.11 b-d shows that this growth came at a reasonable cost in genotype size.

After configuration of the EvoDevo framework to an extent were it was possible to perform some experiments, we ended up with these settings:

- An adaptive linear mutation rate of 2.6 mutations pr. individual, where the adaptation ranges from 100 - 5 % of the original rate, seen in Figure **??** b
- A linear fitness function, as in equation 4.3.
- Elitism of 1 pr. generation.

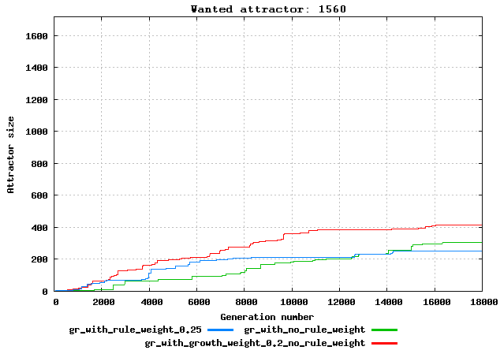


FIGURE 6.11: This represents Graph 6.11 a. b-d contains the growth of genotype for each of the represented lines, and is located in Appendix B. In this graph the lines represent different weighting strategies for the Growing evolution function. The plots are based on the mean value of 3 runs.

- Threshold of 10 for adding of chromosomes.
- Active rules weighting of 0.0 when selecting individuals to a next generation.
- Genome size weighting of 0.2 when selection individuals to a next generation.
- Add chromosome chance of 2%.
- Crossover chance of 2%.
- Quiescent state initiation for Restricted genotype and Full's next state.

Configuring a GA to work as a EvoDevo-system which uses a representation that changes during the process of evolution is a non-trivial task, compared with the configuration of a evolution full evolution method. During the configuration almost all of the settings worked well for a full evolution method, while being highly variable when used on the growing evolution. Having a growing evolutionary method configured properly will take time and a willingness to rethink the framework and adapt when difficulties arise, for instance a lack of growth in genotype, or a failure to achieve successful results. In this case the configuration is achieved by taking some shortcuts, and does not encompass all ways of achieving, regulating, or adjusting a growing genotype evolution function. The configuration and framework which is achieved and used for this thesis is by no means optimal, but adequate for the purpose of proving a point.

6.2 Main experiment results

6.2.1 Result for Comparison experiment

One of the main reasons for trying to grow a genotype representation, is because trying to decide upon a static size which is big enough to contain a solution is difficult when the problem becomes large. So the aspect of getting results which have a good utilisation of the genome compared to a full representation is important. Having a good utilisation shows that the growing function is able to produce solutions of arbitrary complexities but use a reduced search space compared to a full function, which can help by making the process of deciding on a representation size unnecessary. Looking at the graph in Figure 6.12 which shows the average usage and genome size for the successful runs of different attractors for the growing and the full function, it is clearly observable that growing the genotype gives both a smaller genotype size, and a higher utilisation of the genotype than using a full evolution function. For each attractor developed, 20 runs was done with a setting of 3 states pr. site, a von Neumann neighbourhood, and a geometry of 4x4. This tells us that the full rule space is $3^5 = 243$, making a Full evolution function have a genotype size of 243, while a growing starts at a genotype size of 1. The sizes for the Restricted evolution function is found based of the results of the Growing evolution function.

Another usage of Figure 6.12 was to find appropriate values for the restrictions for the restricted evolution experiments. As earlier mentioned there are 3 different restricted runs pr. attractor, one with a lower genotype size, one with and equal genotype size, and one with a bigger genotype size than the growing function achieved. The values that were decided upon for the restricted runs is located in Table 6.3, where the difference from the medium restriction to the low and high restriction is relative to the size.

Having run the restricted, full, and growing experiments it is possible to compare success rates, generation usage, and rule usage for the results. The number of successes were predicted to be higher for the full than for the growing and the restricted, since a full genotype does not need to simultaneously grow both the genome and the fitness, and there is lots of room of exaptation[58] from the start, giving the possibilities for large jumps or changes in fitness or structure for the development. Exaptations is thought to be an important part of evolution[58][4], where the phenotypic response to a mutation is

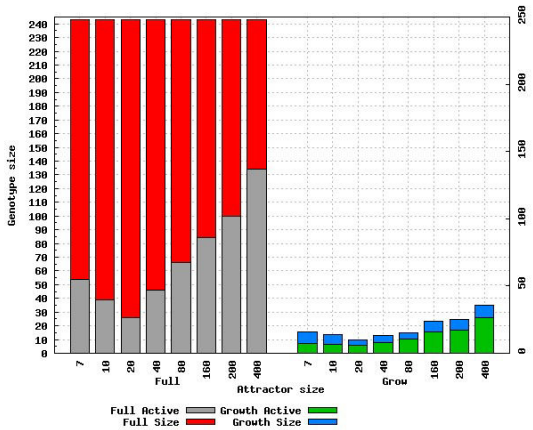


FIGURE 6.12: This graph compares the genotype size to the chromosomes activated during the development process, for the results created by the Full evolution function with the Growing evolution function. The plots are based on the mean value of the successful evolutions of 20 runs.

TABLE 6.3: Chosen genotype size for restricted evolution runs.

	Low Restriction	Med Restriction	High Restriction
Attractor 7	12	16	20
Attractor 10	11	13	15
Attractor 20	8	10	12
Attractor 40	10	13	16
Attractor 80	12	15	18
Attractor 160	20	24	28
Attractor 200	21	25	29
Attractor 400	30	35	40

not obvious, and does not decrease the fitness, but will after some generations make a big impact on the phenotype of the individual and further increase or decrease the fitness. It is the ability to let evolution tinker with a part of the development process which is not yet connected to the organism, so that when it finally gets connected, it makes larger changes than a single gene could do. When looking at the comparison graphs in Figure 6.13 a-h we see that the prediction was mostly accurate. In the graphs we clearly see that the full evolution function reaches the close to the appropriate attractor length in 500 generations or less, while most of the time the growing uses all available

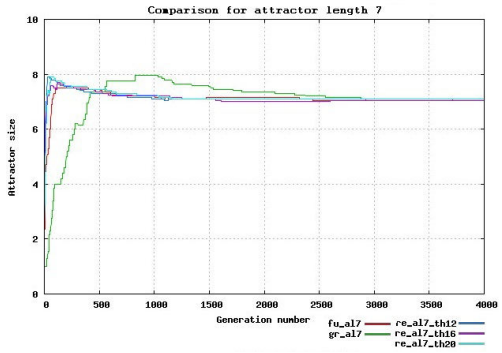


FIGURE 6.13: This is Graph 6.13 a, which shows a comparison of the different evolutions of attractor with length 7. Full evolution function(Brown), Growing evolution function(Green), Restricted evolution function low(Blue), mid(Purple), and high(Cyan). Graphs b - h which have the same colour coding shows the comparisons for the remaining attractors, and is located in Appendix A.

generations to get close to the wanted attractor. We also see that the growing function climbs slowly towards the wanted attractor over the space of many generations, while the full has most of its attractor growth within a handful of generations, which is because the growing evolution function needs to build up the genotype iteratively. The restricted evolution function, which initiates with a fixed size of its genotype we see a growth in attractor which is similar to the full evolution function, with a sharp growth in the early generations, and a convergence and optimisation after this. But the attractor growth for the restricted is shallower than for the full, which is because to the genome size being larger for the full. For most of the graphs the convergence of the restricted evolution ends up around the same area as the growing evolution ended, and looking at the graph in Figure 6.14 we see that the usage of the genome is roughly the same, especially when the restrictions on genome size were the same as the growth, i.e. restricted mid.

Further, looking at Table 6.4 we can see the success rates for the different experiments. Here we see that the full evolution function has an almost perfect success rate, except for attractor length 7. While the growing and restricted evolution functions have a success rate which varies from perfect to bad. For the attractors of 10, 20 and 40 there is a very good success rate for all the evolution function, and looking on graphs in Figure 6.12 and 6.14 we see the genotype size and usage of genotype is the lowest at

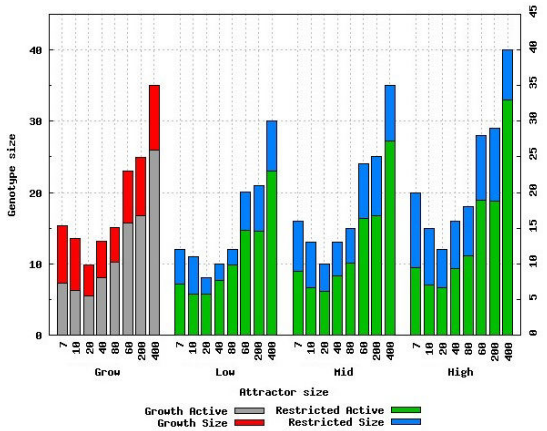


FIGURE 6.14: This graph compares the genotype size to the chromosomes activated during the development process, for the results created by the Restricted evolution function using high, mid, and low restrictions with the Growing evolution function. The plots are based on the mean value of the successful evolutions of 20 runs.

these attractors as well. So these attractors are most likely easy to produce within the current state and geometry setup for the CA. For higher attractors, we see a drop in the success rate for the growing and restricted evolution function, which is related to the increase in complexity. An increase in complexity will either need a larger search space or more generations to optimise a solution. In our case we supplied a fixed amount of generations, and a growing search space. The growth of rules seemed to slow down, or converge after growing to a size with a relative large search space for the wanted attractor. The convergence of the rule growth can be observed in graphs in Figure 6.15 a-h. Since these graphs are averages of 20 runs, it is natural that there is a slowing and stop in the growth as more and more runs reach a solution which stabilises the genotype size. But it is likely that getting closer to an attractor also has something to do with the decreasing growth speed, since the lower attractors grow to a smaller genotype size. One can also observe that even though they run for different amount of generation they have a similar growth speed before it slows down, which is easily seen by comparing the early genotype growth in the graphs with the same amount of generations. Observing Table 6.4, it is also worth noting the growing and restricted evolution functions have a higher success rate on an attractor of 7. This attractor was selected as an attractor

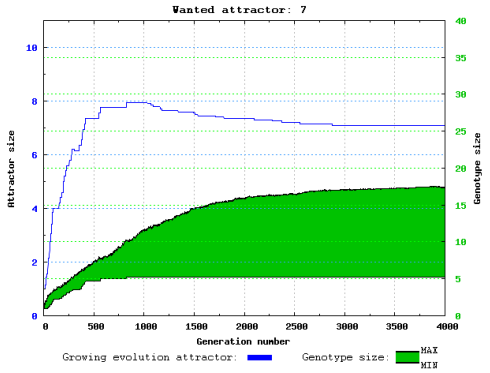


FIGURE 6.15: This is Graph 6.15 a, graphs 6.15 b - h is located in Appendix C. Graphs 6.15 a-h show the evolution of the attractor compared to the growth in genotype size for the different runs in the comparison experiment which used the Growing evolution function. The blue line is growth in attractor, and the green line is growth in genotype size. The change in width for the green line is the difference in minimum to maximum genotype size for the individuals in the generation. The plots are based on the mean value of 20 runs.

which would be difficult to develop within the CA settings, and not because it was large. Having a geometry of 4x4 with cyclic boundaries creates a good environment to create attractors which evenly divides by 4. The number of states also affect what attractors area suitable for the CA and not, but exactly which attractor lengths are suitable and which are unsuitable for our CA is difficult to find out. Using 7, which does not evenly divide with either the geometry, the amount of states, or the amount of states minus the quiescent state, we aimed for one which was difficult. The results for the attractor 7 indicate that our idea was correct, since it has a worse success rate than the similar attractor of 10, and also a higher genotype usage, which can be seen in the graphs in Figure 6.15.

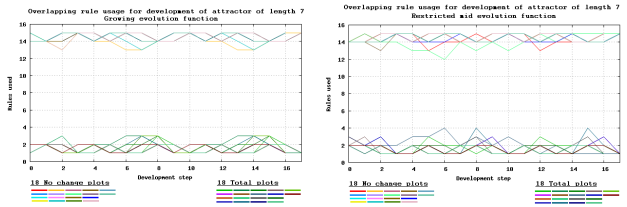
Another thing which is found when looking at the results is the usage of rules throughout the development of the attractor. When looking at the development of attractors there are some obvious points which separates the full from the growing which can all be observed by studying the detailed rule activation graphs in the Appendix E. First, there is an overall higher usage of transient for the attractors evolved with a full genotype than the attractors evolved with a growing or restricted genotype. Second there seems

TABLE 6.4: Success rate for comparison experiments

	Growing	Full	Restricted low	Restricted Mid	Restricted High
Attractor 7	18/20	15/20	18/20	19/20	18/20
Attractor 10	20/20	20/20	19/20	20/20	20/20
Attractor 20	20/20	20/20	20/20	20/20	20/20
Attractor 40	20/20	20/20	19/20	20/20	20/20
Attractor 80	15/20	20/20	10/20	14/20	14/20
Attractor 160	14/20	20/20	9/20	9/20	14/20
Attractor 200	16/20	20/20	13/20	10/20	12/20
Attractor 400	1/20	20/20	1/20	4/20	3/20

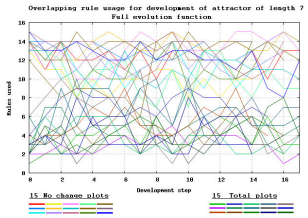
to be an overall lower usage of rules for the growing and restricted, compared to the full. One rule can be used multiple times pr. development step, so simply having room for more rules in the genotype should not be the only factor which makes growing and restricted have a higher average no change value. Third, the results for the growing and restricted seem to be made up of a small set of valid solutions, with small or no alterations, while the solutions for the full have vastly different structures. This can be observed by simply looking at the pattern of "No change" in the graphs in Figure 6.16, which is the lines at the top of the graphs. Last, for the attractors longer than 7, there are distinct patterns in the development which repeat multiple times within the same attractor. These patterns occur in quads within an attractor, which points to the probability that this is because of the cyclic boundaries in the geometry of size 4x4.

The growing evolution function was able to outperform the full evolution function in the aspect of genotype efficiency, which was one of the main reasons to perform the experiment. For attractors of simple complexity growing was able to keep a success rate within the set generations which could compete with the full function. This success rate for the growing got worse when the length of the attractor got high, which tells us it might need more generations, or maybe a better way of adding chromosomes to the genotype. It is also clear that for longer attractors the need for longer periods of optimisation or exaptations might be needed. Another point to mention is that the growing evolution function performed better than the full both on success rate and on genotype efficiency for the attractor of length 7.



(a) Attractor 7 Growing evolution

(b) Attractor 7 Restricted evolution



(c) Attractor 7 Full evolution

FIGURE 6.16: .Overlapping graphs of the rule usage during the development of attractors of size 7. The total rule usage is the amount of sites in the geometry which change in some way pr. development step, while no change shows the number of sites which do not change. Each line is a different run, and the reasons for missing lines or missing parts of lines are overlapping or similar structures. Higher resolution and higher attractors are found in Appendix D

6.2.2 Result for state increase experiment

The results of this experiment did not achieve as good results as the 4 state experiments. As seen in Table 6.5, when the number of states grew the success rate dropped, 3 of the 12 runs had a success rate of 0 of 20 runs, and only 3 experiments achieved over 50% success rate. The genotype efficiency of the successful results are also worse than for the original 3 states CA, which can be seen in the graph in Figure 6.17. Only the single successful run for the attractor of 7 achieved a smaller genotype, and less active rules than the original 3 state. On average the growth in genotype size is larger, but the amount of active rules stay low, meaning that most of the chromosomes added have no effect on the development process. This is contrary to what the purpose of growing a

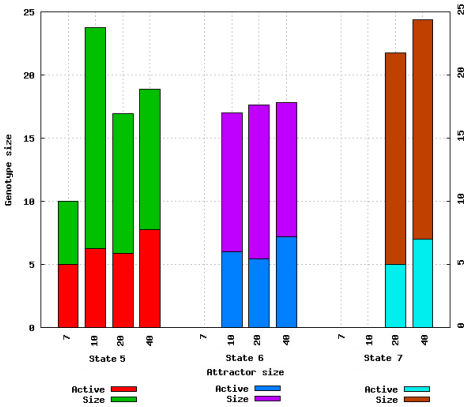


FIGURE 6.17: This graph compares the genotype size to the chromosomes activated during the development process, for the results created by the Growing evolution function when the number of states pr. site increase. The plots are based on the mean value of the successful evolutions of 20 runs.

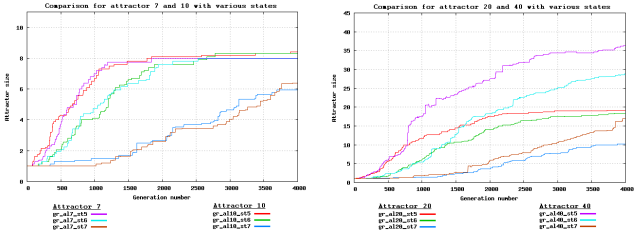
genotype wants to achieve, which is to add a chromosome or two, and make the increase in genotype potential create more suited individuals. But having a continued low usage of rules is an indication that growing a genotype would give efficient genotypes when state space increase, which is the one of the main reasons for running an increased state experiment.

TABLE 6.5: Success rate for increase in state space

	Attractor 7	Attractor 10	Attractor 20	Attractor 40
5 States	1/20	4/20	16/20	16/20
6 States	0/20	1/20	13/20	5/20
7 States	0/20	0/20	4/20	5/20

Observing the graphs in Figure 6.18 a and b we see that increasing the amount of states increases the amount of generations needed to achieve the wanted complexity. This can be seen in that the graph lines for a higher state count grows slower towards the wanted complexity than a lower state amount does. This might be that the sheer amount of rules which needs to be mutated to get one rule which interacts with the current development for an individual, gets larger as the number of states grow. Because there

is a large miss to hit ratio on the mutation, a higher restriction on adding of genes could be a good way of achieving better results. In the graphs showing the genotype growth for these experiments, graphs in Figures 6.19, 6.20, 6.21, and 6.22 a-c, there is an excessive amount of growth compared to the same attractors in the graph in Figure 6.15 a-d. This growth with the large amount of unused genotype space, indicated that the GA needs reconfiguration or some dynamic parameters which adjust the configuration dependant on different variables. Further studying of the graphs tells us that even if they do not reach the wanted attractor within the fixed amount of generations given, there is still a growth, or closing to the wanted attractor, meaning the evolution is not stuck, it just needs more generations. The need of more generations is natural because increasing the number of states, will automatically increase the search space, even for the growing evolution function, meaning that it should have more time to optimise each added chromosome.



(a) Attractor 7; 7states(Brown), 6states(Cyan), 5states(Purple) Attractor 10; 7states(Blue), 6states(Green), 5states(Red)

(b) Attractor 20; 7states(Blue), 6states(Green), 5states(Red) Attractor 40; 7states(Brown), 6states(Cyan), 5states(Purple)

FIGURE 6.18: This is Graph shows a comparison of the evolutions of various attractor lengths when increasing the number of states pr. site. The plots are based on the mean value of 20 runs. For better resolution, see Appendix F.

