NTNU
Norwegian University of
Science and Technology

An investigation into Cellular Automata: The Self-Modifying Instruction-Based Approach

Tom Glover

Master of Science in Informatics
Submission date: December 2015
Supervisor: Gunnar Tufte, IDI
Co-supervisor: Stefano Nichele, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

NTNU – Trondheim
Norwegian University of
Science and Technology

An investigation into Cellular Automata: The Self-Modifying Instruction-Based Approach

Tom Glover

December 15, 2015

MASTER THESIS
Department of Computer and Information Science
Norwegian University of Science and Technology

Supervisor: Stefano Nichele

Abstract

Cellular automata is an idealized complex system, where a parallel grid of basic cells communicates locally in order to provide computation. Cellular automata was first proposed as a medium for replicating machines [1]. This thesis aims to return to these root and tries to emerge Stem-cell like behaviour and abilities in Cellular automata.

Stem-cells are a vital part of a complex development process in any multi-cellular organism. They also serve in maintaining the organism once it is fully developed. A cell is classified as a stem-cell if it has the ability to both replicate and develop into a different cell. This is abstracted to mean, the replication and the morphogenesis problem, as one.

Self-Modification(SM), from Simon L. Harding and Co [48] involves a method very strong on morphogenesis. M. Bidlo's Instruction-based approach [10], is a method very strong on replication. This thesis aims to combine the two and show that they work well in combination, and that it can utilize both the strength of IBA on the replication problem, and the strength of SM on the morphogenesis problem. In addition, SM is closer to how genes behave, as SM forms a regulation of how genes are expressed. Moreover, this thesis also aims to show that stem-cells or hierarchical-like structures are possible, and often emerge naturally.

It was discovered that SM gave great advantage as an extension to IBA. It gave IBA useful tools to solve a number of problems IBA alone could not. Also, through this thesis it was shown that SM with IBA could delay its development and use longer development paths when necessary.

preface

This is a master thesis conducted at the Norwegian University of Science and Technology(NTNU), in the Department of Computer and Information Science (IDI). It is the ten month final conclusion to my Master of Science degree in Informatics, which ended December 2015.

Acknowledgement

This thesis would not be possible without the help of my supervisor Stefano Nichele, who has helped me in all manners, from the original idea to the finished result.

I would like to thank Gunnar Tufte for stepping in for a brief time and Keith Downing for advice and answers to questions.

Gratitude is extended to my friends and family who assisted me by proof reading the language in this thesis. Erik Lothe, Luka Cetusic, Karsten Kjensmo, Brian Glover and Morten Grannes.

Summary English

In this thesis we investigate a method for genotype representation in cellular automata. This method is inspired from gene regulation process in biology and is called self-modification. This is then combined with instruction-based approach to form SMIBA.

In order to test this new method, SMIBA together with IBA and TT was tested on a number of problems relevant in artificial life. This firstly, being the problems of replication and of development, which are seen as vital for self-replicating machines. Secondly, these two problems of replication and development are then combined into a new novel problem, which is then subsequently used to test the different methods.

SMIBA was seen to perform well, in comparison to the other methods, on all problems tested. SMIBA and IBA were also shown to scale exceptionally well when incrementing maximum possible states of the CA, often even performing better. Further properties in SMIBA of delayed development and hierarchy were also identified.

Sammendrag Norsk (Summary Norwegian)

I denne oppgaven undersøkes en ny metode for representasjon av genotype i cellular automata. Denne nye metoden er inspirert fra genuttrykk kjent fra biologi og blir kalt selvmodifisering(SM). Denne metoden er da kombinert med instruksjonsbasert tilnærming(IBA) og blir da til SMIBA.

Med hensyn til å teste den nye metoden, vil SMIBA, i tilegg til IBA og TT, testes på problemer relevante i kunstig liv. Med dette menes problemene replikasjon og utvikling, som er sett på som relevante for selvreplikerende maskiner. Disse problemene er også kombinert til et tredje nytt problem som også blir brukt til å teste SMIBA, IBA og TT.

Det viste seg at SMIBA fungerte meget bra på alle problemene som ble testet, sammenlignet med de andre metodene. SMIBA og IBA viste også en sterk evne til å løse problemer hvis, maximum antall tilstander i CA ble økt. På dette problemet fikk SMIBA og IBA, til og med, ofte en økt evne til å løse problemer når dette skjedde. Videre ble SMIBA egenskapene av hierarki og å kunne utsette utviklingen identifisert

Contents

List of Figures

List of Tables

Abbreviations

CA	Cellular Automata
CGP	Cartesian Genetic Programming
EA	Evolutionary Algorithms
GA	Genetic Algorithm
IBA	Instruction-based approach
RISC	Reduced instruction set computing
SM	Self-Modification
SMCGP	Self-Modifying Cartesian Genetic Programming
SMI	Self-Modification Instruction
SMIBA	Self-Modification Instruction-Based Approach
TT	Transition table

Chapter 1

Introduction

"This is how you do it: you sit down at the keyboard and you put one word after another until its done. It's that easy, and that hard."

Neil Gaiman

In this thesis, five research questions and a research goal was formed to highlight the intentions of this research. For every question, a short introduction is presented to explain the motivation behind the question.

Through several billion years, evolution has managed to generate and develop organisms of high complexity from a relatively simple and small genotype starting from only a small zygote. By what is understood in developmental biology today, stem-cells are identified as a defining feature in the development of a multicellular organism. It therefore stands to reason that if this is true in nature, it should also hold true in artificial development and should be beneficial to our understanding of it.

In artificial development, two common problems are often explored; morphogenesis and replication. Firstly, the problem of replication, where the goal is to make copies of an original structure. Secondly, the problem of morphogenesis, where a structure is built from the most simple beginning. Normally, they are explored as separate or isolated problems, yet in nature they are highly interconnected. Replication is a vital part of the morphogenesis process, and cells and creatures alike need to morph before they can replicate. A system that exhibits both morphogenesis and replicating abilities in unison would therefore be closer to the ideal of artificial life. In nature, cells that have both the ability to morph into other cells and the ability to replicate themselves, are identified as stem-cells. For these reasons we aim in this thesis to combine them into a single new problem.

RQ1: Is it possible to evolve a single genotype in cellular automata that can solve both the replication and the morphogenesis problem together?

In 2008, Bidlo and Skarvada devised a method of encoding Cellular Automata (CA), called Instruction-Based Approach (IBA). This method replaces the traditional transition tables with a RISC programming architecture that runs as a small program in each cell. It demonstrates to be especially able to solve the replication problem and also performed well on the morphogenesis problem [10]. In 2007, Harding et al. devised a method of development called Self-Modifying Cartesian Genetic Programming (SMCGP) [48]. This method infuses CGP with additional self-modifying instructions that change the program at every iteration. SMCGP showed great ability to solve the morphogenesis problem. This thesis aims to combine the two into a self-modifying Instruction-based approach(SMIBA), and study its performance.

RQ2: How would a method combining both IBA and Self-modification from SMCGP perform in comparison to regular IBA and explicit representations (Transition Tables)?

It is useful to note that CA is seen as an idealised complex system, and is commonly used for studying artificial life and development. Many researchers have used it together with several developmental strategies to solve both the replication problem, and the morphogenesis problem. Therefore, there exists a good body of work to compare results with.

In 2000, Bedau et al. defined 14 unanswered open questions about artificial life [28]. Over a decade later these 14 questions still serve as a guideline for artificial life research [29].

In [28], one of the many interesting questions raised is "Determine what is inevitable in the open-ended evolution of life.". Underlying this question, they ask what are the features common to all evolutionary processes, or to a broad class of evolutionary processes. Development from embryo to organism is an important process for all multicellular organisms, and underlying this process is the aforementioned stem-cell, which serves the purpose of replication and a development step, to many different cells. Stem-cells rely on a hierarchy between stem-cells where they build other lower level stem-cells.

Another question is "Evaluate the influence of machines on the next major evolutionary transition of life.". In order to predict the the next evolutionary transition, it is useful to look at the ones before it. One evolutionary transition is the transition into a multicellular organism, and central to this organism is the stem-cell. Therefore, understanding how stem-cells developed could become relevant to the understanding of the next transition. This thesis aims to add to the answer of these questions by looking into the following question.

RQ3: Is a stem-cell like feature necessary or at least useful to develop multicellular organisms, and will it self emerge given incentive?

Gene regulation is present in nature, and without it is hard to imagine stem-cells being possible. SMIBA can regulate itself through the use of modifying instruction. Would stem-cell like behaviour be easier in SMIBA than the other methods?

One problem inherent with Transition Table(TT) as a genotype representation, is that of scaling. The TT will grow exponentially in size, based on the number of states in the CA, as well as the number of neighbours considered in the TT. Nature has made many systems more scalable through the use of modularity, regularity, and/or hierarchy [66]. There is evidence that biological cells are very modular [67], and cells are created through the stem-cell hierarchy. The SMIBA and IBA algorithms will scale better than TT, since the genotype size is fixed. Therefore, their runtime will scale better with CA with a high number of states.

RQ4: Does SMIBA scale better in problem solving?

TT is an explicit genotype representation. The size of the TT will grow exponentially with number of states. It is assumed that an alternative such as SMIBA which is implicit, would scale more easily. If it does, is it the implicit nature that does this or is it through some other mechanism or even a combination of many properties?

One such expected property is that a hierarchy already exists or will emerge in SMIBA where parameters of the instruction will cause small changes in the phenotype, IBA instructions will cause larger, and SM instructions cause larger still.

RQ5: Is a hierarchical genotype a good answer to the problem of scaling?

Hierarchy is identified in biology together with modularity and regularity, as a means to increase evolvability [66]. Therefore, a hierarchy in an evolutionary system would seem advantageous in a changing fitness landscape. In a connected but different notion, could a method due to a hierarchy in the genotype allow for better movement even in a static fitness landscape?

These five research questions are formed to help answer the research goal of

RG: Create a novel algorithm SMIBA, test it against already existing

methods, and identify strengths, weaknesses and properties in SMIBA.

These research questions and goal were devised to answer the original research assignment that was specified as to the following:

"This project aims to study the properties of Cellular Automata (CA) machines. Cellular Automata have been studied by von Neumann in the 1940s, in order to design self-replicating artificial systems inspired from biology. In 1970, Conway proposed the "game of life" based on CAs. Our main interest with CA lies in the fact that they are a model of natural computation: massive parallelism, local interactions, entangled memory and processing, they show self-organization and self-regulation properties, and many other characteristics that resemble living systems. CA can be considered as discrete dynamic systems and approached as networks of sparsely connected units (cells). Such networks can be analyzed and evaluated using the same methods for Boolean Networks and Random Boolean Networks (RBN). This opens the possibility to generate and visualize attractor basins and the trajectories from initial state to attractors, which may represent the system behavior.

Investigation of properties of CA development using different developmental mappings, e.g. traditional transition tables vs. instruction-based development. In particular, it may be possible to use artificial evolution (e.g. genetic algorithms) to evolve a particular type of CA function with an instruction-based approach with the introduction of self-modifying instructions (instructions that can self-modify the program that is present in every cell). "

In order to answer this assignment and the research questions, a general research plan was made as the following:

- Jan/Feb: Reading relevant literature and understanding the research problem.
- Mar/Apr: Short literature review of relevant research topics in the field.
- May: Initial coding of necessary artefacts for experiments.
- June/July: First report draft including introduction to the problem, background literature review, explanation of the experimental setup and results comparable with other research papers to confirm the experimental setup. A future work plan is also included in the draft.

- Aug/Sep: Future coding of new artefacts. Some experiments should also be proposed by the student.
- Oct: Experimenting and analysis of results.
- Nov: Dedicated to writing report.
- Dec: Report submission

To ensure progress through the year, this plan was followed as a guideline. The short literature review is in effect chapter 2 of this thesis. On June 13, a half-way report was delivered to the supervisor.

This thesis has been organised into 5 main chapters.

1. Introduction: Speaks of the motivation behind this research.
2. Background: Overview of research topics relevant to this research
3. Methodology: Explanation of the system built and documentation of the testing done to confirm it.
4. Results: Data and analyses of results from the experiments performed and discussion of said results.
5. Conclusion: Summary of results and interpretation. Includes future work.

Chapter 2

Background

"Quotation is a serviceable substitute for wit."

Oscar Wilde

In this chapter we review many of the topics needed to understand the field of artificial life and complex systems. First, biological and natural systems are reviewed. Secondly, artificial systems are reviewed and linked to their biological and natural counterparts. Thirdly, some state-of-the-art solutions to the problem area will be reviewed. In short, this chapter contains the body of research this thesis builds upon. This chapter can be considered to be a short literature review.

2.1 Biology

This thesis crosses into artificial evolution and development and in this case the art imitates life. In order to understand the art it is useful to examine life. The use of such understanding, John Von Neumann phrased quite well in 1951 [3].

"Natural organisms are, as a rule, much more complicated and subtle, and therefore much less well understood in detail, than are artificial automata. Nevertheless, some regularities, which we observe in the organization of the former may be quite instructive in our thinking and planning of the latter; and conversely, a good deal of our experiences and difficulties with our artificial automata can be to some extend projected on our interpretations of natural organisms"

2.1.1 Evolution

Evolution simplified, is the change in the population over generations. It is a wide field, therefore, focus will mainly be on what drives evolution. In 1868 when Darwin published the "variation of animals and plants under domestication" [53], evolution was already an old idea. What Darwin did was, contributed to it with a multitude of evidence. Darwin also came up with the first theory of why evolution occurs, when he published in 1859 [52], his theory of evolution by natural selection. Evolution by natural selection is the understanding that through (phenotype) variations of individuals in a population, the variations of the individual cause different rates of survival. Over generations, individuals that have high survival rate proliferate at a higher rate then the ones who have lower survival rate. Therefore, they will over many generations replace the variations that have a low rate of survival.

When it comes to the process of how new variations appear, Darwin struggled to answer. This was not surprising, since the explanation, which is mutation, relies heavily on the understanding of genetics. Such understanding was not an accepted nor a known theory at the time. It did originate at around 1860 [56] first with Mendel, but it was built up over a long period of time into the modern understanding we have today.

Mutation

Mutation[56] is random errors that cause permanent changes in ribonucleic acid(RNA) or deoxyribonucleic acid(DNA). DNA is in short the information molecule, which is the source code of a cell. RNA is a copy of DNA that is involved in protein creation and also used in transmission of genetic information. We will not go into details as to how mutation may change them, but suffice it to say that if it does occur, DNA or RNA that is copied from changed DNA or RNA also carries the same change. In many cases mutation will not necessarily cause a variation, as mutations might change genes that are inactive, or might happen very late in the developmental process.

Genetic drift

Another driving force in evolution is genetic drift, which was first introduced in 1929 by Wright [55]. Genetic drift works by the stochastic way new generations are created. Random chances might cause variations to be forgotten because they are not transferred to the next generation. Over time certain variations might allow the population to drift away from fitness peaks and into new fitness peaks. The theory also states that genetic drift is more likely to happen the smaller the population [13]. This makes it highly relevant to artificial evolution, because small populations are often used.

2.1.2 Development and Genotype to Phenotype

In both artificial and natural development there is a common terminology of genotype and phenotype. They will be explained by using DNA and biology as an example. Every cell in a organism's body contains DNA, and it is built up of many genes. The DNA as a whole defines the entire recipe of the organism from the first second it is created to its very end. The DNA is the Genotype because it is the recipe, or the instructions, of the creature. Phenotype on the other hand, is the organism itself. Explained in more detail, the phenotype is how the organism; looks, behaves, develops and so on, in other words its traits. The genotype is the input, the phenotype is the output. Genotype is the recipe, phenotype is the cake.

It is also important to note that the phenotype is not determined by the Genotype alone. The environment also effects the phenotype. After all, inseminating a cat embryo into a dog yields no cat and a cat raised by a dog is likely to behave differently from one raised by a cat.

In nature, genotype to phenotype mapping is done implicitly and indirectly as genes behave in relation to each other. Genes may even suppress other genes so that they are not activated any more. This can be very useful for development and is called methylation [36]. In this process enzymes attach to genes rendering them redundant. This is possible because the change happens in the genes, any descendant of the cell where methylation has happened will copy the genes with the enzyme included. This differs from mutation because methylation is intended. This is one of the processes that is utilized in gene regulations.

Development in nature is very implicit [51] in its manner. In artificial evolution we are free to use methods that are not implicit or explicit. In this thesis both methods that are implicit and explicit will be tested.

2.1.3 Stem-cell

The human body contains about 10^{13} cells, which contains more then 200 distinct types of cells [20], but at our earliest stage of creation, humans are created from one cell called the zygote. The zygote splits, goes through several stages, and after several days, it turns into the blastocyst. The blastocyst consists of an outer cell layer and an inner cell layer. The inner layer cells are also called embryonic stem-cells and are pluripotent, which means they can turn into any other type of cell. Throughout the development of the body there are other types of stem-cells, which have varying levels of potency. This means that the different stem-cells have a different set of daughter cells they can turn into. What is common between stem-cells is they all exhibit the two following properties.

- **self-renewal:** the cell has the ability to retain itself. This can be done in two ways, by splitting into two cells where one is the same cell as the original called the mother cell and the other is a differentiated cell called the daughter cell. Alternatively, it can split into two differentiated cells and another cell of the same type as the mother cell splits into two identical cells.

- **potency:** The ability to turn into other types of cells. Different types of stem-cells have different level of potency. Ranging from totipotent, which have the ability to turn into the complete organism to unipotent, which only have the ability to turn into one other type of cell.

Stem-cells are found also in the adult human body and serve to maintain and repair damaged tissue. In other animals, stem-cells can be found in the adult that has very high level of potency. One such example is the salamander that has the ability to turn stem-cells into higher potency stem-cells in order to regenerate limbs. Stem-cells were first proven in 1960 [40, 41] and this discovery moved stem-cell research into the very active and controversial field it is today. This was marked well when Science in 1999 named stem-cells as the breakthrough of the year [42]. This was due to several breakthroughs, such as discovering that adult stem-cells have potential to turn into more then one form of cell (multipotent)[42]. Also a method for halting the development of embryonic stem-cells[42] and more was discovered.

2.2 Open-ended evolution

Open-ended evolution is a bit of a hot topic within the science community, both among biologists and artificial life researchers. In 2000 Artificial Life held an open forum in order to draw up a list of open problems to help guide the research community [28]. Many of the topics discussed related to open-ended evolution.

What is open-ended evolution?

There is a little bit of a disagreement about what is necessary for open-ended evolution and also the definition of what is open-ended evolution [31]. Much of the scientific community uses the term synonymously with complexity growth in organisms. Some even argue that some complexity is necessary, but is not the driving force [30]. What the community seems to agree on is that a system with open-ended evolution has the ability to continuously produce novel organisms and it is this definition that was chosen for this thesis.

2.3 Complex systems

This thesis falls under the theme of complex systems. Before one understands what is a complex system, it is useful to examine what complex means. Complex is defined by the Oxford online dictionary as:

> 1: Consisting of many different and connected parts:
> 'a complex network of water channels'
>
> 2: Not easy to analyse or understand; complicated or intricate:
> a complex personality
> the situation is more complex than it appears
>
> *Oxford online dictionary*

What is meant by different but connected parts? In comparison, could not a simple system also be built up by parts? [34] The keyword is therefore connected. By connected, they mean that complexity arises from parts that interact with one another in a manner that is not apparent from the parts. This is the notion of emergence, which is in many ways the opposite of reductionism. Reductionism is a scientific method that in order to understand a system tries to divide the system into as small parts as possible and tries to understand the small parts on their own. A system is merely the sum of its parts. Although reductionism has helped science understand many systems, when a system is highly interconnected it is no longer a useful tool[36]. Complexity is simply more than the sum of its parts. As complexity is "not easy to analyse or understand" it becomes hard to define it in simple and general terms, it might therefore be best to explain it by examples.

2.3.1 The brain

One such example is the brain, which is an interesting topic of on its own, but it is also a very good example of a complex and interconnected system. As a computer scientist it becomes tempting to explain the brain in a simplified manner similar to processors, with hertz, latency and memory. Such models are in fact made and used, (see Figure: 2.1), where you can see the human mind abstracted into something much like a computer. They can be useful in predicting human behaviour but the abstraction makes it lose the very essence of what makes the brain complex. In reality the brain is not so centralized and the model falls short in explaining why and how the brain works.

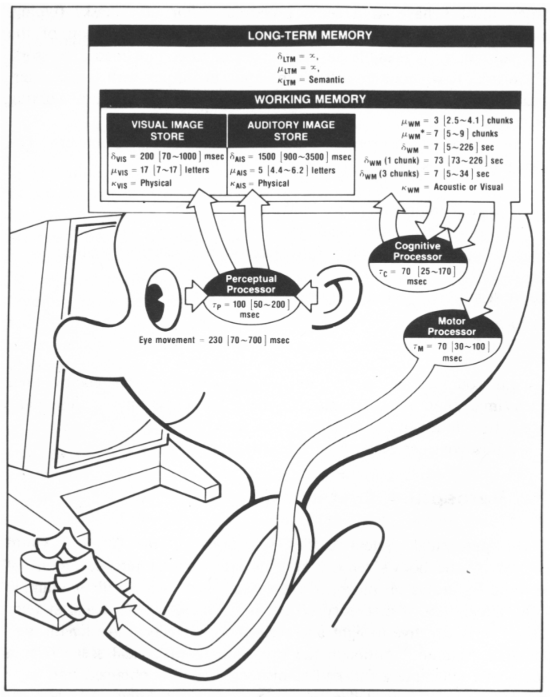

LONG-TERM MEMORY
$\delta_{LTM} = \infty$,
$\mu_{LTM} = \infty$,
κ_{LTM} = Semantic
WORKING MEMORY
VISUAL IMAGE STORE
δ_{VIS} = 200 [70~1000] msec
μ_{VIS} = 17 [7~17] letters
κ_{VIS} = Physical
AUDITORY IMAGE STORE
δ_{AIS} = 1500 [900~3500] msec
μ_{AIS} = 5 [4.4~6.2] letters
κ_{AIS} = Physical
μ_{WM} = 3 [2.5~4.1] chunks
$\mu_{WM}{}^*$ = 7 [5~9] chunks
δ_{WM} = 7 [5~226] sec
δ_{WM} (1 chunk) = 73 [73~226] sec
δ_{WM} (3 chunks) = 7 [5~34] sec
κ_{WM} = Acoustic or Visual
Cognitive Processor
τ_C = 70 [25~170] msec
Perceptual Processor
τ_P = 100 [50~200] msec
Motor Processor
τ_M = 70 [30~100] msec
Eye movement = 230 [70~700] msec

Figure 2.1: Image example of the Model Human Processor, where the mind is simplified into a machine. Source: [37]

In reality the brain is built up of two types of cells, neurons and glial cells [38]. Glial derives from the Greek word for glue, and is a type of cell that seems to perform support roles like providing structure, removing dead material and much more. The more interesting cell type is the neurons, which can be split in 3 parts [36]

- The soma, which is the cell core and body.
- The dendrites, which stretch outwards and is the neurons source of incoming signals.
- The axon, which also stretch outwards and is the neuron's way for transmitting signals to other neurons.

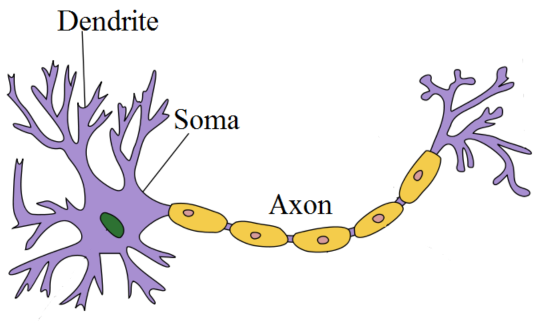


Figure 2.2: A neuron with soma dendrite and axon marked. Source: [73]

A neuron can either be firing or not firing. When a neuron is firing it means that it is sending signals through the axons to other cells, and in order to fire, the neuron needs enough signal from its dendrites. One way to think of a neuron is to think of it as a valve that releases only if there is enough pressure. A single neuron in itself does not do much, but it is through its interactions with countless of other neurons the brain can perform complex behaviour like recognizing this pattern as writing and translating the words you are now reading into thoughts and meaning.

2.3.2 Ant colonies

Ant colonies and other insect colonies are a fine example of a complex system. They behave in many ways similar to the brain. The colony as a whole can be seen as a hive mind, but before one can understand that, one needs to know how ant colonies work. An ant itself is quite simple. It follows its simple genetic imperative to march around in seemingly random patterns doing simple tasks like gathering leaves, food, or other materials for the hive by scrounging along the forest floor. Yet together, the entire hive can fend off intruders, gather resources and build large hives consisting of many deep hallways as seen in Figure: 2.3. The deep hive is often used as a breeding ground where the temperature is controlled using heat from decaying nest materials [36].

Another example to consider is the fire ants, which struggle if it is alone in water, but when together they congregate and form rafts to increase their buoyancy [58]. The ants are not smart enough to consciously make these decisions on the basis of the hive, rather they communicate locally and respond to local interactions between other ants or their senses.

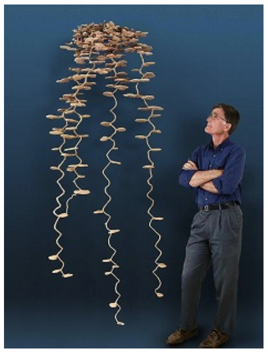


Figure 2.3: Casting of an underground ant nest showing deep structures, Source: [57].

2.3.3 Central themes and Common properties

Two short examples of complex systems have been given. Besides these two, there are many more that could be mentioned, like the immune system, the economy and the internet. Common properties of these examples mentioned is emergence [34], but also properties like signalling and information processing and adaptation. These topics will be reviewed later in this subsection. [36].

Emergence

Emergent complexity is when simple parts work together to form on a larger scale more complex behaviour. A formal definition of emergence can be found in [19] and goes as following. "Emergence is a non-trivial relationship between the properties of a system at microscopic and macroscopic scales. Macroscopic properties are called emergent when it is hard to explain them simply from microscopic properties." The brain and ant colonies are good examples of this. For example how the fire ants generate buoyancy above their sum by forming rafts together. The opposite is also something that can be seen to occur. When complex systems interact and through their properties self-organize, this is called emergent simplicity or self-organisation. One good example can be found in magnetism. Materials such as iron consist of many small magnets called spins. Normally these small magnets point in random directions such that their magnetic fields cancel each other out. The higher the temperature of the material, the more random movement in the molecules, which effect the spin and causes them to behave randomly. If the temperature is low enough, the spins line up such that they all point in the same direction, which as a result emerge magnetism. This happens because spins that point in opposite directions repel each other [35]. If enough pressure from other spins are in alignment, then spins that are not, will be forced into alignment through this repelling force from the ones that are aligned.

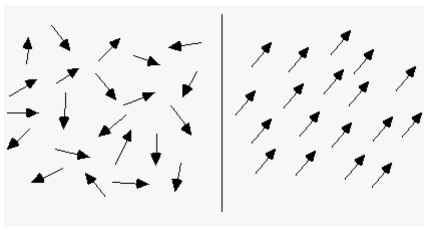


Figure 2.4: How magnets work. Left side shows a magnetic field with random spin. Right side shows a magnetic field where spins have aligned themselves. Source: [35]

Many systems exhibit this emergent simplicity where the chaotic nature of the smaller parts becomes unimportant in the whole. Another such example includes how the stellar body Earth moves around the sun in a neatly formed orbit, while ignoring the countless complex actions of its inhabitants [34].

Complexity

A property of a complex system is also its complexity. There are sadly no commonly accepted definition on how to quantify complexity. Bar-Yam proposed that complexity is the amount of information necessary to describe a system [34], but this was not entirely accepted by the community. How to define and quantify complexity is still an ongoing discussion as claimed by Mitchell in [36]. Just this year, the journal Artifical Life, included an article proposing a new combination method [59]. The method combines Kolmogorov Complexity and Minimum message length. Kolmogorov Complexity, states that the complexity of a string is the shortest program that can be used to describe a given string [59]. In other words, how hard it is to compress the information. This can abstract nicely to mean Bar-Yams definitions of describing a system. The minimum message length follows a similar notion, as it considers the increased information added to the output compared to the given input [59]. The new method proposed uses a combination of the two where Kolmogorovs method is used to define the quantity of complexity of an output. The input is considered in relation to its output using a minimum message length inspired method. If the community will accept such a definition or not remains to be seen. Therefore, this thesis remain at the description presented at the start of this section.

Processing

Complex systems have a tendency to contain some form of information signalling and processing, but not in a traditional sense. In a complex system signalling is very often highly distributed. The brain is a good example of this. It is considered the central control mechanism for the body and is our processor, but the brain itself is highly distributed. There is no single neuron that is the authority over a given action, but rather a network of neurons. This becomes quite clear when we consider that minor damage suffered from brain-surgery, accidents or tumours do not normally remove functionality [35].

Adaptation

Adaptivity is often a property of complex systems, that they to some degree react to input or environment. The brain clearly has some form of adaptive reaction to damage as mentioned above. It is also designed to make us more adaptive to our environment by learning from it. Much like humans, ants create structures and a home to protect themselves from the environment, by shaping the very environment they live in. The other complex systems mentioned, but not explained, also include some level of adaptation. The immune system has many adaptive methods to fend off an infection. One such example is how the first line of defence, the white blood cells or the lymphocytes act [36]. Normally these cells float around the blood not doing anything, but through special receptors on the cell body, they will react when encountering invading bacteria. These receptors vary from cell to cell such that each cell detects a set of invaders. If invaders are detected, the white blood cell will release antibodies and also start to divide and replicate at an increasing rate. One daughter of the white blood cell will also remain and remember the detected invader in case it is encountered again. Suffice it to say, the notion of adaptivity relates very nicely to the measure of complexity between input and output. Also adaptation has to do with the resilience of the system. This is very apparent in self-organising systems. If we were to randomly flip the spin of a magnetic particle, the pressure from surrounding magnetic spins will quickly bring the flipped spin back in line [35].

2.4 Cellular automata(CA)

Now that we have looked at some natural complex systems and their properties, a designed complex system might provide some contrast. A great example of such a system would be Cellular Automata. Here the rules are purposely simple, but the interaction is greatly complex. Cellular automata is an architecture first devised in 1940 by John Von Neumann and Stanislaw Ulam. Von Neumann used it in order to examine if a machine is capable of replicating itself. The work was published finally by Arthur W. Burks in 1966 [1].

CA is an idealized version of a complex system, and contains massive parallelism and distributed computation [2]. CA is an architecture, which consists of a grid of 1 or more dimensions(not counting time). Every cell in the grid can be in 2 or more states. The cells change state discreetly by considering on its own state and other neighbouring cells state. Normally all possible combinations of neighbour and self states are mapped out and put into a table called a transition table(TT), which serves as a look-up table for the cellular automata. There are alternative ways to specify cell behaviour, which we will come back to in the later sections.

The simplest form is named elementary cellular automata and is a 1 dimensional two state cellular automata. Here a cell uses its two adjacent neighbours and itself to make up its TT. It was extensively examined by Stephen Wolfram [4]. Most notably in [5] he showed that elementary cellular automata fit into one of 4 classes. Examples of behaviour in these 4 classes can be seen in Figure: 2.5.It might be important to note that CA is classified based on its behaviour on most initial condition. Therefore there exist in all classes initial conditions that are exceptions to the nature of the class.

- The 1st class covers CA that settle quickly into a unique homogeneous state, which is largely independent of the initial condition.
- The 2nd class covers CA that settle into stable or short cyclic states on a local level. The states are built up of a set of simple structures.
- The 3rd class covers CA that behave in pseudo-random structures. CA in this class have long loops of states that they go through and can be called chaotic. This class is very sensitive to its initial condition such

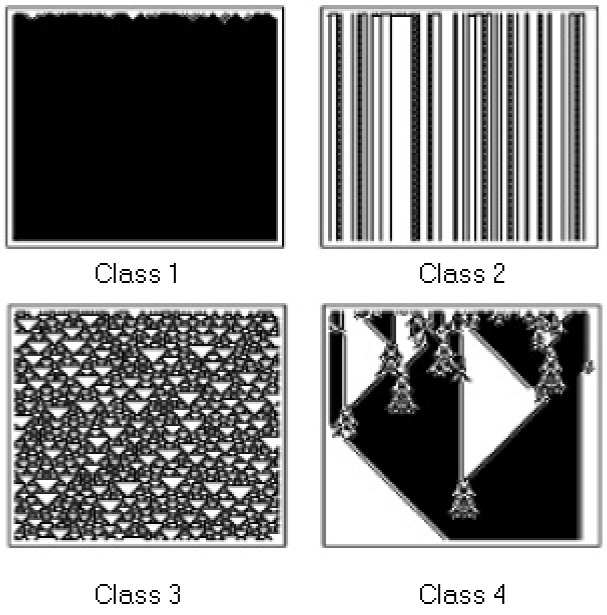


Figure 2.5: Examples of class 1-4. Acquired from [18]. Reformatted and relabelled to fit page.

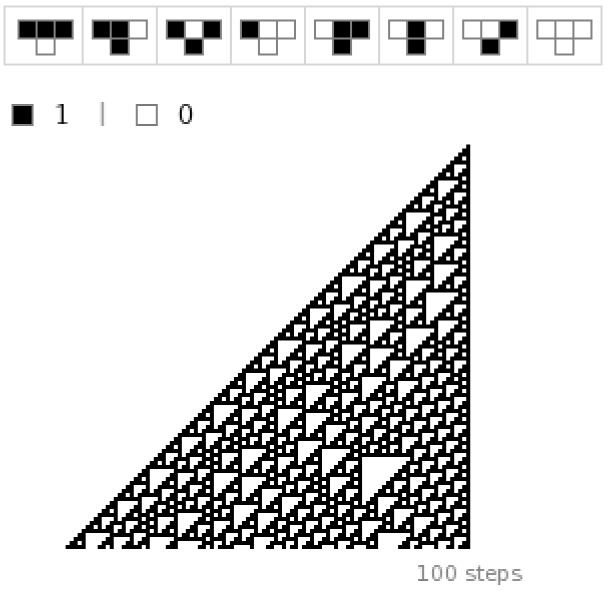


Figure 2.6: Rule 110 with transition table, which is proven to be computationally universal.(acquired from Wolfram Alpha)

that the same rules provide different results depending highly on the initial condition [18]. Members of this class can be used as random number generators [16, 44].