Because of the easier growth in the increased state experiments, they were run again, but this time the weighting with for genotype size was tuned down, and the weighting for number of active rules was added again. In these runs the genotype size weight was 0.05 and the weight for active rules were 0.15. As for the results of this extra experiment, the success rate fell on every point. having such a low success rate means using the values of the successful runs as a average value will be flawed, and therefore not usable. The

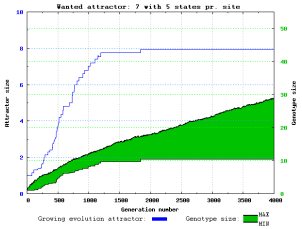


FIGURE 6.19: This is Graph 6.19 a. Graph a, b, and c, represent 5, 6, and 7 states respectively, and is available with higher resolution in Appendix G. The graph shows the growth of attractor and the genotype growth for the wanted attractor 7.

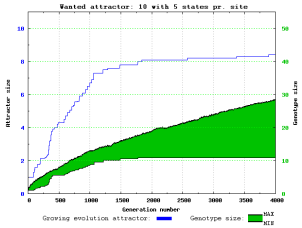


FIGURE 6.20: This is Graph 6.20 a. Graph a, b, and c, represent 5, 6, and 7 states respectively, and is available with higher resolution in Appendix G. The graph shows the growth of attractor and the genotype growth for the wanted attractor 10.

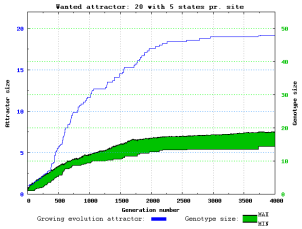


FIGURE 6.21: This is Graph 6.21 a. Graph a, b, and c, represent 5, 6, and 7 states respectively, and is available with higher resolution in Appendix G. The graph shows the growth of attractor the genotype growth for the wanted attractor 20.

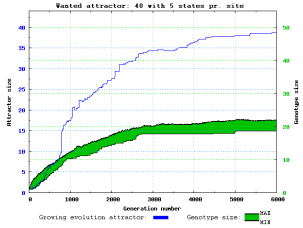


FIGURE 6.22: This is Graph 6.22 a. Graph a, b, and c, represent 5, 6, and 7 states respectively, and is available with higher resolution in Appendix G. The graph shows the growth of attractor and the genotype growth for the wanted attractor 40.

successful results did not achieve genotypes which were significantly more efficient than when this weighting was not included.

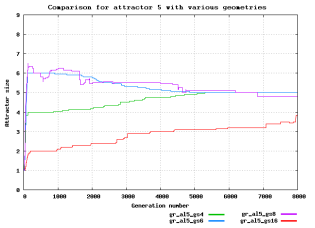
6.2.3 Result for geometry increase experiment

The average attractor values for the different runs is represented in graphs in Figure 6.23 a-c. One can see that most of the runs achieved good to acceptable results, while the odd one out would be the geometry of 16, which is a double of geometry size with regards to the second largest geometry size. Having a large geometry size gave a higher

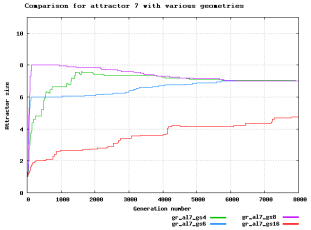
offset for the average attractor length from the wanted attractor, and scored lowest on the success rate in Table 6.6.

TABLE 6.6: Success rate for increase in geometry size

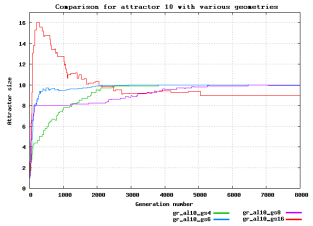
	Grid 4	Grid 6	Grid 8	Grid 16
Attractor 5	20/20	20/20	14/20	5/20
Attractor 7	20/20	20/20	20/20	8/20
Attractor 10	20/20	20/20	19/20	9/20



(a) Wanted attractor 5



(b) Wanted attractor 7



(c) Wanted attractor 10

FIGURE 6.23: Graphs show a comparison of the evolution of various attractors when changing the geometry size. The plots are based on the mean value of 20 runs. Coloring: Geometry size 16(Red), 8(Purple), 6(Blue), and 4(Green). For better resolution, see Appendix H

A valid point for the 16 geometries mediocre success rate is that each of the attractors developed were of a length which relatively shorter than the geometry length in one dimension, making an attractor which simply uses boundary conditions difficult to evolve, it would also need some structural setup in the form of a transient. Almost like building

a geometry in which an attractor of wanted size can be obtained. This dependance of a structure is also needed in the smaller grids, the size of the structure is smaller and thus faster to evolve, giving a higher success rate. Table 6.7 shows the average transients for these experiments. When looking at the effectiveness of the solutions created here, seen in Figure 6.24 , we see that these experiments are performing similar to the default setup for growing and restricted evolution in a 4x4 geometry, seen in the graph in Figure 6.14. These experiments should in theory be performing slightly better on the effectivity for the 4x4 geometry, since there is a weighting for the active rules in these runs, but this is not visible in Figure 6.24 so the margins are too small to notice. On the other hand there are some of the experiments which stick out, by having a very low genotype size and a very high efficiency of said genotype, this is the attractors for the 16x16 geometry, and the attractor of 5 on the 8x8 geometry.

TABLE 6.7: Average transients of different attractors when the geometry size grows

	Grid 4	Grid 6	Grid 8	Grid 16
Attractor 5	3.4	5.75	8.571	24.00
Attractor 7	3.3	4.45	6.55	27.00
Attractor 10	0.85	9.2	14.737	32.11

Looking at the genotype graphs for these attractors, Figure 6.25, 6.26, 6.27 d, and 6.25 c, there is a distinct difference in the growth of genotype compared to all the other runs. There seems to be difficult for the growth of genotype to hold on through natural selection. Had a solution not been present within the low number of rules in which the genotype was able to grow to, the evolution process might not have been able to increase the number of rules enough to achieve a reasonable result, i. e. get stuck at a local maxima. The fact that the grid size of 8 is able to achieve higher genotype sizes in runs for other attractors, and that the grid size of 16 with an attractor of 10 achieves slightly higher genotype than the other attractors for grid 16, can indicate that the growth of genotype stabilises, or slows down when reaching a genotype size which is capable, of a correct, or a close to correct solution. There is also reason to think that because of longer transients, which entails larger structures, the number of sites in the geometry which contain different rule conditions increases, making the addition of a rule to the genotype have a higher probability of interfering with the current structure, which according to Kenneth O. Stanley[4] normally reduces the fitness for the individual.

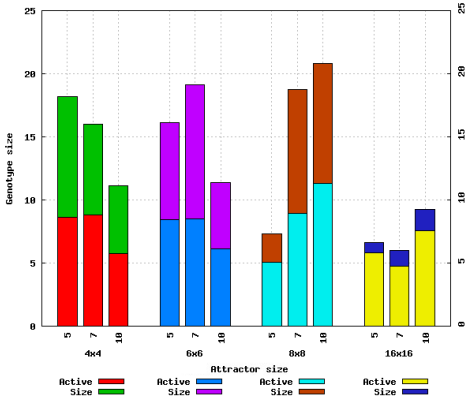


FIGURE 6.24: This graph compares the genotype size to the chromosomes activated during the development process, for the results created by the Growing evolution function when the size of the geometry increased. The plots are based on the mean value of the successful evolutions of 20 runs.

The growing evolution function is able to produce successful result when the geometry changes, the results achieve similar efficiency as earlier experiments except for some results which are superior. Whether or not these superior results are because of a slightly different weighting scheme, or because of more thorough optimisation of the genotype, or simply lucky with the selection of geometry, states and attractor is difficult to say. But for a growing evolution to produce good results on different geometries, it should be designed and configured for this ability. Looking for scalability behaviour in the different geometries there was first check to see if the results for different geometries were somewhat equal in the development phase as the earlier experiments. This meaning that they had an overall low active site count in the geometry pr. development step, and that many of the same type of attractors were found, which showed similar patterns in rule usage. Looking at the graphs in Figure 6.28 a-d we see that many of the same ideas hold. Also in larger geometries are some attractors more likely than other, while some of the attractor on different geometries have a larger set of available attractors. While the geometry of 16 with an attractor of 7 only had 8 successful runs, there were only 5 different attractors developed, where 3 were duplicates and 2 were unique. For the geometry of 8 with an attractor of 7 there were 20 successful runs, out of these 8

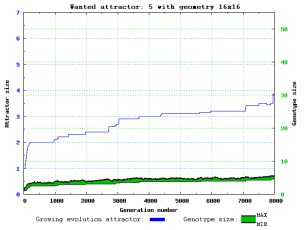


FIGURE 6.25: This is Graph 6.25 d. Graph a, b, c, and d, representing geometry 4, 6, 8, and 16 respectively, is available with higher resolution in Appendix I. The graph shows the growth of attractor and the genotype growth for the wanted attractor 5.

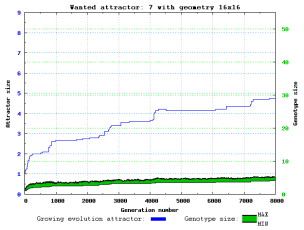


FIGURE 6.26: This is Graph 6.26 d. Graph a, b, c, and d, representing geometry 4, 6, 8, and 16 respectively, is available with higher resolution in Appendix I. The graph shows the growth of attractor and the genotype growth for the wanted attractor 7.

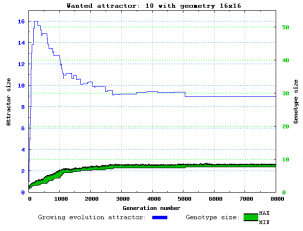
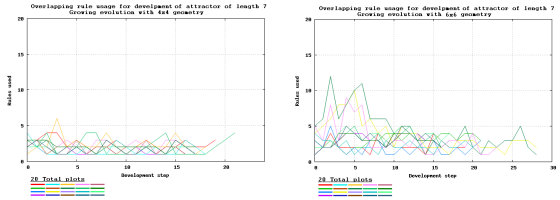


FIGURE 6.27: This is Graph 6.27 d. Graph a, b, c, and d, representing geometry 4, 6, 8, and 16 respectively, is available with higher resolution in Appendix I. The graph shows the growth of attractor and the genotype growth for the wanted attractor 10.

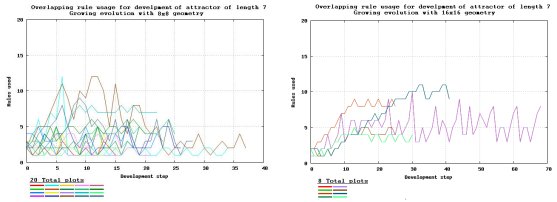
had the same, or related pattern in the rules used pr development step. Looking at the graphs in Figure 6.28 a and b for the geometries of 4 and 6 we see that they have similar behaviour, since the number of colours shown in the graph is not equal to the number of lines drawn, meaning that some completely overlap with another one. Even if they do not completely overlap, they can still be closely related since the lines are the total amount of rules used, and this line can be changed while still keeping the same development by using the mod functionality for duplicates of same chromosome. All of the geometries keep a low rule usage in for all of the attractors, even the ones with long attractors. Low is here relative since the maximum amount of rules active is the number of sites in a geometry, which growths with the power of two in a two

dimensional space. As mentioned earlier, there is a higher usage of transients in the larger geometries, but this is not like the full evolution functions usage of transients, since here it is necessary to be able to create small enough attractors. The last point, where there were repeating structures for attractors which are much larger than the geometry, does not have attractors of sufficient length to be considered.



(a) Evolved on geometry of 4

(b) Evolved on geometry of 6



(c) Evolved on geometry of 8

(d) Evolved on geometry of 16

FIGURE 6.28: Overlapping graphs of the rule usage during the development of attractors of size 7 for different geometries. The total rule usage is the amount of sites in the geometry which change in some way pr. development step, while no change shows the number of sites which do not change. Each line is a different run, and the reasons for missing lines or missing parts of lines are overlapping or similar structures.. For better resolution, see Appendix J

Looking for signs of inherent or emergent scalability or stability during redevelopment of successful solution in new geometries yielded mostly bad results. Scalability in redevelopment was defined as an attractor which regulated its length based on the geometry size it was developed in, but which also regulated the size in what could be observed as a clearly functional pattern. For instance an attractor of length 5 developed in a geometry of 4x4, which when redeveloped gave attractors of length 5, 10, 15, and 35 on geometries of 4, 6, 8, and 16 respectively. Looking for these signs of scalability means

we were looking for attractors which show signs of scalability for all the different geometry sizes, and disregard the ones which might scale for specific increase or decrease in geometry. Looking for different kinds of scalability signs could be possible, for instance visual patterns created as part of the transient which can make the solutions scalable to specific increases or decreases in geometry. This was not done. Signs of stability was defined as an attractor keeping its size when redeveloped in another geometry. When looking for stability there was no need for the stability to show in all the redevelopments, in this case we looked at each redevelopment, and compared it to the original, not like in scalability, where we looked on all the redevelopments of a result together. The redevelopment was done in geometries of size 4, 6, 8, and 16, which should in theory make 6 the difficult number, seeing that it is not in the sequence of the power of 2 like the 3 other values. Most of the developments resulted in did not show any apparent signs to why they achieved or did not achieve any either scalability or stable result. The lack of coherence between the stability and scalability for different attractors, seen in Table 6.8, 6.9, 6.10, and 6.11, means that the settings and wanted result, not the evolution function, have more impact on inherent scalability or stability when redeveloping in other geometries. Redevelopments where there were many either stable, or signs of scalability in the redevelopment, was mainly because a growing evolution function creates multiple similar structures, and in some cases the most likely ones show signs of either scalability or stability. There does not seem to be any emergent behaviour which occurs by growing the genotype that encourages scalability or stability within geometry changes based on this experiment. But some of the attractors were using patterns with fractal qualities to achieve wanted attractors, and fractals do hint at scalability. These did not scale well as attractors, but could mean that looking for structures in the development instead of attractors could shed another light on the prospect of geometry scalability in growing genotypes.

Table 6.8: Stable and scaling attractors when redeveloping successful results in geometry of 4.

	original 4		
	attractor 5	attractor 7	attractor 10
stable in 4x4	N/A	N/A	N/A
stable in 6x6	2 of 20	17 of 20	0 of 20
stable in 8x8	3 of 20	16 of 20	0 of 20
stable in 16x16	2 of 20	16 of 20	0 of 20
shows signs of scaling	1 of 20	0 of 20	16 of 20

Table 6.9: Stable and scaling attractors when redeveloping successful results in geometry of 6.

	original 6		
	attractor 5	attractor 7	attractor 10
stable in 4x4	1 of 20	1 of 20	0 of 20
stable in 6x6	N/A	N/A	N/A
stable in 8x8	4 of 20	2 of 20	1 of 20
stable in 16x16	4 of 20	2 of 20	0 of 20
shows signs of scaling	0 of 20	0 of 20	1 of 20

Table 6.10: Stable and scaling attractors when redeveloping successful results in geometry of 8.

	original 8		
	attractor 5	attractor 7	attractor 10
stable in 4x4	0 of 14	1 of 20	0 of 19
stable in 6x6	1 of 14	3 of 20	4 of 19
stable in 8x8	N/A	N/A	N/A
stable in 16x16	1 of 14	6 of 20	6 of 19
shows signs of scaling	4 of 14	2 of 20	3 of 19

Table 6.11: Stable and scaling attractors when redeveloping successful results in geometry of 16.

	original 16		
	attractor 5	attractor 7	attractor 10
stable in 4x4	4 of 5	2 of 8	0 of 9
stable in 6x6	3 of 5	3 of 8	0 of 9
stable in 8x8	4 of 5	4 of 8	0 of 9
stable in 16x16	N/A	N/A	N/A
shows signs of scaling	0 of 5	2 of 8	3 of 9

Chapter 7

Analysis/Discussion

7.1 Configuration

During the configuration there were done some shortcuts to be able to find suitable parameters in reasonable time. The configuration reached was used as a setup for multiple experiments, ranging from different evolutionary functions, to change in geometry and change in the number of states. Getting a growing genotype evolution function to work was much harder than getting the full genotype evolution function to work. For the full evolution, the results were good with almost all configurations, while the result varied when using a growing evolution. This tells us that changing some parameters can have big effects on results when using a growing evolution, and since we did change parameter for the different experiments, the framework should have had different configurations. Most of the experiments performed reasonably well, which tells us that the setup was most likely not very bad. But for some of the experiments, the results could have had improvements from reconfiguration. Another point to review is the usage of elitism. In the configuration tests, elitism did well, and was chosen because of it. There is also the added benefit of genotype regulation in the elitism. But the negative effects of elitism were not seen until the analysis of the result started. In a growing evolution the negative effects of elitism is much higher than for a full, or a restricted evolution. Elitism works like an anchor, fastening a generation to the currently "best" individual, which for growing evolution not only anchors the fitness, but also the amount of chromosomes. Makes the fitness landscape exploration have a limit on how far from the current fitness, and genotype size it can go. This basically sets a limit of how far forwards and backwards

the algorithm can explore. For a full or a restricted evolution, the anchoring in genotype size means nothing since they are unable to grow, and they will normally have enough room to perform exaptations so that larger jumps in the fitness landscape can happen. This is the large limitation elitism poses on a growing evolution, the hindrance of moving away from local optima if stuck. For this to make sense, just think that 2 evolutionary processes have gotten stuck at the same local maxima, one is a full evolution, the other is a growing. For the processes to get away from this local maxima, each needs to create a individual in their population which have similar fitness to the elite, but is at another point in the fitness landscape. To create such an individual, there is a need for 10 mutations within the set of active chromosomes, but for each generation, only 3 mutations happens pr. individual. For the full evolution, to achieve this, it can simply do part of these mutations in chromosomes which are not used, which can work like a buffer, so when these finally gets mutated to become part of the development, it would be as if 10 mutations happened in a single generation. While for the growing evolution, it does not have the room for these mutations to happen in a "buffer", so it has to add chromosomes, but the anchoring prevents the adding of enough chromosomes to achieve the same functionality as the full evolution. Further, if the growing evolution tries to use the active chromosomes, and not do the mutations in a "buffered area", the fitness will drop such an amount that natural selection will remove it. In this way is elitism bad for a growing evolution. So it could be beneficial to remove elitism, and rather implement another genotype size regulation. Stanley and Miikkulainen [4] created a type of gene regulation, which made added chromosomes have a better ability to stay in generations and not get rooted out by natural selection. By doing this they could ensure that added chromosomes got optimised for a while and not being eliminated. Using a similar function it could be possible to have a much higher threshold for adding chromosomes, giving the evolution time to optimise. The way Stanley and Miikkulainen achieved this behaviour in the algorithm was inspired by nature, in that creatures evolve into different species. Different species are unable to reproduce if they are very different. In our case very different would be have a large difference in genotype size. This kind of regulation could be interesting to add to the current framework to achieve a better exploration of the potential which is added for each chromosome.

7.2 Main experiments

7.2.1 Plan

The main idea was that a growing genotype evolution function would essentially create genotypes which did not explore an unnecessary amount of the state space. Doing this can be used as an alternative to the process of deciding upon a static representation which is large enough to contain a solution to the desired problem. A static representation is often bloated compared to the needed representation, because finding a size which is large enough to contain a solution, but small enough to not feel bloated is difficult. Since deciding upon a solution/fitness function makes large parts of the search space implicitly redundant, the idea of growing an effective genotype seems plausible. Another point we were hoping the growing evolution would handle well was local optima. Since the process consists of optimising a genotype size, and increase the search space by adding a chromosome to the genotype if the optimisation fails, creates an iterative process for the evolution of a solution. For each iteration the fitness landscape increases since new possible combination of rule emerge, making previous generations local optima, no longer an optima but somewhere in between. This continued change in fitness landscape would grow new optima which would hopefully be close to a generations fittest individuals.

7.2.2 Effective

The solutions found are mostly successful in the effective usage of the genotype, which is good based on growing a non bloated representation. This tells us that growing a genotype is a valid opinion to defining a static representation. Since comparing the results to the restricted results tells us they perform approximately identical, we see that the solutions found in the growing evolution is reasonably optimised. How this approach works on larger attractors is difficult to know since our experimenting did not include them. There was some test runs on longer attractors, and they have a similar problem which can be seen in the experiments with longer attractors. The growth in genotype size slows and almost converges, creating a need for extremely many generations to be able to achieve close to reasonable results. Having an increasing amount of generations to optimise solutions before added genes become integrated in generations is good for

effectiveness, could in our framework use some polish, so it would scale to encompass attractors of with the length of thousands.

7.2.3 Inherent effects of growing genotype

7.2.3.1 Low transients

The results from the experiments show very short average transient values. This can be thought of as an inherent behaviour of the growing evolution, since it is a result of the iterative optimisation. Growing a genotype is achieved by adding chromosomes, and optimising them, for so to repeat the process. This means that each added chromosome should in theory maximise its potential for the individual it is developing. In the process of growing attractors, the transient is of great importance to achieve every type of wanted attractor, but of little use when simply counting the size of the attractor. So for a framework which does not intentionally specify the importance of a transient while evolving attractors, will have a system where attractors mean nothing for end result. When evolving a growing genotype the importance is in achieving a best possible result with the fewest amount of rules, giving the transient very little value compared to coming closer to the actual wanted result. This gives most of the results achieved through the experiments a very low transient, but also gives lower success rates to results which are dependant on transients to be successful.

7.2.3.2 Focused evolution

Because of the focus on improvement with every step, the different starting patterns for the first few rules have an implicit weighting. The number of rules which give immediate improvement in fitness are mostly non transient rules. Because of natural selection, and elitism, these rules will spread in the generations, and be represented in most of the individuals when an increase in genotype happens. This makes the starting rules not uniformly chosen, but weighted towards these structures. The implicit weighting in the evolution will be part of each fitness increase, until this path is as close to the wanted solution as it can get, making the evolution focused on improving taken choice for each fitness increase. Focusing on the path taken, and forcing it to be the correct path is a good way of solving the problem, as long as there is a solution with the start choices

taken. Having elitism further strengthens evolutions choice of path, by making it harder for evolution to backtrack and try again, which it sometimes might have to do. This is simply can be thought of by having a solution which is impossible to achieve without a transient and 15 chromosomes, but it is easy to achieve a very close solution with no transient by using only 8 chromosomes. For the growing, close solution will be reached first, but going from the close to the correct solution might need many mutations, more than is likely to occur in an individual in a few generations. So to get from close to correct, there needs to be multiple rounds of mutation, but every round decreases the fitness to such a point it will get removed by natural selection. This amounts to how much the evolution process can decrease the fitness of an individual, and make the individual still be part of future generations. Elitism decreases the amount of change possible, since a future generation will always have an individual which stabilises the fitness, and in turn makes local optima be a problem also for the growing evolution. The focus of the evolution within the growing evolution makes the production of early transients unlikely, which again strengthens the need for a generation to have the ability to backtrack a little. This means elitism might be the wrong way to go for the growing genotype evolution, since it strengthens progress, but hinders backtracking. The thought of growth reducing the amount of evolutionary paths which get stuck at local optima, might be flawed because it depends on a solution being reachable with the currently evolved structure as a starting point. This being said, this is also a problem for restricted and for full genotype evolutions, but they have larger chances of exaptations, or starts which includes transients making giving a better exploration of the whole search space. The problem will most likely be part of the growing evolution function, but there are things like removal of elitism and weighting which can help make this problem smaller. For instance, removal of elitism will help the growing evolution with this problem, since it can in theory allow whole generations to shrink in fitness and by doing that rework early configured chromosomes.

7.2.3.3 Similar structures

The solutions for both growing and restricted on the same attractor lengths share multiple solutions. They often find the same solution patterns, which is part of a small repertoire of patterns. For the restricted, the repertoire is usually a little larger than for

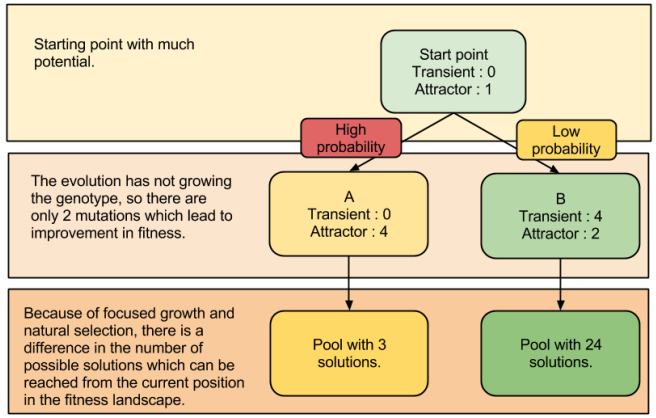


FIGURE 7.1: Displaying how a growing evolution function ends up with a low amount of different solutions, and essentially gives similar structures.

the growing evolution. Compared to the full genotype representation which is very varied in the results, this seems like an inherent behaviour of the growing evolution. There are multiple reasons for this behaviour. First, being strict on the size of the representation immediately removes many solutions possible in larger genotypes. This decrease in search space makes the set of possible solutions shrink, making each of the now possible solutions more likely. Second, if looking at the solution space as a spectre of solutions, ranging in the amount of rules used, from 0 to x, the focus on using few rules which is inherent in a growing evolution makes the the high range of the spectre more difficult to create, which further shrink the number of results which are likely. Third, the blatant focus on growth after optimisation which creates a focused evolution, gives many solutions which are outside the focus area a low to zero probability of being evolved, which further decreases the set of structures which get evolved, illustrated in Figure 7.1. These three behaviours shrink the set of reachable solutions, as well as makes some of the reachable solutions more likely than others. When evolving multiple solutions, this will make the same solutions, or solution structures appear over and over.

We can see, in graphs in Figure 6.16b and in Appendix D the graph for restricted with attractor length 20, that this true also when using restricted evolution. Since

the restricted evolution is based of the genotype size which a growing evolution finds, the structures found are similar to the ones found in a growing evolution. But there are normally some other solutions mixed in with the solutions found with restricted evolution, which are not found in growing evolution. The reason for this is that a growing evolution has a focused evolution, as explained above, which skews the found solutions to the low attractor, low amount of rules used side of the above explained spectre. A Restricted evolution is able to have a different focus while evolving a solutions, because it has a fixed amount of chromosomes to begin with, and knows a solution is within the fixed rules. This growing and restricted comparison becomes more interesting when looking at the comparison graphs for attractors of length 200 and 400 in the graph in Figure **??**g-h. Here one can see the growing achieve better results than the restricted for length 200, and for length 400 a higher and lower size genotype achieves worse than growing, while the same amount of genotype preforms about equal. For the attractor of 200, this most likely means that the majority of the attractors close to a wanted attractor in the search space, is easily reachable through focused evolution, and low transients. While the usage of longer transients only lead to the minority of the solutions, making them harder to evolve. This gives the growing evolution an edge for finding attractors of this complexity. For the attractor of 400, we see that reducing and increasing the genotype by 5 compared to the size which the growing evolution found, both reduces the average attractor value for the restricted runs, while keeping it even to the size found keeps the average value also even. Here we have a genotype size which is very effective in regards to usage and size, but the size still gives a large search space, which makes the amount of solutions in the search space probably quite small. When decreasing this search space with a lower genotype many of the solutions can have gotten cut out. And increasing the genotype can lead to an increase in almost solutions, which are close to the wanted solution within the current representation, to get the almost solution to become a wanted solution one needs a slightly larger representation. This means that there might be ”sweet spots” in the genotype size, or at least ”sweeter spots” than others, where the idea of ”all roads lead to Rome” would work for the wanted attractor. What we mean by this is that, with a genotype size close to a ”sweet spot” most of possible ways of increasing the fitness has the ability to lead the the wanted attractor.

7.2.3.4 Usage of geometry regulation

On all the experiments run on this framework the resulting solutions have been taking advantage of geometry regulation. By geometry regulation we mean actively using the geometry to make structures which achieve attractors. These structures have directional movement, but because of periodic boundary conditions, they cycle around and returns to sites which have previously been handled by the rules in the genotype. In this way the structures can either move in at a speed which combined with the geometry results in an appropriate attractor. Or it can move but leave behind structures with which it can interact when cycling around the periodic boundary, essentially creating an environment for itself to evolve within. These types of moving structures often resemble a game of life glider, which moves in a single direction, but each site it moves, takes x development steps. This makes the repeating structural patterns within an attractor understandable. In graph in Appendix D for the attractor of size 20 we see that for the different solutions there is a pattern repeating 4 times within the length of an attractor. The pattern repeats 4 times because the geometry is of size 4x4, which makes a movement of 4 sites in any direction result in coming back to the starting point. For a larger geometry there will be another number of repeats. This is not the case for the experiments we ran on larger geometries, since this behaviour occurs when the attractor length is much larger than the geometry size, and the attractor length divides evenly on the geometry size, which was not the case for those experiments. The large geometry experiments was done on small attractors, resulting in attractors which did not obtain a pattern repeating within an attractor. But these results were also dependant on geometry regulation, but the kind which creates an environment which it comes back to after a round trip. This usage of the geometry boundaries has two main reasons. First, creating attractors that stay still, and do not use the geometry conditions, requires a larger set of rules than an attractor which walks around and stops by crashing into itself. the construction of stopping mechanism needs ”extra” rules which make them less effective, and thus solutions which are unable to be found until sufficient amount of rules are added. But because after adding a sufficient amount of rules, the now best solution is most likely much closer to a wanted solution than one which is finally able to try to take form, this new kind of solution will simply be passed by. Second, is the restrictions of the CA on the amount of location fixed attractors available when using zygote development. A location fixed attractor means, an attractor which stays within a fixed geometric

space even if the total geometry is limitless. Zygote development means developing from a single site with a non quiescent state, in a surrounding field of quiescent state sites. To be able to achieve an attractor there needs to be, for every set of rules which causes the propagation of a state in any direction, a similar set of rules which stop the propagation, since in a limitless geometry, having an eternally propagation state makes it impossible to create an attractor. The problem is how far can a state get propagated away before it is impossible to stop it, within the boundaries of the CA? For a 3 state CA with a von Neumann neighbourhood, this seems like a short distance. Creating a structure which slowly propagates in a direction, which would be caught by a faster moving propagating structure which destroys each others propagation when the tips of the propagating pattern collides could work. But the actual process of catching up and stopping a propagating pattern is difficult. Using a von Neumann neighbourhood gives limited possibilities to interact with the edges of patterns which is what stopping a propagating pattern needs. This means that to create location fixed attractor the easiest way is for the rules to build a structure, which does not propagate over distances, wherein the attractor is located. Such a structure is very limited in size, at least when the smaller number of states pr. site is concerned, and being very limited in size makes it limited in possible attractors. So both because the rule usage is less, and because of the restrictions the CA settings apply to the development, the solutions created take advantage of the geometry regulation.

7.2.4 For increased state

All the experiments which were ran with an increased number of states pr. site, showed a usage of rules for successful results which was promising. The genotype size was larger than for the experiments with a low number of states pr. site, but the number of rules used to develop the attractor was on the same size range. This suggests that this growing evolution is capable of producing effective and compact solutions for both high and low number of states pr. site. When increasing the number of states, the dynamic of the evolution process changes, which the framework should be wary of. Since there is a very high increase in the rule space, and automatically a much larger increase in search space for each added chromosome, the potential for each chromosome in the genotype is higher. This means that a CA with more states should require less or an equal amount of chromosomes to create equal size attractor to a CA with less states. But this does

not mean that we should have results for the growing evolution with an on average much larger genotype. The large genotype is mostly because of the configuration being specialised for a lower number of states. It seems the main reason for the drop in success rate when increasing the number of states is the optimisation time, i.e. the number of development steps in which an generation can improve before adding further genes. There is a need for a larger optimisation time with the increase of states, because of the increased miss to hit rate in the development process. Being a zygote development, there are some chromosomes which achieve interaction with the development process, and some which do not. Both of these types of rules are important, but for a growing evolution the importance is at different parts of the evolution process. Early in the process a zygote needs to be able to have chromosomes which interact with the single site, and with the simple patterns which emerge from the first interactions. These rules make up a small percentage of the total rule space, and this percentage grows smaller the larger the number of states get. So the higher the number of states grow, the lower the chance of a chromosome hitting one of the rules which gives an impact gets. By hitting and missing it is meant mutation, crossover, and selection giving new individuals whose chromosomes contain these abilities. These hit and miss chances tells us that we should give evolutions with a higher number of states a longer optimisation time, which essentially means slower addition of chromosomes. There are two main reasons for the need of a slower addition of chromosomes. First, if one is continuously adding chromosomes without an adequate amount of time to process the added chromosome, the solution found will have a larger amount of chromosomes, and effective solutions might be passed. Second, since the growing evolution uses a mutation scheme which implies a number of mutations pr. individual instead of some kind of global mutation, the amount of mutations pr. chromosome will shrink with a growth in genotype. Having a high miss, low hit rate we do not want low mutation chance for our chromosomes, since the GA have to sift through many chromosomes to get a hit so we want changes to happen. Increasing the amount of chromosomes simply shrinks the mutation chance pr. chromosome, and ensures that not every chromosome can get a mutation, which in the early phases is bad since the GA is simply looking for a few chromosomes within a large set. Also spreading the search over multiple chromosomes is a bad thing for this algorithm, since one of the main purposes for the growing evolution is to create a small genotype with an effective usage of chromosomes. Speculating further in the hit/miss rate for chromosomes which impact the development, and the time for optimising these,

this rate is dynamic depending on the size of the attractor being evolved, the usage of the geometry, and how far in the process the evolution has gotten. This means that in the early stages, of evolution, there is difficulty mutating chromosomes to affect the development process, bus as the attractor length for the individuals grow, number of rules can possibly be activated during the development process increases. When the number of possibly activated rules go up, the miss rate for mutation of chromosomes goes down, making it easier to affect the development process. But as earlier stated, by adding chromosomes it is highly likely that fitness will initially drop[4]. So in later stages of evolution, the number of hits which positively affect the development grows smaller, while the number of hits which affect the development in any other way grows larger. But when the evolution has gotten to this point, a single hit might not be enough to increase the fitness, there might be a need for 3, or 4 hits at the same time. For the evolution process to add multiple chromosomes, each adde chromosome needs to not get dropped due to the fitness drop, This could happen due to the miss rate, for instance having three add chromosome cycles which all add chromosomes which miss the development process, and thus do not decrease the fitness enough to be removed by natural selection. So in early evolution stages, it is important to be able to get hits in the hit rate, while in the later stages, it might be important to get hits in the miss rate, so that the genotype size can grow to a size which is able to increase fitness. This can explain some of the problems the GA has had on adding chromosomes for long attractors.

7.2.5 For increased geometry

When growing attractors on a larger geometry than the default experiments there was a significant drop in success rate. Larger geometries gave worse results, seen as a geometry of 16 had the worst result while the geometry of 4 and 6 had quite equal results where both were good. All the reached solutions had a good and effective usage of the genotype which they were able to grow, and they showed the same signs as the earlier experiments by having similar structures. This tells us that the growing of genotype will provide equally effective solutions on larger geometries. Had this behaviour not been persistent, and for instance shown worse effectiveness for larger geometries, the growing genotype evolution would not be a good way of evolving solutions for systems

on larger scales. The reduced success rate is troubling, since it gets very low on a geometry size of 16, which might not be very large if used for highly intricate and complex systems. Increasing a geometry increases the state space, and changes the fitness landscape, making the development of a sets of chromosomes be different than for smaller geometries. Because of the inherent usage of geometry regulations in a growing evolution function, there is a higher need for transients when the geometries gets larger. The growing genotype evolution is not very good using transients, because transients is seen as waste of genotype space before it is know whether or not it is needed to achieve the wanted attractor, and this knowledge is not known or taken advantage of in the GA. As the geometry grows, there becomes a need for both more, and longer transients to achieve solutions. By looking at Table 6.7 one can see that this is true, and it makes sense. There are limitations on how large attractors can be developed from a zygote in a limitless geometry, based on the settings for the CA. These limitations in a 3 state ca make for very small attractors, which are most likely lower than 5 (since not one of the solutions on attractors of size 5 have a fixed location attractor which does not use the geometry boundaries.). The creation of an attractor shorter than the geometry but larger than this CA limitation there needs to be a transient. But as soon as the length of the wanted attractor gets past the size of the geometry the possibilities for low transients increase. Attractors whose length divide evenly by the size of the geometry is the simplest to achieve with low transient, i.e. simpler for the growing evolution, while the attractors with an odd division by the geometry size will have a harder time since they are more dependant on the transient. This even and odd division is not final, since the movement of a glider structure do not have to move at a CA max propagation speed, the speed can be reduced by many different factors, depending on the number of states and neighbourhood of the CA. In the experiments which produced attractors of length 10 for a geometry of size 4, there were some examples of this, where glider structures which moved 2 sites every 5th development step occurred, giving transients of 0, even if the attractor length is outside the length for possible fixed location attractors, and does not evenly divide by the geometry. There were also examples where the attractors of length 5 in a geometry of size 4 achieved wanted results without transients. These solutions are fixed location attractors which take advantage of the small geometry, using the boundary conditions to stop the propagation of a state, which means that this is not some special behaviour, just a coincident which occurred because of attractor length, geometry size, and number of states. As for emergent scalability within the growing

evolution function, there does not seem to be any special effect towards it. The ideas to point out is a result with an attractor of length 1 which uses the geometry boundaries is more likely to achieve scaling when size of geometry grows, than being stable. This scalability is most likely destroyed on some increases in geometry, like going from an even to an odd sized geometry and vice versa, because the development of when it impacts itself would be skewed. Attractors which are stable are most likely to either have a transient, most likely a transient which is longer or of equal size to geometry, or be a fixed location attractor. Longer or of equal size is because a transient which circles the geometry and comes back to the starting point before the attractor starts, is an attractor that uses the first round around the geometry to build an environment where the attractor can be created, and as long as this structure is intact the size of the geometry does not matter, the attractor will stay the same. From our redevelopment experiments, we see that the growing genotype evolution function does not have any special emergent behaviour or properties which enable these kinds of scalability, but neither does it have anything to discourage them either, since both scaling and stable results were found.