- The 4th and last class covers CA that are of a complex nature. Here cells can over time have global interaction. This class seems to appear most commonly between class 2 and 3 [18]. This class of CA is the truly interesting class and will be elaborate further on it.

Wolfram speculated that some members of the 4th class have the capability of universal computation. One rule was proven in 2004 when M. Cook presented proof that rule 110 (Figure: 2.6, which is a member of the complex class), is capable of universal computation [7]. The proof was devised by showing that rule 110 can emulate a cyclic tag system, which has previously been shown to be computationally universal [8, 9], and therefore it follows that rule 110 must also be computationally universal.

The CA discussed so far is 1-dimensional CA. This thesis will mostly focus on 2-dimensional CA and moving from one to two dimensions seems like a big change, but when it comes to the behaviour of complexity and classes CA behaves much in the same manner[18]. One thing that does change is the neighbourhood scheme. In two dimensions there are two more common schemas, Von Neumann neighbourhood and Moor's neighbourhood, which is shown in the Figure: 2.7.

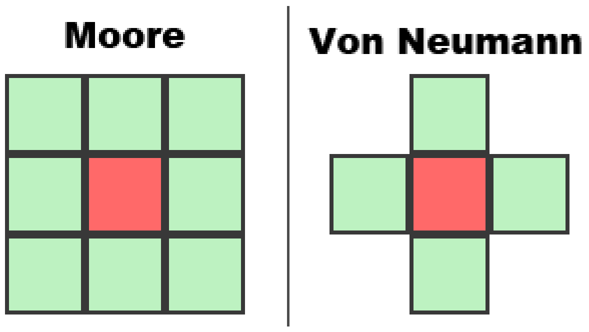


Figure 2.7: Illustration of different neighbourhood schemes, the red cell is changed depending on the values in the green and red cells.

Edge of chaos and λ(Lambda)

The edge of chaos is a controversial theory that was proposed by Christopher Langton in [26]. It states that CA with capability for universal computation is most likely found at the phase transitions between class 2 and 3 behaviour. He also proposes a function for identifying genotypes with ability for computation. He does this by calculating a value called the λ. On a cellular automata with K distinct states and a neighbourhood of N cells, the λ is found by firstly picking an arbitrary state from the set of unique states possible in the CA and call this state the quiescent state. Count the number of transitions to the quiescent state and call this value n. Calculate λ by the following formula:

$$\lambda = \frac{K^N - n}{K^N}$$

In practice the values of K^N and n can be a bit comprehensive to find. As K^N scales quickly for larger numbers of K and N. Luckily, they can be quite accurately estimated by testing uniformly and randomly over the possible K^N states. Using this method $\lambda = 0$ where $K^N = n$, then all transitions lead to the same state and the CA is therefore homogeneous. While λ values where n=0 and $\lambda = 1 - \frac{1}{K}$ are the most heterogeneous. Langton then showed that by traversing

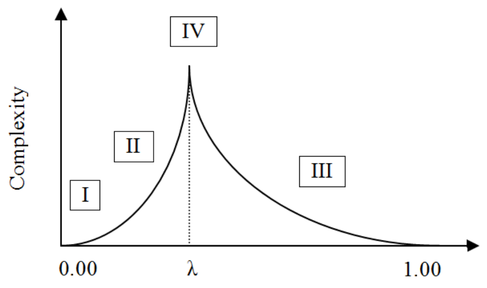


Figure 2.8: Class distribution over lambda values, Source:[60] which is adapted from [27].

through λ values one observes all 4 classes of CA described by Wolfram as shown in Figure: 2.8.

2.4.1 The allure of an alternative architecture

Today the most common computer architecture is based on Von Neumann architecture. It is quite a testament to Von Neumann's genius as he devised both the common and the alternative. Regardless, the Von Neumann architecture comes with certain limitations. One such vital limitation being the Von Neumann bottleneck[17].This bottleneck is the limitation put on the system through the bus that memory, CPU and I/O share. This bottleneck is currently to some degree mediated through the memory hierarchy also referred to as caching.

Moore's law is a well known computer science observation that states that every two years the density of transistors double. As was predicted [21, 22, 23], it can be argued that the law no longer applies, but is being artificially kept by multi-core processors. Having two cores of 1 GHz does not guarantee the same capabilities as one core of 2 GHz. Multi-core processes were implemented due to the problem of heat leakage [24]. It seems, multi-core processors only make the Von Neuman bottleneck worse [25], making the bottleneck problem even more relevant.

On the other hand, memory and processing in CA are, entangled into one system. Since it was shown that CA can be universal computational [7], it must therefore be capable of anything a conventional computer is capable of. CA only communicates locally and therefore takes a long time to do certain tasks, but It is completely parallel and can be simulated effectively across both cores and machines [6].

CA applications

Aside from attempting to replace the conventional computer, CA can be used for several other tasks like, noise cancelling in image processing [43], or how rule 30 can be used as a random number generator [44].

Another application more related to this thesis is the study of biology. Local interaction giving rise to global information processing that occurs in nature, such as the examples of the brain and insect colonies. Many of the features found here are also found in cellular automata such as massive parallelism, locality of interaction and simple components [65].

2.5 Evolutionary Algorithms(EA)

A complex system like CA comes with a certain feature of scaling. In order to achieve desired behaviour of the CA, in all but the most simplest versions, a very large search space needs to be explored. This becomes apparent when considering the scale of possible genotypes in cellular automata. The size of a TT or the genotype size(gS) can be calculated by taking the number of states(n) to the power of the number of neighbours(nH) $n^{(nH)} = gS$. Such a genotype can be set up in a number of ways, we can calculate this be taking number of states(n) to the power of the gS $n^{(gS)}$. The two formulas can be combined to be $n^{n^{nH}}$. An elementary cellular automata which was discussed earlier contains a genotype size of $2^3 = 8$. This genotype can be set up $2^8 = 256$ different ways . That is only the simplest form of CA; if we were to scale it up to a simple 2-dimensional 2-state uniform cellular automata with a Von Neumann neighbourhood, one cell needs to consider $2^5 = 32$ neighbourhoods. This means we now have $2^{32} = 4\ 294\ 967\ 296$ (4.29 billion) ways to set it up. The largest CA that was considered in this thesis is a 4-state CA with Von Neumann neighbourhood, which has $4^{4^5} \approx 3.23x10^{616}$. We therefore need some form of algorithm that can effectively navigate the very large set of candidates. This brings us to the topic of this subsection; EA.

Evolutionary Algorithms(EA) are a set of algorithms inspired by evolution. [39] EA is seen as a set of function optimisation algorithms. Much of the their strength lies in that their ability to solve almost any problem without knowing more about the problem than the difference between a good or bad solution. This is done by the programmer designing a unique fitness function for the problem. This is a function that is made to evaluate a candidate solution. EA need to be able to generate new candidate solutions continuously. It does this be modifying previous candidate solutions either by mutation, crossover or both.

Crossover

Crossover is the strategy for creating new novel candidate solutions. It uses two or more parent solutions, splits them up and recombines their genotypes to make a new child solution. There are several strategies for crossover [62], and most strategies vary on how often to split between the different parent solutions. The simplest form is called one point crossover. It simply creates a random point and recombines the child by using one side of the genotype from one parent and the other side from the other parent. Other common strategies are two point crossover and multipoint crossover, which in effect simply formed the child using smaller subsections of the parents. In addition there exists some adaptive methods where the crossover rate change depending on other properties [63, 64]. Crossover alone does cause some genetic material to be irreversibly lost in the process so it is very commonly used in combination with mutation. Some argue that crossover is non-essential as it only provides evolutionary adaptability in specific problem areas [12] and that the true source of novelty is mutation.

Mutation

Mutation is another strategy to create novel solutions. Sometimes it is used alone such as in evolutionary strategies. There are also several strategies for mutation, but most of them simply move, change or flip single bits of the genotype based on some form of randomness [62]. In this thesis, mutation is combined with crossover in a genetic algorithm.

Local Optima

One essential weakness to EA is that they have a tendency to get stuck in a local optima that is not necessarily the global optima. A local optima (sometimes called a local maxima/minima) is a fitness peak where the neighbouring solutions all cause some decrease in fitness [54]. Most evolutionary algorithms are adept hill climbers. If properly balanced, they exhibit some ability to avoid getting stuck in a local optima. Much of this strength derives in the drift and pseudo-randomized population. It is important to note that even if the EA is well balanced, getting stuck in a local maxima is still a relevant problem. The ability to exit local maxima is a very relevant ability when developing desired behaviour in CA. It is a

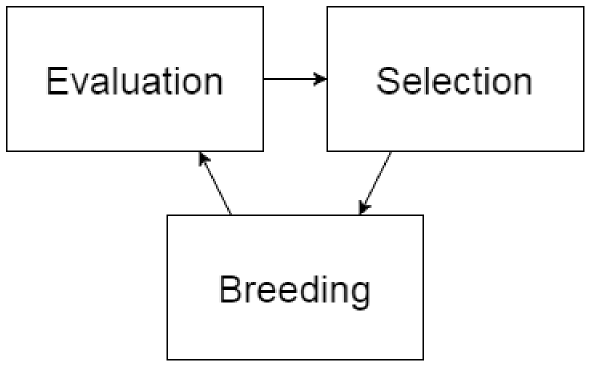


Figure 2.9: Main loop of a genetic algorithm. Created using draw.io

likely problem that one will face when working with CA, as the fitness landscape in CA is rarely a straight line.

2.5.1 Genetic Algorithm(GA)

Genetic Algorithm(GA) is a form of EA and was first investigated and introduced by John Holland [32] in 1975. It is implemented by using a population pool of random candidate solutions. Iteratively the pool is valued solution by solution, and solutions that have a higher fitness value have a higher chance of breeding part of their solution into the next generation. This is why GA is considered to mimic natural selection. Breeding is done by a combination of mutation and crossover. First a new solution generated from part of two other solutions using crossover, then the new child solution may mutate further altering it. The new child solution is then passed to evaluation again as seen in Figure: 2.9. It continuously runs through this loop until a satisfactory result is achieved or the limit of maximum generations is reached.

Selection

There are several ways to select breeders for the next generation. The two most common practices are tournament selection and fitness proportion selection.

Tournament selection picks random candidates from the population, which competes in a tournament based on fitness. The winner of the tournament is selected for breeding (crossover). New tournaments are held until the next population has reached sufficient size.

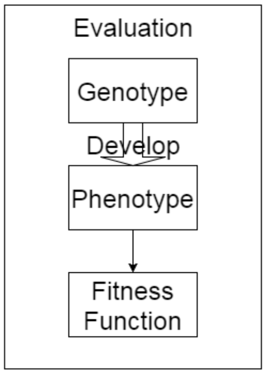


Figure 2.10: Ilustration of how evaluation works on CA in a GA. Created using draw.io

Fitness proportion selection is done by normalising the fitness of the entire previous generation so that the sum of fitness shared by the population totals 1. The population is then sorted by fitness and distributed over a range of 0-1 such that every candidate's fitness is, every candidate before it in the distribution summed with its own. Then the breeders are selected by finding a random number between 0 and 1 and selecting the first candidate with a larger number. This thesis uses the fitness proportion form of selection.

Evaluation

Evaluation with regard to cellular automata becomes a little more complicated when it comes to the evaluation. Normally the fitness function can be applied directly to the genotype. Due to the genotype to phenotype mapping in CA, the phenotype needs to be developed before a candidate can be evaluated by the fitness function as seen in Figure: 2.10.

Elitism

Between every generation all the old solutions are removed. It might be favourable to ensure that the best solution carries on, so that quality of every generation is not lost between generations. In order to do this, normally, the GA utilizes something called elitism. Elitism simply ensures that the best solution(s) of the old generation is retained into the new generation.

Some studies indicate that elitism can be quite vital for GA performance [33].

Stochastic

One problem with genetic algorithm is that it is stochastic, and therefore yields unpredictable results. It is therefore hard to find critical problems that illustrate the strength of GA. This is shown well in [14] where they actively tried to find a problem where GA will outperform hill climbing algorithms. The results on the other hand were not conclusive to a high degree. GA's stochastic nature is one of the properties that makes it and other EA's so interesting. This is because it can come up with very creative, but hard to predict and maybe also hard to understand solutions.

No free lunch

An important theorem in optimisation algorithms is the no free lunch theorem presented in [72]. This theorem states that for any algorithm strength of solving a specific problem, must be paid for in a weakness on a different set of problems. This means that no algorithm is superior to other algorithms, but they can be better only at specific problems.

2.6 Instruction-based approach(IBA)

Normally cellular automata use transition tables as a genotype, but there are other proposed alternatives, and one of them is the instruction-based approach. This approach was introduced by M. Bidlo and Z. Vasicek [10]. The method replaces the TT with a small RISC (Reduced instruction set computing) program that is run in every cell. It gives the advantage of a higher level of abstraction in the encoding and should therefore make the genotype and phenotype mapping more understandable, even if it uses a implicit genotype to phenotype matching. In addition, the method seems to complement GAs very well and in the experiment that was performed in [11]. Here it is compared to the standard transition tables, and showed improved results in regards to chance of finding a complete solution. Moreover, it showed that on some problems it proved to find the solution quicker.

The method of IBA will be both implemented and built upon in this thesis.

2.6.1 Genome growth

IBA was used later in [47], by Tufte and Nichele where they used it together with a implicit version of TT. The implicit TT is done by using TT entries that are generally stated and cover more then one case. Here they grew the program both in IBA and TT by adding more and more rules/entries over the course of evolution. One observation from this paper relevant to this thesis, is that for replication, only one or two IBA instruction was needed to solve the problem.

2.7 Cartesian Genetic Programming(CGP)

Genetic programming is a form of EA where a program is evolved in the same loop of evaluate, select and breed. Normally the genotype or program is represented as a tree graph, where nodes interact with simple operations of values [15]. CGP is much like genetic programming. It differs in that it allows for inactive genetic material that is in the genotype, but does not directly alter the phenotype. It does this by allowing any node to point to a lower node in the graph and as such every node does not need to be connected to a output node. This method gives greater freedom in terms of drift, and should increase the chance of leaving local optima [15].

Self-Modifying Cartesian Genetic Programming(SMCGP)

SMCGP is a further extension to CGP, which allows for the genotype to iteratively change itself. This is done by including in the program special rules that modify the program itself. Within the development of the phenotype the genotype might modify itself while it is developing by example, copying, moving or removing other code in the program. This method was first suggested in 2009 in [48] and further work and testing showed SMCGP's ability to solve a wide range of computational problems [49, 50].

In this thesis the SM method will be combined with IBA. This will be done by extending the rule list of IBA with a number of non-encoding instructions that instead alter the rules of the program itself. This allows for the possibility of creating a genotype that include both encoding genes and non-encoding(self-modifying) genes. Self-modifying genes can be considered as a form of gene regulation mechanism, where different parts of the genome may be active at different stages of development.

2.8 Modularity and Evolvability

Marc Kirschner and John Gerhart defined evolvability as "an organism's capacity to generate heritable phenotypic variation" [68]. In order to have any improvement in evolution, novelty must be produced, but most changes often have negative effect. If there is a capacity to produce many new working solutions, the organism is more likely to be able to adapt to a changing fitness landscape. Nature has managed to increase evolvability with use of modularity, regularity and hierarchy in its systems[66]. The modularity of a solution helps it create working novel solutions and gives advantages in open-ended evolution where the fitness change. Modularity also seems to exist in the gene network and this is proven quite creepily, but also elegantly when Walter J. Gehring [69] caused eye like structures to develop on fly wings that have a single gene activated by genetic engineering. This is quite impressive as there are at least 2500 genes active in the formation of an eye [70].

The notion of modularity and evolvability was put into practice by Ben Kovitz, who experimented with a cascading design [71]. Kovitz revolved different and simple mathematical problems where an input value was modified using simple operations into several output values. The experiment showed increasing evolvability over time.

Taking inspiration from modularity, this thesis aims to make CA more modular with the help of self-modification. This could be done by the self-modification genes creating a genome regulation mechanism.

Chapter 3

Methodology

"Real stupidity beats artificial intelligence every time"

Terry Pratchett - Hogfather

"If you try and take a cat apart to see how it works, the first thing you have on your hands is a non-working cat."

Douglas Adams

In this chapter we will go through the system that has been implemented at a high level and how it was tested. The implemented artefacts or modules of this thesis will be explained in a chronological order. This order is also a bottom up order, which occurred naturally as many of the modules are necessary for other subsequent artefacts or modules.

The system is implemented in Java. The language was chosen simply because the developed system is not directly performance-dependant and Java is the language the author has the most experience with.

CA was implemented with three different genotype representations, namely TT, IBA and SMIBA. TT is the original method where every possible neighbourhood value, the results are explicitly stated. IBA is a newer implicit method where the cell runs a short RISC program. SMIBA is the new method presented in this thesis, which is IBA infused with self-modifying instructions that modify the program itself. Finally, a number of problems were implemented, which were solved using the different representations.

3.1 Cellular Automata

In order to do anything that is intended, a CA framework or engine needed to be implemented. The CA system that was developed supports both one and two dimensional cellular automata with up to 9 different states for cells. The CA can use both Von Neumann and Moore neighbourhoods in the case of 2-dimensional CA. The two aforementioned neighbourhoods can be seen in Figure: 2.7. The CA is uniform such that all cells operate from the same ruleset.

The cells in the CA are bound in a loop such that the edges of the lattice are connected to the other side of the lattice as illustrated in Figure: 3.1. This ensures that every cell has the sufficient number of neighbours. The CA started with transition table as a base genotype representation and the IBA and SMIBA was implemented later.

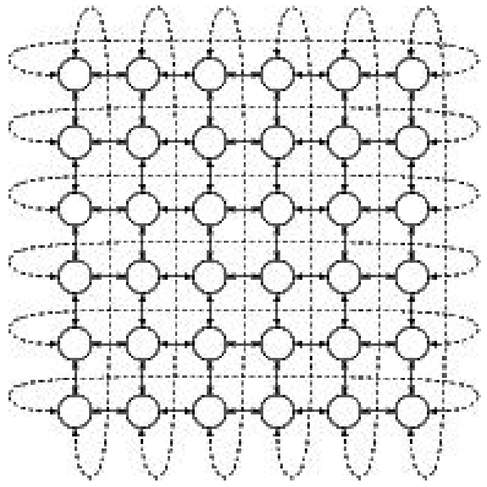


Figure 3.1: Grid that illustrates the edge bounds of the implemented CA. Source: [45] where it was used to illustrate different but similar.

The CA can be set up to write to a .txt-file every iteration, in order to permanently document results. The output file can then be executed through a graphical user-interface to get a graphical overview of the results. This feature was used for inspecting results and to draw many of the figures in this thesis.

3.1.1 Testing

In order to ensure that the CA was working as intended, some testing was performed. First the elementary one-dimensional CA was tested. This was done by applying several known rules and comparing them to output from Wolfram Alpha [75] to see that they were identical. An example of this comparison can be seen in Figure: 3.2.

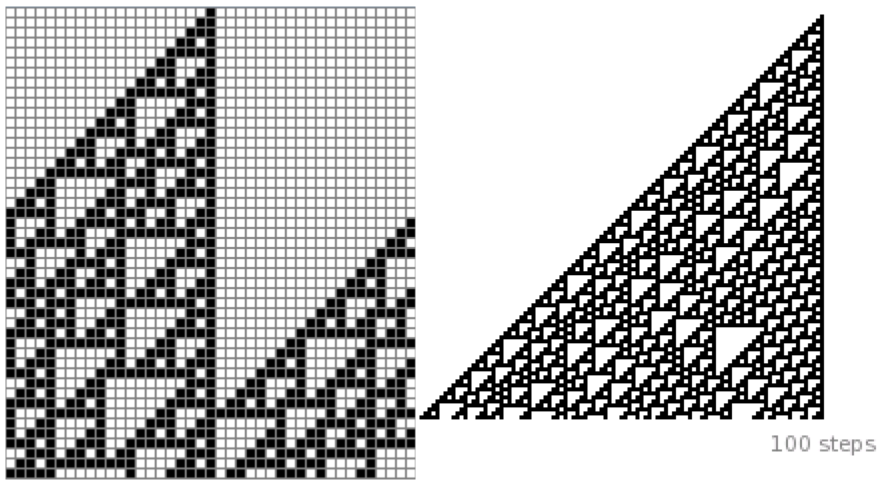


Figure 3.2: Rule 110 taken from the implemented GUI on the left and one taken from Wolfram Alpha on the right. The picture on the left also illustrates the loop mechanism of the lattice.

In order to test that the two-dimensional CA worked, "Game of Life" rules were implemented and the starting board was initialized to the glider gun to see if it operates as intended. First 20 steps of this can be seen in Figure: 3.3. After the first 20 steps the glidergun made several more gliders, which ultimately looped round and crashed into the glidergun, destroying it.

3.2 Genetic Algorithm

Now that there is confidence in the working of the CA, we will move on to the next module, the GA. As mentioned in a earlier section, in CA, the search space of possible rules can be very large. Therefore searching through the entire search space by brute force would take a considerable amount of time. In order to search through this space effectively, the Genetic Algorithm was selected. It is a

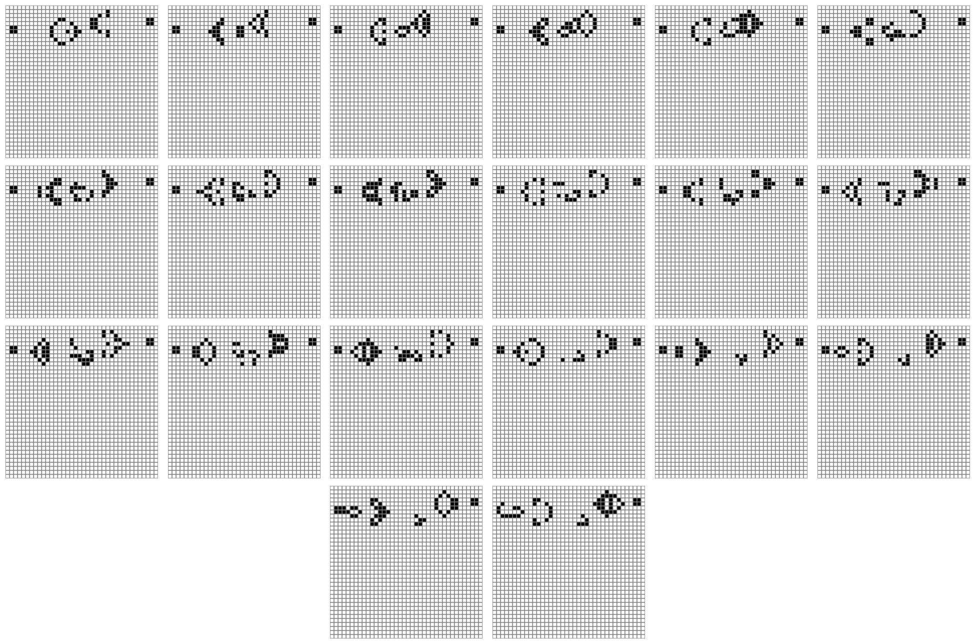


Figure 3.3: First 20 steps of a Glidergun implemented on this system.

robust and capable algorithm for searching through search spaces, but it was also selected because of its inspiration from and similarity to evolution. The GA that was implemented was conventional GA and goes through the following common steps.:

- Evaluation: Testing the population using the fitness function.
- Sorting: Reorder the population into a prioritized list so that better candidates in the population are placed before the worse candidates.
- Normalizing: Distributing the sorted population over a range between 0 and 1 so that the best solution takes up more space of the range than worse solutions.
- Selection: Now that the population is sorted and normalized, breeders can be selected by casting a random number between 0 and 1 and selecting the first candidate in the population that has a larger value.
- Crossover: Creating a new candidate for the next generation of the population by recombining subsections of two breeders' genotype. In this imple-

mentation multi point crossover was used, which means that the parents were split into many multiple subsections.

- Mutation: Every candidate in the next generation is mutated such that every gene in the genotype has a low rate of probability to change. In this implementation, random mutation was used.

3.2.1 Majority Problem($p_c > 1/2$ problem)

In order to test the GA, the majority problem [36], also known as the $p_c > 1/2$ problem, was chosen. This is a problem where the goal is to find a ruleset for the CA that can find out what state n initial condition contain the most of. If the initial condition contains mostly white cells, then the entire CA should turn completely white and if the majority is black cells then the CA should turn completely black. Such a problem may be considered simple for a computer with a centralised system, but in CA nothing has overview of the entire system, therefore it becomes a very hard problem to solve. So hard that there seems to be no ruleset that can solve all possible initial conditions for an 1-dimensional 2-state CA given that the board is of a sufficient length [46].

The fitness function for this problem counted every possible initial condition for a given lattice length. Then, it counted the number of cells that had the correct value/colour at the end of 20 iterations.

Results

Rule found	**Number of times**	**Fitness**
Rule 21	16	150/160
Rule 31	26	150/160
Rule 87	25	150/160
Rule 7	27	150/160
Rule 232	7	140/160

Table 3.1: Resulting solutions found

The GA was executed 100 times on the majority problem with lattice length 5. The chosen population was set to 50, and it ran for 100 generations with elitism. Mutation rate was set at 2% and crossover rate at 10%. With a lattice length of 5,

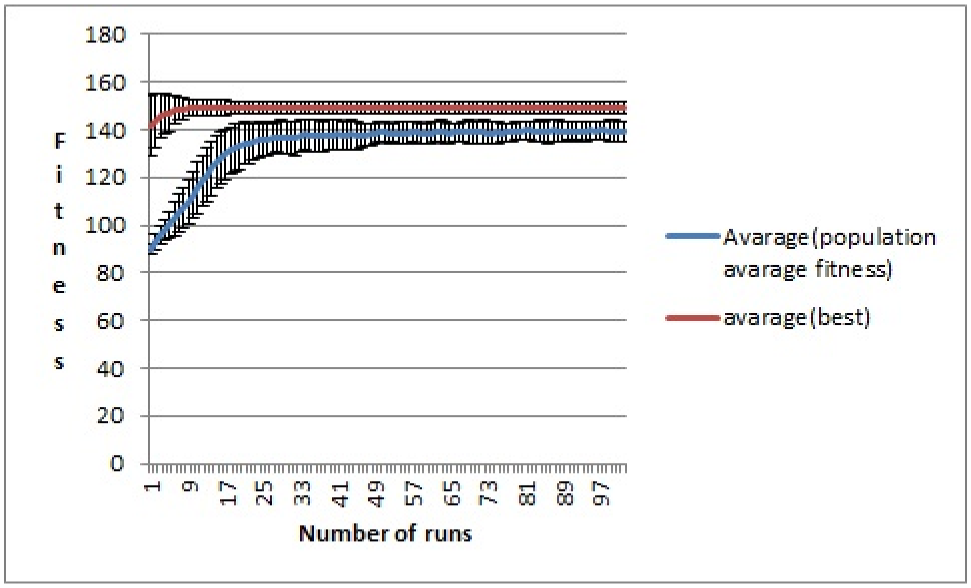


Figure 3.4: Average of 100 GA runs, with Standard deviation on the $p_c > 1/2$ problem

if a rule that solves the problem fully would get a fitness of $(nrstates)^{latticelength} * (latticelength)$ which is $2^5 * 5 = 160$ fitness in this case.

No rule were discovered that got a perfect score, but several were found that got a 150/160 fitness, these were rule 21, 31, 87 and 7. 7/100 times the GA got stuck at a local maximum with 140/160, which is rule 232. Table 3.1 shows the number of times a given rule was the best candidate solution found. Figure: 3.5 shows 4 graphs of the average population fitness and best fitness in the population. These are all cases of executions. two cases were ran without elitism for comparison.

3.2.2 Pixel art Problem(development problem)

The second problem used to test the GA is a well-known development problem. Here the goal is to evolve rules that could develop from a single black cell or the zygote into a predefined cellular automata state, which can be seen as a pixel art image. A few less conventional images were used to test the GA like Luigi and the creeper, but later when comparing results between algorithms and between other literature more conventional images were used. The fitness function for

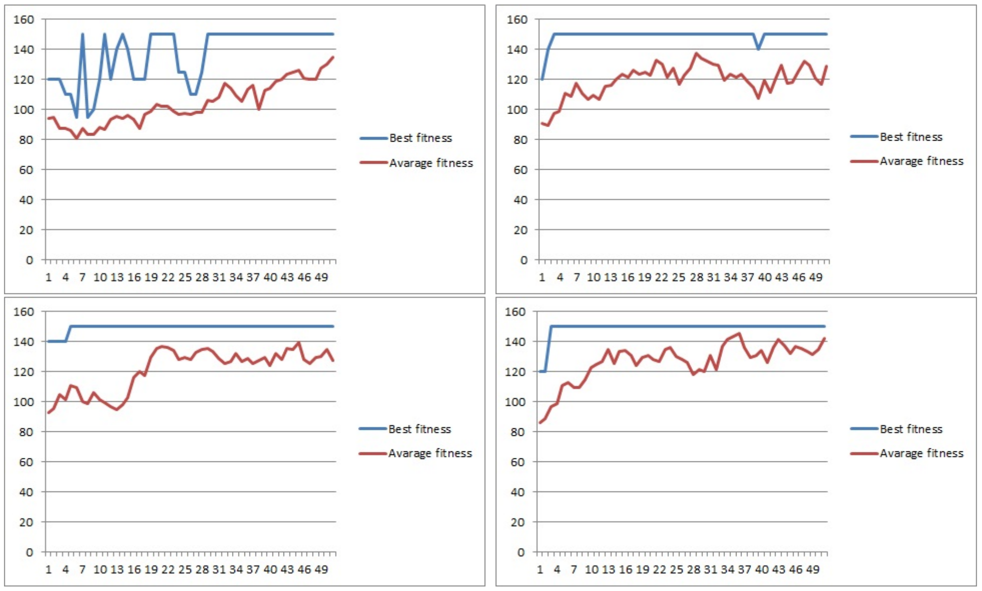


Figure 3.5: 4 examples of GA runs of the majority problem, above two without elitism and the below two with. All had a population of 20

this problem simply iterated the CA 20 steps and matched the resulting CA to the desired CA state and counted the number of correct cells. 4 pictures of different difficulty were identified and chosen, as can be seen in Figure: 3.6. These are named the Luigi, mushroom, creeper and creeper-reduced structures. Deeper results of this problem were explored in the result section, where other target images were also used. These other images are not featured in this section because they were not used to test or balance the GA. Using the same images for testing might result in over-fitness to these structures. All the structures are encoded with the same color corresponding to the same value in the CA. The same encoding is used throughout the thesis, this encoding can be seen in Figure: 3.6.

Luigi structure results

The Luigi structure with a lattice width and height of 16 has a maximum fitness of $16 * 16 = 256$. It was executed 100 times with a population of 50 for 100 generations with elitism. Mutation rate was set at 2% and crossover rate at 10% with Von Neumann neighbourhood. The search space is $4^{4^5} = 4^{1024} \approx 3.23x10^{616}$, which is a very large space and, it may be difficult to find a perfect

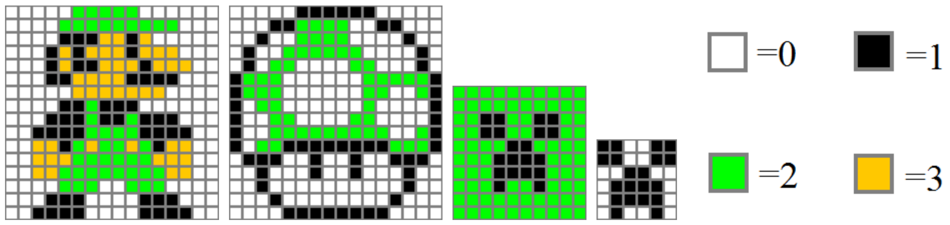


Figure 3.6: Left to right; Luigi 4 colors 16*16 lattice, mushroom 3 colors 16*16 lattice, creeper 3 color 10*10 lattice and creeper-reduced to 2 color and 6*6 lattice. .

solution. The phenotype space depends on the lattice size and the number of states and is at $4^{256} \approx 1.34x10^{154}$.

None of the GA found an optimal solution. The best solution found had a fitness score of 146/256, this solution you can see in action in Figure: 3.7.

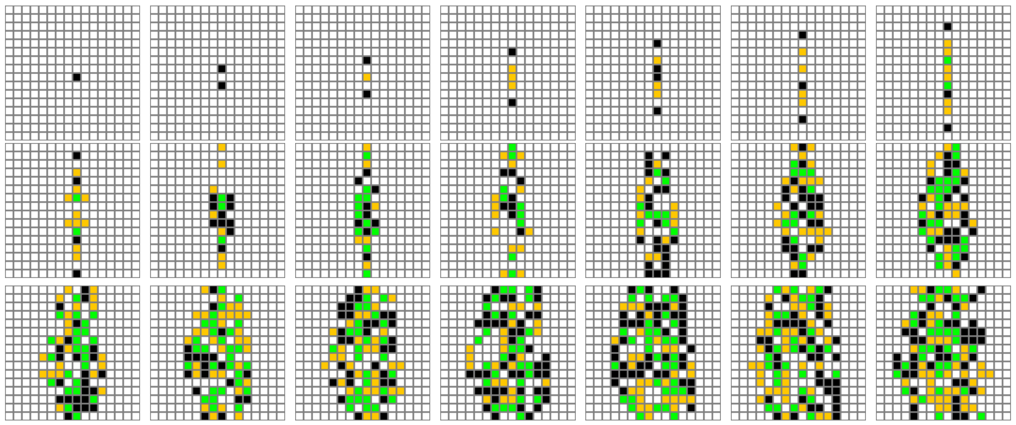


Figure 3.7: The best GA solutions found with 146/256 fitness.