7.3 Future work

The performed experiments does not conclude the research in a growing genotype evolution, since there are still problem to solve, and further experiments to run. Further work on the idea includes, see if a longer optimisation time is what is needed to drop the genotype size for small attractors when the number of states increase. If that does not work, find what does, since most areas where the real world application of EvoDevo systems are thinkable, need more complexity than given with a 3 state automata. Finding and fixing the root to the problem of growing genotypes for large attractors. At some point in the growth process, it becomes too difficult to add chromosomes, which makes long attractors very difficult and luck dependant. If the growing evolution function only supports low complexity solutions, because of a convergence in the rules, this is highly problematic. So to create some better regulation of when to add genes, and how to make them stick through generations needs to be done, so that the growing evolution is able to have multiple unused/unimportant chromosomes to perform exaptations, without creating a bloated solution. It would be interesting to see if growing a genotype is able to still achieve effective solutions in a reasonable timeframe when using higher number of

states on a larger geometry, so that the function does its job in arbitrary settings. All of the experiments for the growing genotype evolution have been performed with the usage of attractors as a measurement of complexity. This proves points with regards to effectivity of results, and ability to achieve results, but for many real world applications this type of complexity is not suited. Another convenient type of result is fixed structures, either after a set amount of development steps, or as the remains after transients for point attractors. These kinds of results is easier to compare to, for instance, evolution of circuits. Another idea would be to let the growth not only change the search space, but the state space, essentially let the growth happen in both genotype and in the number of states pr. site. This would achieve true complexification, in that the boundaries of how complex a problem the system could solve would not be based on the CA settings. When using a growing genotype we are basically adjusting how much of a fixed amount is needed for a solution, while we still have to create the fixed amount, i.e. search space. By having a growth in both genotype and number of states pr. site the solution is no longer bound by the fixed state space. This would of course require an entire new type of regulation within the framework, where the regulation parameters were dynamic based on parameters like attractor length, geometry size, number of states, genotype size, add rate, and optimisation time etc. These kinds of dynamic regulation parameters could also be of good use in growing genotype evolution, when handling the changing hit/miss rate, or could be used for add rate, to create some kind of adaptable add rate, there are many ways to improve the idea. All in all, the experiments run in this thesis indicated that this kind of evolution works, and that it produces genotype representations which have a highly effective usage compared to what a large manual representation would achieve. The effectiveness of the solutions are persistent through changes in geometry, and with some tweaking for changes in number of states pr. site.

7.4 Afterthought

The area of complexification is currently not deeply explored, making most contributions count. For this thesis the contribution is given by achieving an EvoDevo-system which creates non bloated solutions for different complexities, and tries to find behaviours caused by this EvoDevo-system. Using such a system lets a designer skip a major obstacle for fixed length representation, which is figuring out how large the representation

needs to be[4]. This will also remove the chance of designing too small representations based off wrong heuristics, which would make it a safe approach to evolving systems where it is difficult to find an appropriate size for the representation. Further expanding the system to include a growth in the number of states will achieve a system which can evolve solutions to arbitrary complex problems. This can further reduce the difficulty of designing a representation. Since these kinds of solutions are very effective in regards to rulespace, and genotype size, it could be that for complex solutions, the creation of a manual representation gets bloated to the point where it will use longer computation time to find solutions. Having a system which explores an expanding area within an search space and state space can have better runtime than starting off on a extremely large but fixed search and state space. Since this kind of evolution is highly inspired by natures[47] way of evolution and development, the limitations to what this kind of evolutionary process can achieve is very low. This is only a start which shows that this kind of growing genotype evolution is possible within EvoDevo-systems, and could be a good alternative to static representations. Further inspiration by nature could also be an option, for instance within the development process were the addition of cleavage divisions[47] or pattern formation[47] could be an idea, so the restrictions on location fixed attractors would not be so strictly bound to the fact that development happens from a zygote.

Chapter 8

Conclusion

8.1 Process and content

To create this thesis there has been a long literature study, which in the early parts of the thesis is used to give the reader an overview of the field of study. This contains the workings of complex systems, through the world of a CA. Further are examples and overviews of how the development of the CA can be seen as the morphogenesis of an organism, and how different representations of a genotype and phenotype functions. How these kinds of systems can be created by the use of evolution through natural selection, and that this lead to the creation of an algorithm, inspired by complexification and natural evolution. This algorithm is used to look at how growth in genotype instead of a fixed genotype affects the evolution, and development of organisms which is created by the algorithm. Using this evolution process, there was a hypothesis of this creating smaller and more effective genotypes, and that there might be other emergent abilities resulting of this kind of evolution. To check if this was the case, and this algorithm could exhibit some of the proposed attributes, a framework was built with a basis in a default GA. This framework supports both restricted, full and growing genotype evolution, meaning the genotype size is either static at a set size, static at the size of the rule space, or growing from a size of 1. Simply configuring the framework took a long while, since improvements needed to be implemented when discovered. This ended in a having a single configuration for all different evolution settings. When the configuration was finished, experiments were designed so that we could create a comparison between the results of the growing, and the full and restricted. These experiments were also

designed to see if results persisted through different settings of the CA. The experiments contain the evolution of solutions to different complexity problems, with the complexity measured in the attractor length of the attractor which would be the evolved organism. These results were then studied, and analysed to try to see identifiable behaviour or patterns, and changes in patterns and behaviour when using different complexities, and evolution types.

8.2 Reason

Designing a good representation for EvoDevo-systems is a large obstacle which requires lots of heuristics[4]. To have an alternative which automatically creates this representation through the evolution process would be able to save much time, and also help to make sure that the representation is not too small, or bloated. This and the the fact that an EvoDevo-system already is highly influenced by natural evolution and development, so creating a genotype which grows when needed is just a furthering of this influence, were the main reasons for trying to look for possibilities of smaller genotypes. And further looking at the effects of evolution and development with a smaller genotype. There is also the point that a very large state and search spaces, can create a too large fitness landscape, which makes it difficult to find solutions. By only exploring a limited part of the landscape and then expand the exploration area, could achieve better and faster results.

8.3 Result

The results of the experiments shows that this growing genotype algorithm creates small and effective solutions compared to the full evolution function. It also shows that this behaviour is persistent throughout changes in complexity, geometry size, and states pr. site, indicating that this type of evolution would be able to achieve effective solutions with automatically created representations. The comparison between the growing and the restricted evolution show that they perform very similarly, which indicates that a the growing will be able to perform similarly well to a manually constructed representation, given enough generations. But the growing will not have the need of creating a valid and good representation, since it is done automatically. Studying the results tells that the

attractors created by a growing evolution have a tendency to have low attractors, and depend on the geometry regulation to create solutions which have effective and compact genotypes. There is also some problems with local optima, where it is difficult to create enough added rules to perform exaptations to escape the optima. This problem can not simply be solved by adding more rules, since that would entail adding more rules all the time, creating less compact and effective solutions when not getting stuck.

8.4 Remaining

There is still much work to be done before one can safely say that this algorithm will be able to function as a system for creating a representation and a solution in most EvoDevo-systems. The algorithm needs to have a solution on how to solve the addition of chromosomes for longer attractors and how much optimisation time is given, based on how far in the evolution process the algorithm has gotten. For the complexification reached through the experiments in this thesis, it is not true complexification, since the number of states pr. site is set before the algorithm is run. In setting a fixed amount of states pr. site the maximum complexity is implicitly set, making the growing genotype simply select how much of the state space is to be included in the search space. By improving the system to include not only a growing genotype size, but also a growing number of states pr. site, closer to true complexification would be achieved. This would let the system be tested for arbitrary complex problems, which would tell us if it really is a valid option to a manually created representation. Further, there is a need to stop using attractors as a way of measuring complexity, and try to create stable patterns, which can easier be compared to solutions to real world problems created by EvoDevo-systems. There are possibilities that the creation of structures will not yield similar results to attractors, because structures are often reached after long transients, which were found to be more difficult for the growing evolution than short attractors.

8.5 Final conclusions

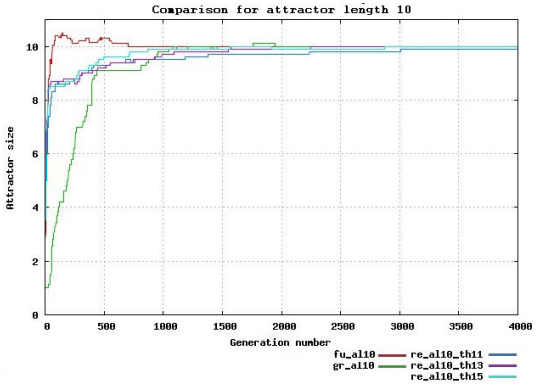
This thesis does not have any definite conclusion in a way like, the hypothesis was correct, or that a growing genotype evolution is the best way to create solutions with a EvoDevo-system. But it does bring to light that this type of evolution could become a

valid option to manually created representations. The thesis gives the growing genotype evolution a good starting potential, by showing that it is able to evolve solutions with different complexities, and the solutions are on average of a low genotype size with an effective usage of said genotype. This genotype creation is persistent when changing the geometry size, and will most likely persistent with a change in the number of states pr. site. On average the growing evolution does use more generations than a restricted or a full evaluation function, but it does not need to use heuristics to create a representation large enough to contain a solution, and it does not create bloated solutions which search unnecessary amounts of the state space. With further experimentation and improvement of the method, it might go from indications of being an alternative, to become a good alternative for solving problems with a EvoDevo-system in real world applications. And the automatic creation of a representation, will help with problems regarding the heuristics and selection of a large enough representation which is a large obstacle todays GA. In society, the EvoDevo systems and CA are still in an early phase, and not utilised to a large degree, they are mostly used in research and experimental projects. This can be said for most of cellular computations modules. A creation of a model which can create solutions for arbitrary complex problems(which is theoretically possible if letting the growth also happen in the state space), could be able to help a tiny step toward a higher utilisation and integration in society, or at least make research or experiments within the field easier.

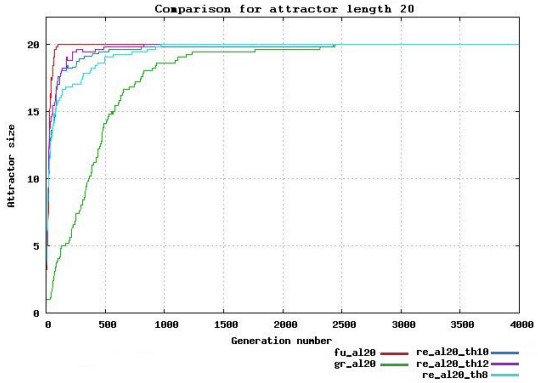
Appendix A

Comparisons of attractor growth

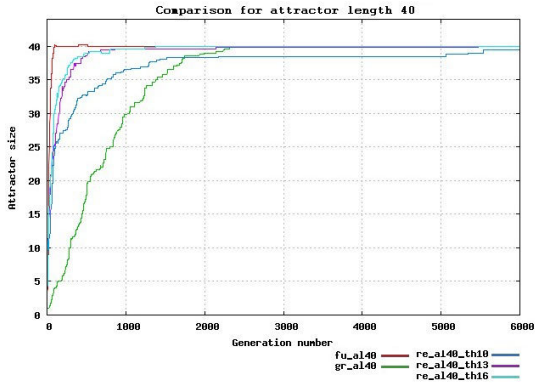
For Full, Growing and Restricted low, mid and high, for different wanted attractors.



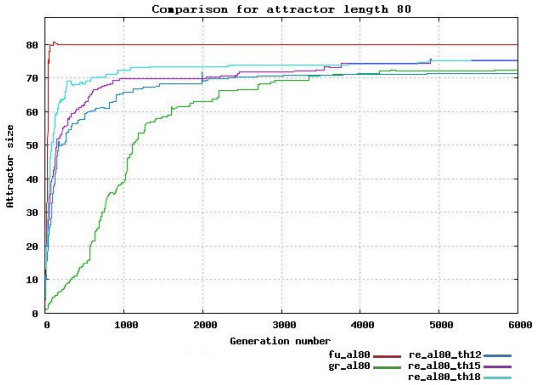
Graph 6.13 b showing attractor length 10.



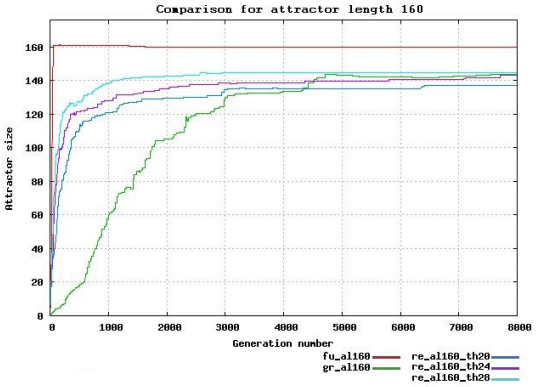
Graph 6.13 c showing attractor length 20.



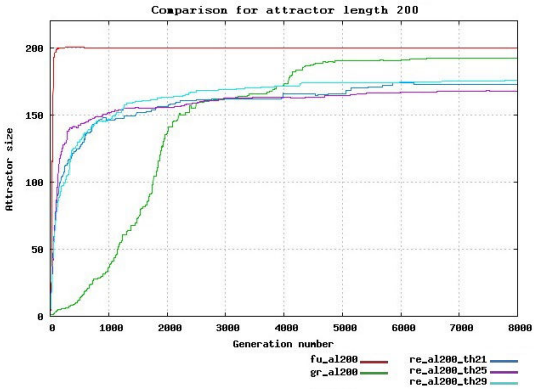
Graph 6.13 d showing attractor length 40.



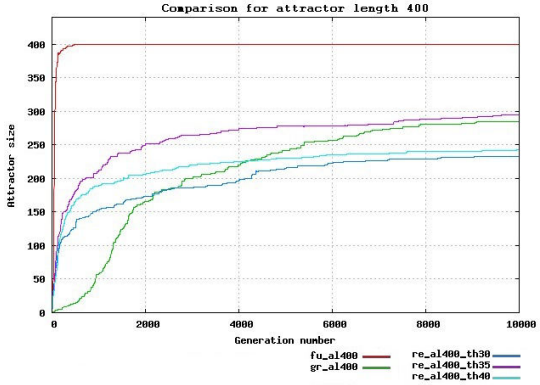
Graph 6.13 e showing attractor length 80.



Graph 6.13 f showing attractor length 160.



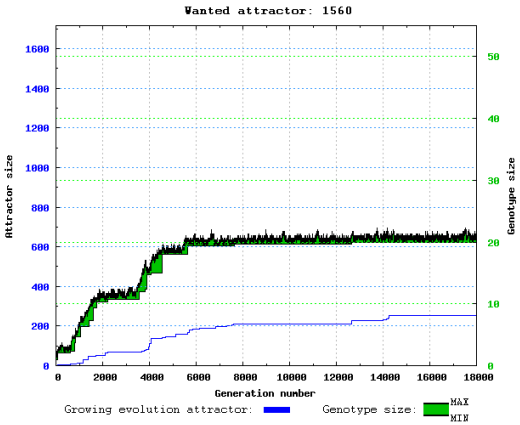
Graph 6.13 g showing attractor length 200.



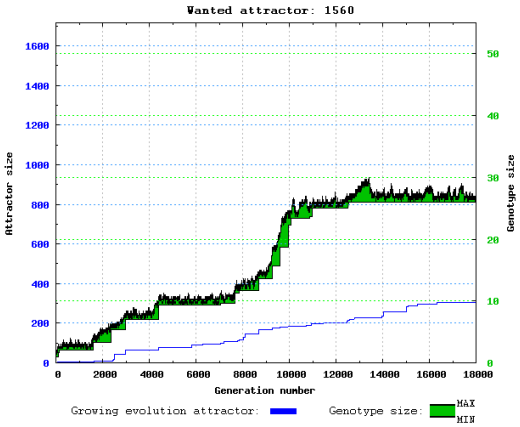
Graph 6.13 h showing attractor length 400.

Appendix B

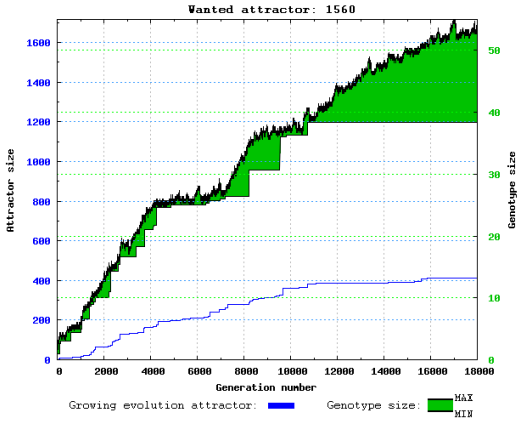
Genotype growth for weighting strategies



Graph 6.11 b with weighting for active rules..



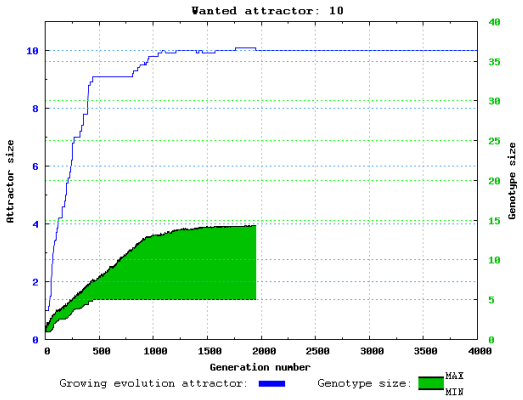
Graph 6.11 c with weighting for active rules turned off.



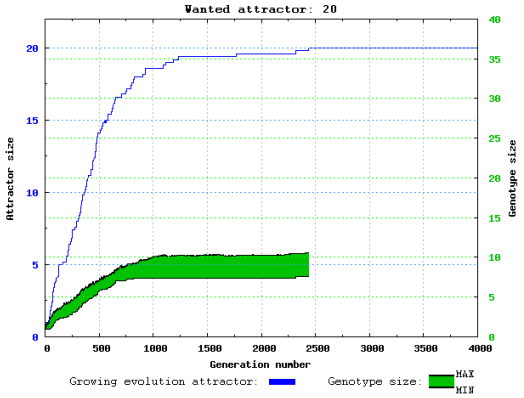
Graph 6.11 d with weighting for active rules turned off, and added a weight to add more rules..

Appendix C

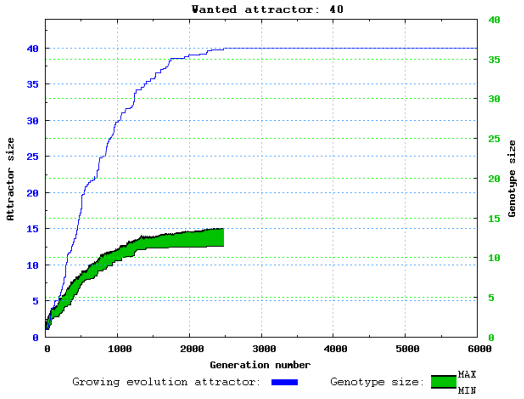
Genotype growth for default settings



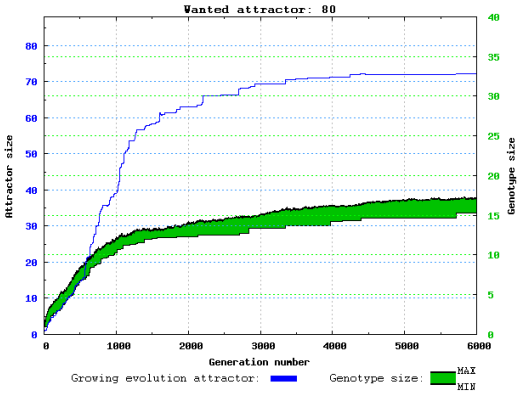
Graph 6.15 b. Attractor length 10.