It is interesting how the GA found a candidate solution that utilizes the large white area at the left and right edges of the lattice. Inspecting the other resulting candidate solution show similar strategies.

Figure: 3.8 shows the average of the best fitnesses and also the average of the population's average fitness out of the 100 runs. Figure: 3.9 shows the distribution solutions in form of their fitness value.

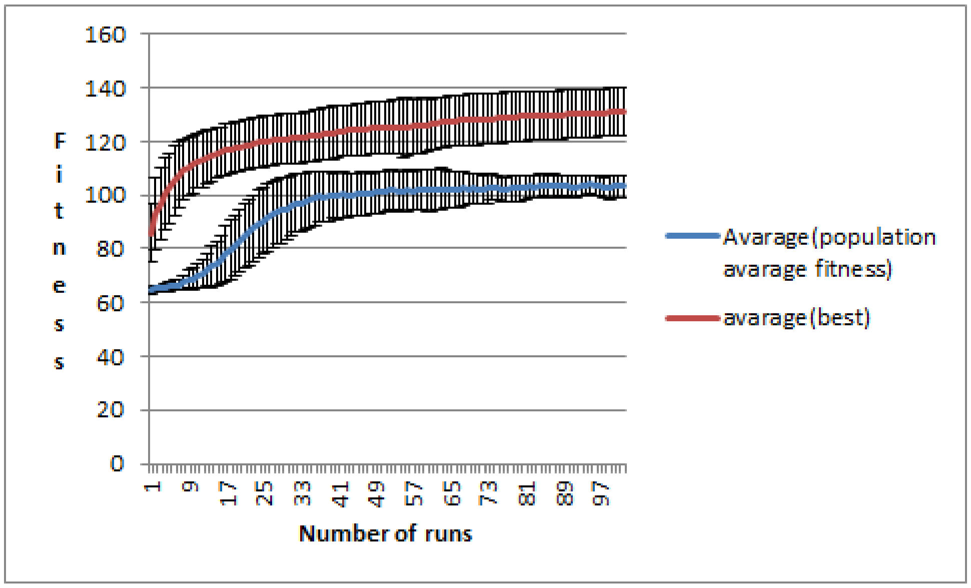


Figure 3.8: Average of 100 GA runs, with Standard deviation on the Luigi structure

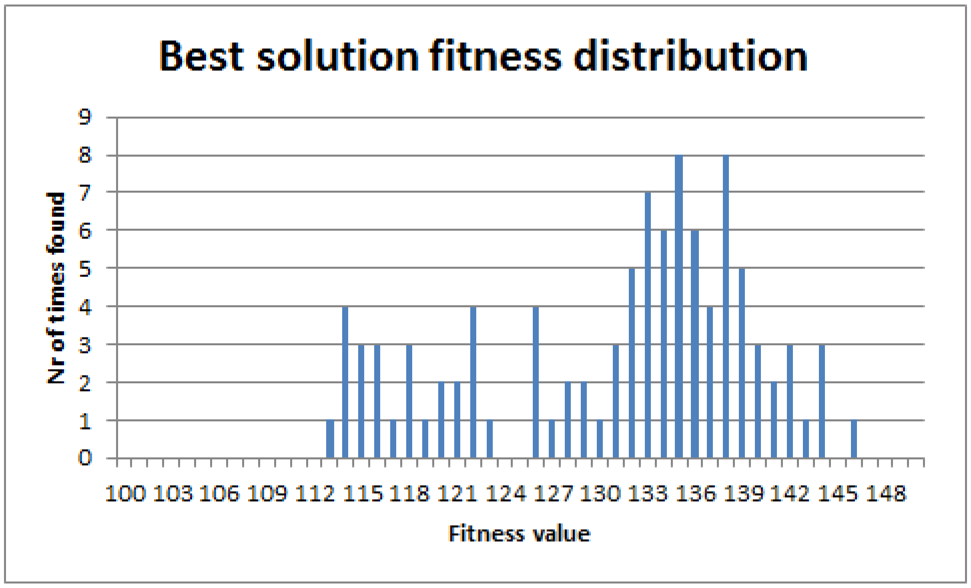


Figure 3.9: Fitness distribution of 100 GA runs, on the Luigi structure. (Graph range compressed from 0-256 to 100-150)

Mushroom structure results

The mushroom structure has a max fitness of $16 * 16 = 256$. It was executed 100 times with a population of 50 for 100 generations with elitism. Mutation rate was set at 2% and crossover rate at 10% and Von Neumann neighbourhood. The search space is $3^{3^5} = 3^{243} \approx 8.72x10^{115}$, which is much smaller than the Luigi structure, which is also true for the phenotype space $3^{256} \approx 1,39x10^{122}$.

None of the runs found an optimal solution, but slightly better solutions on average were found compared to the Luigi structure. The target image has 142 white cells, and some of the candidate solutions utilize this very directly by only changing very few cells as seen in Figure: 3.10.

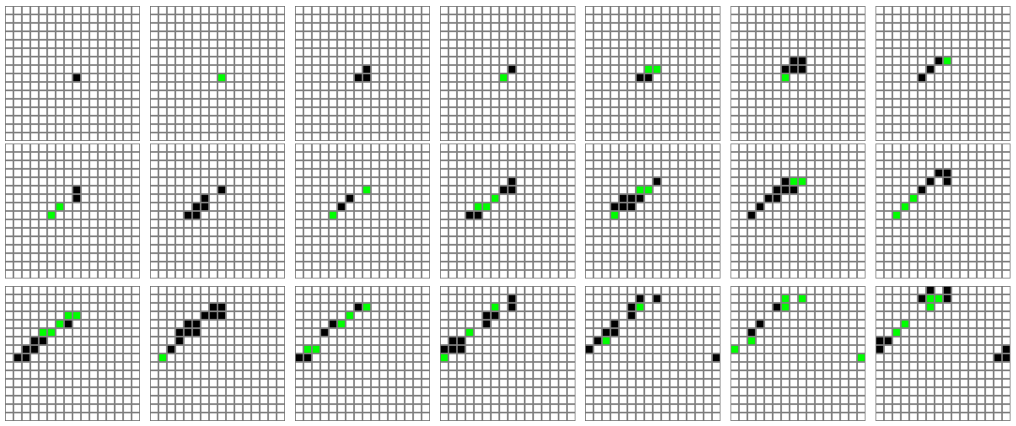


Figure 3.10: One of the best GA solutions found with 152/256 fitness.

Figure: 3.11 shows the average of the best fitnesses and also the average of the population's average fitness out of the 100 runs. Figure: 3.12 shows the distribution solutions in the form of their fitness value.

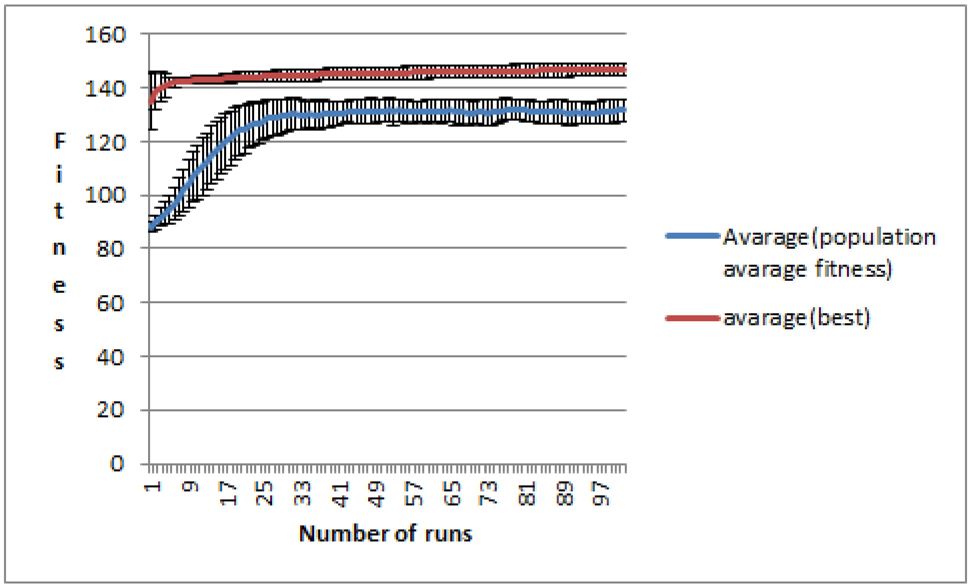


Figure 3.11: Average of 100 GA runs, with Standard deviation on the Mushroom structure

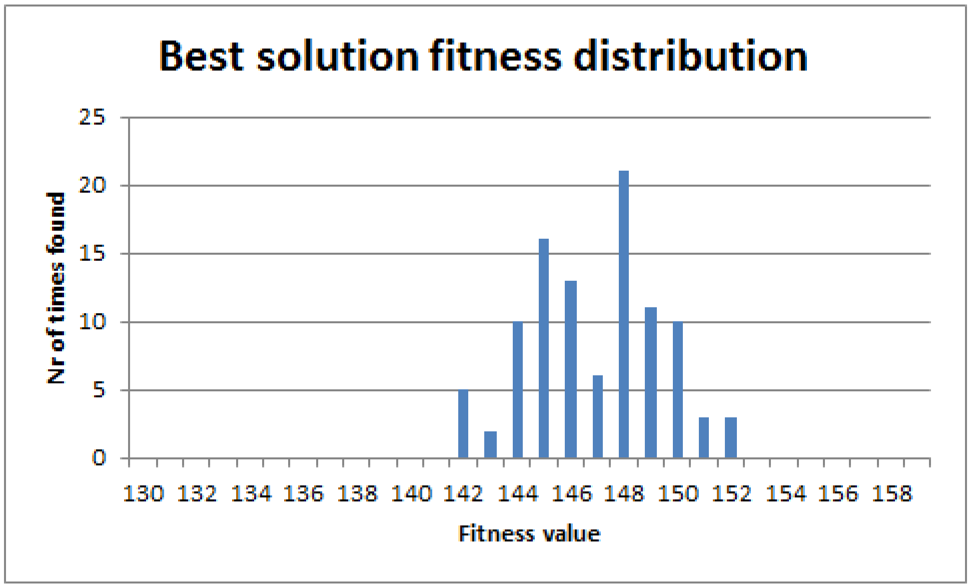


Figure 3.12: Fitness distribution of 100 GA runs, on the Mushroom structure (Graph range compressed from 0-256 to 130-160)

Creeper structure results

The creeper structure is an image taken from a modern game called Minecraft. It has a maximum fitness of $10 * 10 = 100$. It was executed 100 times with a population of 50 for 100 generations with elitism. Mutation rate was set at 2% and crossover rate at 10% and Von Neumann neighbourhood. The search space is $3^{3^5} = 3^{243} \approx 8.72x10^{115}$, which is the same as the mushroom structure, but it has a much smaller phenotype space at $3^{100} \approx 5.15x10^{47}$. The lattice is smaller and the image is simpler so it should be easier to find a solution.

No optimal solution was found, but it did get closer in comparison to the other structures. 7 iterations found candidate solutions with 86/100 fitness. One such example can be seen in Figure: 3.13.

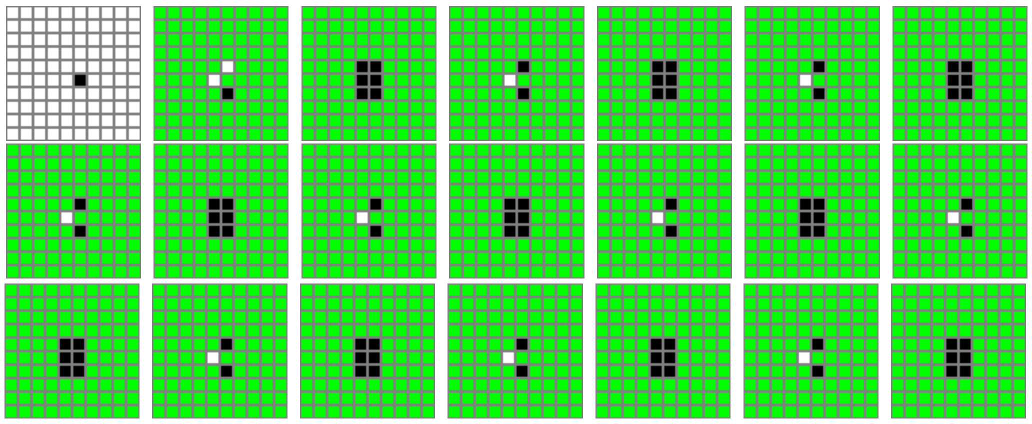


Figure 3.13: One of the best GA solutions found with 86/100 fitness.

Figure: 3.14 shows the average of the best fitnesses and also the average of the population's average fitness out of the 100 runs. Figure: 3.15 shows the distribution solutions in the form of their fitness value.

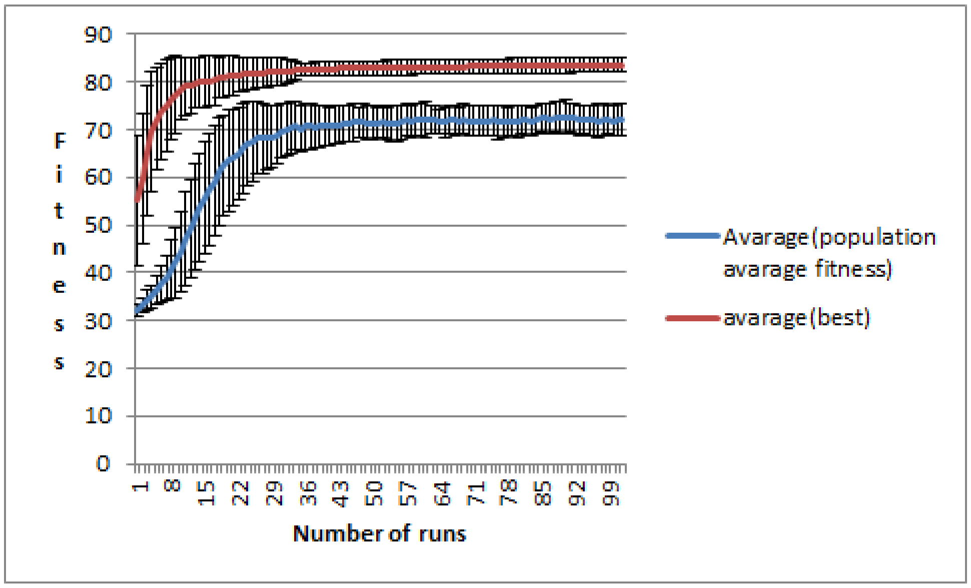


Figure 3.14: Average of 100 GA runs, with Standard deviation on the Creeper structure

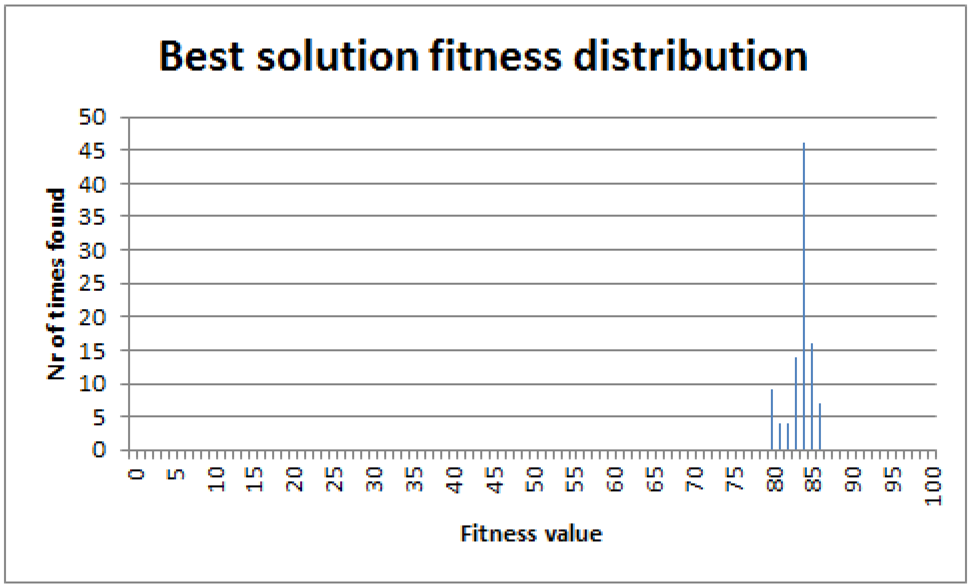


Figure 3.15: Fitness distribution of 100 GA runs, on the Creeper problem

Creeper reduced structure results

The creeper reduced structure has maximum fitness of $6*6 = 36$. It was executed 100 times with a population of 50 for 100 generations with elitism. Mutation rate was set at 2% and crossover rate at 10% and Von Neumann neighbourhood just like the other structures. It has a search space of $2^{2^5} = 2^{32} \approx 4.29x10^9$, which is much smaller in comparison to the other pixel art structures. This is also true for the phenotype space, which is at $2^6 \approx 6.87x10^{10}$. Yet the results show it finds solutions very comparable to the creeper structure (see Figure: 3.18 and Figure: 3.15). The rate at which it approach the solution, is much faster in the reduced structure.

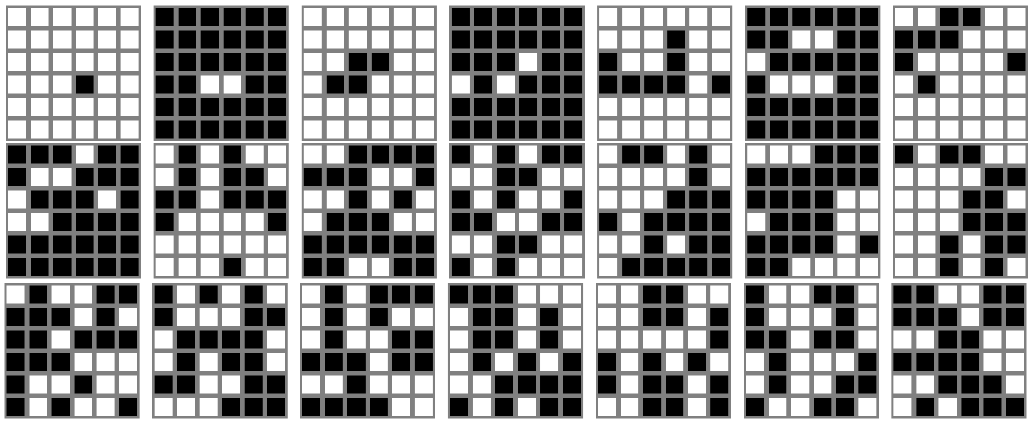


Figure 3.16: One of the best GA solutions found with 30/36 fitness.

Figure: 3.17 shows the average of the best fitnesses and also the average of the population's average fitness out of the 100 runs. Figure: 3.18 shows the distribution solutions in the form of their fitness value.

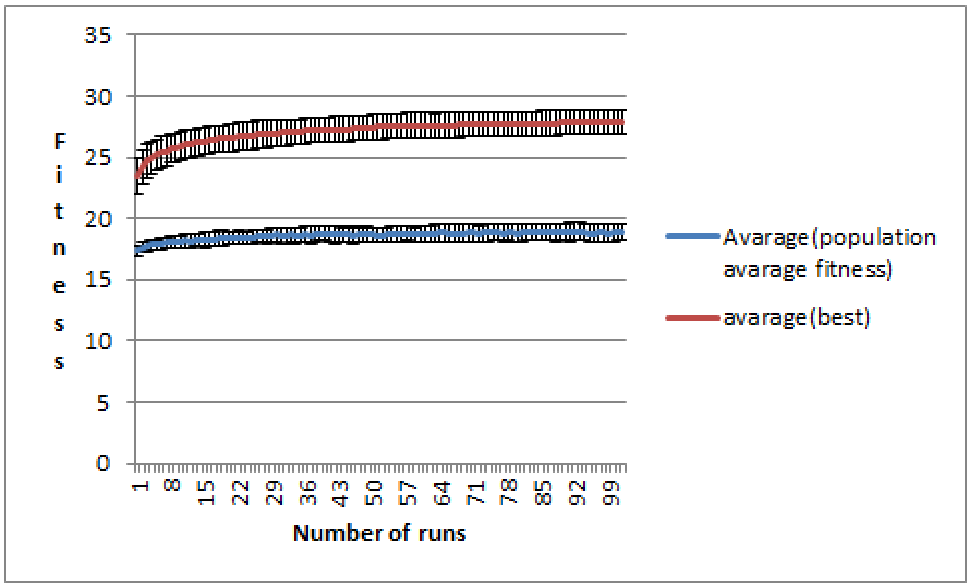


Figure 3.17: Average of 100 GA runs, with Standard deviation on the Creeper Reduced structure

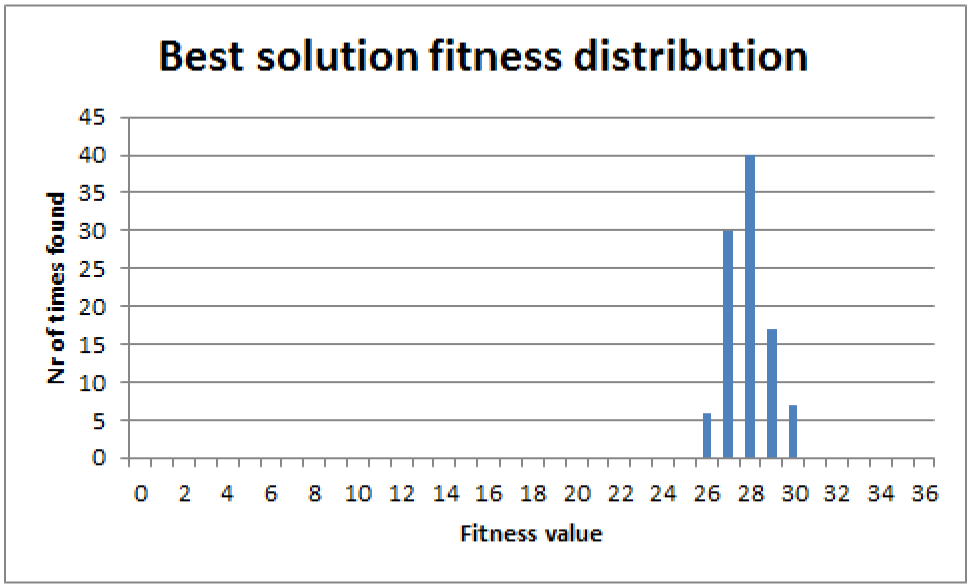


Figure 3.18: Fitness distribution of 100 GA runs, on the Creeper Reduced structure

Creeper reduced structure long run results

The earlier runs underestimate the difficulty of finding a solution to the structures. The fitness function was therefore changed to use the best fitness of the rule between the 5th iteration to the 40 iteration. Iteration 0-4 was skipped because it would require at lest 6 iterations before the CA could spread to the edges. The previous tests were done using a very low population and generation limit. This time the GA ran for 500 generations and had a population of 400, but the same mutation and crossover rate was used.

As expected this test yielded better results than the shorter run. 5 of the iterations found rules with fitness of 34/36 (see Figure: 3.19). None of these five rules settled into a stable image, but instantly went over to a considerably lower fitness state.

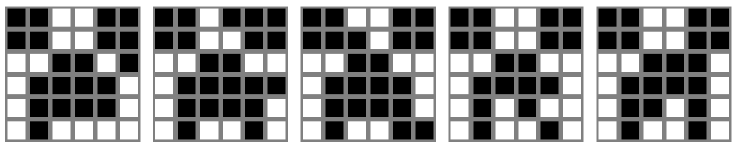


Figure 3.19: 5 of the GA's found a fitness of 34/36. From left to right the image was taken at iteration 18,34,32,12 and 37

Figure: 3.20 shows the average of the best fitnesses and also the average of the populations average fitness out of the 100 runs. Figure: 3.21 shows the distribution solutions in the form of their fitness value.

In general

As the earlier graphs in this section show satisfactory upward mobility, the GA was deemed to be working. Note that the parameter optimisation is outside the scope of this thesis. A little optimisation was nonetheless done. Other values for population, mutation and crossover were tested on the same structures, but with a higher limit on generations. These did not yield better results. Therefore the earlier values of mutation chance at 2% and crossover chance at 10% and 50 population were used in all later experiments. These short tests were all done with using TT as a genotype representation, which means the optimisation, however small, would be in TT favor. This ensures that when comparing TT to the other methods, the GA is not optimised in IBA or SMIBA's favor, which would make the comparisons fairer as this thesis aims to show SMIBA's strength.

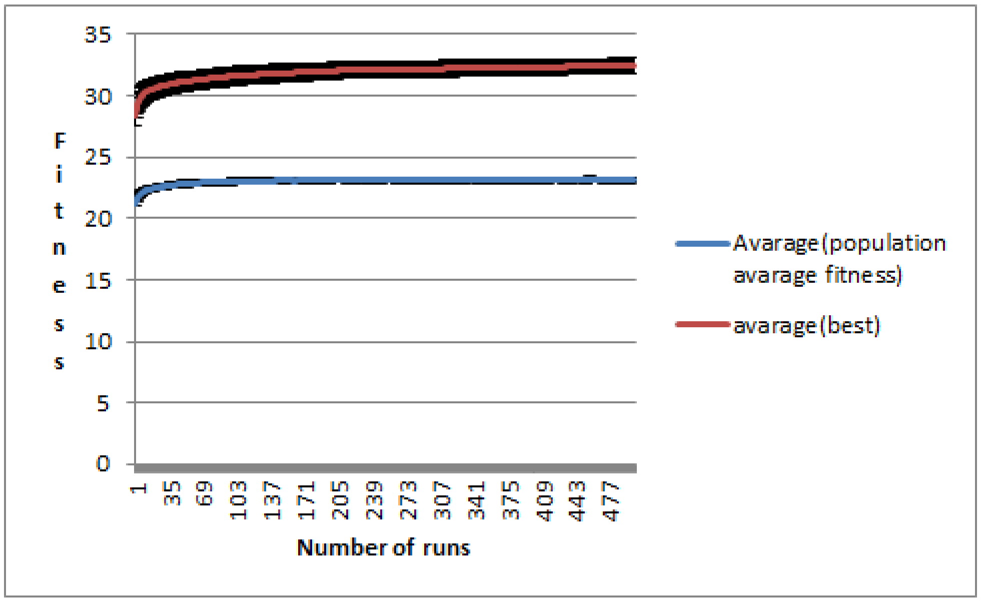


Figure 3.20: Average of 100 GA runs, with Standard deviation on the Creeper Reduced long run structure

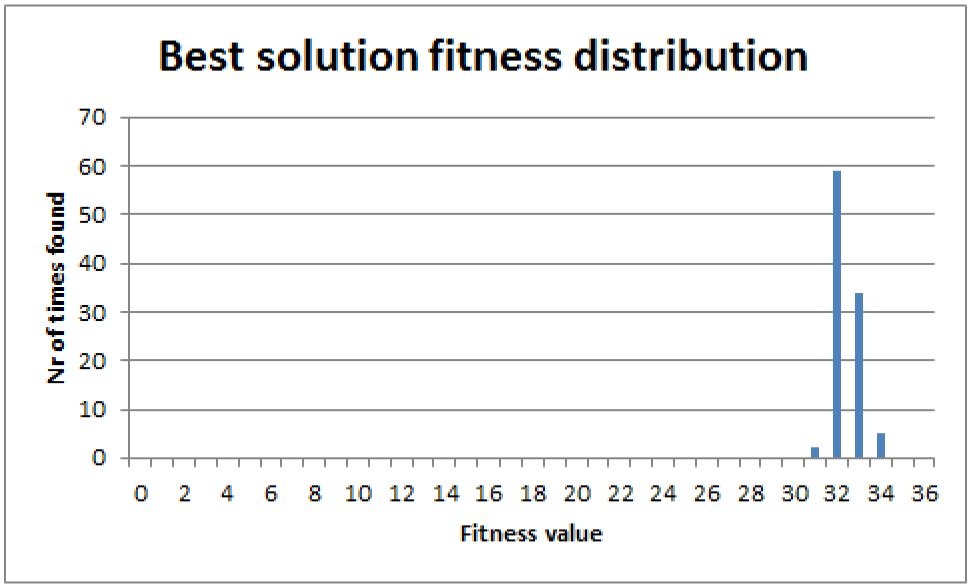


Figure 3.21: Fitness distribution of 100 GA runs, on the Creeper Reduced long run structure

3.3 Instruction-Based Approach

The second genotype representation that, was in line to be implemented, was the Instruction-Based Approach. IBA was implemented by using the same ruleset as [11, 47] with a fixed number of instructions. This ruleset can be seen in Table: 3.2. The instruction set contains many different simple operations for either moving or modifying information. In addition to this, it also contains a No Operations(NoOp) instruction. This empty instruction allows the program to, in effect, use fewer rules if it so wants. The NoOp instruction also provides genetic material that does not alter the phenotype, allowing the values to drift while the program is being evolved. Instructions that modified the information could potentially modify the cell values into values that were outside the scope of the CA. Therefore a modulo operation was applied after instructions to ensure that the CA did not go out of its intended range.

When implemented, the IBA was represented such that every single instruction consisted of 3 values. The first value represented which instruction was to be used. The second and third values represented which cell in the neighbourhood the instruction were applied to, and these values can be seen as parameters to the instruction. All instructions contained exactly 3 values regardless if they do or do not need two parameters to be performed. This can be seen as a plus because, much like the NoOp instruction, the unused genetic material allows the values to drift.

All the instructions were tested individually and inspected to ensure they worked as intended.

3.4 Development Problem

IBA and TT was further tested by the development problem, then the test results were compared to [11, 47]. The same results will also be featured in the Results and Discussion chapter of this thesis. They will also be used to test SMIBA later in this chapter.

3.4.1 Formal Development Problem

The development problem was first used to test out the GA in this thesis. This problem will also be used to test out the different genotype representations in

Instruction	**Operation**	**Description**
AND	$N[i_1] = N[i_1] \wedge N[i_2]$	Bitwise AND
OR	$N[i_1] = N[i_1] \vee N[i_2]$	Bitwise OR
XOR	$N[i_1] = N[i_1] \oplus N[i_2]$	Bitwise XOR
NOT	$N[i_1] = \neg N[i_1]$	Bitwise NOT
INV	$N[i_1] = n - N[i_1]$	Inverse
MIN	$N[i_1] = min(N[i_1], N[i_2])$	Minimum
MAX	$N[i_1] = max(N[i_1], N[i_2])$	Maximum
SET	$N[i_1] = N[i_2]$	Replace
INC	$N[i_1] = N[i_1] + 1$	Increment
DEC	$N[i_1] = N[i_1] - 1$	decrement
SWAP	$N[i_1] \Leftrightarrow N[i_2]$	Swapping
ROR	$LCR \Rightarrow RLC$	Rotate right
ROL	$LCR \Rightarrow CRL$	Rotate left
ROU	$UCD \Rightarrow CDU$	Rotate up
ROD	$UCD \Rightarrow DUC$	Rotate down
NOOP		No Operation

Table 3.2: IBA instruction set. L,C,R,U,D is Left, Center, Right, Up, Down in regard to the cell being updated. n is in this table the number of number of states in the CA

comparison to each other, as it is to be used as a benchmark for these representations it requires a more formal explanation in terms of its fitness function.

Fitness function

The fitness function for the development problem is the same as was used to test the GA. It works in the following manner. Every iteration of the CA the fitness function compares the current state of the CA lattice with a the target structure. This comparison is done cell by cell in the lattice and for every cell that matches the target structure, a point is awarded to the current state. The best value between the states is retained and used as the final fitness value. This means that the greatest fitness a candidate genotype can be awarded is the same as the lattice area.

The structures examined

A number of structures were used and can be seen in Figure: 3.22. All of these problems with the exception of 6a/b are explored in other literature [47, 11, 61].

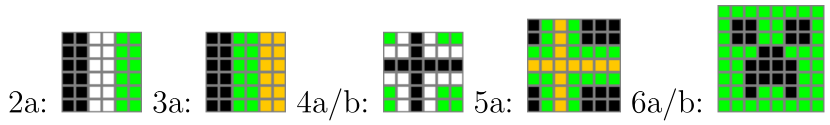


Figure 3.22: left to right; 2a: 3 state French flag, 3a: generic 4 state Flag, 4a/b: 3 state Norse Flag, 5a: 4 state larger Norse Flag, 6a/b: 3 state creeper

Flag structure(3a)

This problem was also used to test if IBA and TT were working by executing a few choice experiments and directly comparing them to the results found in [47, 11]. The first development structure tested is the 4 state flag structure 3a as seen in Figure: 3.22. This time the GA was given an upper limit of 100 000 generations to try to find the solution. This makes it comparable to the results in [47]. In this papir Nichele and Tufte solved the same structure by increasing the number of instructions during evolution, while the test in this thesis's implementation was fixed at 10 instructions. This test also checked every state from iteration 5 to 40, which is 10 more steps then Tufte and Nichele checked. The extra development steps were added in this experiment because the results from this test will be used further in the next chapter. The comparison between results in this thesis and [47] are found in Table: 3.3.

IBA found solutions to this structure by building up columns of diverse colors that moved around the lattice until they hit the target image. Many different ways to build up the columns were used, but nearly all solutions exhibit class 2 behaviour because it ends in a small fixed loop around the lattice. An example of this is seen in Figure: 3.23. One exception was the class 1 CA, seen in Figure: 3.24, where the flag ends in a completely stable state.

TT in comparison often found solutions that build up the flag in a more unexpected manner, as can be seen in Figure: 3.26 and 3.4.1.

The results as seen in Table: 3.3 were not as diverse as expected. IBA barely outperformed TT on this specific structure, but the methods did perform well in relation to [47].

Results	**Our**	**Nichele and Tufte**
Instruction-Based Approach		
Success rate %	45	46
Average	25163	6424
StDev.	28028	1922
Transition Table		
Success rate %	34	19
Average	30676	5002
StDev.	25250	3157

Table 3.3: IBA on the 3a structure comparison. Success rate is out of 100 tries. Average and StDev. is the average and standard deviation of the number of generations the successful runs required to find the solution

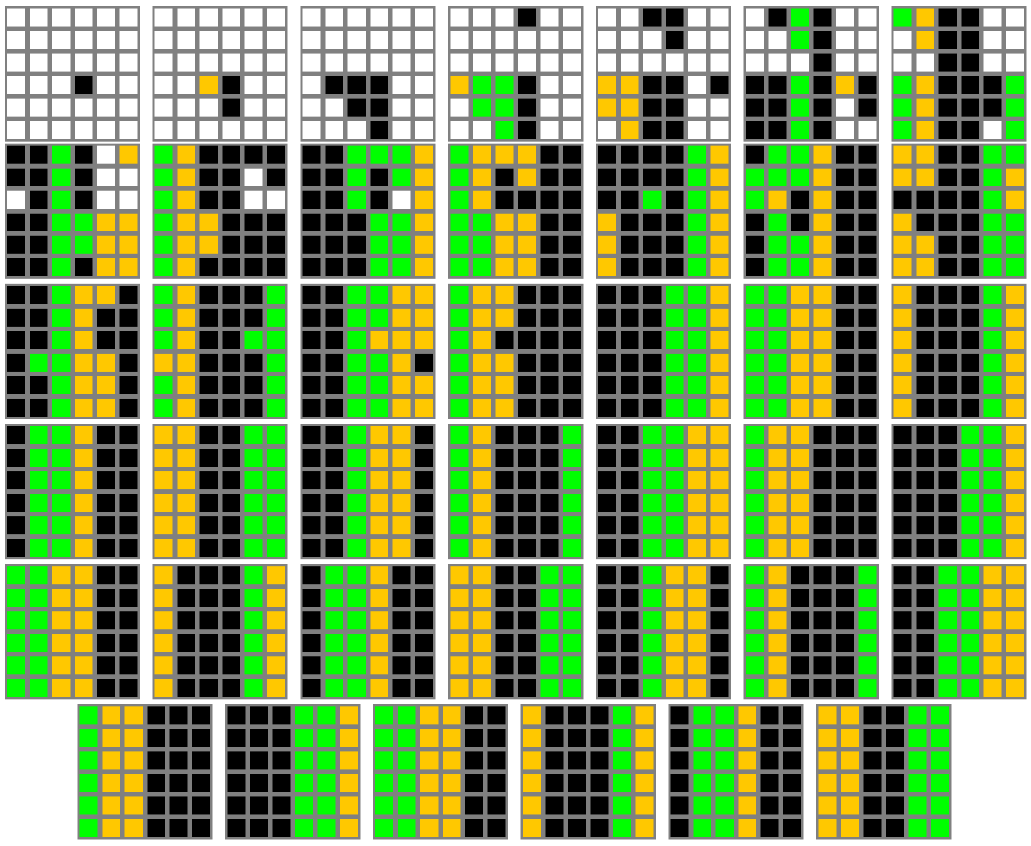


Figure 3.23: Class 2 3a solution to the flag structure.

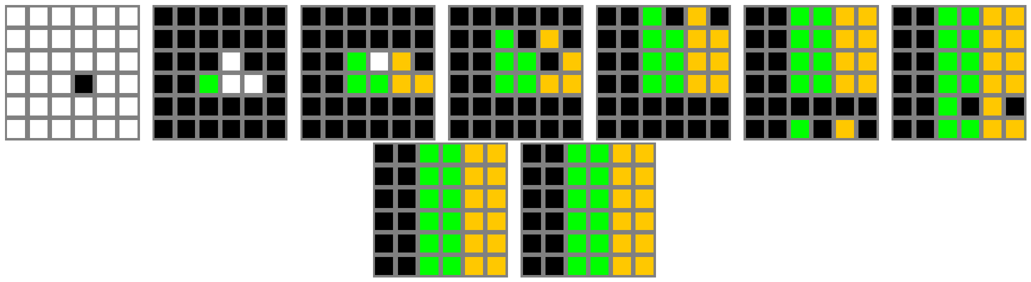


Figure 3.24: Class 1 3a solution to the flag structure, found in the long running GA.

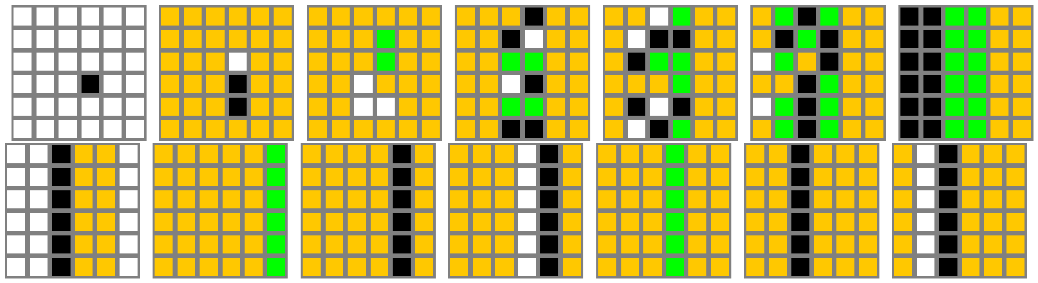


Figure 3.25: One of the many TT solutions to the 3a structure where the flag appears after a seemingly unexpected step. Take extra notice of step 5 as the GA questions its own purpose before lamentation in the understanding that it is to create a simple flag picture.

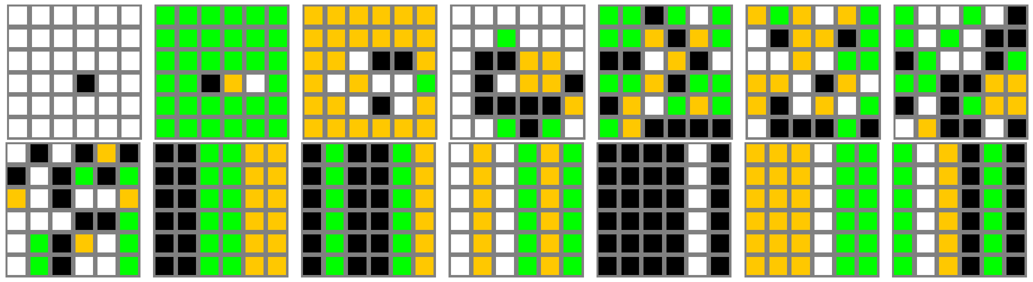


Figure 3.26: Another example where the flag appears unexpectedly when developed using TT.

French Flag (2a)

A structure from [11] was also used. The structure that was selected is the french flag structure as seen in Figure: 3.22. It was tested using the same parameters as the previous flag structure. In both [11] and this thesis, 10 instruction programs were used. In [11] the implementation only checked the first 30 steps while this thesis checked the first 40, which gives the results from this thesis a slight advantage. The purpose of this test was to check if this thesis implementation, was a working implementation. This slight difference was therefore not seen as a problem as long as the results were comparable.

Results	**Our**	**Bidlo and Vasicek**
Instruction-Based Approach		
Success rate %	79	79
Average	15438	37925
StDev.	24508	27171
Transition Table		
Success rate %	86	54
Average	13319	28896
StDev.	18835	26264

Table 3.4: IBA and TT on the 2a structure comparison. Success rate is out of 100 tries. Average and StDev. is the average and standard deviation of the number of generations the successfully runs required to find the solution

The results are compared in Table: 3.4. In this implementation IBA behaves very similar to [11]. TT on the other hand outperforms IBA on this structure, which was unexpected. It also greatly outperforms the TT in [11]. When it comes to the purpose of testing if TT and IBA were working, the results are very satisfactory.

3.4.2 Creeper structure

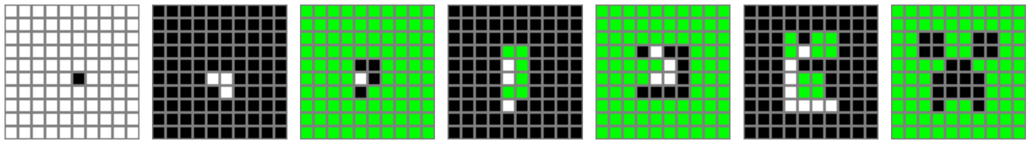


Figure 3.27: Only found creeper solution found using TT

Out of curiosity, another structure was also explored to test the results and this was the creeper structure. The same parameters were used as on the 3a and 2a structure, but this time the results were much more in TT's favour, as only TT managed to find a solution.

Creeper structure comparison

	TT	**IBA**
Success rate %	1	0
Average fitness	93.64	88.4
StDev.	2.013	0.9101

Table 3.5: IBA and TT comparison on the Creeper structure

Speculating on this, the hypothesis was due to that IBA has a much smaller genotype diversity than TT, and IBA therefore had no possible solution to the problem. The genotype variation can be calculated, and a single IBA instruction can be set up in a number of different ways. There are 16 different instructions and the two parameters can point to any of the 5 cells in the neighbourhood scheme. Therefore a single instruction can be set up into $16 * 5 * 5 = 400$ different variations. In this test 10 instructions were used, so this small program can be set up into a total of $400^{10} \approx 1.05x10^{26}$ ways. This value is independent of the number of possible states in the CA. In comparison TT with 3 states can have $3^{3^5} = 3^{243} \approx 8.72x10^{115}$ variations. Therefore IBA has a smaller genotype diversity then TT for 3 states and beyond, given that it contains only 10 instructions. Therefore, it would be logical to assume, that some portion of the possible genotype space is unreachable with this method when compared to a TT. This can be further demonstrated when considering there are only $3^{100} \approx 5.15x10^{47}$ possible phenotype states. This will be further discussed in the Results and Discussion section.

3.5 Self-Modification Instruction-Based Approach(SMIBA)

The third and final genotype representation method that was tested in this thesis is the Self Modification Instruction-Based Approach. This is the new method introduced in this thesis and, to the best of our knowledge, this method has never been applied in this setting before, so it is hard to compare results directly. Therefore testing SMIBA is done by comparing it to TT and IBA and this is further done in the analyses and discussion section.

Instruction	Parameters	Description
SKIP	$N[i_1] = nrofskips$	Skips the next $N[i_1]$ instructions. Not a SM instruction.
MOVE	$N[i_3] = Start$ $N[i_4] = Insert$	Moves the instruction at line nr $N[i_3]$ to before $N[i_4]$.
DUPE	$N[i_3] = Start$ $N[i_4] = Insert$	Copies the instruction at line nr $N[i_3]$ to before $N[i_4]$.
DEL	$N[i_3] = Start$	Deletes the instruction at line nr $N[i_3]$.
CHF	$N[i_3] = Start$ $N[i_4] = Instruction$	Changes the instruction at $N[i_3]$ to the instruction at $N[i_4]$.
CHP	$N[i_1] = parameter$ $N[i_3] = start$ $N[i_{2\|4}] = value$	Changes the $N[i_1]$ parameter at $N[i_3]$ to the value in $N[i_2]$ or $N[i_4]$ depending on $N[i_1]$.

Table 3.6: Instructions added to the IBA instructions in order to exhibit Self modification.

Self Modification was implemented using the IBA as a basic ruleset and extended with a SM ruleset. The additional ruleset can be seen in Table: 3.6. IBA was implemented with parameters that only ranged from $[0 - 4]$, but this would not allow SM instruction to reach the entire genotype, therefore it seemed advantageous to have SM instruction with a higher range. This is why the IBA system with, rule, op1, op2, was extended in SMIBA to rule, op1, op2, op3, op4 where op3/op4 had a range of [0, (maximum Instruction length)] Different rules simply used different op values, and this allowed for easier logic when mutating the genome and also made some instructions much easier to implement.

For example, a MOVE instruction in the genotype might look like {17,1,3,9,4}.

The values 1 and 3 would not be used for anything in this iteration. Even so, they can be useful to retain in some programs, as this instruction might be changed by a CHF instruction at some time. They might also be useful to encourage drift in the GA. As the move instruction is in this example it would simply remove the 9th instruction (10th as 0th is an instruction) and place it before the 4th instruction. This then becomes the new 4th instruction and pushes back all subsequent instructions.

In order to avoid cycles, the SM was first added to a temporary list so that it may be performed in order at the end, after all IBA instructions have been performed. If this was not the case, DUPE and DEL could cause a infinite cycles. This also allowed for performance improvement since the SM need only be performed every iteration and not for every cell. This is relevant since some of the SM instructions are costly to perform in comparison to the regular IBA instructions.

Skip

An additional non-SM rule was added called skip. This instruction was added, due to inspiration from biology. As mentioned earlier, during development genes can regulate which parts of a genome are active by secreting chemicals to suppress specific genes during certain and sometimes all of the development steps. In [51] Bently and Kumar showed strength in an implicit embryogeny system inspired by gene suppression. It relied on rules that required preconditions to be performed. Due to this, it seemed favourable to add a rule that allowed together with SM instruction for code to be available only in a subset of the developmental steps. This is why skip was added. It might be worth noteing that skip is useless in the IBA setting, as it will simply lower the number of instructions used.

Originally it was intended to be a GOTO instruction, but when implementing this instruction loop were detected. It was made so that it could only point to an instruction after itself, this limitation made it into a skip and it was renamed accordingly. It could be worth reimplementing it as a GOTO, but change the instruction implementations to simply count instructions and stop after a certain limit has been reached.

Testing

All the instructions were tested individually by running a SMIBA program and inspecting the genotype from iteration to iteration, to see if the genotype was changed as the instruction intended. The 3a and 2a structure was also tested on this method using the same parameters as the other methods, and the results were favourable. As can be seen in Table: 3.7, SMIBA greatly outperformed both methods on these structures. These results will be featured again, in more detailed, the next chapter.

Transition Table		
Problem	2a	3a
Success rate %	86	34
Average	13139	30676
StDev.	18835	27660
Instruction-Based Approach		
Problem	2a	3a
Success rate %	79	45
Average	15438	25163
StDev.	24508	28526
Self-Modifying Instruction-Based Approach		
Problem	2a	3a
Success rate %	100	96
Average	1912	13309
StDev.	9416	22687

Table 3.7: Comparison on the French flag and flag structure between IBA, SMIBA and TT. Success rate is out of 100 tries. Average and StDev. is the average and standard deviation of the number of generations the successfully runs required to find the solution

3.6 Replication problem

As mentioned in earlier chapters, cellular automata is commonly tested on replication problems and developmental problems. The replication problem requires a different fitness function that tests for successful replication. A larger lattice will also be necessary. This makes it slower than the development problem to perform. Therefore most of the experiments are only allowed to go for 10 000 generations as opposed to the 100 000 in the development problem. This is also done in both [47, 11]. The fitness function implemented, was highly inspired by the one used in [11].

3.6.1 Replication fitness function

After every iteration, the fitness function searches the entire lattice for patterns that match the target image. The lattice is searched by comparing every subsection of size corresponding to the target image. Every cell that is the same as the target image in this subsection awards a point. If n is the value for the desired number of replications, then the n best values are summed for each iteration. Then the iteration with the highest summed value is used as the genotypes fitness value.

Initially, the fitness function retained the values for all possible subsections, sorted them and selected the ones with the highest values as seen in Algorithm: 1. This was very inefficient and this part of the algorithm was improved to only retain the values of the subsections with the highest values as seen in Algorithm: 2. This has no effect on what any given genotype is awarded in fitness, but it makes the function run faster. Therefore the experiments, which used the first fitness function, were not rerun. Note that the algorithm does not consider targets that lie over the borders of the lattice. In addition, the target image has an added white border to it, which ensures that perfect replicas do not overlap.

3.6.2 Alternatives to the fitness function

Much of the design of the fitness function was done to ensure it would not slow down the runtime of the algorithm. This is because it is slow, compared to the single pattern matching, that the development problem needs to do, because replication needs to do a multitude of pattern matching. This makes the fitness function for the replication problem in-practice the main bottleneck of the entire experiment. This is why it does not consider patterns that cross borders, as

Algorithm 1 Old Replication fitness function

```
1: function FitnessOfRule(rule, nrOfReplications)
2:     for Each CA iteration do
3:         partialList ← new List
4:         fitness ← 0
5:         for columnWidth - targetWidth do
6:             for rowHeight - targetHeight do
7:                 partialFitness ← 0
8:                 for targetHeight do
9:                     for targetHeight do
10:                        if Cell match then
11:                            partialFitness+= 1
12:                partialList add partialFitness
13:        Sort partialList
14:        for k = 0 to NrOfReplications do
15:            fitness+= partielList.get(k)
16:        if $fitness > maxFitness$ then
17:            maxFitness ← fitness
18:    Return: maxFitness
```

Algorithm 2 Improved replication fitness function

```
 1: function FitnessOfRule(rule, nrOfReplications)
 2:     for Each CA iteration do
 3:         $bestReplications \leftarrow$ new array[nrOfReplications]
 4:         $fitness \leftarrow 0$
 5:         for columnWidth - targetWidth do
 6:             for rowHeight - targetHeight do
 7:                 $partialFitness \leftarrow 0$
 8:                 for targetHeight do
 9:                     for targetHeight do
10:                         if Cell match then
11:                             $partialFitness += 1$
12:                 for all bestReplications do
13:                     if bestReplication < partielFitness then
14:                         $bestReplications \leftarrow partielFitness$
15:         if $fitness > maxFitness$ then
16:             $maxFitness \leftarrow fitness$
17:     Return: maxFitness
```

this would require either hard coded logic or module operations in cell matching. Nor does it consider the possibility that imperfect partials are overlapping. This might generate a fitness landscape with many local optimas that are far from a perfect answer. Originally, only partial fitness matches that did not overlap were considered in the function, but this slowed down the algorithm a great deal and was therefore removed.

3.6.3 Limitations outside the fitness function

One might also consider that the height of the lattice and the number of desired replications puts limitations on what kind of solutions can be found. If the lattice is not big enough to room all the replicas in one direction, the only solutions possible are ones where the CA develops replications in more than one directional dimension. Therefore, all experiments were run with a few cells over the room required for 3 replications in hight and length.

3.6.4 The structures explored in this thesis

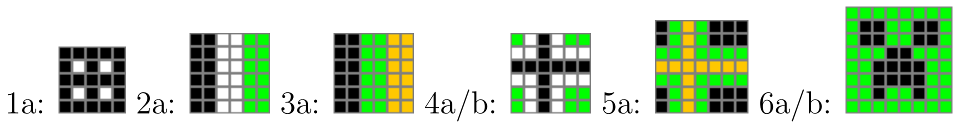


Figure 3.28: left to right; 1a: simple Structure, 2a: 3 state French flag, 3a: generic 4 state flag, 4a/b: 3 state Norse Flag, 5a: 4 state larger Norse flag, 6a/b: 3 state creeper(unbordered)

The structures that were selected can be seen in Figure: 3.28. This is not the first time the replication problem has been inspected, therefore the scientific network provides a plenitude of structures that can be selected. All these structures selected, with the exception of 6a/b, were used also in other articles [47, 11, 61]. The structures were selected so that there are was a range of structures that are symmetrical and non-symmetrical, few to many states and large to small in size. 6a/b were added because of the unexpected results when used in the development problem. a and b versions of the structures use the same target solution, but b structures gives the algorithms one extra state to use, so 4b is the same target solution as the 4a, but the CA is has 4 states instead of 3. This strategy was used when some of the algorithms struggled to find a single solution.

3.7 DevRep problem

In order to try to solve both replication and development in unison, a problem was formulated simply called development replication problem, which is abbreviated to DevRep problem. The problem was defined to be one where the algorithm must find a solution from a zygote to develop and create the target structure(image) atleast once in the lattice. In a later iteration, the same target structure must be present several places in the lattice. In short, first the structure needs to be developed then replicated. No changes were made to the GA or the CA other than the Fitness function. It was given 40 iterations to form the problem and had a population of 50, same as for the other experiments.

One major hurdle that was foreseen, was that if, DevRep was to be at all possible, then development needs to be performed on a lattice that is bigger than the actual target image. During the development, the CA cannot use the border conditions to form the image or to stop growing sections once the section is fully formed. Some simple trials were run to test if TT, IBA and SMIBA could develop with a lattice that is larger then the target. These trials proved that it was possible.

3.7.1 Structures

Due to this problem being hard in comparison to the other two problems explored in this thesis, two new structures were used, 0a and xa/b. These two new structures together with the other structures explored with the DevRep problem can be seen in Figure: 3.29.

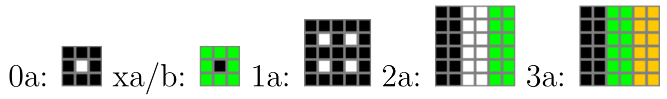


Figure 3.29: left to right; 0a: Simple 2 state, xa/b: simple 3 state, 1a: simple Structure, 2a: 3 state French flag, 3a: generic 4 state flag.

Note that on this problem, as with all problems explored in this thesis, a Von Neumann neighbourhood was used. The Game of Life implementation used to test the CA was a exception to this and there a Moore neighbourhood is necessary. The 0a and xa/b structures are small, and if a Moore neighbourhood was used the central cell in these structures would have a centralised overview of the image, and the problem would no longer be a distributed problem.

3.7.2 Fitness function

The fitness function for the DevRep problem is based on the fitness function for replication. The first 20 iterations are checked for development fitness by using Algorithm:2 and checking for one replication. The development step with the best fitness is stored and the CA is reset and progressed to this stage. In SMIBA the genotype program is also reset to its initial stage before moving to the best development state. Once at the point of the best developmental fitness, the fitness function looks for 3 replications of the target image for the next 20 iterations of the CA. The two scores are weighted so that the fitness of the development counts the same as the fitness of the 3 replications.

3.8 Pajek

Pajek [74] is a tool for large network analyses and visualisation. It is a large and complex program with many features, but in this thesis it was used for visualisation of development pathways. This visualisation was done by executing a specific solution in CA and tracing every unique CA state as a node. Transition to the next state was then marked as a directed edge. These values were then prepared so that Pajek could read it as an input file and draw a graph of the network. This allowed for a visualisation of developmental pathways that could be compared to other developmental pathways.

Chapter 4

Results and discussion

"Cyberspace. A consensual hallucination experienced daily by billions of legitimate operators, in every nation, by children being taught mathematical concepts... A graphic representation of data abstracted from banks of every computer in the human system. Unthinkable complexity. Lines of light ranged in the nonspace of the mind, clusters and constellations of data. Like city lights, receding..."

William Gibson - Neuromancer

In the previous chapter, we had a look into this thesis's experimental setup and how it was tested. In this chapter we will be take an in-depth look into the results of the experiments and discuss the findings. We will also look at some interesting solutions or cases which appeared in the experiments.

Firstly, results from each of the three problems; Development, Replication and DevRep will be presented, then after each of these problems, discussion of findings, properties of the different methods and relevant cases are elaborated on. Later, an additional extra experiment is presented and discussed. Here, once a run finds a perfect solution the fitness function is changed, requiring that the run is re-evolved into a new structure. Finally analysis of the instructions used to solve the different problems is presented.

4.1 Development

The first experiment that was performed in this thesis was the development problem. In this case the goal is to develop a target structure from a small zygote meaning a single black cell in the center of the lattice. The results are summed in Table: 4.1, here SMIBA clearly outperforms the other two methods. SMIBA solves problems easily, that the other methods are struggling with. Structure 2a is solved every run in SMIBA as well as in fewer generations compared to the other methods. 3a was solved nearly every time, while the other methods did not even manage half. Further, it also managed to find solutions to problems the other methods are unable to find a single solution to. 4a and 4b was solved 5 times while only IBA managed to solve 4a once. As a exception to this TT managed to solve 6a once and 5b twice, while none of the other methods managed to solve them.

Transition Table							
Problem	2a	3a	4a	4b	5a	5b	6a
Success rate %	86	34	0	0	0	2	1
Average(numGen)	13139	30676	x	x	x	64506	35528
StDev.(numGen)	18835	27660	x	x	x	833	x
Instruction-Based Approach							
Problem	2a	3a	4a	4b	5a	5b	6a
Success rate %	79	45	1	0	0	0	0
Average(numGen)	15438	25163	32839	x	x	x	x
StDev.(numGen)	24508	28526	x	x	x	x	x
Self Modifying Instruction-Based Approach							
Problem	2a	3a	4a	4b	5a	5b	6a
Success rate %	100	96	5	5	0	0	0
Average(numGen)	1912	13309	57224	47042	x	x	x
StDev.(numGen)	9416	22687	42635	12950	x	x	x

Table 4.1: General results of TT, IBA and SMIBA on the development problem. All results are taken out of 100.

4.1.1 IBA in comparison to TT

In this thesis we have among other things tested IBA and compared it to TT. While running the tests and looking at the results, properties of their relation to each other in the genotype space became clear. It seems IBA has a genotype space that is always a subset of TT's. This conclusion can be reached by understanding, that for every IBA genotype there is a corresponding TT genotype, but this is not necessarily true the other way.

One can know this because an IBA ruleset can be converted into a TT. Simply, for every neighbourhood combination in the TT, run the IBA program on the neighbourhood and store the result in the TT (see Figure: 4.1 for illustration).

IBA also has redundant space. There are many IBA rulesets that do nothing, if the IBA ruleset never places a value in the center of the neighbourhood no change is ever made. There are also many IBA rulesets that do the same as other rulesets. One can know this because there are many instructions that have equivalent functionality that can be built up from other instructions or the same instruction with different parameters. For example an AND instruction on L and C is the same as AND on C and L. This means IBA to TT has a one to one relation, but TT to IBA has a 1 to (0,n) relation.

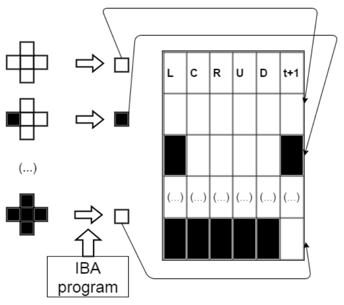


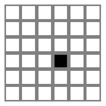
Figure 4.1: converting IBA into a TT

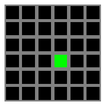
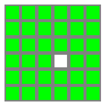
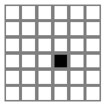
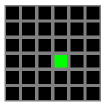
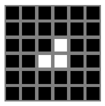
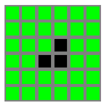
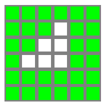
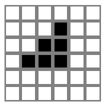
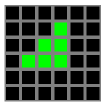
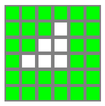
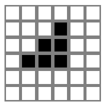
Also it would seem, there are TT that cannot be represented in IBA with 10 instructions. This is supported by the problem structures that TT managed to solve, but IBA was unable to. On the creeper test TT consistently found solutions better then the IBA to a very large degree, as was seen in Figure: 3.5. This raises a new question, what portion of the genotype space is IBA not able to reach?

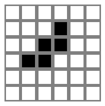
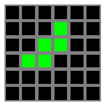
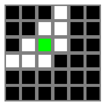
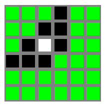
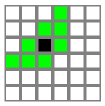
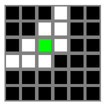
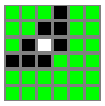
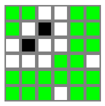
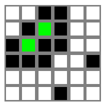
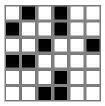
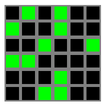
When looking at the best candidate solutions found by the individual runs in development, TT often produces different strategies than IBA. TT appears to find solutions that build up the desired image then descend into chaos, while IBA favours solutions that appear to be class 2 solutions. This would suggest that IBA is either incapable of finding class 3 with its current instruction- set and length or that solutions using these classes are hard to find or rare. In understanding of the classes and their relations found in [26, 18], here it is given evidence that Class 4 is found at the phase transition between class 2 and 3 behaviour. If a method is unable to reach class 3 then, it might also be unable to reach class 4. This is a potential setback for IBA, so it would suggest IBA is weak on complex problems, but strong on others. Rather than this, a different alternative hypothesis for explaining the results is also possible. In this thesis TT is implemented as an explicit table while IBA constantly used 10 instructions as a limit. This was done with intention of showing scalability in IBA and SMIBA, which will be discussed later. This comparison of TT, which used a genotype size relative to the number of states, while IBA used a constant size, might be an unfair comparison for problems that require complex solutions. The diversity of genotypes are allowed to grow with the problem in TT, but not in IBA. This corresponds well with the results from [47] where it was shown that some problems required a larger number of instructions on average to be solved. Both methods also used the same crossover and mutation rates, therefore TT with a larger genotype size could potentially take benefit from this over IBA. This is hard to speculate on without doing a optimisation of GA parameters, which is outside the scope of this thesis. These observations of different strengths between the methods also matches well with the No free lunch theorem [72]. This theorem states that for any increase in performance over one class of problems, will be directly paid for in performance over a different class. If this theorem is also true for encoding methods, this would suggest that IBA's strength comes at a cost of solving, problems that require chaotic CA or alternatively solution variations, which will be discussed later.

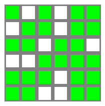
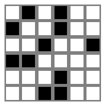
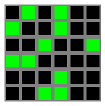
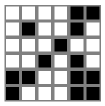
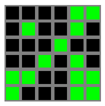
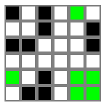
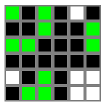
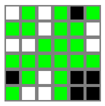
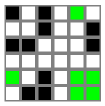
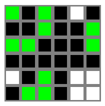
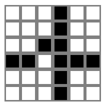
4.1.2 SMIBA

As seen from the Table: 4.1, SMIBA showed the strongest results for the development problem both in solving rate and of structure variation. To a lesser degree SMIBA also found solutions in fewer numbers of generations on average. Yet again, the 5b and 6a structures proves a exception as TT was the only method able to find a solution.

Iteration	Genotype	Phenotype
0	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),**(MOVE, D, D, 4, 8)**,(MIN, L, D, 5, 5),**(AND, L, C, 7, 6)**,(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MOVE, L, C, 2, 6),	

1	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),**(MIN, L, D, 5, 5)**,(INC, C, C, 5, 5),**(OR, D, L, 8, 5)**,(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(MOVE, L, C, 2, 6),	
2	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),**(INC, C, C, 5, 5)**,(ROR, C, C, 8, 5),**(MOVE, D, D, 4, 8)**,(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(MOVE, L, C, 2, 6),	
3	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),**(ROR, C, C, 8, 5)**,(ROL, U, R, 5, 3),**(MIN, L, D, 5, 5)**,(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(MOVE, L, C, 2, 6),	
4	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),**(ROL, U, R, 5, 3)**,(AND, L, C, 7, 6),**(INC, C, C, 5, 5)**,(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(MOVE, L, C, 2, 6),	
5	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),**(AND, L, C, 7, 6)**,(OR, D, L, 8, 5),**(ROR, C, C, 8, 5)**,(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(MOVE, L, C, 2, 6),	
6	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),**(OR, D, L, 8, 5)**,(MOVE, D, D, 4, 8),**(ROL, U, R, 5, 3)**,(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, L, C, 2, 6),	
7	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MOVE, L, C, 2, 6),	
8	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(MOVE, L, C, 2, 6),	
9	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(MOVE, L, C, 2, 6),	
10	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(MOVE, L, C, 2, 6),	
11	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(MOVE, L, C, 2, 6),	

12	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(MOVE, L, C, 2, 6),	
13	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, L, C, 2, 6),	
14	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MOVE, L, C, 2, 6),	
15	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(MOVE, L, C, 2, 6),	
16	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(MOVE, L, C, 2, 6),	
17	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(MOVE, L, C, 2, 6),	
18	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(MOVE, L, C, 2, 6),	
19	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(MOVE, L, C, 2, 6),	
20	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, L, C, 2, 6),	
21	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MOVE, L, C, 2, 6),	
22	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(MOVE, L, C, 2, 6),	

23	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(MOVE, L, C, 2, 6),	
24	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(MOVE, L, C, 2, 6),	
25	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(MOVE, L, C, 2, 6),	
26	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(MOVE, L, C, 2, 6),	
27	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, L, C, 2, 6),	
28	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MOVE, L, C, 2, 6),	
29	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(MOVE, L, C, 2, 6),	
30	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(MOVE, L, C, 2, 6),	
31	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(MOVE, L, C, 2, 6),	
32	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(MOVE, L, C, 2, 6),	
33	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(MOVE, L, C, 2, 6),	

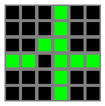
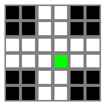
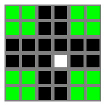
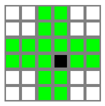
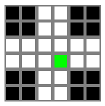
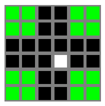
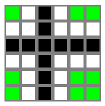
34	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, L, C, 2, 6),	
35	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MOVE, L, C, 2, 6),	
36	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(MOVE, L, C, 2, 6),	
37	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(INC, C, C, 5, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(MOVE, L, C, 2, 6),	
38	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROR, C, C, 8, 5),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(MOVE, D, D, 4, 8),(MOVE, L, C, 2, 6),	
39	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(ROL, U, R, 5, 3),(AND, L, C, 7, 6),(INC, C, C, 5, 5),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(MIN, L, D, 5, 5),(MOVE, L, C, 2, 6),	
40	(OR, R, D, 7, 0),(NOT, D, U, 0, 9),(AND, L, C, 7, 6),(OR, D, L, 8, 5),(ROR, C, C, 8, 5),(MOVE, D, D, 4, 8),(ROL, U, R, 5, 3),(MIN, L, D, 5, 5),(INC, C, C, 5, 5),(MOVE, L, C, 2, 6),	

Table 4.2: A solution to development of the 4a structure. This solution used 2 move instructions to create a 7-step loop of the program. This seems to slow down the development of the structure enough for the flag to be formed. Instructions modified by the MOVE instruction are made bold for the first loop, for better viability.