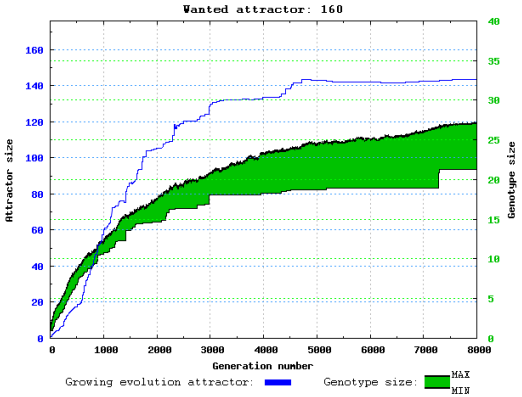
Graph 6.15 c. Attractor length 20.



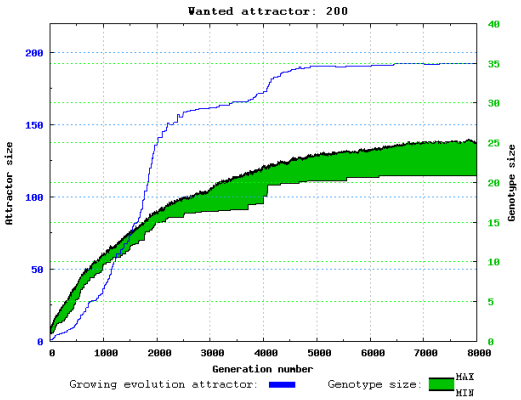
Graph 6.15 d. Attractor length 40.



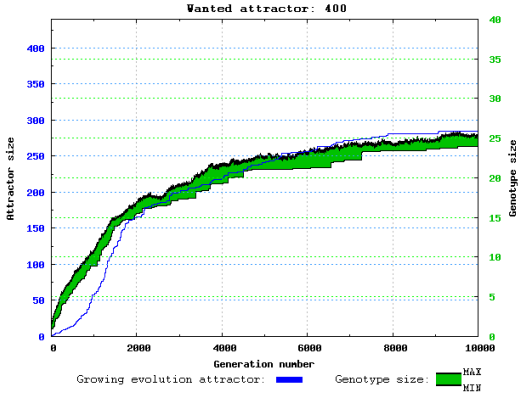
Graph 6.15 e. Attractor length 80.



Graph 6.15 f. Attractor length 160.



Graph 6.15 g. Attractor length 200.



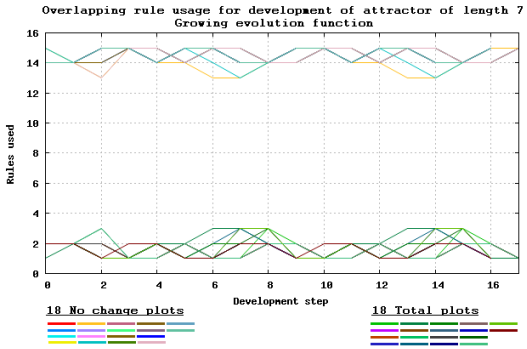
Graph 6.15 h. Attractor length 400.

Appendix D

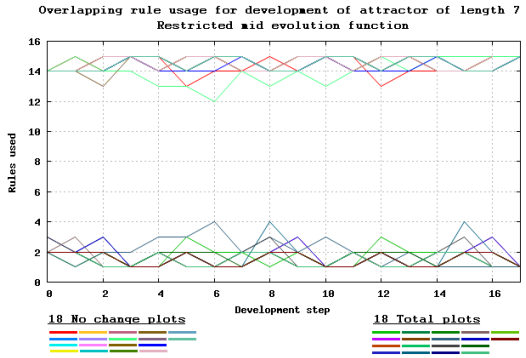
Overlapping development steps

Here we leave out the larger attractors since these show similar results, but will take up much room, and have too many plots to be able to discern any valuable information.

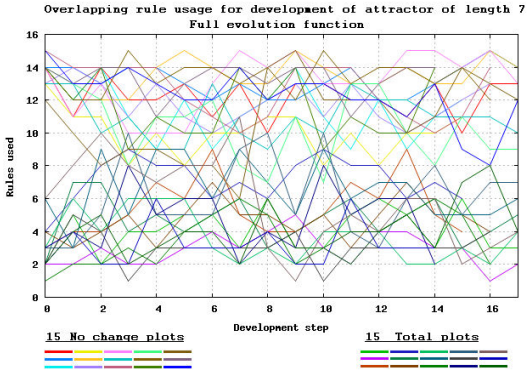
D.1 Attractor 7



Related to Graph 6.16. Attractor 7, Growing evolution.

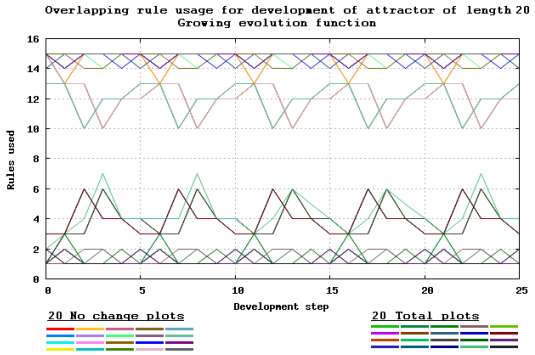


Related to Graph 6.16. Attractor 7, Restricted evolution.

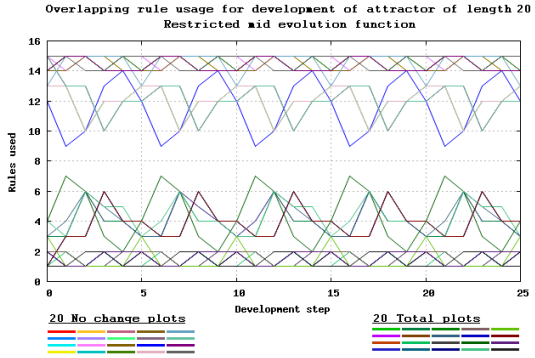


Related to Graph 6.16. Attractor 7, Full evolution.

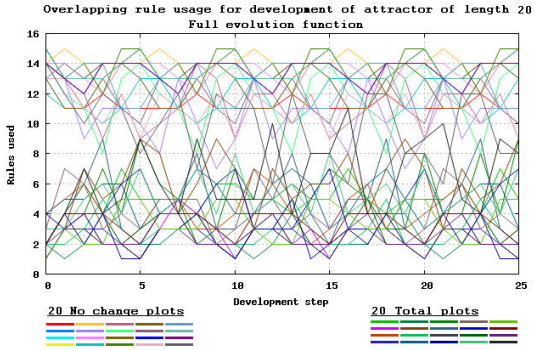
D.2 Attractor 20



Related to Graph 6.16. Attractor 20, Growing evolution.



Related to Graph 6.16. Attractor 20, Restricted evolution.



Related to Graph 6.16. Attractor 20, Full evolution.

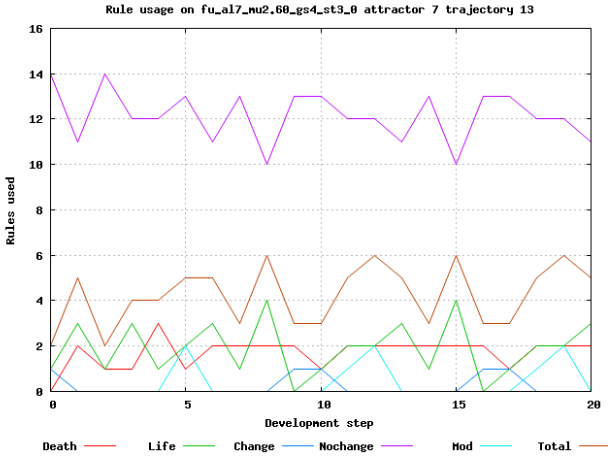
Appendix E

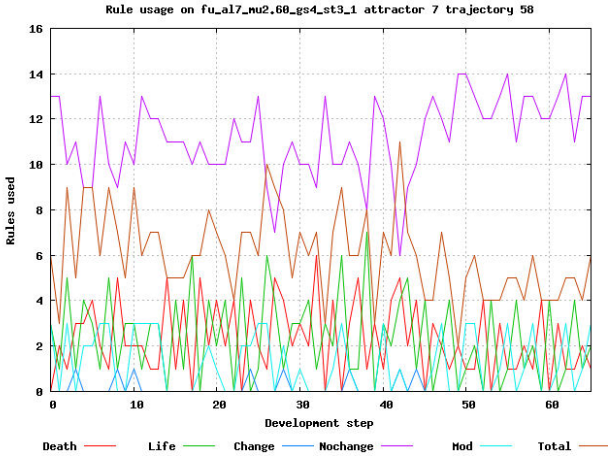
Single development step

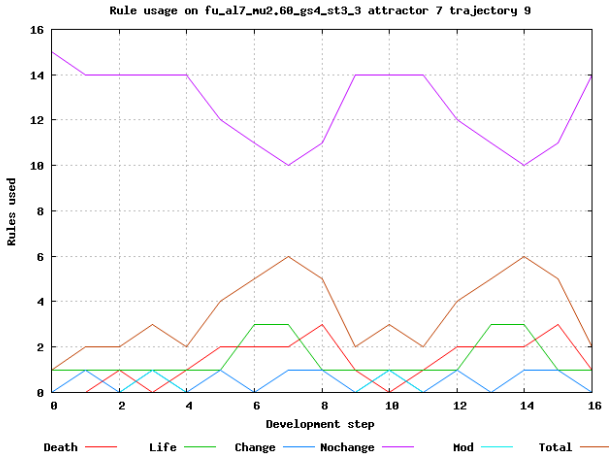
Because of a high amount of graphs. there will only be included 5 graphs from Full, 5 from Restricted mid, and 5 from Growth for the attractors with length 7 and 20. If not there would be approximately 300 graphs, which would take up almost 150 pages. Only include from 7 and 20 since they are one easy, and one difficult attractor, and they are both in a range where it is possible to observe what is going on. 5 of the 20 were included since the idea is to look at many of them, to see patterns, and similar structure, so more than a few are needed, but including 20 would be too much.

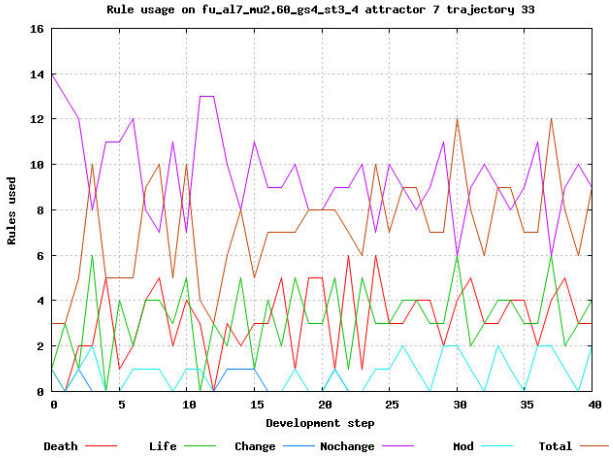
E.1 Attractor 7

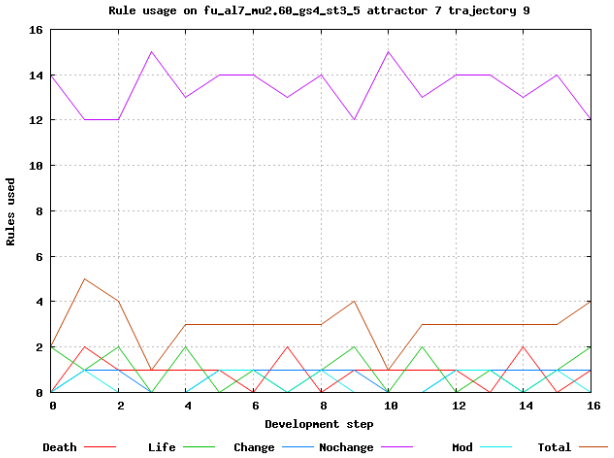
E.1.1 Full evolution



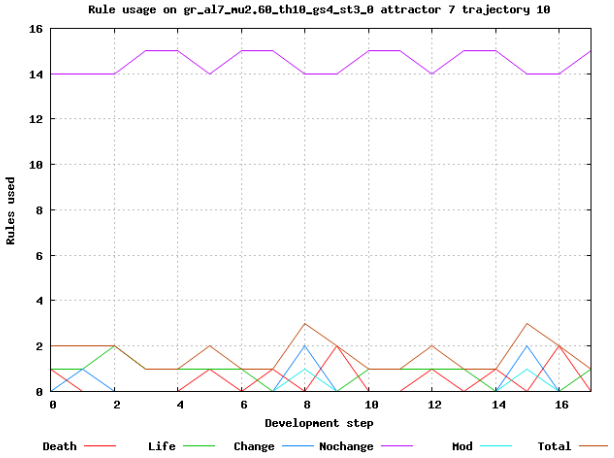


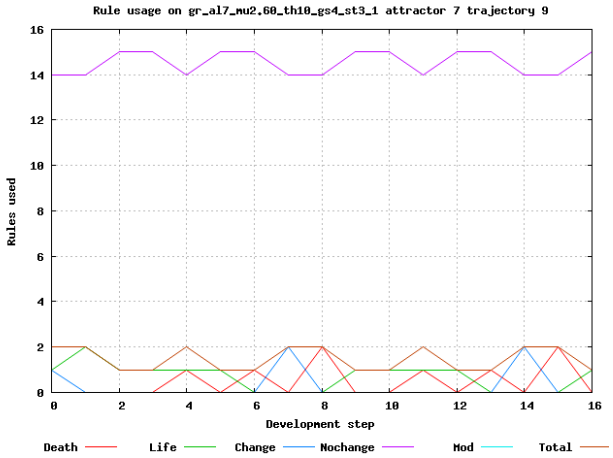


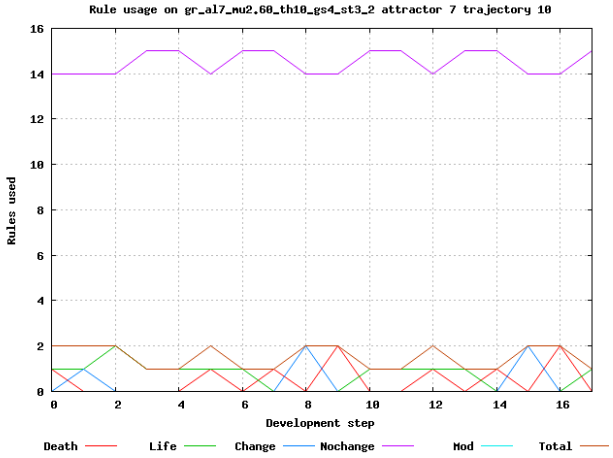


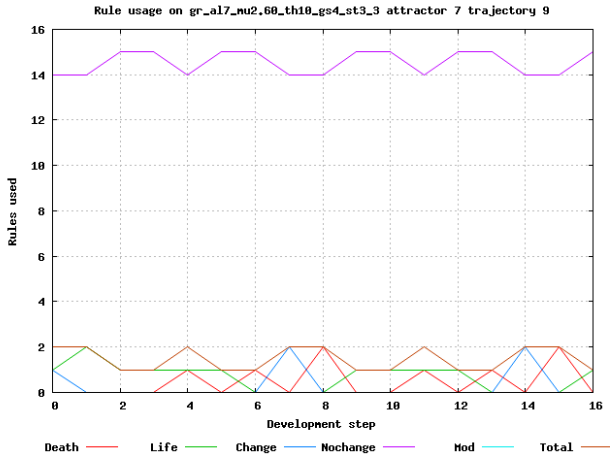


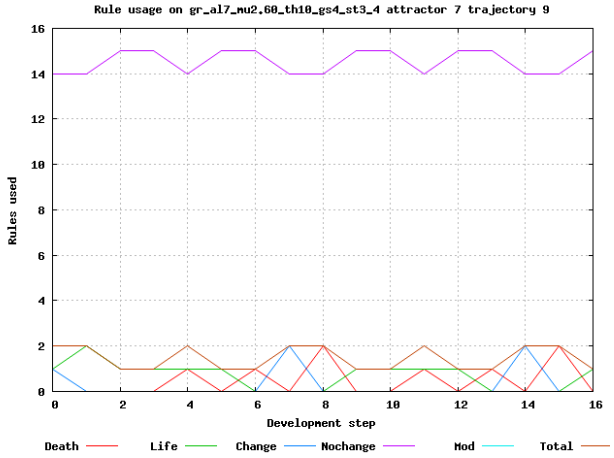
E.1.2 Growing evolution



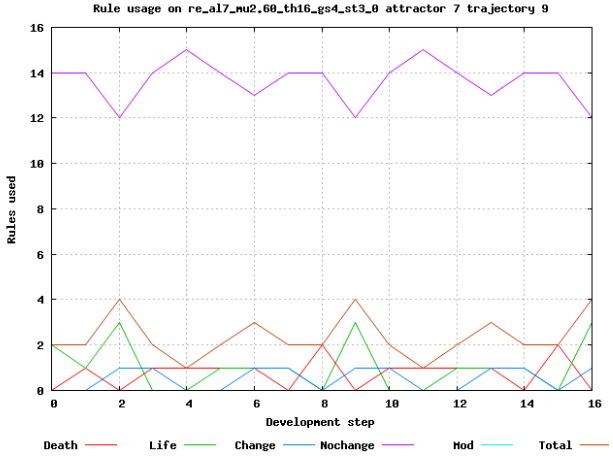

Rule usage on gr_al7_mu2.60_th10_gs4_st3_1 attractor 7 trajectory 9
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

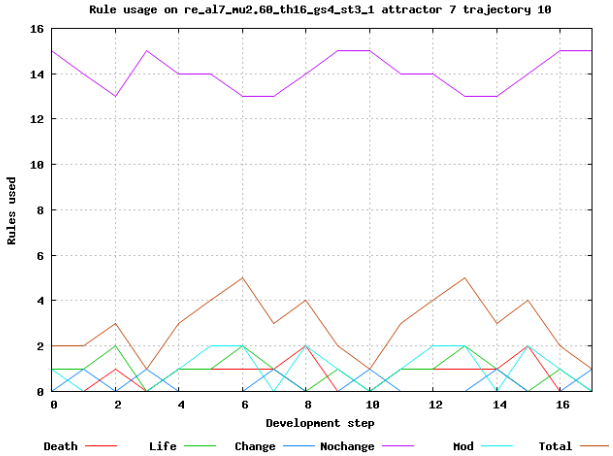

Rule usage on gr_al7_mu2.60_th10_gs4_st3_2 attractor 7 trajectory 10
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