The notion of scaling was also considered briefly in these results. Structure 4a and 5a were executed with maximum number of states incremented in a second attempt. This increased the phenotype variations but on the development problem this did not show any increase in solving efficiency, but neither did it show any decrease in SMIBA 4a to 4b structure. A closer look was taken at the structures solved only by SMIBA. These solutions proved some interesting cases.

One property that SMIBA can utilize on these problems is delaying development, slowing it down. This can be seen well in Table: 4.2. In this case the state of the environment or the general state value loops constantly, while the zygote only grows or develops at certain intervals. In this example the structure is formed by creating a looping structure with two move actions. Together, they form a 7-step loop, which circulates the instructions. This allowed the flag to form in 40 steps, the maximum steps the fitness function will inspect. This property which, was witnessed in many other cases, caused a deeper analyses of the development length in solutions. This will be discussed in the following section.

4.1.3 Only solved in TT

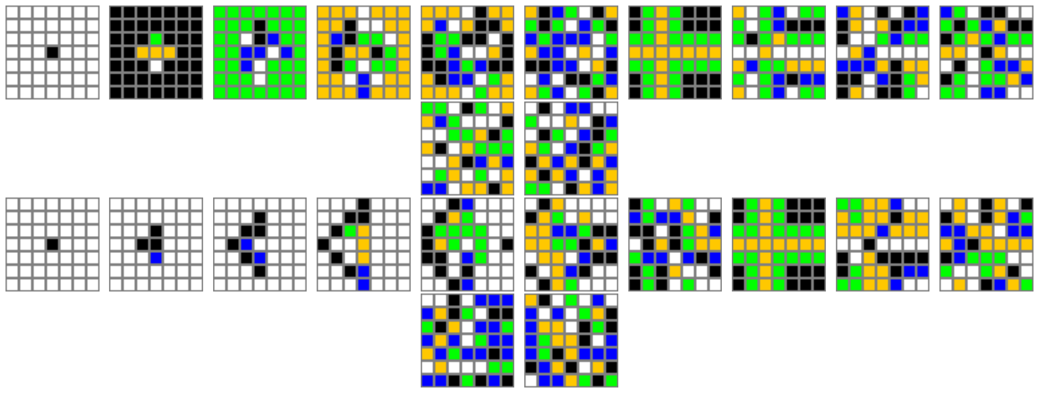


Figure 4.2: TT solutions to the 5b structure. In both cases a flag appears seemingly out of randomness and the CA quickly descend into seemingly random behaviour afterwards. Blue is the fifth color included in this CA.

Slightly different than in [47, 11], a few problems in development were only solved by TT. The 5b and 6a structure were solved in rare cases during the execution of a 100 runs in the experiment. Looking into these solutions all of these cases show a high degree of chaotic CA, as can be seen in Figure: 4.2. Such a statement is hard to prove because identifying classes in CA is very exhaustive, since TT is also very slow when scaled to the level of the 5b structure, and would be very hard to even attempt. There are also very few cases to draw from, the results would only show an indication. One could also argue that the CA used in these cases are chaotic because the structure needs to be formed in very few steps. Tracing a shape through many development steps is hard when a behaviour is highly chaotic and random. Also note, that just running the GA on the 5b took 90 hours, which is 10 times longer when compared to IBA and SMIBA where it only took roughly 9 hours. Such runtime for the TT, IBA and SMIBA were only achieved through the use of threading and code optimisation.

These results still indicate a fundamental difference with solutions found with TT as opposed to, IBA or SMIBA. It could further give some evidence that, IBA and transitively also in SMIBA, avoided class 3, also known as the chaotic class of CA. When looking at the average fitness value in 5b and 6b structures a trend emerge. Even if the TT lose in this regard on the 5b structure, as can be seen in Table: 4.3. When looking at the standard deviation of the results, TT is shown to have a much more varied values for fitness. This is probably why it solved problems the other methods could not.

	TT			**IBA**			**SMIBA**		
	5a	5b	6a	5a	5b	6a	5a	5b	6a
Avg(fitness)	39.1	33.9	93.6	38.8	38.9	88.4	38.5	38.6	89.8
StDev(fitness)	5.93	7.45	2.01	3.22	3.17	0.90	3.54	3.00	0.86
Best(fitness)	48	49	100	43	45	90	48	48	94
Max pos. fit.	49	49	100	49	49	100	49	49	100

Table 4.3: Average Fitness of 5a/b and 6a between the methods.

4.2 Developmental Length

TT, IBA and SMIBA have different developmental strategies and one could expect there to be a discrepancy on the average developmental time of the algorithms. further, a property of SMIBA of delaying development was identified in some cases. Due to this, SMIBA would be expected to have a longer development time than the other methods. Furthermore it is worth noting, that a long developmental time is not a weakness of an algorithm, nor is it necessarily a strength, unless the method is able to utilize it. If an algorithm only uses developmental strategies with very short developmental time, could indicate that it to some degree excludes class 3,4 CA [26, 18]. In order to investigate this, a comparison of the three algorithms was performed, where development length was in focus. This calculation was done by excluding incomplete solutions from the data set. From the remaining complete solutions, the average developmental steps required to form the structure was calculated. 2 different values of upper limit for development of CA iterations were tested(40 and 80). The results are summed up in Table: 4.4.

In this table the two development problems that were solved by all 3 methods were inspected. TT showed a clear use of the extra development steps. This method also showed more executions solved and a faster solving time.

IBA on the other hand was less clear. It did not show any improvement on its ability to solve problems. Even if it did use the extra development steps in its solutions as well as solve the problems in fewer generations. SMIBA did not show any increase in number of solutions found. This was only expected to be the case in 3a, as 2a was already solved every time and therefore, no improvement could be made. Finally, SMIBA also showed solutions that used more development steps, and fewer generations were required to find solutions.

These results are uncertain in telling what kind of solutions IBA excludes, as it was expected to show a lower development time in comparison to the other methods. On the other hand IBA did not show an improvement in its ability to solve the problems when given the extended development steps. It also showed a slightly lower standard deviation of development, which tells us that the solutions found with this method is more uniform. SMIBA was expected to use a longer development time then the other methods due to the property of delayed development. The results might indicate that the property is only used as a last resort by the method. A much larger discrepancy between the methods was expected, it is possible that other structures would provide better comparisons. The structures of 2a and 3a were chosen, because they were the only structures solved by all methods, therefore these provided the only comparable results.

Transition Table				
	2a		3a	
Max Iterations	40	80	40	80
Success rate %	86	96	34	46
Avg(Devo)	18.90	36.59	18.94	33.85
StDev.(Devo)	9.58	23.59	10.83	23.20
Avg(NumGen)	13139	7700	30676	26934
StDev(NumGen)	18835	11437	27660	25533
Instruction-Based Approach				
	2a		3a	
Max Iterations	40	80	40	80
Success rate %	79	81	45	43
Avg(Devo)	23.85	36.03	24.42	31.93
StDev(Devo)	8.10	16.39	8.53	19.87
Avg(NumGen)	15438	12128	25163	15943
StDev(NumGen)	24508	19974	28526	17868
Self-Modifying Instruction-Based Approach				
	2a		3a	
Max Iterations	40	80	40	80
Success rate %	100	100	96	96
Avg(Devo)	22.80	30.46	26.13	38.49
StDev(Devo)	8.93	18.14	9.82	21.61
Avg(NumGen)	1912	1774	13309	6345
StDev(NumGen)	9416	4179	22687	12912

Table 4.4: Comparison on the 2a and 3a structure between IBA, SMIBA and TT. Average and standard deviation of the developmental time.

4.3 Replication

The second experiment explored in this thesis is the replication problem. In this case the goal is to create copies of an initial structure, such that the initial structure replicates into multiple copies of itself. The results are summed up in Table: 4.5.

Transition Table								
Pattern	1a	2a	3a	4a	4b	5a	6a	6b
Success rate %	62	5	0	0	0	0	0	0
Avg(NumGen)	2116	4909	x	x	x	x	x	x
StDev(NumGen)	2533	2009	x	x	x	x	x	x
Instruction-Based Approach								
Pattern	1a	2a	3a	4a	4b	5a	6a	6b
Success rate %	100	100	100	0	100	100	4	100
Avg(NumGen)	32	167	37	x	32	71	5723	29
StDev(NumGen)	22	157	33	x	25	68	3364	25
Self-Modifying Instruction-Based Approach								
Pattern	1a	2a	3a	4a	4b	5a	6a	6b
Success rate %	100	100	100	0	100	100	22	100
Avg(NumGen)	38	279	54	x	37	94	4737	54
StDev(NumGen)	24	344	53	x	25	72	2745	42

Table 4.5: Replication problem comparison

These results correspond well to the results found in [11], where IBA was shown to be especially effective in the replication problem. SMIBA seems to be able to rely on its IBA base for solving the same problems that IBA solves. SMIBA used a few more generations to get there, but this was to be expected as IBA was already exceedingly good at this problem. None of the methods managed to solve the 4a problem, which was surprising, but when transitioned to a 4b structure both IBA and SMIBA solved it with ease. This could indicate that 3 states are insufficient to easily solve a problem that is symmetrical over two axis, because the addition of an extra state made it simple. The transition in efficiency between 4a and 4b does state well in favour of IBA and SMIBA as methods that handle the scaling well. When given additional states TT requires an exponential increase in genotype size, which causes a longer runtime. Also, given the results of 4a/b and 6a/b, showed no increase in ability to find solutions. Furthermore both the average fitness value and the best fitness found in these experiments were seen to go down. This is shown in Table: 4.6.

Transition Table				
Pattern	4a	4b	6a	6b
Avg(fitness)	163.03	143.13	155.12	144.43
Best(fitness)	177	159	170	157

Table 4.6: Replication problem average and best fitness found in scaled structures for TT. A perfect solution would score 192 in fitness value.

In comparison to TT, in IBA and SMIBA the genotype size can remain constant, or however large it is desired to be. An alternative hypothesis the performance could also be caused by an intricate relation between phenotype and genotype variation as there could be an optimal relation between genotype and phenotype size. It might be possible to exploit this with IBA, SMIBA and other implicit methods, as these methods allows for generation of genotypes of a specified size. Phenotype variation could be modified by incrementing the number of states allowed, as was briefly tested on some problems for this thesis. The 6a to 6b transition showed similar results, but differs from 4a to 4b because 6a was solved in some cases. It was never expected that SMIBA would outperform IBA on a structure to the degree it did on 6a. No other structure showed the same result. Even so it still shows that SMIBA has made a contribution to IBA also on replication problem. More advantages and strengths of the SMIBA solutions will be presented later.

General replicator

When inspecting cases in the dataset, a solution to the replication program for the 3a structure with IBA was tested on a 2a structure, and quite surprisingly it solved it perfectly. Following this, more structures were tested by using programs evolved on a specific structure to see if they could replicate other structures as well. Quite interestingly the program could replicate almost any structure. Inspecting even more solution cases showed similar properties, both in IBA and SMIBA.

An example is this program that was randomly drafted from a 3a solution (XOR, L, C), (ROD, L, C), (OR, C, C), (XOR, C, L), (ROL, D, C), (MIN, C, D), (MAX, R, R), (ROU, U, L), (ROD, C, D), (SET, C, L). This program can be seen performed on its intended structure in Figure: 4.3. This program has the ability to solve other structures than its intended designed structure, see Figure: 4.4, 4.5, 4.6 and 4.8. It was also interesting to note that on the 4a problem the program could replicate the structure, but with the wrong colors as seen in Figure: 4.7. In Figure: 4.4 and 4.6, it can be seen to use a support color not present in the target solution to replicate the structure. In fact when Figure: 4.5 was replicated with the same program in 2-state CA, it was still able to solve the structure. This is something that would be very difficult to do in TT, since the explicit definition of the genotype in TT cannot simply be scaled down or up to use on a CA with a different number of states.

4.3.1 Continued replication

From inspecting case solutions in IBA and SMIBA, a general observation was made. Many of the cases were observed to continue replicating after they had produced the minimum amount of replications required by the fitness function. One example of this can be seen in Figure: 4.3, where additional replications were produced. Therefore a deeper analyses of the solutions was considered and a question arose. How good were these solutions at replicating continuously, i.e. are the replications also replicators? The solutions found in the general replication experiment were therefore placed on a lattice 3 times the normal size and inspected for 3 times the normal CA iteration length. This means a lattice size of 75x75 and the CA was evolved for 120 steps. Through every step of the total 120 steps the CA was inspected and the number of replications were counted. Only replications that had a border, and did not touch the edges were counted. The results are summed up in Table: 4.7.

The results were much better then anticipated. In the problems inspected, all methods showed a great ability to replicate beyond the scope of the fitness function they were evolved with. This was especially true for the solutions found in SMIBA, with many extreme cases where as many as 50 replications were produced, as can be seen in Figure: 4.9. Speculating on the cause for the discrepancy in replication ability between the methods, common traits in phenotype solutions were identified. While IBA and SMIBA often solved the problem by replicating in two directions (in different dimensions). TT only solved by replication in 1 or 2 directions (the directions were the same dimension). TT exclusively replicated

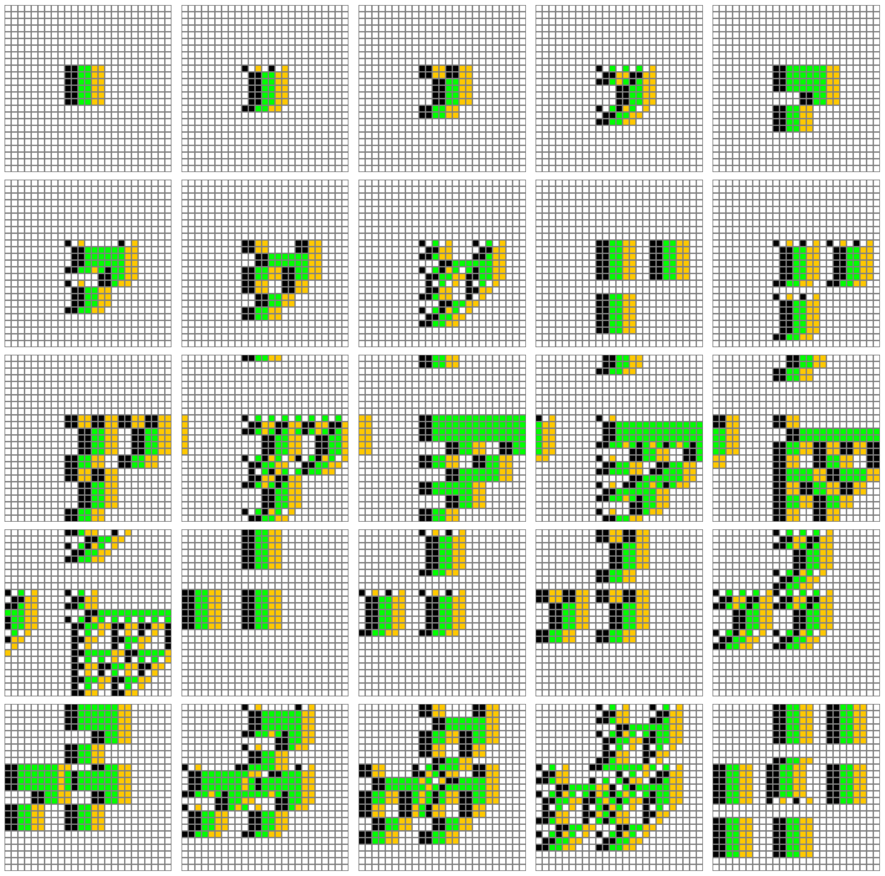


Figure 4.3: 25 steps of a solution to the replication problem for 3a, also continues to replicate after

either left and right or up and down. This caused TT to not be able to use the full lattice. 2a was shown as an exception in the case of IBA and SMIBA, and here TT caused on average the greatest number of replications. This was not because TT was especially strong on the 2a problem, but rather that IBA and SMIBA were especially weak on it.

4.4 A Final Observation

As a final observation, structures that IBA and SMIBA struggled with more then others(2a, 4a and 6a) have a common theme. 2a took longer to evolve

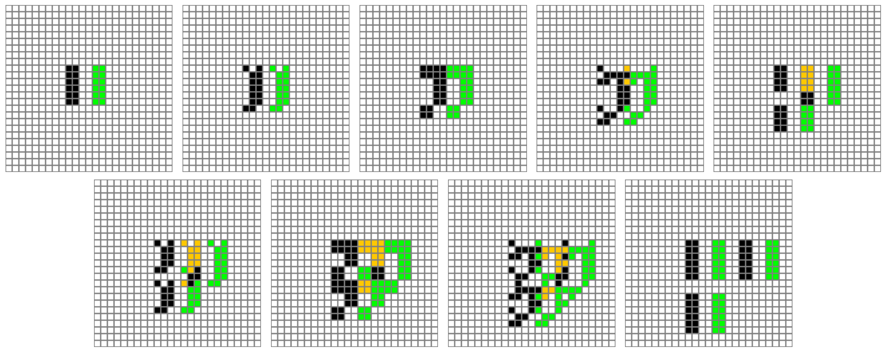


Figure 4.4: 8 steps of a solution to the replication problem for 3a, but used on 2a structure.

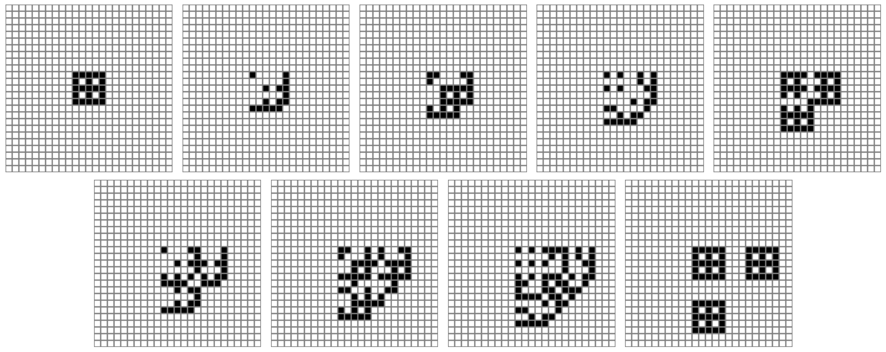


Figure 4.5: 8 steps of a solution to the replication problem for 3a, but used on 1a structure.

in comparison to structures thought to be harder, i.e. 3a. The solutions found with 2a, were also not as good as the other structures as continues replicators. 4a was never replicated and 6a was hard to replicate until they were given an additional state. What the 2a, 4a and 6a structures have in common is that they only have 3-states in the structure. When 4a and 6a were given a additional state for evolution to exploit and turned into 4b and 6b, what caused the difference in efficiency could perhaps not be the increase in phenotype variations. It could in fact be much simpler. 4-states as a binary representation, is a more sensible limit compared to 3-states. Since IBA and SMIBA utilize instructions that are designed for bit operations, a 3-state upper limit could become detrimental. If this is true, then IBA and SMIBA will be expected to show different distributions of instructions used to solve 2a and 6a in comparison to the other structures.

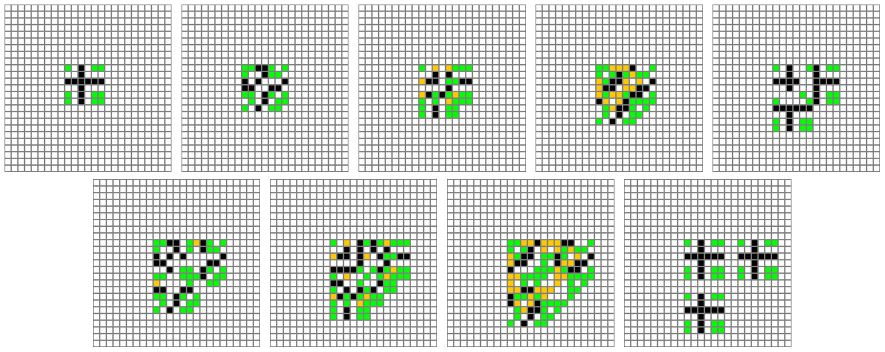


Figure 4.6: 8 steps of a solution to the replication problem for 3a, but used on 4a structure(4 state).

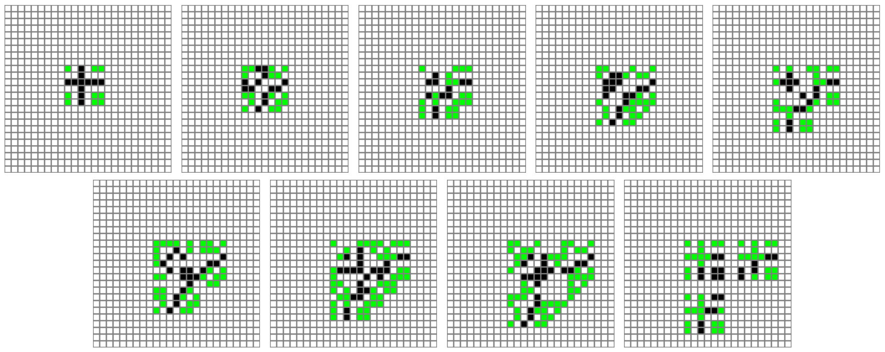


Figure 4.7: 8 steps of a solution to the replication problem for 3a, but used on 4a structure(3 state). It can no longer replicate the image, but the structure is perfectly replicated

This will be inspected later in this chapter.

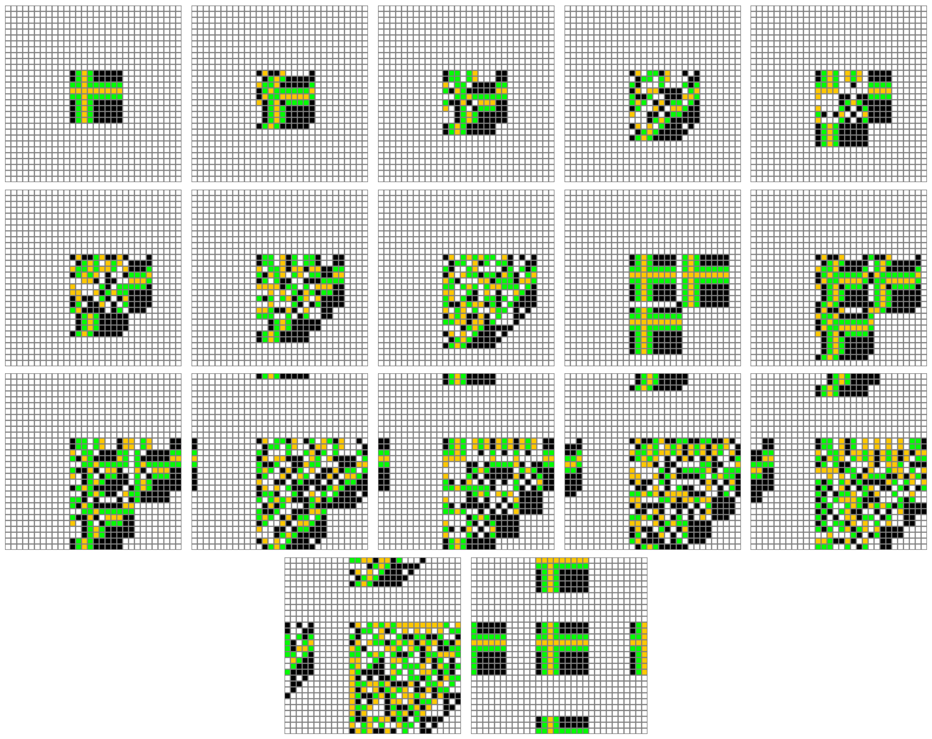


Figure 4.8: 16 steps of a solution to the replication problem for 3a, but used on the larger structure 5a. Note that the lattice had to be a bit bigger to fit the solution and that it required an extra loop to replicate it. Also note that this would not be a solution that the fitness function would find particularly good, as the fitness function was not designed to consider solutions found crossing the lattice bounds. We as humans on the other hand can clearly see the flags are replicated over the boundaries.

Transition Table				
Pattern	1a	2a		
Best replications %	9	9		
Avg(replications)	8.81	5.8		
StDev(replications)	1.06	1.6		
Instruction-Based Approach				
Pattern	1a	2a	3a	6a
Best replications	28	5	23	23
Avg(replications)	14.95	5	16.94	14
StDev(replications)	7.44	0	4.33	9
Self-Modifying Instruction-Based Approach				
Pattern	1a	2a	3a	6a
Best replications	51	9	50	50
Avg(replications)	17.24	5.06	18.25	16.59
StDev(replications)	10.67	0.58	10.46	7.56

Table 4.7: Replication count in the extended solutions, best replications is the case with the highest number of complete replications observed.

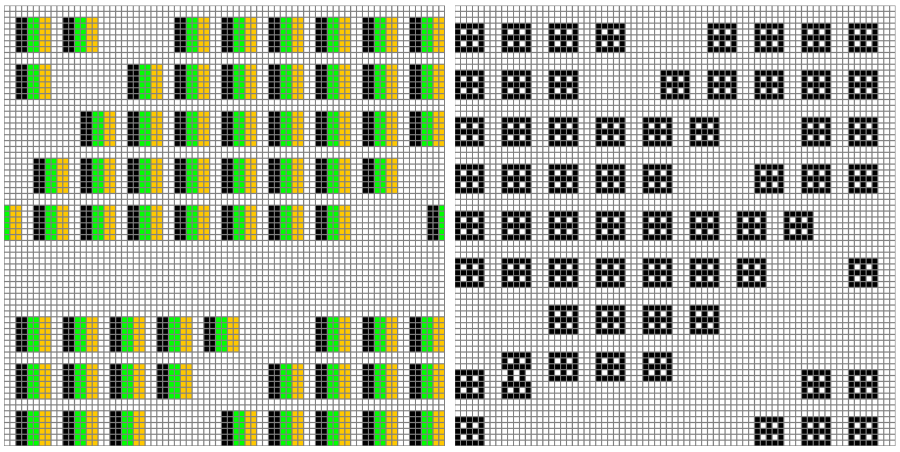


Figure 4.9: 2 examples of mass replications. Many of the replications are not counted as they touch the edge.

4.5 DevRep Problem

The final of the 3 main experiments inspected in this thesis is the DevRep problem. This is a problem which was devised by this thesis. The DevRep is a combination of the two previous problems, replication and development. The goal of the problem is to create a genotype that is capable of developing into a structure, then replicating the structure afterwards. It is a novel problem expected to be hard. It is not studied in the literature, but might be important towards developing self-replicating machines. The general results of this problem can be seen in Table: 4.8.

Transition Table						
Pattern	0a	xa	xb	1a	2a	3a
Success rate %	96	52	23	0	0	0
Avg(NumGen)	877	1756	2005	x	x	x
StDev(NumGen)	1228	1910	1776	x	x	x
Instruction-Based Approach						
Pattern	0a	xa	xb	1a	2a	3a
Success rate %	71	0	0	0	0	0
Avg(NumGen)	3032	x	x	x	x	x
StDev(NumGen)	2737	x	x	x	x	x
Self-Modifying Instruction-Based Approach						
Pattern	0a	xa	xb	1a	2a	3a
Success rate %	94	0	1	0	2	8
Avg(NumGen)	1913	x	9565	x	7652	5818
StDev(NumGen)	1944	x	x	x	2047	2973

Table 4.8: DevRep problem comparison

This problem gave the most distributed results of the three problems explored in this thesis. 0a was solved by all the methods but TT and SMIBA solved it with greater success than IBA. After the successes on the 0a problem, solving the xa problem was expected to be a simple problem that all methods could solve, yet only TT was able to. This structure is the only case were TT out performed the other two methods to a large degree. When scaled to xb, SMIBA managed to solve it while TT saw a decrease in ability. When inspecting the resultset, SMIBA showed an on average decrease in fitness value when scaled on this problem, but showed a increase in standard deviation.

When it comes to the 1a structure none of the methods managed to solve this problem. This in itself would not be strange had it not been for SMIBA's ability to solve the 2a and 3a problem, which was expected to be harder. The only hypothesis made to explain this is that it again has to do with phenotype and or genotype variation. What caused 3a to be easier then 2a, could have been because 3a is a 4-state structure. The DevRep is a hard problem to solve and as such it could be that 2 cell-states are not enough to both develop and replicate the 1a structure, but with 3 cell-states enough variations can be made to create the 2a. This can be further pointed when considering 3a was solved more than 2a. That 3a was easier could also be due to 3a being a 4-state, and therefore better suited for binary operations in IBA, as discussed in earlier. Having one or more additional state may be necessary in hard problems such as the DevRep problem. This is unfortunately not studied in the literature due to the lack of scalability of TT.

As mentioned 0a was the only problem solved by all methods, therefore an inspection of the different strategies utilized by the methods was performed. At least 10 solutions were inspected from all the methods. Firstly, in TT, all exhibited strategies also observed in TT on the development problem. The solution was build up successfully but afterwards the CA descended into a loss of structure and chaotic behaviour, yet often symmetrical over one axis. This can be seen in Figure: 4.10 where the structure is lost after 24 steps and never recovered. This was also observed to happen in some of the SMIBA solutions. Secondly, IBA showed zero phenotypical variations in its solutions. All inspected solutions in IBA showed many different genotype programs, but all solutions gave the same phenotype solution. This phenotype solution can be seen in Figure: 4.11. Finaly, SMIBA showed greater variation in its solutions, it showed solutions similar to TT, but it also showed solutions that exhibited behaviour that was desired. Several solutions in SMIBA developed a single solution first, which then replicated, sometimes seemingly indefinite or as long as there is space, as can be seen in Figure: 4.12. In this figure the structure can be seen to reform in step 3,7,11,15,19,23,27 and 35. This SMIBA program was then moved to a larger lattice and showed continued growth Figure: 4.13, here the program generated a total of 61 replications at one point and showed no sign of stopping. It was not possible to test on a larger lattice then 75, the CA implemented in this thesis was never designed to handle lattices of such size. Note that the given solution

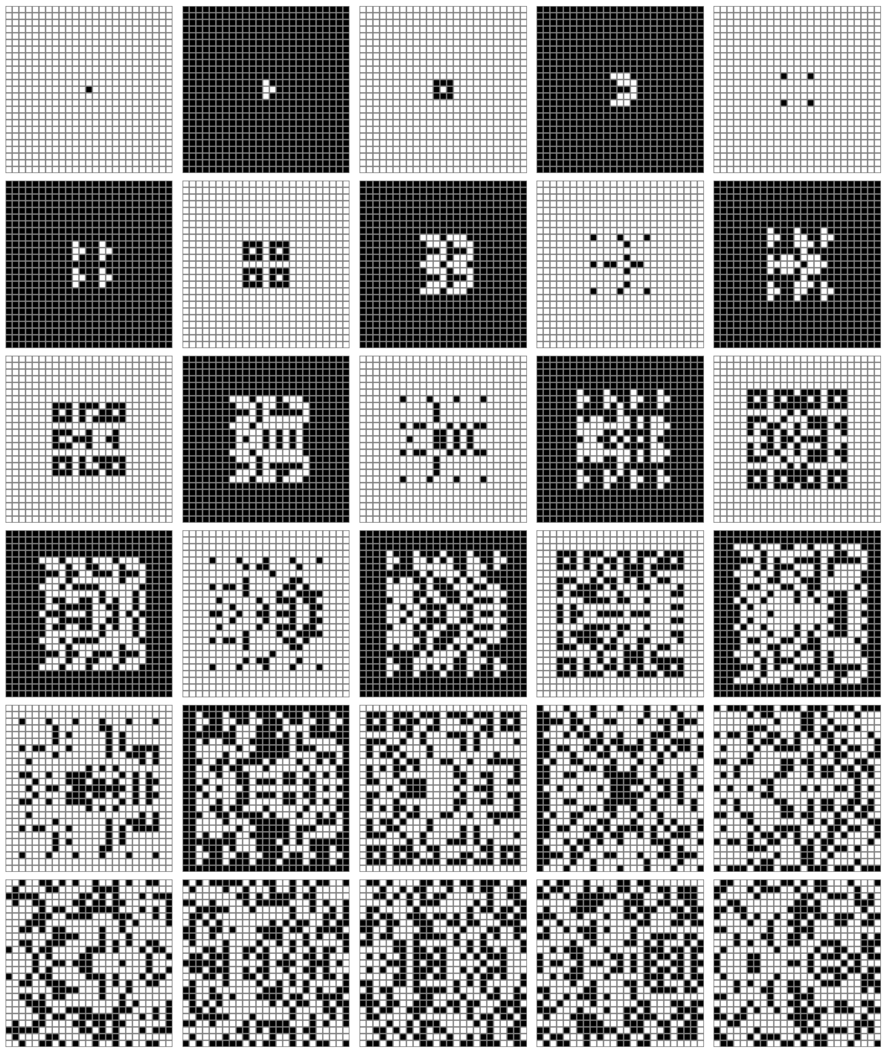


Figure 4.10: TT solution to the EvoDevo problem 0a.

creates replicas as long as there is space. Where space is not available it adopts a strategy to move the structures, so that replication may continue.

4.5.1 Solutions outside the intention

There are times when things do not go according to plan, and it seems the fitness function allowed for several alternative interpretations of the problem than

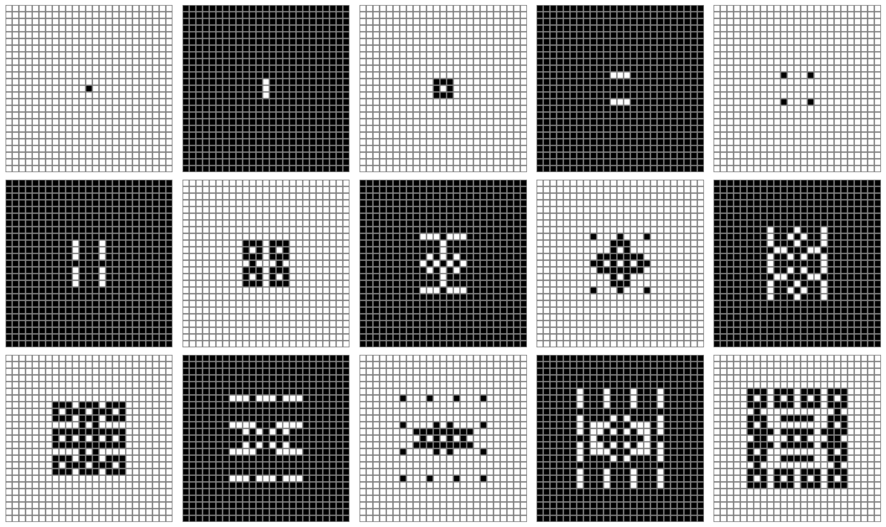


Figure 4.11: IBA only type of solution. All cases inspected gave this type of solution, but many different programs.

was intended. On occasions TT and SMIBA solved the problem by developing multiple structures the first time the target image is formed. This can be seen in Figure: 4.14 where the target structure is formed 6 times while later it is merely formed 4 times. Such solutions are hardly general replicators. This happened exclusively in TT and SMIBA, but one would think IBA would also be capable of this feat, possibly on different structures. They are able to develop artistic patterns instead of randomized behaviour. In this case heart shaped structures emerge. Outside the scope of this thesis such solutions may be used in picbreeders[76] where complex structures could develop and replicate.

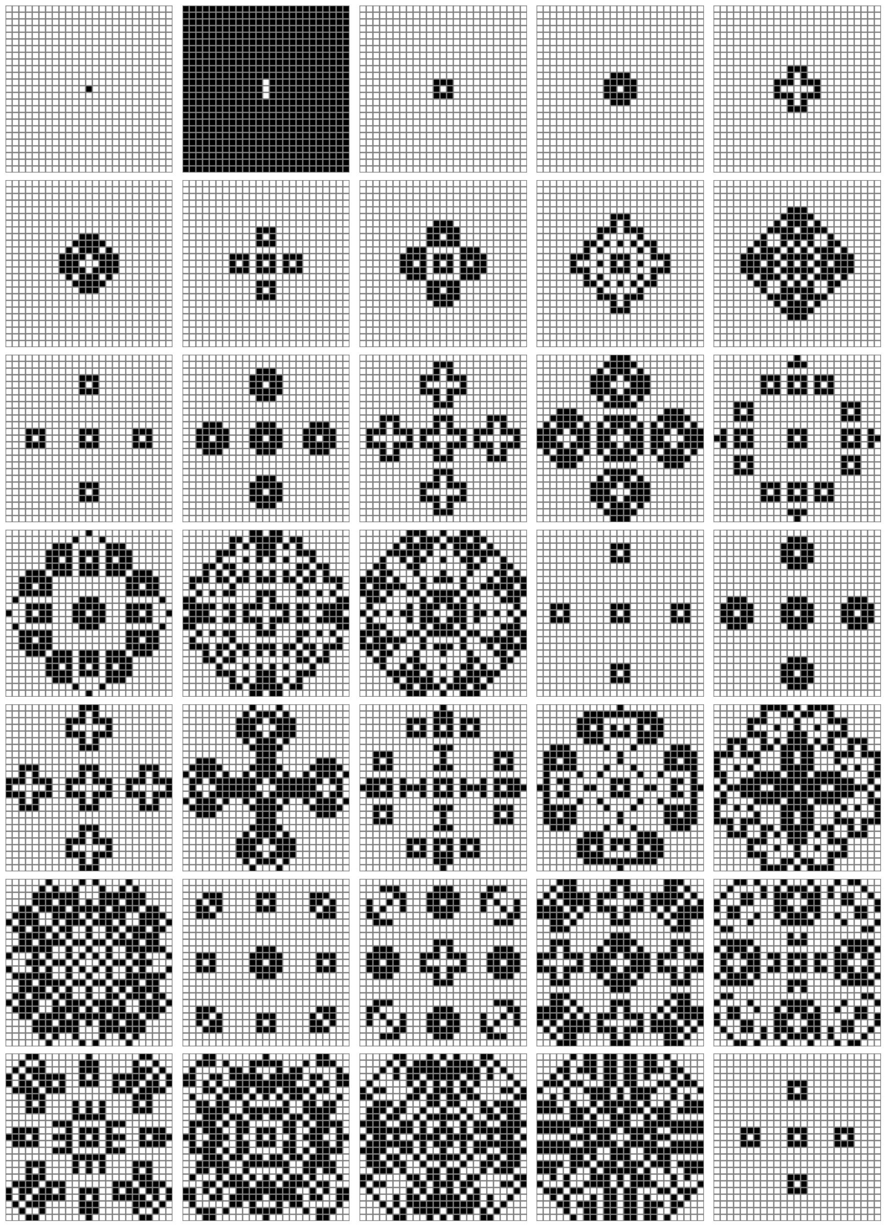


Figure 4.12: Example SMIBA solution to 0a.

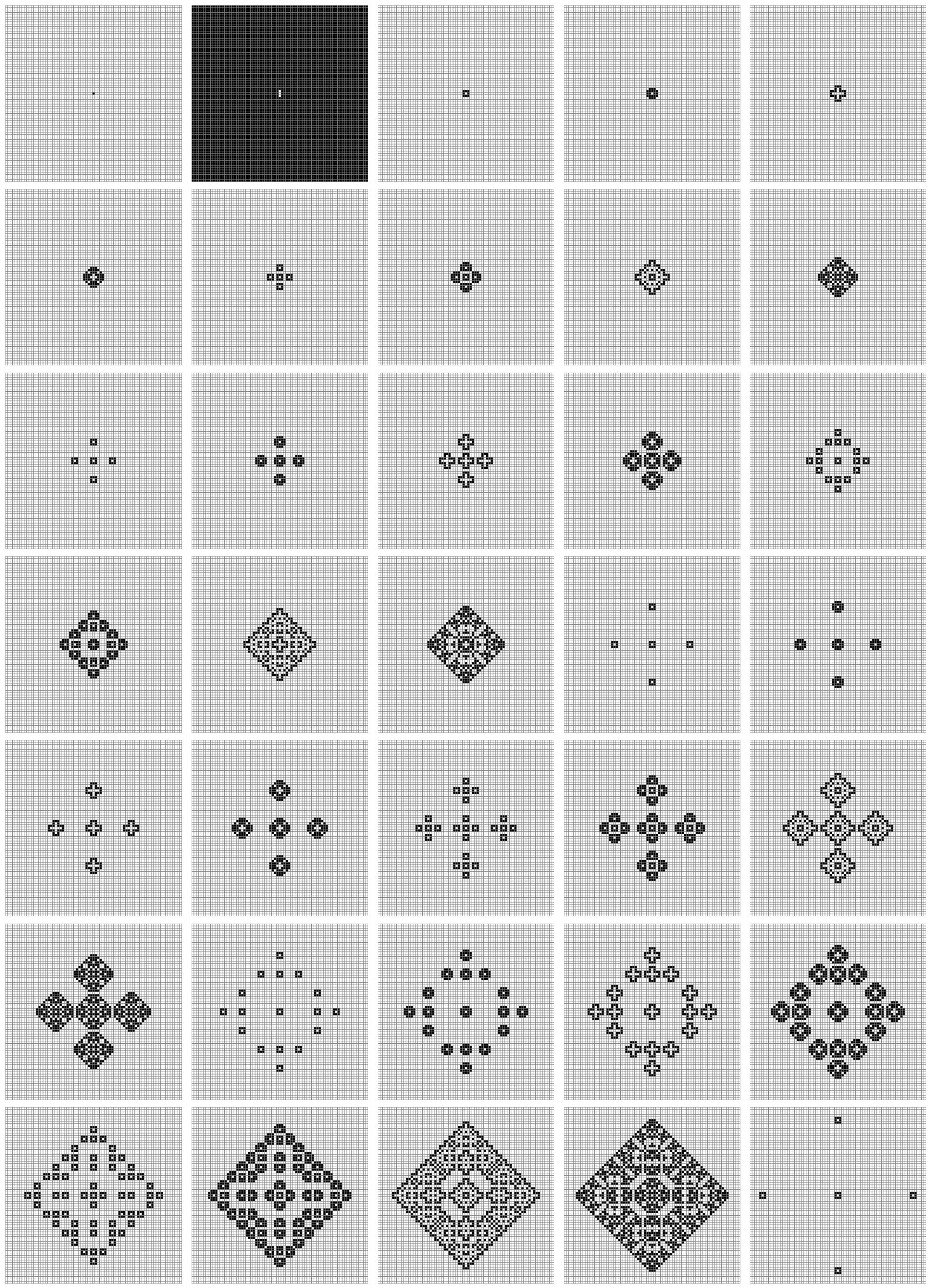


Figure 4.13: SMIBA solution to 0a. The previous solution on a large 75*75 lattice. At state nr. 30 a total of 61 replicates despite the fitness function only ever asking for 3 the program replicates and spreads out as long as there is room to grow across the lattice.

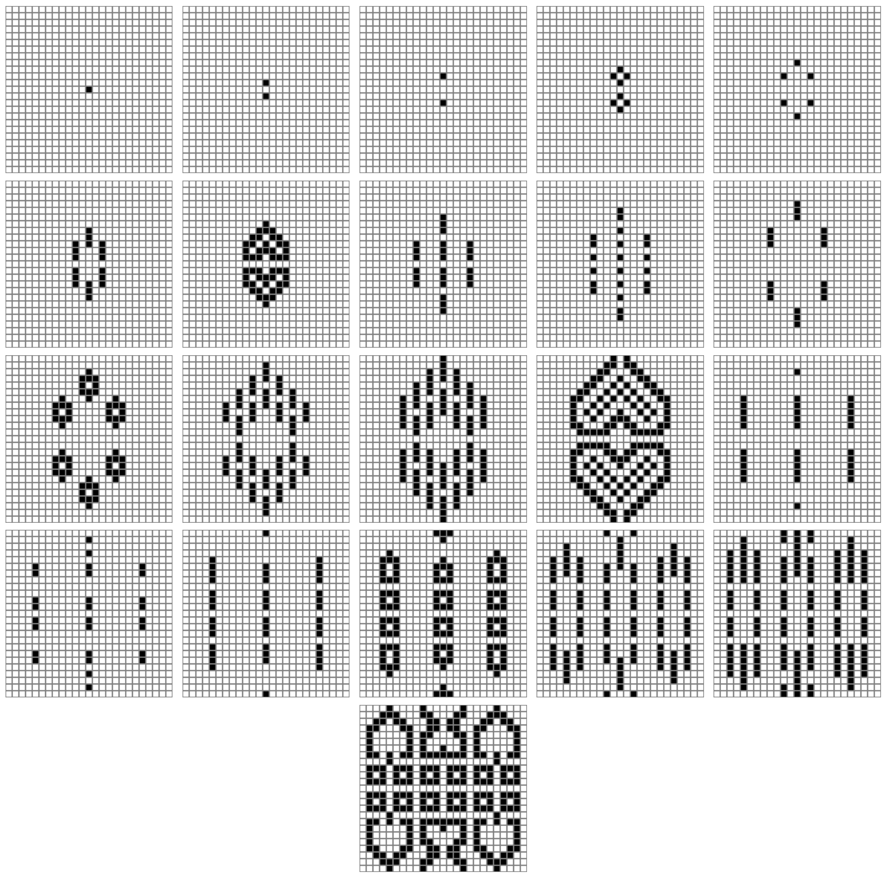


Figure 4.14: A (lovely) solution in SMIBA to the 0a structure.

4.6 Re-Evolving the phenotype, and the notion of modularity

Taking inspiration from [71] where Kovitz experimented with re-evolving a problem to inspect evolvability, a similar, yet different experiment was performed in this thesis. In replication, solutions that could solve several structures at with the same genotype have been identified. Therefore, re-evolving replication was expected to be a trivial task. For this reason re-evolving was performed on development instead. In addition, re-evolving development problems, allow for tracking the developmental trajectories in solutions, since all solutions start from the same zygote. A downside is, that development problems are shown as quite hard. Re-evolving to different problems several times would require a lot of computation, and therefore take a long time to perform. In this implementation problems that were inspected are harder then the problems inspected in Kovitz paper [71]. Therefore only a single re-evolve was performed. This does not allow us to inspect if solutions are drawn to evolvability, but it does allow us to see if the system intrinsically has some evolvability.

This experiment used the same parameters as the development problem, the only change was that if an execution found a solution to the first problem, it would by using the same population try to find a solution to a second and final problem.

Explaining the graphs

From these experiments a few case solutions were inspected and plotted in Pajek. These figures are plotted using the following rules. The first problem solution is marked with black vertices and black edges. The second is marked with blue vertices and blue edges. The dots are labelled with "a" for first problem solution and "b" for second solution. The number in the label is the first time the state occurred in the iterations of the CA. Vertices marked b22 for example are vertices in the second solution and occurred at step number 22 in the CA. If a "b" state points to an "a" state, this means that the same state was observed in both solutions. Note that in SMIBA solutions cycles are very common. These cycles do not behave as normal cycles, as they are in fact escapable. This illustrate well the property of delayed development, identified in SMIBA earlier.

4.6.1 Evolving from $3a \to 2b$

The first problem explored is one where the GA first evolves a 3a structure then the target of the fitness function is changed to a 2b. It was a 2b because it has the same number of states as a 3a structure. Due to the similarity of the two structures this re-evolving was expected to be easy compared to other structures. This experiment was performed in both SMIBA and TT and the results are summarised in Table: 4.8. In the aforementioned table a hemming distance calculation is performed on TT and SMIBA. Hemming distance is a measurement of how different two equal length strings are. It is measured by comparing each character of the string to the same character position in the other string. Simply put, it is how many changes need to be made to a string to turn it into the other sting.

in [2] and everything marked (R), the SMIBA has much of its unused genotype material removed from the calculations. It was removed in this manner, if the SMIBA program did not include a CHF or CHP instruction, then in each instruction, every unused parameter was not considered in the hemming distance calculation. Table: 4.9 also contains a deeper analyses of hemming distance with regard to self-modifing instructions(SMI), regular IBA instructions and parameters.

TT

After having executed this experiment 100 times in TT, only 27 of these runs managed to solve the first problem, and 24 managed to solve both.

SMIBA

When this was attempted with SMIBA the first structure was formed in 89 of the 100 attempts. Of these 89, only 84 managed to find a solution to the second problem.

[1] Average Hemming Distance

[2] SMIBA, if no CHF or CHP instruction is present in the program parameters not used by the instructions was not counted and if no Move or Dupe neither in the program, the end of the programmed was trimmed of Instructions that did not do anything(never changed the center). The second case was very rare

[3] (average)

[4] Self-modifying Instruction

SMIBA vs TT			
	1st	2nd	
TT	27	24	
SMIBA	89	84	
Genotype change			
	A.H.D.[1]	Max	%
TT	401	1024	39%
SMIBA	24.46	50	49%
SMIBA(R)[2]	14.73	(34.61)[3]	42.55%
Instructions(R)	3.54	(9.988)	35.40%
Param(R)	11.19	(24.62)	45.45%
IBA→SMI[4](R)	0.46	(7.70)	5.97%
IBA→IBA(R)	2.40	(7.70)	31.17%
IBA(R)	2.86	(7.70)	37.14%
SMI→IBA(R)	0.57	(2.29)	24.89%
SMI→SMI(R)	0.11	(2.29)	4.80%
SMI(R)	0.68	(2.29)	29.67%

Table 4.9: SMIBA and TT comparison, on the 3a→2b problem.

4 of the solutions managed to solve both problems with the first solution. These solutions would have too high degree of trajectory overlap to make them interesting as a case. The solutions that quickly, but not instantly, solved the second problem were more likely to contain a degree of overlap, without overlapping completely(i.e. same solution). 9 of the executions found a solution to the second problem within 100 generation,and these cases were inspected. As expected these cases often contained overlapping trajectories, especially in the initial steps, as can be seen in Figure: 4.15. Another example can be seen in Figure: 4.16 where initial steps are similar, before splitting ways. In the aforementioned case note how the overall trajectory structure is a mirror of each other.

In Table: 4.9 an interesting property was observed. When splitting hemming distance into subcategories of the SMIBA, a hierarchical pattern emerged. When re-evolving more parameters were changed than instructions, and more IBA instructions then SM instructions were also changed. This suggests that SM instructions make large changes, IBA instructions medium changes and parameters small changes. This variation might be one of the properties that makes

SMIBA vs TT			
	A.H.D.	Max	%
SMIBA	24.2	50	48.40%
SMIBA(R)	16.6	(34.4)	48.26%
Instructions(R)	2.20	(10)	22.00%
Param(R)	14.4	(24.4)	59.02%
IBA→SMI(R)	0.60	(8.2)	7.32%
IBA→IBA(R)	1.4	(8.2)	17.07%
IBA(R)	2.00	(8.2)	24.39%
SMI→IBA(R)	0.20	(1.8)	11.11%
SMI→SMI(R)	0.00	(1.8)	0.00%
SMI(R)	0.20	(1.8)	11.11%

Table 4.10: SMIBA and TT comparison. On the 4b→2b problem

SMIBA strong across the board of problems inspected in this thesis. Through the hierarchical structure the genotype might evolve where small changes to the genotype might cause large or small changes in the phenotype, allowing SMIBA to make large leaps in the fitness landscape, while also retaining the ability for small detailed leaps.

4.6.2 Evolving from $4b \to 2b$

A second re-evolution experiment was also performed with the SMIBA on the 4b to the 2b structure. Only 5 of the 100 were successful in finding the 4b structure. All of them then successfully found the second 2b structure. In the previous 3a to 2b experiment, the 3a and 2b has a similar column based structure, but 3a is a 4-state structure while 2b is a 3 state structure represented in 4 state CA. In this experiment 4b and 2b are both 3-state structures in 4-state CA, but 4b is symmetrical over two axes while 2b over one. The results are summarised in Table: 4.10, in the same manner as the previous 3a-2b experiment. Here the hierarchical property in SMIBA is even more apparent. Also in these cases the trajectories often showed a similar development path initially as seen in Figure: 4.17. These results are built from only 5 cases, and therefore some doubt exists about the accuracy of these cases as representative of a general behaviour.

Between the two experiments many cases with a shared initial trajectory were

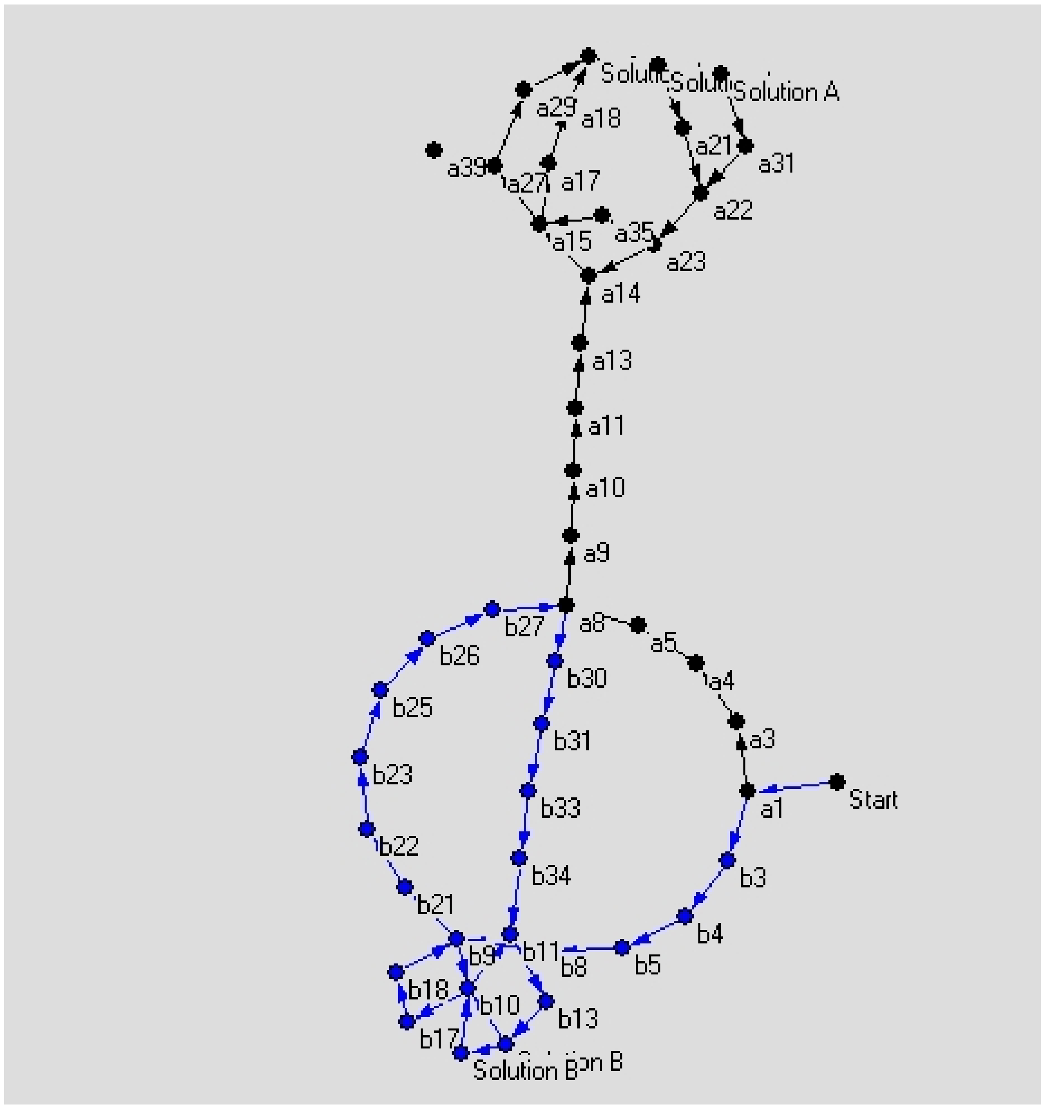


Figure 4.15: Using Pajek to draw a network for $3a \rightarrow 2b$. A highly interconnected case. Solutions to both the first and second problem were found several times, and are re-marked as a new state to highlight this.

identified. They were also found in TT cases of these experiments. These shared states are not replicators so it would be a stretch to call them stem-cell states, but they could rather be identified as "stem states". Through the same manner stem-cells may turn into different cells, these states could turn into different

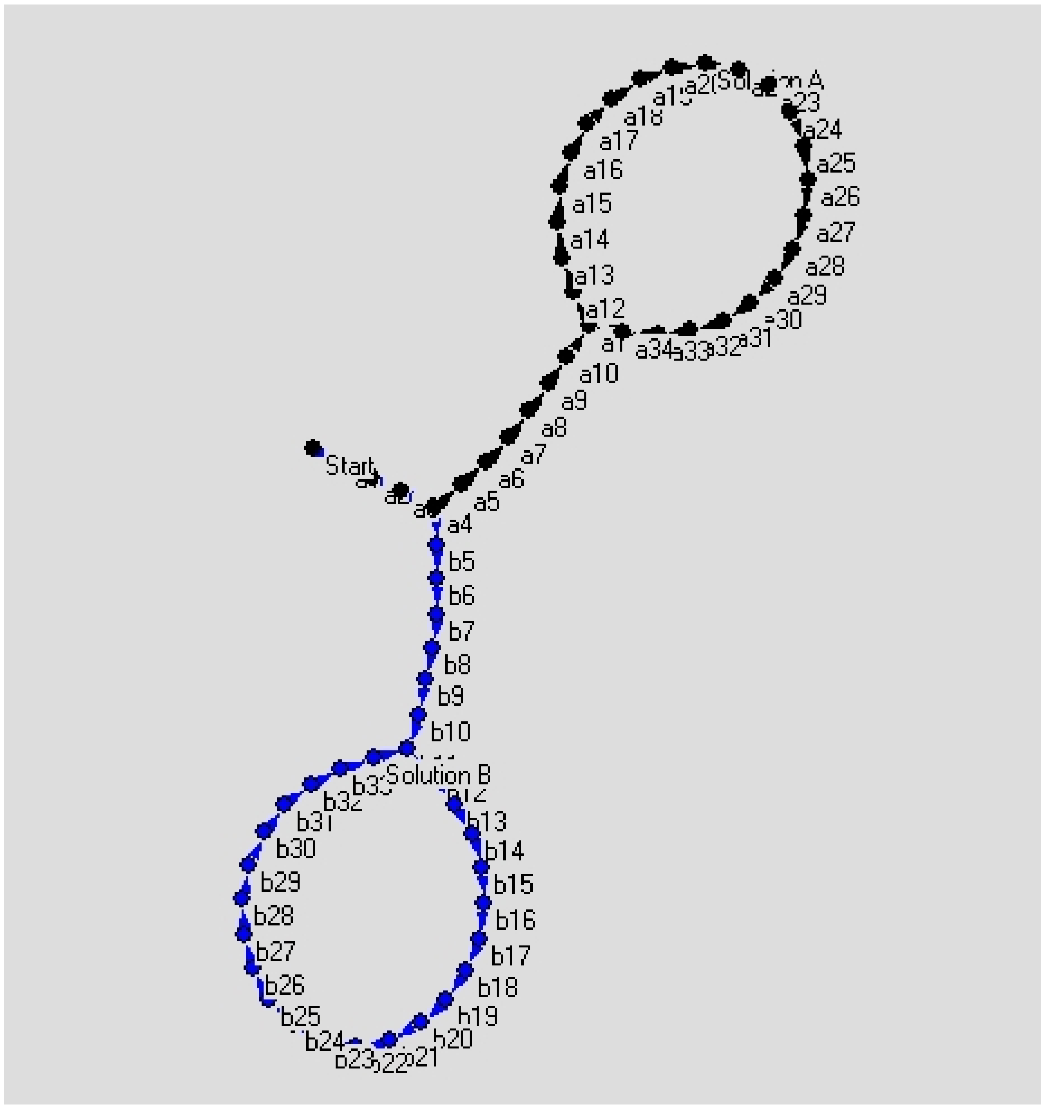


Figure 4.16: Using Pajek to draw a network for $3a \rightarrow 2b$. A case that found the second solution after 9 generations shows a similar developmental structure. The first 5 states are identical, then they move separately but similarly.

shapes, depending on the genotype.

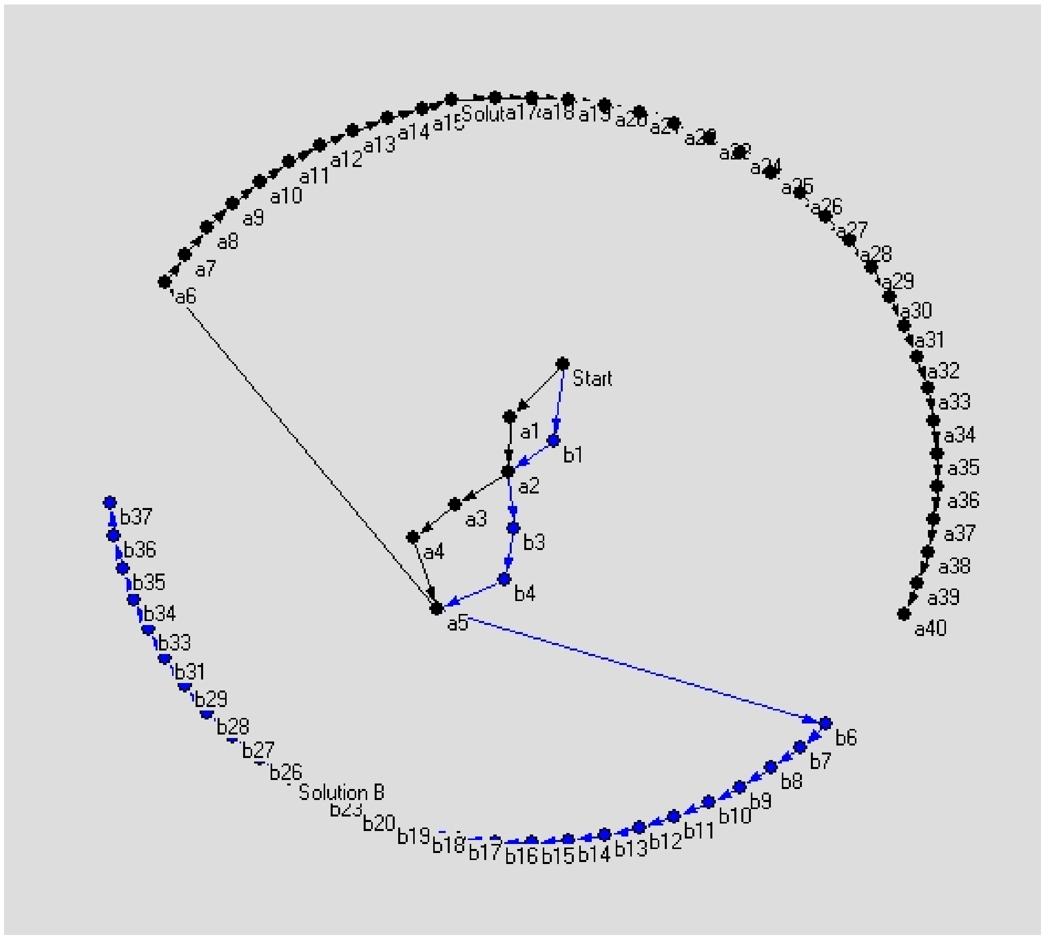


Figure 4.17: Pajek network example for $4b \rightarrow 2b$. This example showing an intertwined start.

4.7 Instruction Distribution

Through the previous experiments analysed in this chapter, a good few case solutions have been found. In IBA and SMIBA the solutions discovered through the experiments give us an opportunity to take a deeper look at what instructions or calculations are necessary or useful to solve the problems investigated in this thesis. This was done by filtering the data of any unsolved runs. From the resulting filtered dataset some simple statistical data was calculated in the following manner. For every case(program) in the dataset, if a specific instruction is in said case, it is counted. If a program contains more than one instruction of the same kind, only the first one is counted. The information is then summed up for each structure and each problem. This information can be seen in Table: 4.11, 4.12, 4.13, 4.14, and 4.15.

IBA Development					
Pattern	2a	3a	4a	total	%
AND	37	27	1	65	52%
OR	59	37	1	97	77.6%
XOR	52	28	1	81	64.8%
NOT	33	13	0	46	36.8%
INV	27	16	0	43	34.4%
MIN	50	38	1	89	71.2%
MAX	42	24	1	67	53.6%
SET	17	10	0	27	21.6%
INC	48	24	1	73	58.4%
DEC	45	26	0	71	56.8%
SWAP	25	4	0	29	23.2%
ROR	19	9	0	28	22.4%
ROL	33	14	0	47	37.6%
ROU	28	8	0	36	28.8%
ROD	21	14	0	35	28%
NOP	14	9	0	23	18.4%
Max	79	45	1	125	

Table 4.11: Development IBA instruction distribution

The first two tables presented concern the development problem. In this thesis, development problems required a good number of evolutionary generations to complete in comparison to the replication problems.

Taking this into consideration, one could expect the instruction distribution in this problem to be highly specialised or uniform. The statistics in the tables do not show this, since the instructions are in fact very evenly distributed, at least in comparison to what was expected. This is also true when considering the individual structures. Even so, some instructions are used more prominently than others.

Looking at the instructions NOOP, SKIP and DEL, which are instructions that in effect shorten the genotype, one notices that they are not very commonly used in the development problem. A slight decrease in use of these instructions also occurred in harder problems. This would indicate that the harder the problem, the more instructions would be useful. There are also some slight discrepancies in how instructions are used in comparison between IBA and SMIBA. considering SMIBA has a larger instruction set, the rotating instruction ROR, ROL, ROU and ROD are more often used in SMIBA. Also note that it seems that MOVE instruction is a clear winner among the SM instructions.

Moving on to the replication problem, this problem was often very quickly solved, many of the structures averaged less then 100 generations. The exception to this quick evolving speed is the 6a structure. In [47] it was shown that the many replication problems were easily solved, and could be solved with only 2 or 1 instructions, in some cases. Given that in all problems replicated included a XOR instruction as seen in Table: 4.13 and 4.14, it seems likely, that it is this instruction that causes IBA to be strong in replication. Note that the XOR instruction can be replicated using other instructions. For example $p \oplus q \equiv (p \vee q) \wedge \neg(p \wedge q)$, but such a relation would be much harder to create for the GA used in this thesis.

Given that the replication problem seems too easy to solve and relies heavily on the XOR instruction, a short experiment was performed on whether or not IBA would be able to solve the replication problem without the use of XOR. This was done by simply making any XOR instruction act exactly like a NOOP instruction and evolving the replication problem as earlier using this small modification. This short experiment used only 10 runs as opposed to the normal 100 performed in all other experiments. This was performed on the 1a structure, and the results were that none of the runs managed to solve the problem. In fact all of the runs ended at 135/150 as fitness values. This shows quite strongly that XOR is very useful in replication.