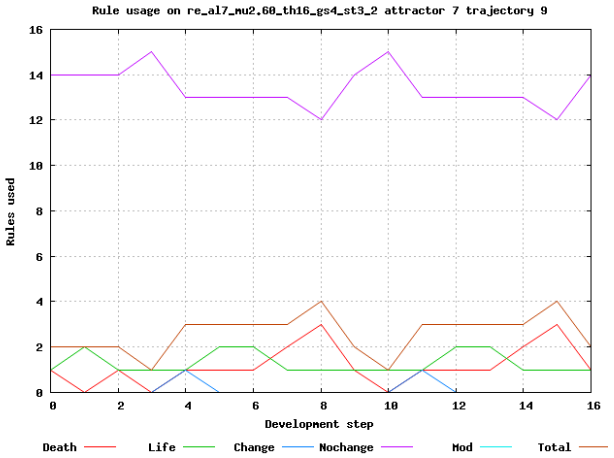

Rule usage on gr_al7_mu2.60_th10_gs4_st3_3 attractor 7 trajectory 9
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

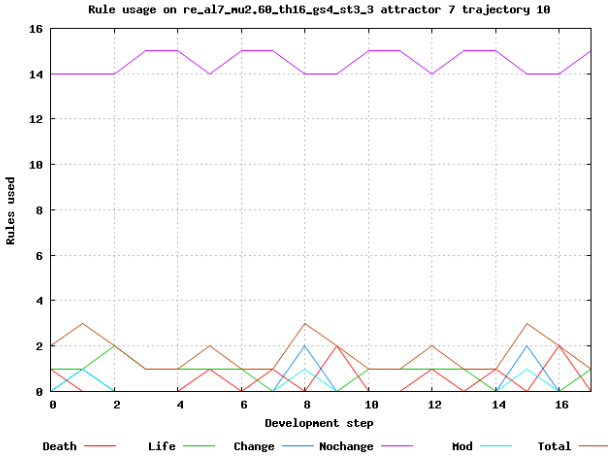

Rule usage on gr_al7_mu2.60_th10_gs4_st3_4 attractor 7 trajectory 9
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

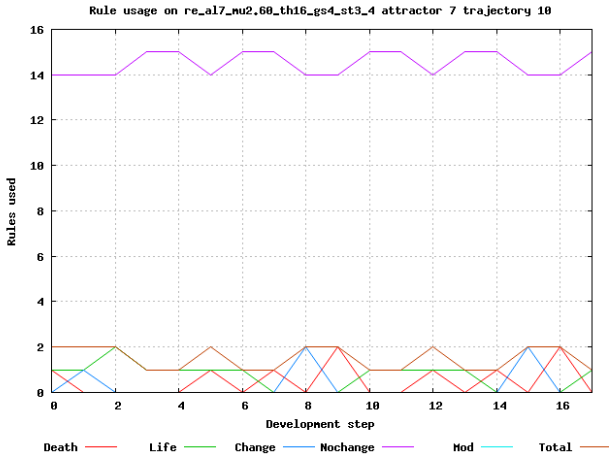
E.1.3 Restricted mid. evolution





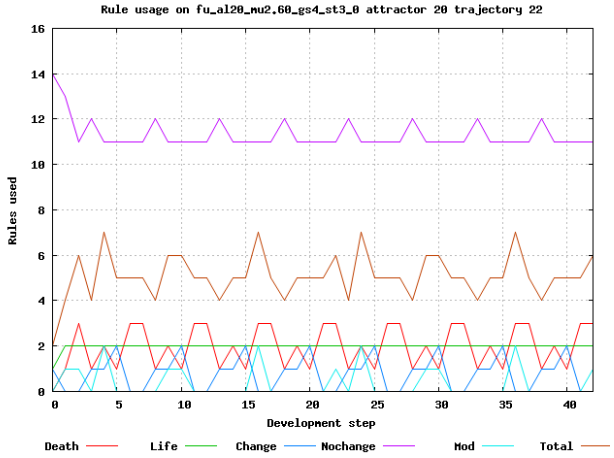


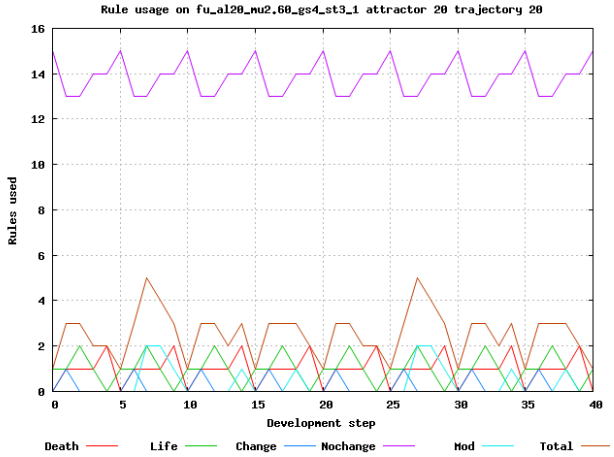


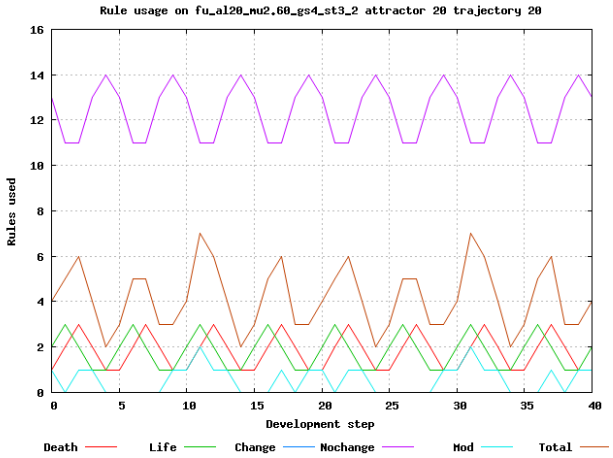


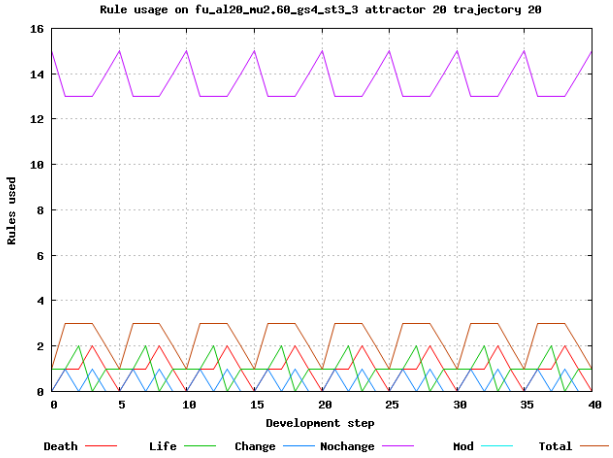
E.2 Attractor 20

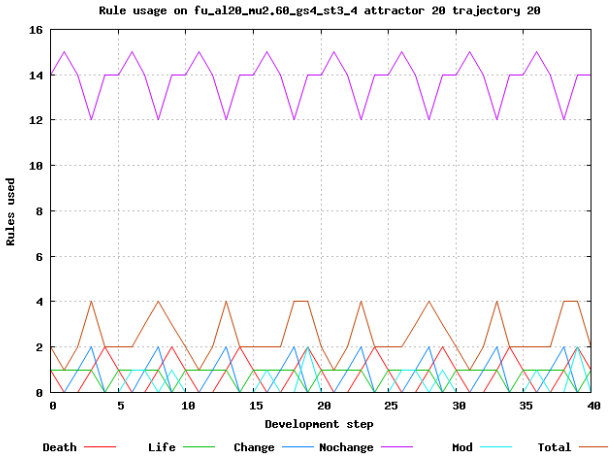
E.2.1 Full evolution



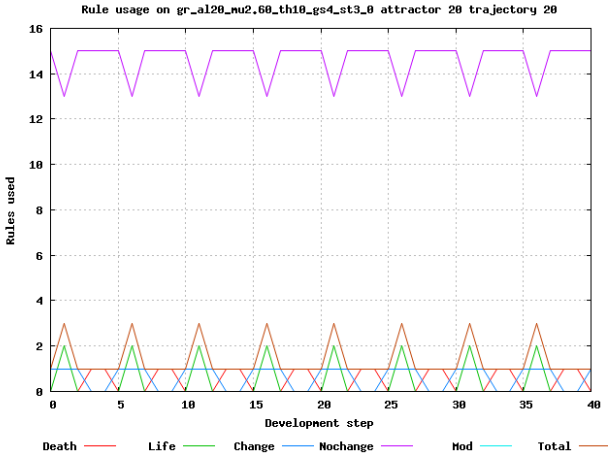


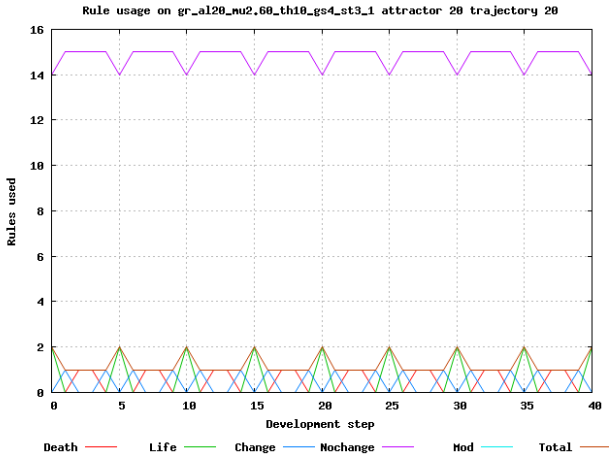


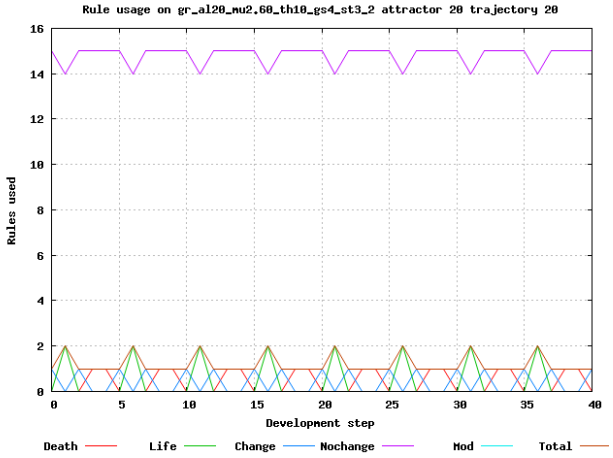


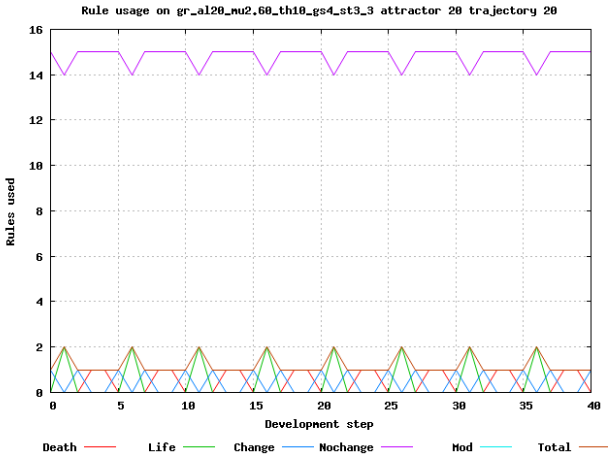


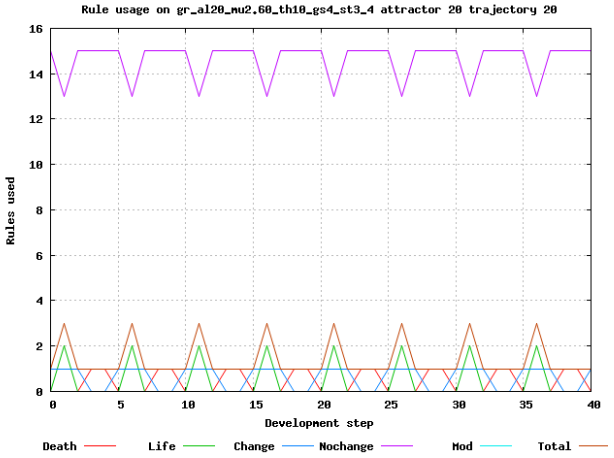
E.2.2 Growing evolution



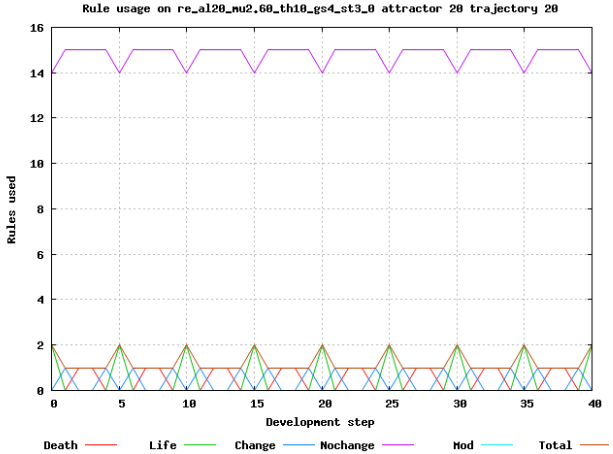

Rule usage on gr_al20_mu2.60_th10_gs4_st3_1 attractor 20 trajectory 20
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

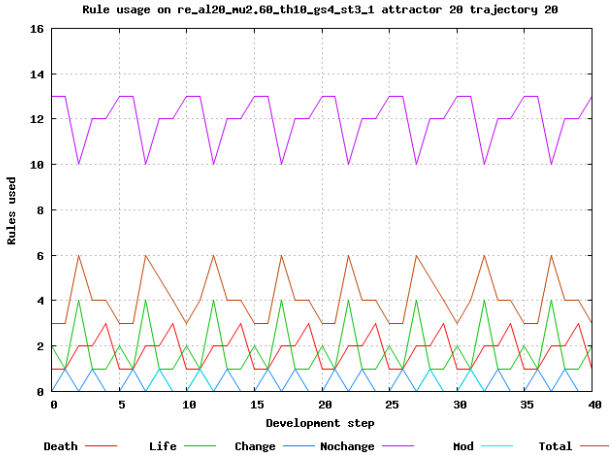

Rule usage on gr_al20_mu2.60_th10_gs4_st3_2 attractor 20 trajectory 20
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

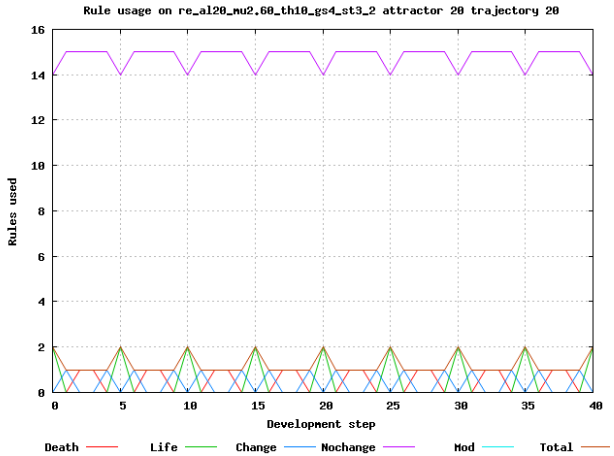

Rule usage on gr_al20_mu2.60_th10_gs4_st3_3 attractor 20 trajectory 20
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

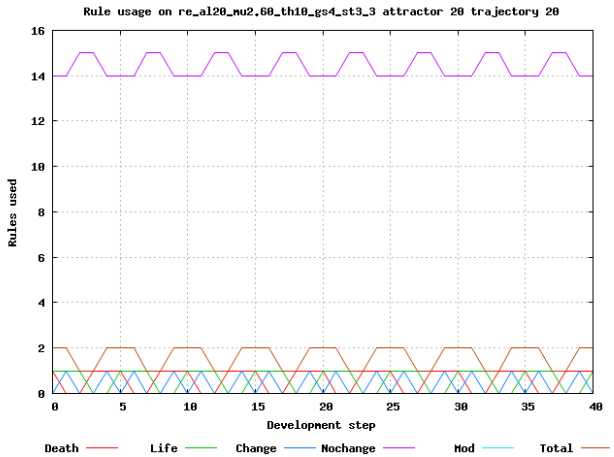

Rule usage on gr_al20_mu2.60_th10_gs4_st3_4 attractor 20 trajectory 20
Rules used
Development step
Death
Life
Change
Nochange
Mod
Total

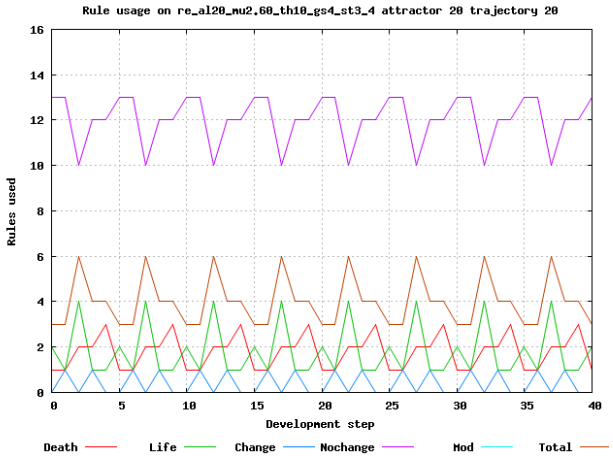
E.2.3 Restricted mid. evolution










Rule usage on re_al20_mu2.60_th10_gs4_st3_4 attractor 20 trajectory 20
Rules used
Development step
0
2
4
6
8
10
12
14
16
0
5
10
15
20
25
30
35
40
Death
Life
Change
Nochange
Mod
Total

Appendix F

Attractor growth for increased state

Better resolution images for Figure 6.18.

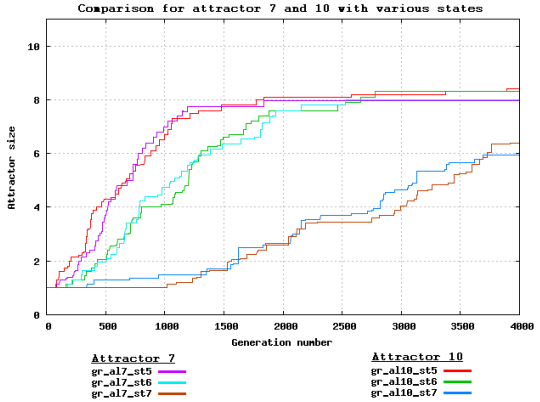


Figure 6.18 a. Showing wanted attractor 7 and 10.

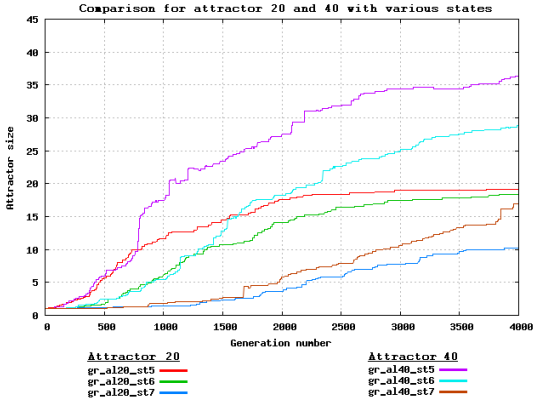


Figure 6.18 b. Showing wanted attractor 20 and 40.