The 6a structure was solved using both a DEC and a XOR in all cases both in

IBA and SMIBA. MOVE and ROU also had a strong presence in SMIBA solutions on this structure.

The replication problem could easily be solved without the use of SM instruction. This is also present in Table: 4.14, where it was found that some of the solutions were solved without the presence of a single SM instruction.

It was theorised earlier that 2a and 6a would show a different distribution of instructions used in replication due to them being 3-state as opposed to the other problems. The data show only small trends in 2a IBA where the NOT instruction is much less common, and the rotating instructions ROR, ROL, ROU and ROD are slightly more common. In SMIBA 6a showed some strong trends discussed earlier, but 2a showed a small trend of not relying on the MOVE instruction to the same degree as other structures.

Finally we move to the DevRep problem as seen in Table: 4.15. In this table the IBA and SMIBA are seen side by side on the 0a problem. In this statistic some very large differences are apparent. Many of the same trends seen in the other two problems are present here. The AND instruction is very commonly used in IBA, but in SMIBA it is very uncommon. It is also interesting to note that IBA solved the DevRep problem for the 0a structure without using XOR in a handful of the cases, despite very heavily relying on that instruction in the replication problem. Also note that in the DevRep problem SMIBA had a much larger use of the rotating instructions ROR, ROL, ROU and ROD compared to IBA.

SMIBA Development						
Pattern	2a	3a	4a	4b	total	%
AND	12	22	5	4	43	20,87%
OR	69	61	3	2	135	65,53%
XOR	53	59	2	2	116	56,31%
NOT	29	23	0	1	53	25,73%
INV	56	28	0	4	88	42,72%
MIN	56	65	0	2	123	59,71%
MAX	20	31	0	0	51	24,76%
SET	41	19	1	2	63	30,58%
INC	32	60	4	2	98	47,57%
DEC	26	52	2	2	82	39,81%
SWAP	35	24	0	3	62	30,10%
ROR	41	37	4	2	84	40,78%
ROL	52	40	3	2	97	47,09%
ROU	41	31	3	0	75	36,41%
ROD	36	36	3	3	78	37,86%
NOP	22	5	1	2	30	14,56%
SKIP	20	13	1	0	34	16,50%
MOVE	64	66	4	4	138	66,99%
DUPE	51	40	2	1	94	45,63%
DEL	16	14	1	0	31	15,05%
CHF	28	22	1	1	52	25,24%
CHP	27	29	0	1	57	27,67%
One or more SM	100	96	5	5	206	100,00%
Max	100	96	5	5	206	

Table 4.12: Development SMIBA instruction distribution

IBA Replication									
Pattern	1a	2a	3a	4b	5a	6a	6b	total	%
AND	31	38	40	36	23	1	41	210	34.77%
OR	30	35	29	35	41	4	34	208	34.44%
XOR	100	100	100	100	100	4	100	604	100.00%
NOT	48	21	44	35	35	4	38	225	37.25%
INV	47	36	41	40	42	1	42	249	41.23%
MIN	34	36	36	41	30	3	37	217	35.93%
MAX	38	33	35	39	32	3	38	218	36.09%
SET	43	47	47	47	52	0	40	276	45.70%
INC	48	39	49	36	30	2	34	238	39.40%
DEC	39	39	36	45	38	4	47	248	41.06%
SWAP	58	49	47	47	49	1	49	300	49.67%
ROR	36	42	42	53	46	1	56	276	45.70%
ROL	43	54	44	49	47	3	49	289	47.85%
ROU	41	52	45	40	45	1	44	268	44.37%
ROD	42	50	39	43	46	0	48	268	44.37%
NOP	47	49	44	49	42	0	36	267	44.21%
Max	100	100	100	100	100	4	100	604	

Table 4.13: Replication IBA instruction distribution

SMIBA Replication									
Pattern	1a	2a	3a	4b	5a	6a	6b	total	%
AND	32	33	30	26	28	2	24	175	28,14%
OR	23	33	23	28	30	6	32	175	28,14%
XOR	100	100	100	100	100	22	100	622	100,00%
NOT	34	25	29	26	26	10	37	187	30,06%
INV	38	37	29	35	34	3	33	209	33,60%
MIN	32	24	25	20	20	7	32	160	25,72%
MAX	26	31	31	23	36	1	20	168	27,01%
SET	41	32	43	40	34	9	35	234	37,62%
INC	27	34	29	24	21	4	33	172	27,65%
DEC	23	25	29	33	21	22	39	192	30,87%
SWAP	39	37	38	44	43	5	37	243	39,07%
ROR	31	29	34	44	22	10	37	207	33,28%
ROL	42	31	41	32	30	7	29	212	34,08%
ROU	40	42	33	38	52	19	44	268	43,09%
ROD	40	41	40	43	36	13	29	242	38,91%
NOP	40	43	37	32	46	5	36	239	38,42%
SKIP	37	41	43	47	37	4	38	247	39,71%
MOVE	43	22	32	41	49	20	40	247	39,71%
DUPE	32	25	30	25	32	4	24	172	27,65%
DEL	16	21	32	17	22	1	16	125	20,10%
CHF	28	29	25	28	27	4	34	175	28,14%
CHP	36	39	36	36	34	3	37	221	35,53%
One or more SM	90	90	93	94	92	21	92	572	91,96%
Max	100	100	100	100	100	22	100	622	

Table 4.14: Replication SMIBA instruction distribution

Development Replication(DevRep)				
Pattern	0a IBA	%	0a SMIBA	%
AND	42	59.15%	17	18.09%
OR	40	56.34%	43	45.74%
XOR	61	85.92%	94	100.00%
NOT	32	45.07%	10	10.64%
INV	18	25.35%	23	24.47%
MIN	32	45.07%	43	45.74%
MAX	33	46.48%	13	13.83%
SET	9	12.68%	23	24.47%
INC	26	36.62%	16	17.02%
DEC	28	39.44%	13	13.83%
SWAP	21	29.58%	31	32.98%
ROR	16	22.54%	33	35.11%
ROL	24	33.80%	39	41.49%
ROU	18	25.35%	41	43.62%
ROD	19	26.76%	43	45.74%
NOP	24	33.80%	20	21.28%
SKIP			15	15.96%
MOVE			50	53.19%
DUPE			54	57.45%
DEL			36	38.30%
CHF			41	43.62%
CHP			37	39.36%
One or more SM			94	100.00%
Max	71		94	

Table 4.15: Development Replication SMIBA and IBA

Chapter 5

Conclusion

"If you trust in yourself... and believe in your dreams... and follow your star... you'll still get beaten by people who spent their time working hard and learning things and weren't so lazy."

Terry Pratchett - The Wee Free Men

In this chapter, we will summarise the findings of the previous chapter, and present them in relation to the research questions and goal introduced in the introduction. Finally, a section where possible directions for future work is described.

5.1 SMIBA in comparison

Through the build up of this thesis, the second research question is answered first. The research question was formulated in the following sentence.

RQ2: How would a method combining both IBA and Self-modification from SMCGP perform in comparison to regular IBA and explicit representations (Transition Tables)?

This question was answered by the results from the development, replication, and DevRep problems. In development, SMIBA showed itself to be stronger than the other methods on a number of structures in solving effectiveness and efficiency. Structures 5b and 6a were a small exception, as they were solved in rare cases by the TT. In replication, SMIBA gave evidence of relying well on its IBA base, but also improved on IBA by solving 6a more often. It also showed itself to be better at producing replications that were also replicators. In the

DevRep problem, SMIBA struggled with the xa structure but managed to solve it when scaled. It was the only method that was capable of finding a solution with the 2a and 3a structures. When inspecting cases, SMIBA showed a much more desirable behaviour on the 0a problem when considering CA behaviour of class given in [26, 18]. Given these results, SMIBA has shown itself a strong addition to IBA, as it outperforms IBA on a number of structures and problems. In comparison to TT, SMIBA showed stronger results on most problems with xa in DevRep, and 6a and 5b in development being the only exceptions.

5.2 Development Replication

The first question of this thesis was formulated in the following manner.

RQ1: Is it possible to evolve a single genotype in cellular automata that can solve both the replication and the morphogenesis problem together?

The DevRep problem was devised to answer this question. All the methods managed to solve this problem with the simplest structure. However, as soon as the structure scaled in number of states or cell size, the problem became very hard to solve for any of the methods. On this problem, SMIBA showed the most advantageous behaviour when looking at it in detail, as it could regulate its genotype through the development phase and replicate very successfully afterwards.

In general, the answer to this question would be yes, it is possible, but it is very hard.

5.3 Stem-Cells

The third question raised in this thesis is formulated in the following manner.

RQ3: Is a stem-cell like feature necessary or at least useful to develop multicellular organisms, and will it self emerge given incentive?

Through the use of Pajek, some stem-states were identified in the trajectories. Therefore, some stem-cell like behaviour is already inherent in the methods, as the cases which quickest solved the second problem often had overlapping trajectories. SMIBA was devised to be strong on replication and development, as is key in stem-cells. Further, SMIBA showed itself to be more effective than the other methods when moved out of the original bounds of the environment it was evolved in. Another property in stem-cells is a hierarchy, which we will look into in a later section.

5.4 Scaling

The fourth question regards scaling and is formulated in the following sentence.

RQ4: Does SMIBA scale better in problem solving?

On this question, SMIBA showed great promise when faced with scaling problems. This in the sense that it managed to solve a number of problems that seemed too hard for the other two methods, it also often improved in efficiency in problem solving ability when the number of states was increased. SMIBA together with IBA has the ability to scale linearly in runtime. This is because the program size may remain constant when the problem is scaled, but often a harder problem would benefit from more instructions.

5.5 Hierarchy

The fifth and final question is about hierarchy in relation to the previous question.

RQ5: Is a hierarchical genotype a good answer to the problem of scaling?

In this regard, SMIBA is the most hierarchical genotype representation as was discussed in the Re-Evolve section. The analyses of hemming distance in this experiment showed, SMIBA to use this potential hierarchical structure and through the previous question, SMIBA was shown to scale exceptionally well. This is enough to claim that there is a correlation between hierarchy and scaling. SMIBA, as a more hierarchical method, showed itself to scale better than IBA, since it could solve a number of harder problems IBA could not. On the other hand, the scaling could also be caused by some or one of SMIBAs other properties, not found in IBA.

5.6 Similarities of SMIBA and biological systems

SMIBA splits the genotype into two parts of encoding and non-encoding genes. The encoding genes, i.e. the IBA instructions, directly alter the phenotype, while non-encoding, i.e. SM instructions, regulate the genotype. The non-encoding genes modify the coding genes by activating, deactivating, changing, or removing them. This allows for different parts of the genome to be active or inactive at different stages during development. This is much like in biology, where the genome regulated in the same way of encoding and non-encoding genes.

5.7 Overview

The five previous questions were all created to answer the research goal as was formulated in the following manner.

RG: Create a novel algorithm SMIBA, test it against already existing methods, and identify strengths, weaknesses and properties in SMIBA.

Through these questions and deeper analyses as presented in the results section, it was shown that SMIBA had many unique strengths and properties. During development it was shown that SMIBA could regulate its genotype in order to slow down development, allowing the structure to form more slowly and determined. SMIBA also showed use of its hierarchical structure when re-evolved. It was shown to be very strong on a number of problems the other methods struggled or failed at. SMIBA seemed weak on problems that seemingly could only be solved though chaotic CA, and this would conclude well in relation to the "No Free Lunch" theorem [72]. Note that the same could be said for IBA. Overall SMIBA was shown to be an excellent extension of IBA.

5.8 Future Work

Many of the results in thesis were unexpected and surprising in a number of ways. This opens up a number of potential directions, which can be examined as they were outside the scope of this thesis.

- The instruction distribution showed a very clear favour of XOR in the replication problem, and once removed, the problem became too hard for IBA to solve. Given this information, it could be possible that XOR-like mathematical operations are being performed in biological replication and they may even be necessary for it.

- Another direction that could be explored is through the IBA to TT converter described in the previous chapter. It could be implemented and used to trace IBA to TT relations. These relations could be used to figure out what IBA lacks when compared to TT, and areas of the possible TT space could be used to develop additional instructions which would be useful to add to the IBA instruction set. As a transitive property these instructions would also be useful in SMIBA. For example it could prove that IBA is bad at accessing chaotic CA. It could then be used in combination with testing different alternative instructions, to figure out what would be required for IBA to be able to access it.

- In scaling, IBA and SMIBA use implicit representation, and through this definition they can scale in genotype size on demand, as was put into effect in [47]. In this thesis it was shown that IBA and especially SMIBA could scale when phenotype size was increased, i.e. increasing the number of possible states in the CA. In fact many times when the number of states was increased, SMIBA and IBA solved problems they previously were unable to solve. That being said it could be possible to let evolution decide on a phenotype size when solving problems. This could then be extended to let evolution decide genotype size as well. It could be possible that through such an experiment a relation between phenotype and genotype size could be found that would be useful when designing complex systems.

- When it came to the DevRep problem presented in this thesis, it was shown to be very hard. Through the course of this thesis a few alternative approaches to solve this problem have been discussed. One could solve the development problem and replication problem separately. Due to IBA and SMIBA having free control of genotype size, one could concatenate the programs from the previous replication and development solutions to each other. This could, in theory, create a good starting point for the GA to evolve from. Alternatively, the set of programs from the separate several replication and development runs could be placed in the same population and provide case solutions for the GA to freely select from.

Bibliography

[1] Von Neumann, John, and Arthur W. Burks. "Theory of self-reproducing automata." IEEE Transactions on Neural Networks 5.1 (1966): 3-14.

[2] Mitchell, Melanie. "Life and evolution in computers." History and philosophy of the life sciences (2001): 361-383.

[3] Von Neumann, John. "The general and logical theory of automata." Cerebral mechanisms in behavior (1951): 1-41.

[4] Wolfram, Stephen. "Statistical mechanics of cellular automata." Reviews of modern physics 55.3 (1983): 601.

[5] Wolfram, Stephen. "Universality and complexity in cellular automata." Physica D: Nonlinear Phenomena 10.1 (1984): 1-35.

[6] Toffoli, Tommaso, and Norman Margolus. Cellular automata machines: a new environment for modeling. MIT press, 1987.

[7] Cook, Matthew. "Universality in elementary cellular automata." Complex Systems 15.1 (2004): 1-40.

[8] Minsky, Marvin L. "Recursive unsolvability of Post's problem of "tag" and other topics in theory of Turing machines." Annals of Mathematics (1961): 437-455.

[9] Cocke, John, and Marvin Minsky. "Universality of tag systems with P= 2." Journal of the ACM (JACM) 11.1 (1964): 15-20.

[10] Bidlo, Michal, and Jaroslav Skarvada. "Instruction-based development: From evolution to generic structures of digital circuits." KES Journal 12.3 (2008): 221-236.

[11] Bidlo, Michal, and Zdenek Vasicek. "Evolution of cellular automata using instruction-based approach." Evolutionary Computation (CEC), 2012 IEEE Congress on. IEEE, 2012.

[12] Fogel, David B., and J. Wirt Atmar. "Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems." Biological Cybernetics 63.2 (1990): 111-114.

[13] Lanfear, Robert, Hanna Kokko, and Adam Eyre-Walker. "Population size and the rate of evolution." Trends in ecology & evolution 29.1 (2014): 33-41.

[14] Mitchell, Melanie, and John H. Holland. "When will a genetic algorithm outperform hill-climbing?." (1993).

[15] Harding, Simon L., Julian F. Miller, and Wolfgang Banzhaf. Cartesian Genetic Programming. Springer Berlin Heidelberg, 2011.

[16] Hortensius, Peter D., Robert D. McLeod, and Howard C. Card. "Parallel random number generation for VLSI systems using cellular automata." Computers, IEEE Transactions on 38.10 (1989): 1466-1473.

[17] Backus, John. "Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs." Communications of the ACM 21.8 (1978): 613-641.

[18] Wolfram, Stephen. A new kind of science. Vol. 5. Champaign: Wolfram media, 2002.

[19] Sayama, Hiroki. "Introduction to the Modeling and Analysis of Complex Systems." Open SUNY textbooks, Milne Library, State University of New York at Geneseo (2015).

[20] Ho, Anthony D., Ronald Hoffman, and Esmail D. Zanjani. Stem Cell Transplantation: Biology, Processing, and Therapy. Wiley-Blackwell; 1 edition, 2006.

[21] Kish, Laszlo B. "End of Moore's law: thermal (noise) death of integration in micro and nano electronics." Physics Letters A 305.3 (2002): 144-149.

[22] Powell, James R. "The quantum limit to Moore's law." Proceedings of the IEEE 96.8 (2008): 1247-1248.

[23] Mann, Charles C. "The end of Moores law." Technology Review 103.3 (2000): 42-48.

[24] Esmaeilzadeh, Hadi, et al. "Dark silicon and the end of multicore scaling." Computer Architecture (ISCA), 2011 38th Annual International Symposium on. IEEE, 2011.

[25] Chen, Trista P., and Yen-Kuang Chen. "Challenges and opportunities of obtaining performance from multi-core CPUs and many-core GPUs." Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on. IEEE, 2009.

[26] Langton, Chris G. "Computation at the edge of chaos: phase transitions and emergent computation." Physica D: Nonlinear Phenomena 42.1 (1990): 12-37.

[27] Langton, Christopher G. "Self-reproduction in cellular automata." Physica D: Nonlinear Phenomena 10.1 (1984): 135-144.

[28] Bedau, Mark A., et al. "Open problems in artificial life." Artificial life 6.4 (2000): 363-376.

[29] Aguilar, Wendy, et al. "The Past, Present, and Future of Artificial Life." Frontiers in Robotics and AI 1 (2014): 8.

[30] Standish, Russell K. "Open-ended artificial evolution." International Journal of Computational Intelligence and Applications 3.02 (2003): 167-175.

[31] Maley, C. C. "Four steps toward open-ended evolution." Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999). Vol. 2. 1999.

[32] Holland J.H., "Adaptive in Natural and Artificial Systems", University of Michigan, Ann Arbor, Michigan, USA, 1975

[33] Zitzler, Eckart, Kalyanmoy Deb, and Lothar Thiele. "Comparison of multiobjective evolutionary algorithms: Empirical results." Evolutionary computation 8.2 (2000): 173-195.

[34] Bar-Yam, Yaneer. Dynamics of complex systems. Vol. 213. Reading, MA: Addison-Wesley, 1997.

[35] Heylighen, Francis. "The science of self-organization and adaptivity." The encyclopedia of life support systems 5.3 (2001): 253-280.

[36] Mitchell, Melanie. Complexity: A guided tour. Oxford University Press, 2009.

[37] Card, Stuart K., Thomas P. Moran, and Allen Newell. "The model human processor: An engineering model of human performance." Handbook of Human Perception 2 (1986).

[38] Kandel, Eric R., James H. Schwartz, and Thomas M. Jessell, eds. Principles of neural science. Vol. 4. New York: McGraw-Hill, 2000.

[39] Deb, Kalyanmoy. Multi-objective optimization using evolutionary algorithms. Vol. 16. John Wiley & Sons, 2001.

[40] Siminovitch, Louis, Ernest A. McCulloch, and James E. Till. "The distribution of colony-forming cells among spleen colonies." Journal of Cellular and Comparative Physiology 62.3 (1963): 327-336.

[41] Becker, Andrew J., Ernest A. McCulloch, and James E. Till. "Cytological demonstration of the clonal nature of spleen colonies derived from transplanted mouse marrow cells." (1963).

[42] Vogel, Gretchen. "Capturing the promise of youth." Science 286.5448 (1999): 2238-2239.

[43] Rosin, Paul L. "Training cellular automata for image processing." Image Processing, IEEE Transactions on 15.7 (2006): 2076-2087.

[44] Wolfram, Stephen. "Random sequence generation by cellular automata." Advances in applied mathematics 7.2 (1986): 123-169.

[45] Poli, Riccardo, et al. A field guide to genetic programming. Lulu. com, 2008.

[46] Land, Mark, and Richard K. Belew. "No perfect two-state cellular automata for density classification exists." Physical review letters 74.25 (1995): 5148.

[47] Nichele, Stefano, and Gunnar Tufte. "Evolutionary growth of genomes for the development and replication of multicellular organisms with indirect encoding." Evolvable Systems (ICES), 2014 IEEE International Conference on. IEEE, 2014.

[48] Harding, Simon, Julian Francis Miller, and Wolfgang Banzhaf. "Self modifying cartesian genetic programming: Parity." Evolutionary Computation, 2009. CEC'09. IEEE Congress on. IEEE, 2009.

[49] Harding, Simon L., Julian F. Miller, and Wolfgang Banzhaf. "Self-modifying cartesian genetic programming." Cartesian Genetic Programming. Springer Berlin Heidelberg, 2011. 101-124.

[50] Harding, Simon, Julian F. Miller, and Wolfgang Banzhaf. "Developments in cartesian genetic programming: self-modifying CGP." Genetic Programming and Evolvable Machines 11.3-4 (2010): 397-439.

[51] Bentley, Peter J., and Sanjeev Kumar. "Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem." GECCO. Vol. 99. 1999.

[52] Darwin, Charles. "On the origins of species by means of natural selection." London: Murray (1859): 247.

[53] Darwin, Charles. The variation of animals and plants under domestication. Vol. 2. O. Judd, 1868.

[54] Knowles, Joshua D., Richard A. Watson, and David W. Corne. "Reducing local optima in single-objective problems by multi-objectivization." Evolutionary multi-criterion optimization. Springer Berlin Heidelberg, 2001.

[55] Wright, Sewall. "The evolution of dominance." American Naturalist (1929): 556-561.

[56] Suzuki, David T., et al. An introduction to genetic analysis. No. Ed. 3. WH Freeman and Company, 1986.

[57] Tschinkel, Walter R. "The nest architecture of the Florida harvester ant, Pogonomyrmex badius." Journal of Insect Science 4.1 (2004): 21.

[58] Mlot, Nathan J., Craig A. Tovey, and David L. Hu. "Fire ants self-assemble into waterproof rafts to survive floods." Proceedings of the National Academy of Sciences 108.19 (2011): 7669-7673.

[59] Lui, Leong Ting, et al. "Complexity Measurement Based on Information Theory and Kolmogorov Complexity." Artificial life (2015).

[60] Nichele, Stefano. "Evolvability, Complexity and Scalability of Cellular Evolutionary and Developmental Systems." (2015).

[61] Trefzer, Martin, et al. "On the Advantages of Variable Length GRNs for the Evolution of Multicellular Developmental Systems." Evolutionary Computation, IEEE Transactions on 17.1 (2013): 100-121.

[62] Sivanandam, S. N., and S. N. Deepa. Introduction to genetic algorithms. Springer Science & Business Media, 2007.

[63] Schaffer, J. David, and Amy Morishima. "An adaptive crossover distribution mechanism for genetic algorithms." Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms. Hillsdale, NJ: Lawrence Erlbaum Associates, Inc, 1987.

[64] Srinivas, Mandavilli, and Lalit M. Patnaik. "Adaptive probabilities of crossover and mutation in genetic algorithms." Systems, Man and Cybernetics, IEEE Transactions on 24.4 (1994): 656-667.

[65] Sipper, Moshe. Evolution of parallel cellular machines. Heidelberg: Springer, 1997.

[66] Lipson, Hod. "Principles of modularity, regularity, and hierarchy for scalable systems." Journal of Biological Physics and Chemistry 7.4 (2007): 125.

[67] Hartwell, Leland H., et al. "From molecular to modular cell biology." Nature 402 (1999): C47-C52.

[68] Kirschner, Marc, and John Gerhart. "Evolvability." Proceedings of the National Academy of Sciences 95.15 (1998): 8420-8427.

[69] Gehring, Walter J. "The master control gene for morphogenesis and evolution of the eye." Genes to Cells 1.1 (1996): 11-15.

[70] Halder, Georg, Patrick Callaerts, and Walter J. Gehring. "Induction of ectopic eyes by targeted expression of the eyeless gene in Drosophila." Science 267.5205 (1995): 1788-1792.

[71] Kovitz, Ben. "Experiments with Cascading Design."

[72] Wolpert, David H., and William G. Macready. "No free lunch theorems for optimization." Evolutionary Computation, IEEE Transactions on 1.1 (1997): 67-82.

[73] Neuron. Digital image. Wikipedia. N.p., n.d. Web. 01 Dec. 2015. `https://en.wikipedia.org/wiki/Neuron#/media/File:Neuron_Hand-tuned.svg`

[74] Batagelj, Vladimir, and Andrej Mrvar. "Pajek." Pajek - Program for Large Network Analysis. Web. 14 Oct. 2015. `http://mrvar.fdv.uni-lj.si/pajek/`

[75] Wolfram—Alpha. Wolfram Alpha LLC. Web. 04 May 2015. `http://www.wolframalpha.com/`

[76] Secretan, Jimmy, Nicholas Beato, David B. D'Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O. Stanley. "Picbreeder." Picbreeder. Web. 1 Dec. 2015. `http://www.picbreeder.org`

Chapter 6

Appendix A: Figure encoding

This appendix chapter contains a list of genotype source encoding used in cases presented in figures. Here TT are represented only by its output column, i.e. last values of the table, as this is the minimum necessary information of the TT. In the TT of this thesis the empty neighbourhood is the first entry, and the neighbourhood increment from the end of the following list: left, center, right, up and down.

- **Figure: 3.23** — (INV, R, R),(SET, L, R),(ROU, U, R),(MIN, U, D),(DEC, R, U),(MAX, U, R),(SWAP, U, R),(XOR, L, R),(NOT, C, D),(ROR, C, L)
- **Figure: 3.24** — (XOR, R, L),(ROD, R, U),(INV, L, U),(MIN, D, U),(ROL, L, C),(ROL, R, U),(OR, C, L),(INC, C, U),(XOR, C, D),(INC, L, C).
- **Figure: 3.25** — 3 3 3 3 3 3 2 2 3 3 0 3 2 2 0 2 0 0 2 1 0 1 2 0 1 2 2 1 0 3 2 1 2 1 2 2 0 0 0 2 3 0 3 2 2 2 1 0 3 2 2 0 3
  2 1 1 1 1 1 0 2 0 0 2 1 1 1 3 3 2 2 0 2 0 3 0 3 0 1 2 1 3 1 2 3 2 2 2 2 2 1 2 2 0 0 3 2 2 0 2 2 0 2 3 1 1 3 0 1 3 3 3 0 0 0 2 1 2 0 3 1 1
  1 0 0 3 2 1 0 1 1 0 2 0 0 1 2 2 3 1 3 1 0 3 2 2 2 1 2 2 0 1 2 0 2 2 1 1 0 1 3 3 0 2 2 0 2 0 1 2 2 2 3 1 2 0 2 0 1 3 0 1 2 1 2 2 1 0 0 1 3
  2 1 3 1 1 0 3 0 2 1 2 2 3 2 1 1 1 1 2 2 3 0 1 1 2 1 3 1 0 0 1 3 1 0 3 1 2 0 3 3 0 3 3 3 2 3 3 0 1 1 2 0 3 0 2 1 3 2 0 3 3 3 1 1 1 1 3 1 3
  2 3 0 0 2 3 1 3 1 2 1 3 1 0 2 1 2 2 0 3 0 2 1 3 3 2 1 2 0 2 3 1 3 2 2 1 0 1 3 2 1 3 2 2 0 3 0 0 0 2 2 0 2 3 3 3 0 2 1 1 3 3 2 2 1 0 0 3 1
  0 0 2 3 0 0 2 3 3 1 3 0 1 0 1 0 1 2 2 1 0 0 1 1 1 0 0 3 0 2 3 3 2 2 1 3 2 0 2 2 0 3 3 2 0 1 1 0 3 0 0 1 1 2 0 3 1 1 2 1 3 0 2 3 3 2 2 3 0
  2 3 0 1 1 1 3 2 0 3 2 0 2 1 2 1 3 1 0 0 1 1 2 2 0 2 0 2 0 1 2 0 1 0 0 1 2 1 3 2 3 3 3 1 2 0 3 3 0 2 0 3 1 0 3 2 0 0 3 1 1 0 2 1 0 2 2 2 2
  2 1 3 1 3 3 1 1 3 2 0 0 3 2 1 1 2 2 3 2 2 2 1 0 2 2 0 2 3 1 0 2 3 0 3 0 0 2 1 2 2 0 3 3 3 3 3 2 0 3 1 2 3 3 0 0 0 0 3 2 1 3 0 0 2 0 3 1 1
  2 3 0 0 2 0 3 1 3 3 2 1 2 0 2 2 0 0 1 1 3 3 2 1 1 0 3 2 1 3 1 1 1 1 0 0 1 2 0 3 2 3 3 3 1 0 1 0 0 1 1 2 3 0 3 2 1 1 1 3 3 3 3 2 0 3 3 2 3
  2 3 2 0 0 2 2 0 0 3 1 3 3 2 0 2 1 1 2 1 2 2 1 0 2 3 1 0 1 2 2 1 1 0 1 3 2 1 0 0 2 3 2 0 2 0 1 0 1 0 0 3 2 0 2 2 2 3 2 0 1 1 1 2 1 2 2 2 2
  0 2 1 3 1 2 3 1 2 3 3 1 0 3 3 1 2 0 3 1 2 3 3 2 3 1 2 0 0 2 0 3 0 1 1 0 0 3 3 2 2 2 0 3 1 2 1 3 3 2 2 3 2 1 2 2 1 3 2 1 3 3 0 3 3 1 3 2 3
  2 1 0 3 3 2 1 0 2 2 3 2 0 3 1 2 1 3 1 3 3 3 1 0 1 2 3 1 1 1 2 3 3 3 2 2 2 0 0 3 0 0 2 1 1 2 1 3 0 3 3 1 3 3 1 0 2 2 0 0 0 3 3 1 3 2 2 2 0
  1 3 1 1 3 0 2 2 3 3 0 0 2 2 1 0 1 2 1 0 2 0 0 3 2 0 3 3 1 0 1 0 2 1 2 3 1 1 1 2 3 3 2 2 1 1 3 0 2 0 2 2 2 2 3 0 2 1 3 3 2 1 2 2 1 2 0 0 2

2 0 0 0 0 0 0 2 1 2 0 0 2 0 2 1 2 3 2 3 0 0 1 0 3 2 0 1 3 1 1 0 1 3 1 0 1 3 1 3 1 0 0 2 0 2 2 0 1 2 1 0 1 3 1 0 2 2 1 0 0 3 0 2 0 3 0 0 2
0 3 2 2 0 3 2 3 2 2 3 2 2 0 3 2 0 0 1 1 2 1 0 1 2 2 1 1 3 2 1 3 2 0 0 2 0 1 0 0 1 0 0 2 3 2 2 0 0 3 0 0 2 3 3 0 2 1 1 1 1 3 3 0 0 3 2 1 1
3 3 0 1 3

- **Figure: 3.27 —** 1 0 0 1 1 1 1 0 1 1 2 2 1 2 1 2 1 1 1 0 2 2 2 0 2 1 0 0 0 1 0 2 2 1 1 2 0 2 1 1 2 1 2 0 2 2 1 2 2 2 0 0 2
  2 1 2 2 2 1 1 1 1 0 2 2 2 1 2 2 2 1 0 2 0 0 1 1 0 1 0 0 0 1 0 0 1 1 1 1 1 2 1 0 0 2 2 1 0 2 0 0 2 0 0 2 2 1 0 0 1 0 0 1 2 1 1 2 1 2 1 2 2
  1 2 1 2 2 0 2 2 2 1 2 0 2 2 2 2 2 0 0 2 2 2 2 0 1 1 2 0 0 2 0 0 2 1 2 1 2 1 1 0 2 0 0 0 2 2 1 2 2 2 2 0 1 2 2 2 0 0 2 1 1 2 0 0 2 0 0 0 2
  0 1 2 2 0 0 2 2 2 1 0 2 2 0 2 1 1 1 1 2 1 1 2 0 0 0 1 1 1 1 1 2 1 1 1 2 2 0 1 1 0 1 2 2 0 1 1 1 1 1 1 1