Appendix G

Genotype growth for various states

G.1 Attractor 7

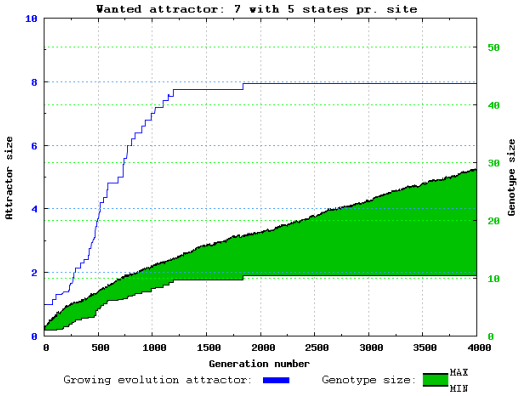


Figure 6.19 a. Showing growth of genotype and attractor for wanted attractor 7 and 5 states.

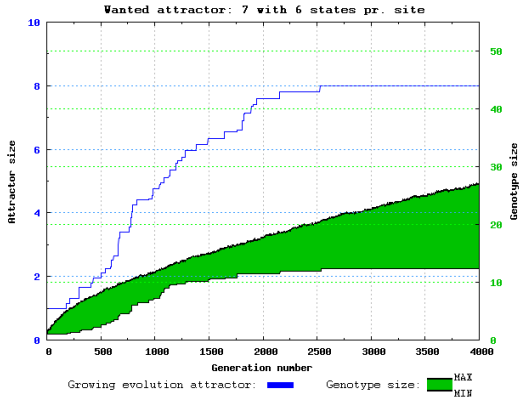


Figure 6.19 b. Showing growth of genotype and attractor for wanted attractor 7 and 6 states.

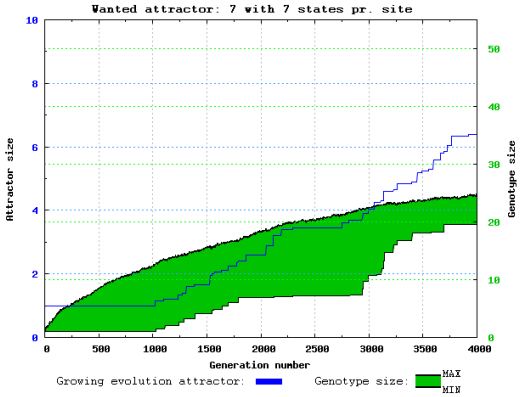


Figure 6.19 c. Showing growth of genotype and attractor for wanted attractor 7 and 7 states.

G.2 Attractor 10

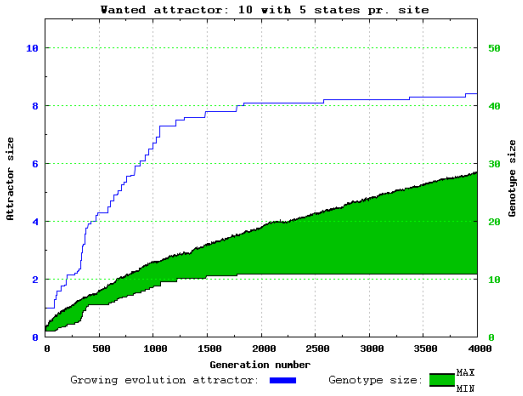


Figure 6.20 a. Showing growth of genotype and attractor for wanted attractor 10 and 5 states.

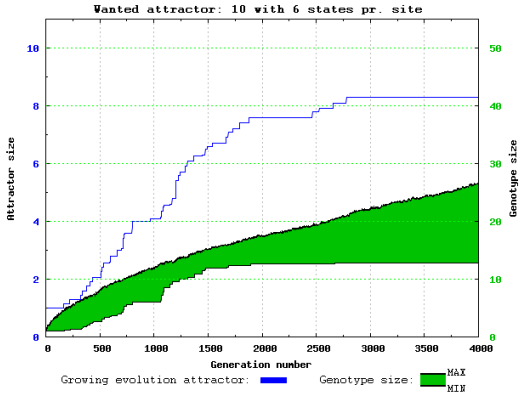


Figure 6.20 b. Showing growth of genotype and attractor for wanted attractor 10 and 6 states.

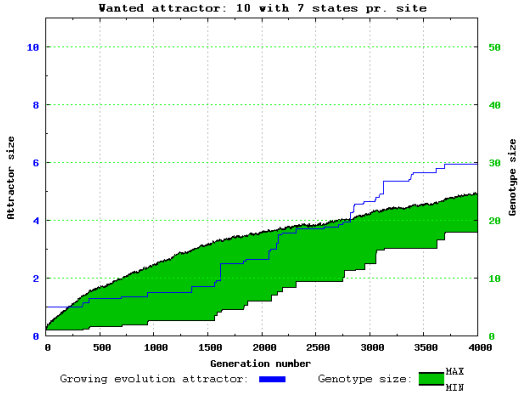


Figure 6.20 c. Showing growth of genotype and attractor for wanted attractor 10 and 7 states.

G.3 Attractor 20

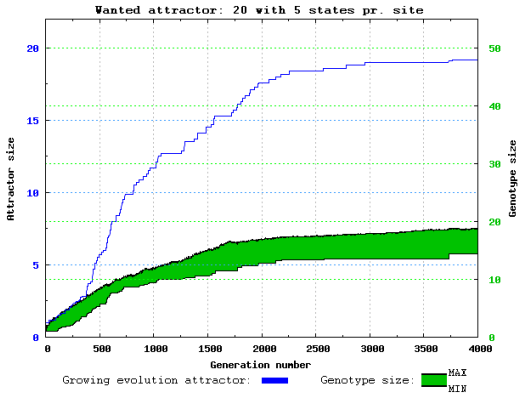


Figure 6.21 a. Showing growth of genotype and attractor for wanted attractor 20 and 5 states.

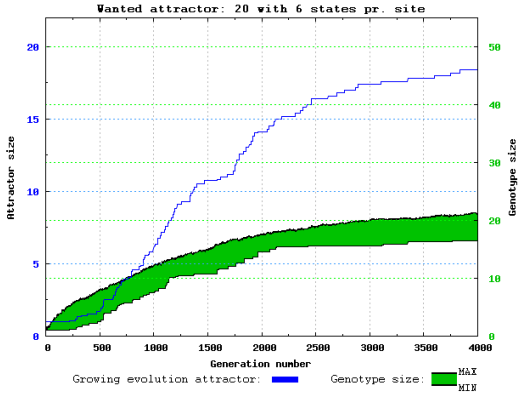


Figure 6.21 b. Showing growth of genotype and attractor for wanted attractor 20 and 6 states.

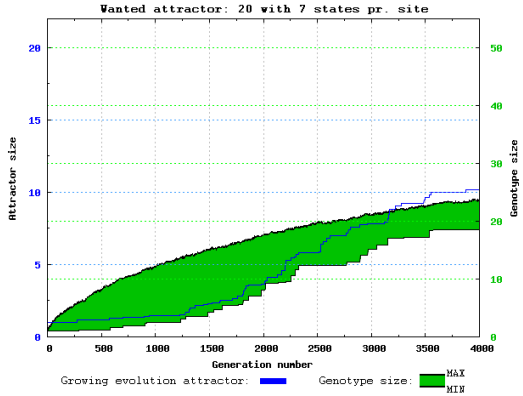


Figure 6.21 c. Showing growth of genotype and attractor for wanted attractor 20 and 7 states.

G.4 Attractor 40

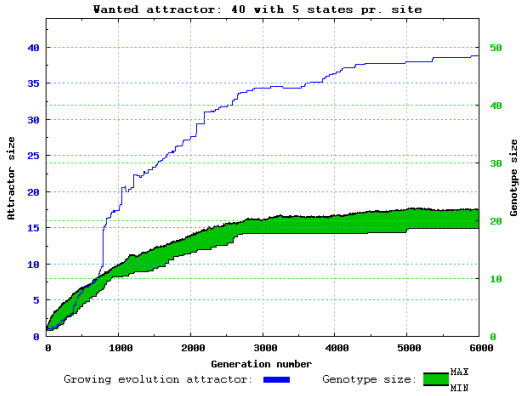


Figure 6.22 a. Showing growth of genotype and attractor for wanted attractor 40 and 5 states.

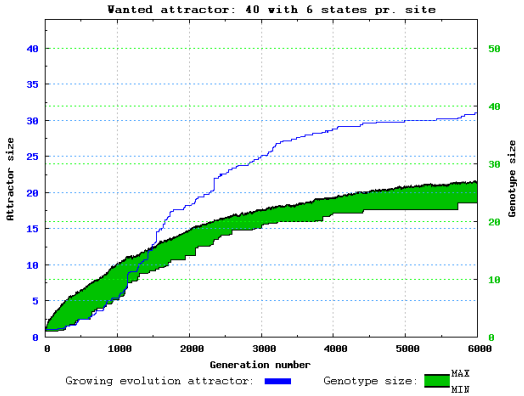


Figure 6.22 b. Showing growth of genotype and attractor for wanted attractor 40 and 6 states.

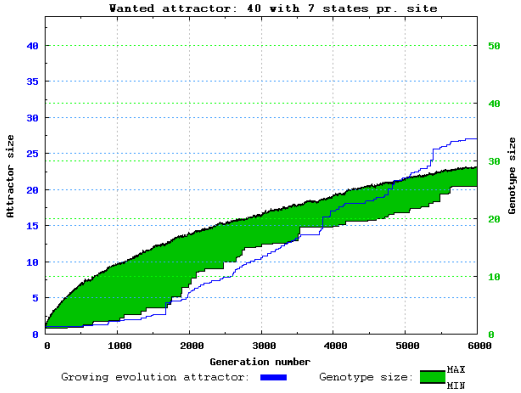


Figure 6.22 c. Showing growth of genotype and attractor for wanted attractor 40 and 7 states.

Appendix H

Attractor growth for various geometries

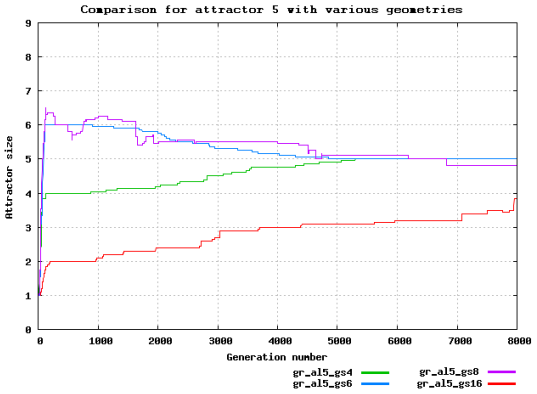


Figure 6.23 a. Showing growth of attractor for wanted attractor 5 for various geometries.

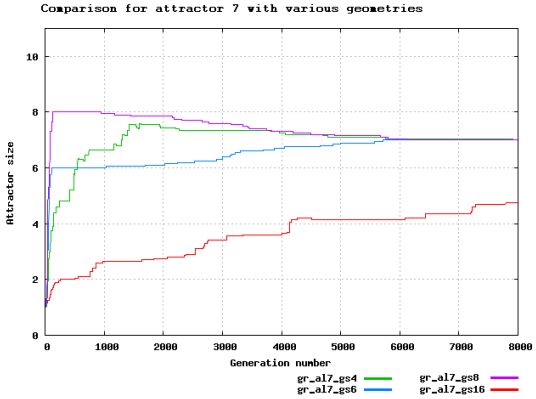


Figure 6.23 b. Showing growth of attractor for wanted attractor 7 for various geometries.

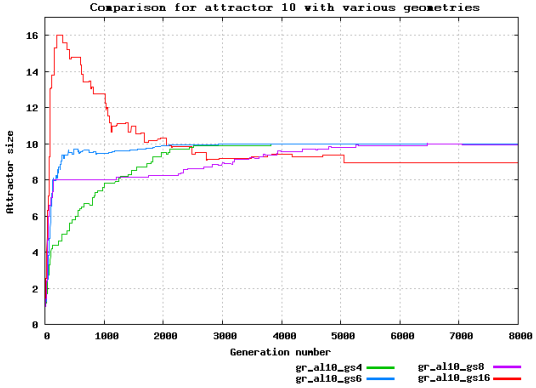


Figure 6.23 c. Showing growth of attractor for wanted attractor 10 for various geometries.

Appendix I

Genotype growth on different attractors and geometries

I.1 Attractor 5

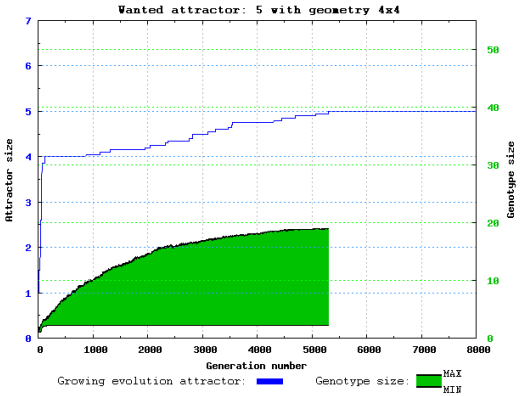


Figure 6.25 a. Showing growth of attractor and genotype for wanted attractor 5 on a geometry of 4.

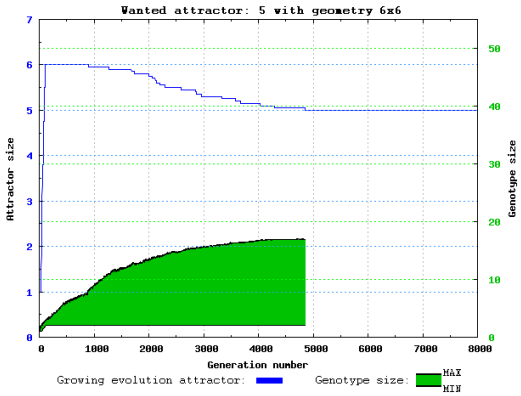


Figure 6.25 b. Showing growth of attractor and genotype for wanted attractor 5 on a geometry of 6.

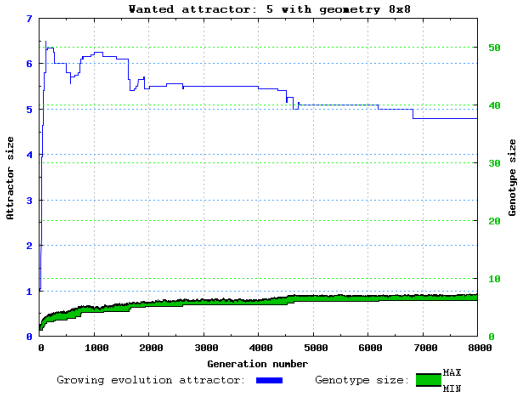


Figure 6.25 c. Showing growth of attractor and genotype for wanted attractor 5 on a geometry of 8.

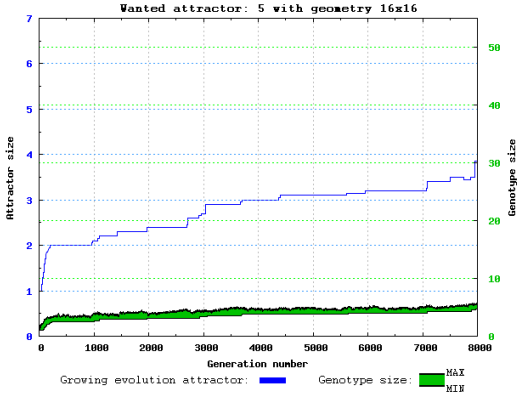


Figure 6.25 d. Showing growth of attractor and genotype for wanted attractor 5 on a geometry of 16.

I.2 Attractor 7

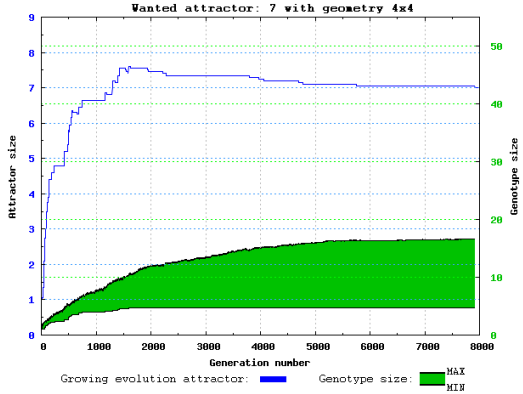


Figure 6.26 a. Showing growth of attractor and genotype for wanted attractor 7 on a geometry of 4.

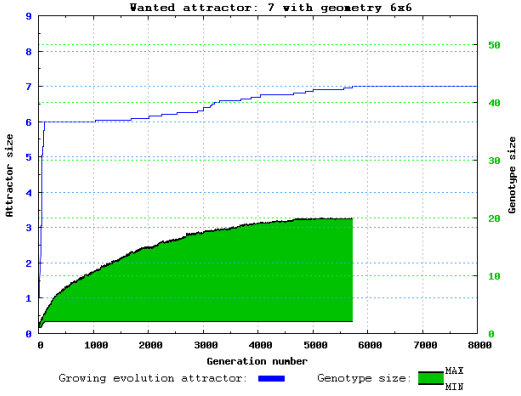


Figure 6.26 b. Showing growth of attractor and genotype for wanted attractor 7 on a geometry of 6.

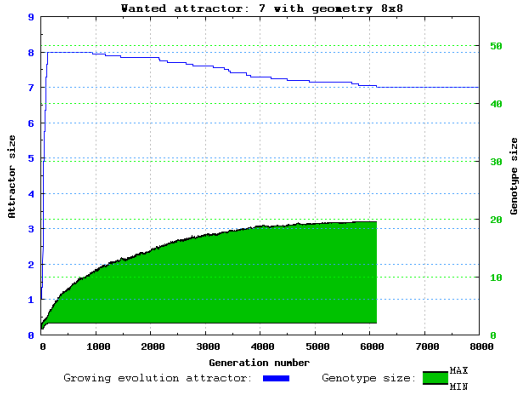


Figure 6.26 c. Showing growth of attractor and genotype for wanted attractor 7 on a geometry of 8.

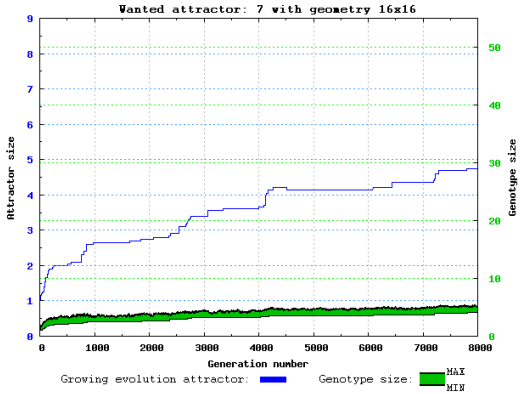


Figure 6.26 d. Showing growth of attractor and genotype for wanted attractor 7 on a geometry of 16.

I.3 Attractor 10

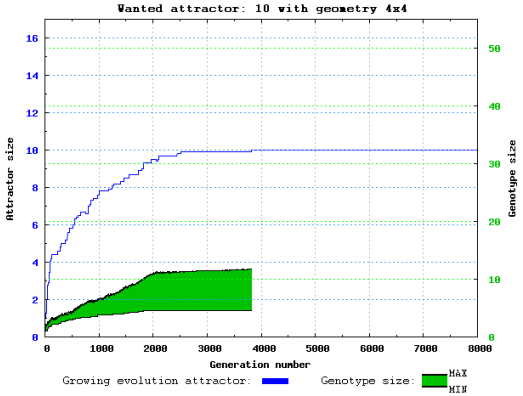


Figure 6.27 a. Showing growth of attractor and genotype for wanted attractor 10 on a geometry of 4.