- **Figure: 4.2 —**
  First:
  0 1 0 1 1 0 4 3 3 2 1 3 0 2 3 2 1 1 0 0 0 1 4 3 1 1 1 3 3 0 1 3 0 0 1 3 0 0 4 1 0 1 3 1 2 2 3 2 1 0 4 4 1 2 0 4 0 2 4 0 3 2 1 1 2 3 1 3 0
  3 1 1 2 1 4 2 0 1 0 4 0 1 3 2 2 0 2 3 4 2 4 1 2 1 3 0 4 1 2 0 3 1 0 4 2 0 2 1 2 2 3 0 4 4 3 3 3 3 4 0 2 2 3 4 4 1 4 2 2 0 3 1 4 4 0 4 3 4
  4 1 2 4 3 0 2 1 0 3 1 2 1 1 3 2 3 1 2 0 1 2 4 3 3 2 1 4 3 4 2 3 2 2 4 0 3 3 3 2 3 4 0 2 0 0 3 2 3 4 2 1 0 2 1 1 1 0 4 4 1 2 4 3 1 4 3 0 3
  4 1 4 2 3 0 4 4 2 2 2 4 2 3 4 3 1 4 4 2 1 4 1 3 0 4 4 0 3 2 1 4 2 0 3 2 2 3 4 3 2 3 4 1 1 4 0 0 0 3 3 2 3 0 3 0 3 1 1 3 3 3 0 2 4 0 1 1 0
  0 4 1 2 1 2 2 3 1 1 3 0 4 3 1 0 2 3 1 3 4 4 3 2 1 2 0 4 1 2 2 0 0 2 2 4 1 4 2 3 4 4 3 2 1 2 2 2 0 1 0 1 3 3 0 3 4 3 3 2 0 0 1 2 1 0 1 4 3
  1 2 4 1 3 1 1 1 4 1 1 2 0 3 4 0 0 0 0 1 2 2 0 1 0 1 3 2 1 2 1 0 1 3 1 2 1 0 0 0 4 1 0 4 2 3 1 3 0 3 3 0 0 0 4 3 2 3 3 1 2 3 2 1 1 0 1 1 0
  2 1 3 4 2 1 1 0 0 3 3 3 2 1 3 4 1 3 4 4 2 0 1 1 2 3 3 1 0 2 4 3 2 2 0 1 4 0 2 2 3 4 3 2 4 4 1 1 4 0 2 4 4 3 2 3 2 1 2 1 4 1 4 2 4 0 2 3 3
  3 3 2 4 3 1 2 0 0 0 2 3 3 2 1 0 1 1 2 3 0 1 3 4 1 2 0 2 3 3 0 3 3 3 4 0 4 0 3 4 2 4 2 3 0 4 2 3 0 3 0 2 0 2 2 3 0 1 2 2 1 3 0 1 2 1 4 3 2
  3 0 1 0 2 3 4 0 0 2 2 3 2 3 3 4 4 3 3 4 1 2 0 0 0 4 0 2 2 4 2 1 2 4 1 4 4 3 0 1 1 4 3 1 0 2 4 2 3 3 4 0 0 0 3 4 3 3 1 4 3 4 0 2 1 1 2 2 0
  2 2 0 0 4 1 4 1 1 0 4 0 3 0 2 2 3 0 0 3 0 0 4 1 4 0 2 4 0 3 4 3 4 1 3 2 3 0 0 3 0 2 0 0 2 4 2 3 1 4 0 4 2 3 1 0 4 2 3 0 0 2 2 1 2 4 2 3 4
  0 3 2 0 3 0 3 2 2 4 4 1 0 0 4 0 4 0 3 1 3 3 1 3 3 4 0 4 4 3 0 1 3 2 2 2 0 0 0 0 3 4 4 4 2 2 3 2 3 1 4 1 1 1 0 3 0 2 0 3 0 0 4 1 3 3 2 0 1
  0 3 3 3 2 2 0 4 3 0 2 4 0 1 2 2 0 3 1 0 3 3 0 2 3 0 0 4 4 1 3 3 2 4 3 0 3 0 2 0 1 1 3 3 3 1 1 3 3 2 0 4 1 2 2 1 0 0 0 4 4 1 3 2 1 0 2 2 1
  2 3 2 1 3 1 1 0 0 2 1 3 1 4 4 0 4 3 1 4 0 2 3 4 1 3 2 0 4 4 1 0 0 1 1 4 3 1 4 3 0 3 3 4 0 4 1 0 0 4 3 0 3 3 4 2 2 0 2 0 1 0 3 3 1 2 3 4 2
  2 1 4 3 1 3 2 3 3 1 3 0 4 4 4 0 2 3 3 2 4 3 3 2 4 1 2 4 4 3 0 3 1 3 4 0 2 2 1 2 1 0 0 3 3 0 2 4 4 3 0 2 2 1 2 2 0 3 2 1 3 2 0 2 2 3 2 0 1
  1 1 1 1 2 2 2 4 0 4 1 3 0 1 2 4 2 0 0 2 1 4 3 3 1 2 3 4 3 3 1 1 0 3 4 0 4 3 4 3 3 2 4 3 0 3 3 1 1 0 1 3 2 4 0 2 4 0 3 3 2 3 2 4 4 2 3 3 3
  3 0 3 1 3 1 4 3 4 4 3 0 1 3 3 4 2 2 2 2 0 0 2 4 2 1 3 1 3 3 1 4 1 2 1 3 3 1 0 1 2 2 2 0 0 1 4 2 1 3 2 0 3 0 0 0 2 0 4 4 3 4 1 1 4 1 4 1 0
  3 3 3 0 4 1 3 3 4 3 0 0 3 2 3 1 0 2 3 3 2 4 4 3 2 0 0 0 1 0 1 4 2 2 0 1 2 0 3 3 2 4 0 4 0 2 1 4 4 2 4 0 2 1 3 0 4 2 3 3 2 2 3 3 3 4 0 0 4
  3 0 3 4 1 0 1 4 2 2 1 0 2 3 0 4 2 2 0 3 1 2 4 4 1 0 3 4 3 2 1 2 4 0 2 3 4 3 0 4 2 1 0 2 0 0 4 3 4 1 3 2 4 3 1 0 3 2 3 1 2 0 0 3 1 3 2 4 1
  2 1 4 4 2 2 0 1 2 4 0 2 0 1 4 1 2 3 3 1 1 4 2 3 2 1 1 0 1 4 1 2 2 2 4 2 1 1 1 3 0 0 1 4 2 3 1 1 3 0 0 2 2 0 3 2 3 2 3 2 2 2 2 4 3 2 1 2 2
  2 1 4 1 4 2 1 2 0 0 2 4 1 1 3 0 3 0 2 2 3 2 3 1 3 0 3 2 4 0 2 0 1 1 1 3 2 0 4 3 3 3 0 3 4 3 0 2 3 4 2 2 0 1 3 4 1 3 0 0 1 3 4 2 0 2 3 3 2
  4 2 2 3 1 1 3 3 0 2 0 1 2 3 2 2 4 4 4 0 1 2 4 1 2 2 4 1 2 2 3 1 2 0 4 1 1 4 2 1 3 1 2 4 3 4 1 4 1 3 0 1 2 3 4 3 3 3 4 2 4 1 1 3 2 1 1 2 1
  4 2 3 3 4 1 0 1 1 2 1 3 3 1 3 0 4 4 1 2 3 2 3 3 4 4 2 0 0 2 2 2 2 2 1 0 2 1 0 2 0 4 3 4 3 1 0 4 3 0 2 0 4 3 0 3 4 1 2 1 0 0 2 4 4 3 2 4 3
  0 1 4 3 2 4 4 4 3 0 0 4 2 3 1 4 3 3 1 3 4 0 4 4 3 3 4 1 3 1 2 2 1 0 0 0 2 2 2 2 2 4 4 1 3 3 3 1 3 3 1 2 0 4 1 1 4 2 4 0 4 3 0 2 4 4 4 4 1
  0 3 2 1 3 4 3 2 0 0 4 2 4 3 4 2 1 4 3 0 3 3 3 2 4 1 4 2 0 0 2 1 4 0 1 3 4 1 1 2 1 2 0 3 1 3 0 0 1 1 3 4 0 4 0 0 4 1 3 2 4 2 0 0 3 3 2 0 2
  1 3 2 4 1 3 2 0 0 1 0 2 2 0 1 1 2 0 4 0 4 2 4 2 1 4 2 1 3 0 0 2 1 1 0 0 1 0 4 1 0 4 1 0 4 3 0 1 1 3 2 3 1 3 4 0 3 4 1 4 1 1 4 1 3 1 4 1 4
  3 0 3 2 3 2 4 2 1 4 4 0 0 1 0 2 4 2 4 3 1 3 4 3 4 0 2 1 3 4 1 2 4 3 0 1 1 2 2 1 2 4 4 4 2 3 4 2 0 1 2 0 1 0 2 3 2 3 1 3 1 3 0 4 3 2 2 0 0
  3 2 3 3 1 1 3 3 3 1 3 0 3 0 1 0 0 2 4 2 3 4 2 0 0 0 4 4 4 4 0 2 3 2 2 3 3 2 4 1 0 4 1 3 2 2 4 2 0 0 1 3 2 4 4 2 1 4 3 1 3 1 1 2 0 4 2 2 3

1 1 1 0 4 2 0 3 1 1 4 4 0 4 4 2 0 1 4 1 2 4 0 0 1 2 2 3 0 3 0 1 3 0 3 3 4 3 0 4 0 2 2 4 3 2 3 1 4 1 1 2 0 2 4 4 3 4 2 3 3 3 1 4 4 3 4 3 4

1 2 3 0 2 0 2 1 0 4 1 0 0 2 1 2 1 0 3 1 2 3 0 4 3 2 2 2 3 2 4 2 2 4 3 1 3 4 2 0 3 1 1 4 3 0 4 3 4 2 2 2 0 0 3 0 1 4 4 4 4 4 3 4 2 4 1 2 0

2 0 2 1 1 0 0 3 1 3 2 2 1 1 3 3 0 2 0 2 1 3 3 2 1 2 0 1 1 3 0 4 1 2 3 2 3 3 3 4 4 1 3 4 0 2 4 4 3 2 2 4 1 3 3 1 0 0 2 0 0 2 4 1 0 3 4 4 4

1 2 3 4 2 1 4 2 4 3 4 2 2 3 2 2 2 4 3 0 1 3 2 1 1 1 2 1 2 3 4 3 0 2 2 2 2 0 0 2 1 0 1 3 0 3 2 1 0 3 1 1 3 2 2 2 2 1 4 3 3 0 2 3 2 3 4 3 1

4 3 4 4 3 2 2 4 3 4 3 0 4 4 0 2 3 1 4 0 2 0 3 1 3 4 3 4 4 2 3 0 2 0 3 3 2 3 2 4 3 1 3 0 2 3 4 4 4 0 0 1 0 1 1 1 3 2 1 4 4 3 1 0 3 3 3 4 1

0 2 4 3 1 4 1 1 2 3 2 3 3 1 4 0 2 3 0 1 2 2 3 3 4 1 3 0 2 3 0 3 4 4 3 3 1 2 0 2 4 3 1 1 3 4 2 1 2 4 2 0 0 4 2 3 0 1 0 3 4 4 0 3 1 0 1 3 3

4 2 4 3 0 2 1 2 3 4 3 3 0 4 3 1 1 3 4 4 2 1 0 0 0 2 2 4 0 2 3 0 3 3 4 1 4 1 3 4 2 2 1 0 4 0 3 1 0 0 3 0 1 4 3 1 1 1 1 0 3 0 3 4 0 1 1 1 0

4 1 3 0 4 1 0 4 0 3 1 3 0 4 4 4 2 0 0 3 1 2 1 3 1 2 2 1 4 1 0 0 4 3 3 3 2 4 2 4 1 0 1 2 1 4 3 3 0 1 4 4 2 3 0 3 3 0 0 4 0 2 3 2 0 2 0 2 1

3 2 2 2 0 3 3 0 2 4 0 2 2 0 0 0 4 0 3 0 3 4 2 1 2 3 2 2 0 3 2 0 3 4 3 3 3 1 3 2 4 2 1 1 4 2 4 4 1 0 1 3 2 0 3 0 1 4 0 2 3 3 4 4 4 1 0 1 2

1 4 2 2 3 2 0 2 4 1 2 1 0 3 1 4 1 3 2 1 2 0 0 4 4 4 1 4 4 3 4 3 3 0 4 3 3 3 4 4 2 2 4 0 0 2 3 4 1 4 1 4 2 3 4 0 2 0 2 1 1 2 0 4 1 1 1 2 2

1 4 3 0 2 1 0 0 0 3 1 3 1 0 0 1 4 1 4 1 4 2 4 1 4 1 2 2 3 4 4 1 3 2 0 2 1 1 2 3 4 2 3 4 4 4 4 2 0 2 4 0 4 0 0 4 4 1 2 2 0 1 4 4 1 3 4 3 3

1 0 4 4 1 1 1 0 4 2 4 3 1 4 0 2 0 0 0 2 2 0 3 0 2 3 3 3 4 1 2 3 3 2 3 3 3 2 2 0 3 3 1 3 1 2 1 4 4 3 0 3 4 0 4 3 0 4 1 3 2 4 1 2 0 0 3 0 1

3 2 3 0 2 4 4 0 4 3 1 3 2 0 1 4 4 1 2 4 0 4 4 4 3 1 4 3 4 0 3 1 1 1 4 3 4 4 4 0 1 3 4 4 1 4 0 1 0 0 0 0 1 0 4 2 3 4 4 1 0 2 1 2 2 2 3 3 0

0 1 4 2 4 0 4 4 1 0 2 3 3 4 1 1 4 0 3 4 0 0 1 3 4 1 2 0 3 3 3 3 0 1 3 3 4 3 1 1 0 0 2 3 2 4 2 1 2 0 1 3 2 2 2 0 0 1 3 0 4 4 0 3 2 2 4 0 3

4 3 0 0 0 2 4 3 2 2 1 1 3 3 1 1 0 3 2 2 0 1 2 2 4 1 2 2 2 3 2 1 4 3 3 2 0 3 2 0 3 0 4 1 0 3 2 3 3 4 0 0 4 0 0 3 2 3 3 1 0 1 3 0 4 1 1 3 2

0 4 1 2 2 4 1 2 4 2 2 3 2 0 4 0 3 2 0 3 4 4 3 2 2 4 3 2 1 1 1 1 0 0 3 1 4 1 0 2 1 3 1 1 2 0 2 4 2 0 2 3 2 0 3 2 3 4 3 0 3 2 2 2 1 2 0 3 3

0 0 2 2 3 3 0 2 4 4 3 3 4 4 3 3 3 1 2 3 0 4 2 3 0 2 3 3 2 3 4 1 0 4 0 4 0 3 0 4 4 2 3 3 3 1 3 4 0 0 3 2 3 4 0 0 2 3 4 1 0 1 4 2 1 4 1 4 4

4 3 2 0 3 4 2 3 4 1 2 0 1 4 4 1 2 1 1 0 2 0 4 0 3 1 2 1 3 2 3 4 4 1 3 2 3 4 2 4 0 4 1 4 2 2 0 2 0 3 4 2 2 4 0 3 2 1 0 4 4 0 3 1 1 2 4 3 1

0 4 4 1 0 3 4 1 1 4 1 3 3 3 0 3 1 1 0 0

Second:

1 3 1 2 0 3 4 3 4 2 2 4 3 0 0 4 0 3 2 2 3 1 3 0 4 2 1 2 0 3 0 4 2 3 1 1 1 3 0 2 1 3 2 2 2 0 0 0 1 3 1 3 3 4 0 3 1 3 0 1 2 0 4 0 1 2 4 3 2

1 0 1 4 1 3 3 2 3 1 1 0 2 0 2 4 3 2 3 0 4 1 2 0 4 3 0 1 2 4 1 3 1 3 3 3 4 1 0 2 2 2 0 3 3 4 1 1 3 3 0 3 4 3 0 2 3 3 0 4 1 3 2 4 1 2 2 1 0

1 2 2 2 0 4 0 1 0 0 2 1 4 4 0 0 2 2 0 0 0 1 1 4 4 4 2 0 3 0 4 1 0 2 0 0 4 3 2 1 2 2 3 1 1 0 0 4 0 0 1 3 3 0 1 2 1 1 1 0 3 4 0 0 2 2 1 4 3

1 3 3 2 0 4 1 1 1 2 2 2 2 4 4 2 0 1 0 2 3 3 4 4 2 3 0 3 2 3 1 3 4 3 4 2 2 3 4 4 3 0 4 1 0 4 3 2 1 1 0 0 3 3 0 3 4 1 4 0 1 3 3 2 3 3 3 1 2

0 1 2 2 3 1 2 4 4 1 0 4 4 4 0 2 1 1 0 4 4 2 4 2 2 1 2 3 0 2 1 0 3 3 4 1 4 2 1 4 2 4 2 4 0 4 3 4 3 2 1 2 4 1 1 2 1 0 2 1 4 3 0 0 2 0 1 2 2

0 3 2 2 3 0 3 3 1 3 0 3 0 0 2 0 4 1 3 4 3 2 0 3 0 0 0 0 4 4 3 3 3 3 0 3 4 1 2 1 3 3 0 0 1 4 2 3 3 1 0 4 2 2 3 2 0 3 1 4 3 2 1 3 0 3 2 0 1

4 1 1 1 0 0 0 4 2 1 3 0 2 3 0 1 2 0 4 4 3 1 4 4 3 1 0 2 4 4 4 2 1 0 3 3 4 4 1 0 0 4 0 2 1 3 3 0 3 0 1 3 2 1 2 3 1 2 3 2 2 0 2 2 2 3 2 1 4

2 0 4 0 1 3 1 0 4 1 3 4 4 2 0 3 1 3 2 2 3 2 4 2 4 0 3 2 0 0 4 0 1 0 4 4 1 1 4 3 1 2 3 4 3 3 2 2 2 3 2 4 0 2 3 4 1 0 1 4 1 0 2 0 4 0 2 0 0

2 3 3 0 4 3 4 3 1 0 4 3 4 1 4 2 0 2 3 1 1 3 1 3 0 2 3 3 1 0 0 0 0 3 2 4 2 3 3 2 3 2 4 0 0 4 3 2 4 1 1 4 0 3 4 3 3 0 3 2 3 1 3 0 4 2 2 3 3

1 1 0 0 0 1 3 4 1 3 0 0 1 1 1 3 2 0 0 0 1 0 0 1 3 4 1 4 2 2 3 1 2 4 3 1 4 1 2 1 0 4 3 1 4 3 3 1 0 3 3 2 2 0 2 4 0 3 3 3 2 3 1 0 3 0 0 3 3

4 3 3 0 0 0 1 2 3 3 1 2 1 1 1 2 0 3 4 0 1 3 0 2 3 2 0 4 3 3 2 4 4 3 1 2 1 4 3 1 0 1 2 4 1 4 3 2 0 2 4 4 3 3 3 2 2 0 0 3 4 3 4 1 4 4 2 4 0

2 1 1 3 2 1 2 0 2 2 4 2 2 2 4 2 2 2 2 3 0 1 2 3 2 0 1 0 0 2 3 0 4 4 3 4 4 2 2 3 1 0 1 3 1 3 0 1 2 3 2 0 1 2 1 1 1 3 0 0 0 1 4 1 4 4 1 4 4

2 4 4 1 0 1 3 1 4 4 0 4 0 2 1 1 4 1 2 2 0 0 4 2 2 2 2 4 3 4 4 3 0 2 3 1 0 3 4 4 0 2 1 1 3 1 4 3 1 4 4 2 1 2 3 3 0 1 2 2 1 4 3 2 4 2 2 0 1

1 3 0 1 3 3 2 2 4 2 0 2 2 4 1 3 0 3 2 2 2 2 4 3 2 4 1 4 1 0 1 3 1 2 1 0 1 1 3 4 2 1 4 3 3 2 0 3 1 4 4 4 1 1 3 0 2 4 2 1 2 0 4 1 3 2 4 3 0

4 0 1 2 4 0 2 0 4 2 1 4 1 2 1 2 4 2 3 3 1 0 3 2 0 4 2 3 4 1 0 1 4 2 1 1 4 3 3 1 3 1 2 4 4 4 2 3 1 0 4 4 3 4 1 4 0 0 1 2 0 4 0 0 3 4 2 4 4

3 0 2 1 1 0 0 0 1 0 4 1 2 0 4 2 4 0 2 4 3 0 1 4 2 2 0 3 1 2 3 1 1 1 3 2 2 1 2 1 2 0 0 4 2 2 0 2 1 3 1 4 1 4 4 1 0 1 2 3 1 1 1 1 4 1 1 4 3

0 3 0 0 2 3 4 2 3 0 1 2 1 0 0 4 3 0 2 0 3 4 2 4 0 0 0 1 3 4 4 3 4 4 1 1 3 2 3 0 4 3 1 2 0 1 2 1 4 1 3 2 0 1 0 1 3 1 3 0 0 4 3 1 0 3 2 3 0

4 1 2 0 4 0 0 3 1 1 0 1 2 3 3 0 1 1 2 3 1 3 0 3 0 2 2 4 2 1 2 1 3 2 4 0 1 1 2 2 1 4 0 0 0 2 2 1 2 2 4 0 0 3 2 0 3 1 4 0 2 2 3 1 2 3 1 4 2

```
4 3 3 3 3 0 4 2 1 0 0 0 2 3 1 1 1 1 4 4 1 0 2 1 3 3 1 4 3 1 2 4 1 3 2 0 3 1 1 2 0 4 3 0 1 0 2 0 0 0 3 0 2 3 3 1 2 0 3 3 2 1 0 1 4 1 0 1 2
3 0 0 1 2 4 1 0 4 2 0 0 0 1 0 0 4 2 2 4 4 0 0 1 0 4 1 2 3 4 4 2 0 4 1 1 2 0 1 1 1 1 3 3 0 2 2 0 2 0 1 0 3 2 2 3 0 3 3 4 2 3 2 0 3 4 3 4 2
1 1 4 4 2 2 2 1 4 4 4 1 0 0 3 2 1 3 3 3 0 3 4 0 2 0 4 2 2 1 2 1 0 1 2 1 1 4 3 2 0 4 0 4 1 2 1 3 3 1 4 4 4 2 3 2 4 4 1 2 4 3 0 2 4 0 3 3 3
0 3 0 2 0 1 3 1 1 2 2 1 2 0 4 2 2 4 0 3 0 1 3 3 3 1 1 3 1 4 0 1 3 4 1 0 2 3 4 3 2 4 4 2 3 3 3 0 1 1 4 2 2 2 1 1 3 3 2 2 0 0 1 0 3 3 4 0 2
4 4 1 3 4 0 1 1 3 1 1 1 0 0 0 3 4 1 2 0 4 3 1 1 0 0 2 2 4 3 4 4 2 2 0 4 0 0 0 2 0 0 4 4 3 3 1 0 4 4 1 2 2 4 3 4 2 0 0 1 4 3 3 4 0 0 0 2 4
1 1 3 0 4 3 0 2 4 1 3 2 0 1 4 0 3 2 3 3 0 4 1 1 3 4 3 2 3 0 4 0 3 0 3 0 4 0 3 4 3 2 0 1 4 1 3 2 3 3 2 3 0 0 0 1 4 1 4 1 0 0 3 0 0 4 2 1 0
1 4 0 4 4 2 4 2 1 1 1 0 0 1 2 0 4 2 3 4 0 3 4 0 2 3 3 0 4 3 0 4 0 4 1 3 2 0 4 1 1 1 4 4 0 4 2 2 2 1 0 3 3 1 1 4 0 1 4 3 4 0 1 3 0 4 4 1 4
4 3 0 1 2 3 2 4 2 4 3 1 4 2 1 2 1 0 0 1 0 2 3 1 1 3 2 0 1 4 4 0 2 3 4 1 4 3 2 0 4 2 0 4 3 1 2 3 4 1 1 4 3 3 3 3 2 4 2 1 0 1 4 1 2 1 3 4 3
3 3 1 4 4 3 4 3 2 2 3 3 1 3 1 3 0 2 4 0 2 3 1 2 0 0 2 0 4 1 4 0 1 3 1 0 1 3 1 1 4 1 0 2 3 1 4 4 1 4 0 0 2 4 0 4 3 4 2 2 2 2 3 0 2 2 4 4 0
4 3 4 2 4 3 1 4 2 3 0 4 0 4 4 1 0 3 4 3 1 2 3 2 0 0 1 3 0 3 3 1 1 1 2 0 1 1 3 4 1 1 3 0 1 0 4 4 4 2 4 4 4 0 4 0 4 1 2 3 2 0 4 0 4 1 1 4 2
0 4 2 3 4 2 0 3 0 0 0 2 0 1 4 4 2 2 2 3 4 1 4 2 2 1 4 1 0 1 4 0 1 0 3 2 1 4 3 2 0 3 3 2 0 2 3 2 4 0 4 0 1 2 0 2 2 0 0 3 3 1 4 2 3 2 4 4 2
1 4 3 3 4 2 3 2 1 4 4 2 3 2 3 0 2 1 3 3 3 3 2 1 3 2 0 4 0 4 4 4 0 0 0 4 3 3 4 0 0 1 3 0 1 0 4 3 2 0 1 2 4 0 2 0 1 4 2 2 1 1 4 2 0 3 0 4 2
3 3 0 4 4 0 0 2 3 4 4 1 1 0 4 3 3 0 4 0 2 0 2 2 3 2 1 1 1 0 4 0 2 2 1 4 0 2 0 0 0 1 0 4 1 0 3 2 0 1 1 1 0 0 4 4 3 4 2 2 4 2 1 4 1 0 0 3 3
1 3 3 0 3 4 4 4 2 1 3 2 0 2 2 2 0 1 4 0 1 3 3 3 1 2 4 2 3 1 2 3 0 4 0 2 2 0 2 4 0 1 2 3 3 1 3 4 0 4 0 3 4 2 3 0 1 2 4 2 3 1 2 2 3 3 0 3 2
4 4 3 0 1 2 2 0 3 3 4 4 1 1 4 1 4 3 1 0 2 1 1 1 3 4 2 0 1 2 4 0 2 4 1 3 2 0 4 3 1 0 3 4 4 1 4 1 2 0 2 0 3 4 0 3 0 4 3 1 4 4 1 3 0 4 0 1 0
0 3 4 2 0 3 1 4 1 4 1 3 0 3 4 2 3 2 0 1 1 2 1 0 1 3 1 0 3 3 2 0 1 4 3 1 4 0 1 1 0 1 2 3 1 3 3 2 0 2 3 1 0 4 1 3 0 3 1 1 2 1 3 2 1 2 3 1 2
2 3 1 0 0 2 2 2 1 0 2 1 4 0 3 1 0 4 0 0 4 1 1 1 4 2 2 2 2 3 1 4 1 0 1 1 1 3 2 2 1 0 3 4 3 3 1 3 1 4 4 1 0 4 2 2 1 1 1 1 4 0 2 2 0 2 3 3 0
0 2 0 1 2 3 1 2 2 1 0 3 2 1 1 2 3 0 0 2 2 1 1 1 4 1 4 2 1 0 3 4 3 3 0 0 1 0 3 2 0 3 1 0 0 2 1 3 1 3 3 0 2 2 2 2 2 0 3 4 2 1 3 0 2 1 2 0 2
2 0 0 1 1 1 4 1 2 0 1 2 2 0 4 3 1 0 3 1 2 4 4 3 4 0 0 2 1 1 0 4 1 0 0 1 4 4 0 0 3 3 3 1 0 1 2 4 0 0 1 1 0 1 3 3 2 0 4 2 1 0 1 2 1 3 1 3 4
0 3 2 3 0 3 4 4 4 1 3 0 3 4 3 3 4 1 3 1 3 1 2 4 3 1 3 1 4 4 3 0 3 2 4 0 3 4 2 2 0 3 4 3 1 4 0 3 3 3 3 0 1 0 0 3 1 0 3 3 4 3 0 1 1 3 2 4 3
3 3 2 0 2 3 3 2 0 3 3 4 0 0 3 3 4 0 1 0 3 1 1 1 4 2 3 3 1 4 3 2 3 4 2 2 1 4 3 4 2 3 2 3 1 0 2 3 0 3 1 2 4 4 3 4 4 1 3 2 2 4 1 2 1 3 4 4 0
0 0 4 0 3 4 3 3 4 4 2 4 3 2 2 3 1 2 0 1 3 3 2 1 2 3 1 3 2 1 3 4 4 1 0 1 3 0 2 1 0 4 1 2 2 1 3 2 1 4 2 0 3 1 2 0 4 4 3 2 0 1 3 0 4 4 4 2 2
2 2 1 2 2 4 2 0 2 3 2 3 2 3 0 0 0 0 0 1 4 3 3 1 3 2 2 2 2 0 4 4 0 2 2 2 4 2 4 4 0 2 2 1 1 1 3 2 4 2 0 0 2 0 4 3 2 1 3 2 1 2 0 3 0 0 4 0 0
1 0 2 3 2 1 4 2 2 3 0 2 1 0 4 0 1 3 4 4 3 0 3 2 1 3 3 4 2 1 3 1 1 4 2 2 1 3 2 1 1 0 0 4 4 2 3 0 0 3 3 4 4 1 1 0 3 4 2 3 0 1 3 2 1 1 4 0 4
0 3 3 3 0 3 1 2 2 1 1 3 0 4 2 1 2 4 2 4 1 0 4 2 4 4 3 3 2 0 2 3 1 2 0 2 2 4 2 4 3 2 1 0 1 0 1 0 1 2 0 3 3 0 2 4 3 1 4 4 1 3 4 4 3 2 3 4 0
4 2 3 2 4 4 1 1 0 1 3 2 2 2 4 2 3 3 2 4 0 4 3 2 3 3 3 1 0 3 1 1 4 0 1 4 4 0 0 0 4 4 3 0 1 0 1 0 4 4 2 0 3 1 3 2 0 4 1 3 4 1 4 2 2 3 0 0 1
2 3 0 0 1 1 2 4 0 1 0 2 3 4 1 1 4 4 1 4 1 3 1 0 2 1 4 4 0 3 0 4 0 0 2 1 1 1 4 2 1 2 1 3 3 4 4 2 0 1 3 1 0 3 4 1 1 1 0 2 1 1 4 3 4 1 1 3 3
2 0 4 4 2 1 1 3 0 4 1 4 1 0 1 3 0 3 4 0
```

- **Figure: 4.3, 4.4, 4.5, 4.6, 4.7 and 4.8** —
  (XOR, L, C), (ROD, L, C), (OR, C, C), (XOR, C, L), (ROL, D, C), (MIN, C, D), (MAX, R, R), (ROU, U, L), (ROD, C, D), (SET, C, L)
- **Figure: 4.9** —
  First:
  (SKIP, U, C, 4, 3),(ROL, U, R, 1, 3),(MIN, U, C, 4, 1),(ROD, L, D, 4, 7),(DEC, D, L, 2, 5),(XOR, D, R, 8, 3),(ROU, C, L, 3, 9),(MIN, D, C, 7, 1),(MOVE, U, C, 4, 3),(NOT, R, C, 4, 0),

Second:
(SKIP, D, R, 2, 1),(SET, C, U, 5, 0),(CHP, D, L, 5, 2),(SKIP, D, U, 2, 2),(MIN, D, R, 2, 4),(ROU, C, U, 8, 0),(CHP, D, R, 0, 1),(XOR, C, L, 3, 0),(MOVE, L, U, 0, 3),(XOR, D, D, 3, 3),

- **Figure: 4.10** — 1 1 0 0 0 0 0 1 0 0 0 0 1 1 0 0 0 0 0 1 0 1 0 1 1 1 0 0 1 0 1 0

- **Figure: 4.11** — ((DEC, R, C),(MIN, D, U),(DEC, L, L),(XOR, C, D),(OR, L, L),(OR, R, L),(ROD, L, U),(SWAP, C, R),(NOP, R, D),(XOR, C, D)).

- **Figure: 4.12** —(XOR, R, L, 3, 5),(MIN, U, D, 7, 0),(XOR, U, C, 7, 8),(ROD, C, C, 9, 3),(DEC, C, R, 3, 9),(INV, D, C, 1, 2),(CHF, D, C, 5, 6),(SWAP, R, R, 6, 1),(CHF, C, D, 4, 0),(INV, L, D, 2, 5),

- **Figure: 4.13** — (XOR, R, L, 3, 5),(MIN, U, D, 7, 0),(XOR, U, C, 7, 8),(ROD, C, C, 9, 3),(DEC, C, R, 3, 9),(INV, D, C, 1, 2),(CHF, D, C, 5, 6),(SWAP, R, R, 6, 1),(CHF, C, D, 4, 0),(INV, L, D, 2, 5),

- **Figure: 4.14** — ((XOR, R, L, 5, 2),(XOR, U, D, 5, 1),(ROD, D, D, 0, 2),(XOR, R, C, 8, 7),(MAX, R, C, 4, 4),(MOVE, U, R, 2, 8),(SET, L, L, 0, 1),(CHP, L, U, 1, 1),(XOR, U, R, 4, 3),(MIN, C, R, 8, 6)).

- **Figure: 4.15** —
  First:
  (MOVE, C, U, 7, 5),(MOVE, U, R, 6, 0),(MIN, C, U, 9, 6),(SET, L, C, 8, 7),(CHF, C, D, 1, 6),(CHF, U, L, 0, 6),(INC, U, L, 4, 8),(ROD, C, L, 5, 8),(SET, L, L, 4, 9),(SET, U, R, 7, 5),.
  Second:
  (MOVE, C, U, 7, 5),(MOVE, U, C, 6, 0),(AND, C, U, 9, 6),(SET, L, C, 8, 6),(CHF, C, D, 1, 6),(CHF, U, L, 0, 6),(INC, U, L, 4, 8),(ROD, C, L, 5, 8),(SET, L, L, 4, 9),(SET, U, R, 7, 5),.

- **Figure: 4.16** —
  First:
  (MAX, D, C, 1, 5),(MAX, U, U, 8, 0),(CHF, R, D, 5, 9),(AND, U, L, 3, 2),(CHF, L, D, 6, 3),(ROU, R, D, 2, 8),(OR, R, L, 9, 7),(MIN, R, D, 5, 3),(ROL, U, U, 2, 3),(DEC, C, R, 0, 6),.
  Second:
  (MAX, D, C, 6, 5),(MAX, U, U, 8, 0),(CHF, R, D, 5, 9),(AND, U, L, 3,

4),(CHF, L, D, 6, 3),(ROU, L, D, 2, 8),(OR, R, L, 9, 8),(MIN, R, D, 5, 3),(ROL, U, U, 2, 3),(DEC, C, R, 0, 7),.

- **Figure: 4.17 —**
  First:
  (INC, R, R, 1, 4),(INV, D, L, 4, 0),(AND, C, D, 7, 9),(CHP, D, C, 6, 0),(SET, U, R, 7, 5),(MOVE, U, R, 8, 2),(ROD, R, C, 3, 3),(NOP, L, U, 7, 4),(INV, C, R, 5, 6),(XOR, U, U, 6, 2),
  Second:
  (INC, R, C, 0, 0),(CHP, D, R, 8, 1),(AND, C, D, 8, 8),(NOP, L, R, 7, 9),(SET, U, R, 1, 2),(MOVE, C, D, 8, 3),(ROD, L, R, 4, 0),(MOVE, C, L, 3, 0),(INV, R, U, 5, 0),(XOR, D, L, 1, 2),