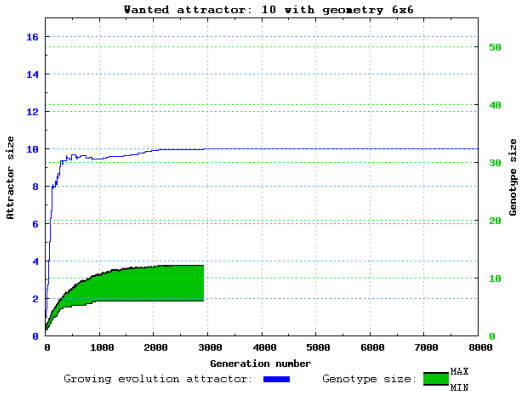


Figure 6.27 b. Showing growth of attractor and genotype for wanted attractor 10 on a geometry of 6.

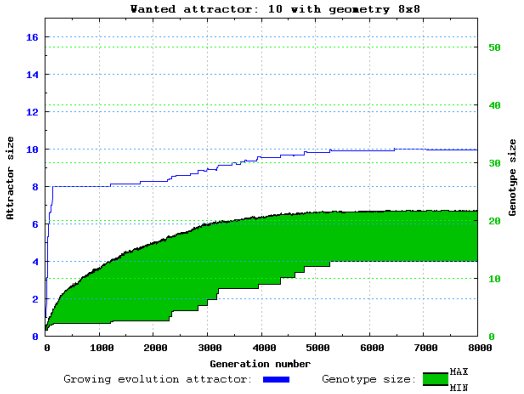


Figure 6.27 c. Showing growth of attractor and genotype for wanted attractor 10 on a geometry of 8.

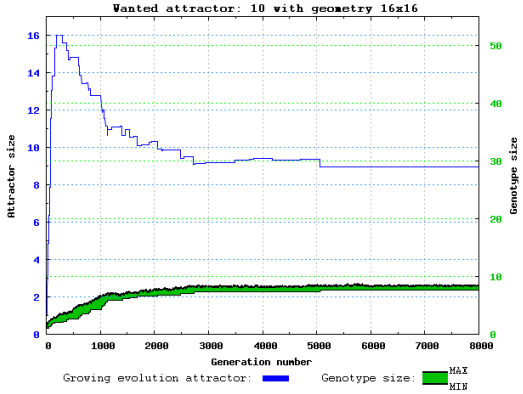


Figure 6.27 d. Showing growth of attractor and genotype for wanted attractor 10 on a geometry of 16.

Appendix J

Overlapping development steps for different geometries

J.1 Attractor 5

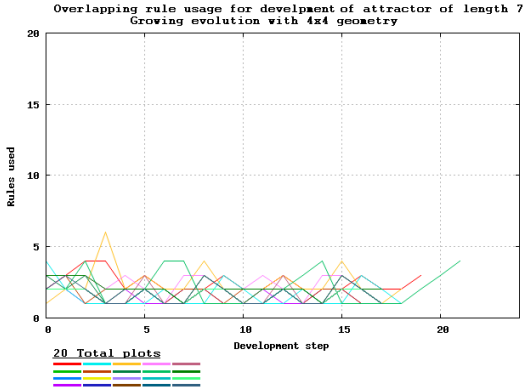


Figure 6.28 a. Showing the overlapping rule usage for each development step for the attractors of length 7 which was evolved on a geometry of 4.

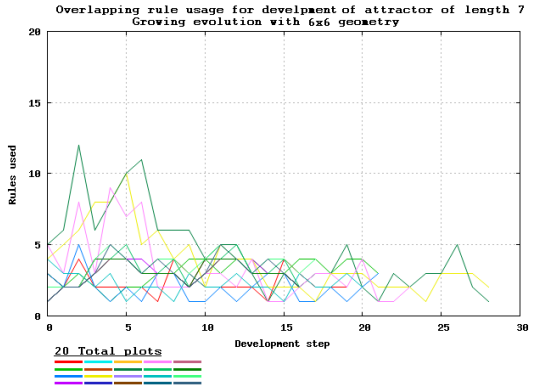


Figure 6.28 b. Showing the overlapping rule usage for each development step for the attractors of length 7 which was evolved on a geometry of 6.

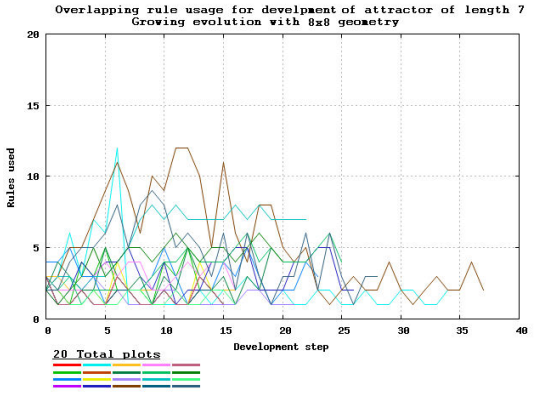


Figure 6.28 c. Showing the overlapping rule usage for each development step for the attractors of length 7 which was evolved on a geometry of 8.

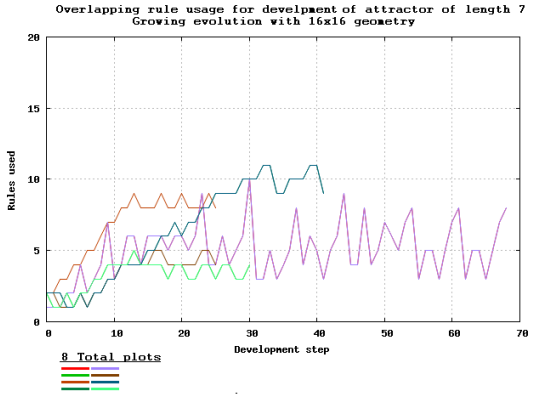


Figure 6.28 d. Showing the overlapping rule usage for each development step for the attractors of length 7 which was evolved on a geometry of 16.

Bibliography

[1] Moshe Sipper. The emergence of cellular computing. *Computer*, 32(7):18–26, July 1999. ISSN 0018-9162. doi: 10.1109/2.774914. URL http://dx.doi.org/10.1109/2.774914.

[2] M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*. Springer-Verlag, 1997.

[3] Gunnar Tufte and PaulineC. Haddow. Towards development on a silicon-based cellular computing machine. *Natural Computing*, 4(4):387–416, 2005. ISSN 1567-7818. doi: 10.1007/s11047-005-3665-8. URL http://dx.doi.org/10.1007/s11047-005-3665-8.

[4] Kenneth O. Stanley and Risto Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2002.

[5] Gunnar Tufte. Discovery and investigation of inherent scalability in developmental genomes. In *ICES*, volume 5216 of *Lecture Notes in Computer Science*, pages 189–200. Springer, 2008. ISBN 978-3-540-85856-0. URL http://dblp.uni-trier.de/db/conf/ices/ices2008.html#Tufte08.

[6] Julian F. Miller. Evolving a Self-Repairing, Self-Regulating, french flag organism. In Kalyanmoy Deb, editor, *GECCO '04: Proceedings of the Sixth Annual Conference on Genetic and Evolutionary Computation*, volume 3102 of *LNCS*, pages 129–139, Berlin, Heidelberg, 2004. Springer. doi: 10.1007/978-3-540-24854-5_12. URL http://dx.doi.org/10.1007/978-3-540-24854-5_12.

[7] Niloy Ganguly, Biplab K. Sikdar, Andreas Deutsch, Geoffrey Canright, and Pal P. Chaudhuri. A survey on cellular automata. 2003. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.107.7729.

[8] C. G. Langton. Studying artificial life with cellular automata. *Physica D*, 22D(1-3): 120–49, 1986.

[9] E.F.Codd. *Cellular automata*. Academic Press, first edition edition, 1968.

[10] E. R. Banks. Information processing and transmission in cellular automata. Technical report, Cambridge, MA, USA, 1971.

[11] Robert T. Wainwright. Life is universal! In *Proceedings of the 7th conference on Winter simulation - Volume 2*, WSC '74, pages 449–459. Winter Simulation Conference, 1974. doi: 10.1145/800290.811303. URL http://dx.doi.org/10.1145/800290.811303.

[12] Andrew Wade. First replication creature spawned in life simulator,. URL http://conwaylife.comandhttp://www.newScientist.com.

[13] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55(3):601–644, July 1983. doi: 10.1103/revmodphys.55.601. URL http://dx.doi.org/10.1103/revmodphys.55.601.

[14] Robert L. Kurucz. Elementary physics in the cellular automaton universe. 2006.

[15] Palash Sarkar. A brief history of cellular automata. *ACM Comput. Surv.*, 32(1): 80–107, March 2000. ISSN 0360-0300. doi: 10.1145/349194.349202. URL http://dx.doi.org/10.1145/349194.349202.

[16] Alvy Ray Smith III. Introduction to and survey of cellular automata or polyautomata theory. In A Lindenmayer and G Rozenberg, editors, *Automata, Languages, Development*, pages 405–422. North-Holland Publishing Company, 1976.

[17] Norman H. Packard and Stephen Wolfram. Two-dimensional cellular automata. *Journal of Statistical Physics*, 38(5):901–946, March 1985. ISSN 0022-4715. doi: 10.1007/bf01010423. URL http://dx.doi.org/10.1007/bf01010423.

[18] Chris G. Langton. Computation at the edge of chaos: phase transitions and emergent computation. *Phys. D*, 42(1-3):12–37, June 1990. ISSN 0167-2789. doi: 10.

1016/0167-2789(90)90064-V. URL http://dx.doi.org/10.1016/0167-2789(90)90064-V.

[19] Stefano Nichele. rajectories and attractor basins as a behavioral description and evaluation criteria for artificial evo-devo systems. Master's thesis, Università degli Studi dell'Insubria, Varese (Italy), 2009.

[20] Stephen Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):1–35, January 1984. ISSN 01672789. doi: 10.1016/0167-2789(84)90245-8. URL http://dx.doi.org/10.1016/0167-2789(84)90245-8.

[21] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 1 edition, May 2002. ISBN 1579550088. URL http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/1579550088.

[22] Stephen Wolfram. Cellular automata. In *Los Alamos Science*, volume 9, pages 2–21, 1983.

[23] T. Toffoli and N. Margolus. *Cellular Automata Machines: A New Environment for Modelling*. Mit Press Series in Scientific Computation. 1987. ISBN 9780262200608. URL http://books.google.no/books?id=HBlJzrBKUTEC.

[24] Michael J. Young. Typical uses of cellular automata, November 2006. URL http://mjyonline.com/CellularAutomataUses.htm.

[25] Sang Il Pak and Tomohisa Hayakawa. Forest fire modeling using cellular automata and percolation threshold analysis. pages 293–298, 2011.

[26] Antonia Mavroudi. Simulating city growth by using the cellular automata algorithm. Master's thesis, Bartlett School of Graduate Studies, University College London, September 2007.

[27] G. Qiu and P. M. A. Sloot. Understanding the complex dynamics of stock markets through cellular automata, 2006.

[28] May Lim, Richard Metzler, and Yaneer Bar-Yam. Global pattern formation and Ethnic/Cultural violence. 317(5844):1540–1544, September 2007. doi: 10.1126/science.1142734. URL http://www.sciencemag.org/cgi/content/abstract/317/5844/1540.

[29] Saifallah Benjaafar, Kevin Dooley, and Wibowo Setyawan. Cellular automata for traffic flow modeling, 1997.

[30] Peter J Bentley. Systematic computation, November 2009. URL http://syscom.wikidot.com.

[31] T. Ostoma and M. Trushyk. Special relativity derived from cellular automata theory: The origin of the universal speed limit, October 1998. URL http://arxiv.org/abs/physics/9810010.

[32] Todd Rowland. 'computation.' from mathworld–a wolfram web resource, created by eric w. weisstein. URL http://mathworld.wolfram.com/Computation.html.

[33] Eric W. Weisstein. 'universality.' from mathworld–a wolfram web resource., . URL http://mathworld.wolfram.com/Universality.html.

[34] K. Culik, L. P. Hurd, and S. Yu. Computation theoretic aspects of cellular automata. *Phys. D*, 45(1-3):357–378, October 1990. ISSN 0167-2789. doi: 10.1016/0167-2789(90)90194-T. URL http://dx.doi.org/10.1016/0167-2789(90)90194-T.

[35] Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1):1–40, 2004.

[36] How to make zuse's z3 a universal computer, 1998.

[37] Timothy O'Connor and Hong Yu Wong. Emergent properties. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2012 edition, 2012.

[38] Tom De Wolf and Tom Holvoet. Emergence versus self-organisation: Different concepts but promising when combined. pages 1–15. Springer-Verlag, 2005.

[39] Y. Bar-Yam. *Dynamics of Complex Systems*. Addison-Wesley stydies in nonlinearity. Westview Press, 1997. ISBN 9780813341217. URL http://books.google.no/books?id=EjeKVJKvd2IC.

[40] Francesco Berto and Jacopo Tagliabue. Cellular automata. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2012 edition, 2012.

[41] Alistair N Boettiger and George Oster. Emergent complexity in simple neural systems. *Commun Integr Biol*, 2(6):467–70, 2009. ISSN 1942-0889. URL http:

//www.biomedsearch.com/nih/Emergent-complexity-in-simple-neural/20195452.html.

[42] Stephen Wolfram. Cellular automata as models of complexity. *Nature*, 311(5985): 419–424, October 1984. doi: 10.1038/311419a0. URL http://dx.doi.org/10.1038/311419a0.

[43] Lemont B. Kier, Paul G. Seybold, and Chao-Kun Cheng. *Modeling Chemical Systems using Cellular Automata*, volume VIII. Springer Publishing Company, Incorporated, 2005.

[44] Stefano Nichele and Gunnar Tufte. Genome parameters as information to forecast emergent developmental behaviors. In *UCNC*, pages 186–197, 2012.

[45] Taras Kowaliw. Measures of complexity for artificial embryogeny. *10th annual conference on Genetic and evolutionary computation (GECCO)*, pages 843–850, 2008.

[46] Howard A. Gutowitz. Transients, cycles, and complexity in cellular automata. *Phys. Rev. A*, 44:44–7881, 1991.

[47] Sanjeev Kumar and Peter J Bentley. 1 - an introduction to computational development. In Sanjeev Kumar and Peter J. Bentley, editors, *On Growth, Form and Computers*, pages 1 – 43. Academic Press, London, 2003. ISBN 978-0-12-428765-5. doi: http://dx.doi.org/10.1016/B978-012428765-5/50034-7. URL http://www.sciencedirect.com/science/article/pii/B9780124287655500347.

[48] Nils Roll-Hansen. Sources of wilhelm johannsen's genotype theory. *Journal of the history of biology*, 42(3):457–493, 2009. ISSN 0022-5010. URL http://www.ncbi.nlm.nih.gov/pubmed/20027784.

[49] M. Shackleton, R. Shipman, and M. Ebner. An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 493–500 vol.1, 2000. doi: 10.1109/CEC.2000.870337.

[50] Rob Shipman, Mark Shackleton, Marc Ebner, and Richard Watson. Neutral search spaces for artificial evolution: A lesson from life. 2000-04-06. URL http://

www.biomedsearch.com/sci/Neutral-Search-Spaces-Artificial-Evolution/0007478449.html.

[51] Lionel Barnett. Ruggedness and neutrality - the NKp family of fitness landscapes. In *Proceedings of the sixth international conference on Artificial life*, ALIFE, pages 18–27, Cambridge, MA, USA, 1998. MIT Press. URL http://portal.acm.org/citation.cfm?id=286143.

[52] Peter J. Bentley and Sanjeev Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In Wolfgang Banzhaf, Jason M. Daida, A. E. Eiben, Max H. Garzon, Vasant Honavar, Mark J. Jakiela, and Robert E. Smith, editors, *GECCO*, pages 35–43. Morgan Kaufmann, 1999. ISBN 1-55860-611-4. URL http://dblp.uni-trier.de/db/conf/gecco/gecco1999.html#BentleyK99.

[53] Charles Darwin. *On the origin of species by means of natural selection, or, The preservation of favoured races in the struggle for life.* London Henry Frowde, 1907. URL http://www.biodiversitylibrary.org/item/18405. http://www.biodiversitylibrary.org/bibliography/2109.

[54] M. Jamshidi. Tools for intelligent control: fuzzy controllers, neural networks and genetic algorithms. *Philosophical Transactions of the Royal Society of London A*, 361(1809):1781–1808, July 2003. ISSN 1364-503X. doi: 10.1098/rsta.2003.1225. URL http://dx.doi.org/10.1098/rsta.2003.1225.

[55] Gunnar Tufte and J Thomassen. Size matters: Scaling of organism and genomes for development of emergent structures. In *Genetic and Evolutionary Computation (GECCO*, ACM conference series. ACM, 2006.

[56] Chris P. Bowers. Simulating evolution with a computational model of embryogeny: Obtaining robustness from evolved individuals. In *In Advances in Artificial Life, Proceeding of the 8th European Conference on Artificial Life: ECAL 2005 (2005*, pages 149–158, 2005.

[57] Gunnar Tufte. From evo to EvoDevo: Mapping and adaptation in artificial development. *Development*, 2008.

[58] Melanie Mitchell, James P. Crutchfield, and Rajarshi Das. Evolving cellular automata with genetic algorithms: A review of recent work. In *In Proceedings of the*

First International Conference on Evolutionary Computation and Its Applications (EvCA'96). Russian Academy of Sciences, 1996.

[59] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30, January 1998. ISSN 1049-3301. doi: 10.1145/272991.272995. URL http://doi.acm.org/10.1145/272991.272995.

[60] Eric W. Weisstein. 'elementary cellular automaton.' from mathworld–a wolfram web resource., . URL http://mathworld.wolfram.com/ElementaryCellularAutomaton.html.

[61] H. Kitano. Designing neural networks using genetic algorithms with graph generation systems. In *Complex Systems*, volume 4, pages 461–476, 1990.

[62] Lee Altenberg. Evolving better representations through selective genome growth. In *In Proceedings of the IEEE World Congress on Computational Intelligence*, pages 182–187. IEEE, 1994.

[63] DouglasC. Wallace and HaroldJ. Morowitz. Genome size and evolution. *Chromosoma*, 40(2):121–126, 1972. ISSN 0009-5915. doi: 10.1007/BF00321457. URL http://dx.doi.org/10.1007/BF00321457.

[64] T R Tiersch and S S Wachtel. On the evolution of genome size of birds. *J Hered*, 82(5):363–8, 1991. ISSN 0022-1503. URL http://www.biomedsearch.com/nih/evolution-genome-size-birds/1940280.html.

[65] T. CAVALIER-SMITH. *The Evolution of Genome Size*. John Wiley, 1985. URL http://books.google.no/books?id=mUG5XwAACAAJ.

[66] Ming Li and Paul M.B. Vitnyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Publishing Company, Incorporated, 3 edition, 2008. ISBN 0387339981, 9780387339986.

[67] W.M. Spears, V. Anand, and United States. Naval Research Center. Navy Center for Applied Research in Artificial Intelligence. *A Study of Crossover Operators in Genetic Programming*. AIC-. Navy Research Laboratory, Navy Center for Applied Research in Artificial Intelligence, 1991. URL http://books.google.no/books?id=gIRoXwAACAAJ.