Discrete Tranformation of Output in Cellular Automata

Aleksander Lunøe Waage

Master of Science in Computer Science
Submission date: July 2012
Supervisor: Gunnar Tufte, IDI

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

Abstract

Cellular automata (CA) is an example of cellular computing: large numbers of simple components, no central control, and limited communication among components. A CA consists of an array of cells, each in one of a finite number of possible states. The cells are updated synchronously in discrete time steps, according to a local, identical interaction rule. CA have been studied for years due to their architectural simplicity and the wide spectrum of behaviors.

It can be difficult to design CA to exhibit a specific behavior, and there is no programming paradigm for implementing parallel computations in CA. This makes CA a prime candidate for adaptive programming methods such as evolutionary algorithms (EA). EA have been successfully used to evolve CA to perform computations, such as the majority problem. In problem solving like this, CA are treated as input-output systems and evolved to visit certain states given certain initial states. The n-th state of a CA is commonly treated as its output, such that one input gives one output.

In this report a way to transform the output of CA such that multiple outputs emerge is investigated. The evolution of transformed CA is also investigated. Useful applications of this include controlling multi-variable systems such as a pole balancing system.

The transformation process was tested in the elementary and 3-state rule spaces. Evolution of transformed CA was successful in the elementary rule space. This suggests there is a correlation between rules of CA and their transformed output. The evolution of transformed CA was unsuccessful in the 3-state rule space. This suggests the chosen evolutionary algorithm doesn't scale with the size of the rule space.

Preface

I would like to thank my supervisor prof. Gunnar Tufte for creating this exciting task description, and for giving me the opportunity to live with my loving girlfriend in South Korea while working. It has been a fantastic year filled with culture, people and cellular computations.

Contents

Chapter 1

Introduction

The von Neumann architecture has dominated computing technology for the past 50 years[7]. It is based upon the principle of one complex processor that sequentially performs a single complex task[7]. More recently work on other computation systems has received renewed interest[5].

Cellular computing is a computation system that share some features with natural systems: large numbers of simple components, no central control, and limited communication among components[5][7]. It could do computation more efficiently[7].

Cellular automata (CA) was the first example of cellular computing[7], explored by von Neumann in the 1940's[5] as a self-reproducing universal computer[4][3][1]. A CA consists of an array of cells, each in one of a finite number of possible states[7]. The cells are updated synchronously in discrete time steps, according to a local, identical interaction rule[7]. CA have been studied for years due to their architectural simplicity and the wide spectrum of behaviors[2][9].

It can be difficult to design CA to exhibit a specific behavior[7], and there is no programming paradigm for implementing parallel computations in CA[5]. This makes CA a prime candidate for adaptive programming methods[7] such as evolutionary algorithms (EA). EA have been successfully used to evolve CA to perform computations[5], such as the majority problem[2][5]. In problem solving like this, CA are treated as input-output systems and evolved to visit certain states given certain initial states. The n-th state of a CA is commonly treated as its output, such that one input gives one output.

In this report a way to transform the output of CA such that multiple outputs emerge will be investigated, and the evolvability of transformed CA explored. Useful applications of this include controlling multi-variable systems such as a pole balancing system.

To limit the scope of the project, some decisions are made. First, the dimension of CA is restricted to one dimension. Second, the size of CA neighborhoods is restricted to 3. Third, the transformation of interpreted states is restricted to discrete Fourier transformation. Fourth, the evolution of transformed CA is restricted to an evolution strategy with 1+4 configuration. The consequences of these decisions are discussed in chapter 5.

The report is organized as follows. The first chapter introduces the idea of transforming the output of cellular automata. The second chapter presents theory for understanding the transformation process and evolution of CA. The third chapter describes the transformation process and evolution of transformed CA in detail. The fourth chapter contains experiments regarding the transformation process and evolution of of CA. The final chapter contains discussions about the transformation process, scope and results.

Chapter 2

Background

This chapter contains theory for understanding how cellular automata work, as well as some concepts needed to understand the transformation process and evolution of CA.

2.1 Cellular Automata

A CA consists of an array of cells, each in one of a finite number of possible states[7]. The cells are updated synchronously in discrete time steps, according to a local, identical interaction rule[7].

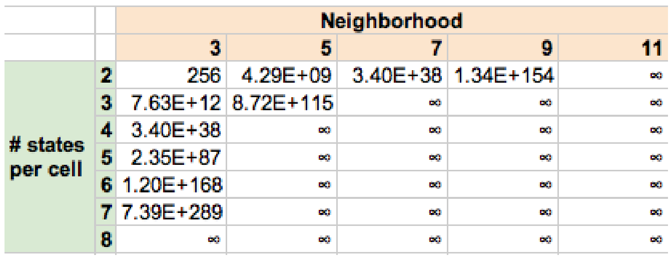
2.1.1 Rule space

The rule space, or the amounts of unique CA rules that exist, is a function of the number of states per cell and the size of the neighborhood:

$$unique\ rules = states\ per\ cell^{unique\ neighborhoods} = states\ per\ cell^{statespercell^{size\ of\ neighborhood}}$$

Increasing either the number of states per cell or the size of the neighborhood has the exact same result, namely expanding the number of rules that exist. Figure 2.1 shows the number of rules given different combinations of neighborhood sizes and states per cell.

Stephen Wolfram popularized a very simple class of CA rules called elementary CA (ECA). An ECA has 2 states per cell and a neighborhood of 3. Its number of unique rules is then $2^{2^3} = 256$.



		Neighborhood				
		3	5	7	9	11
# states per cell	2	256	4.29E+09	3.40E+38	1.34E+154	∞
	3	7.63E+12	8.72E+115	∞	∞	∞
	4	3.40E+38	∞	∞	∞	∞
	5	2.35E+87	∞	∞	∞	∞
	6	1.20E+168	∞	∞	∞	∞
	7	7.39E+289	∞	∞	∞	∞
	8	∞	∞	∞	∞	∞

Figure 2.1: The amounts of unique rules given the size of the neighborhood and the number of states per cell. Numbers bigger than 1.8×10^{193} are denoted as ∞.

Wolfram also came up with a rule naming system for ECA rules called Wolfram Code[9]. A generalization of the Wolfram Code can be used to name more complex rules. Figure 2.2 exemplifies its use with an ECA: the unique neighborhoods are ordered by their decimal value, their next states are then merged and converted to decimal, which is the name of the rule.

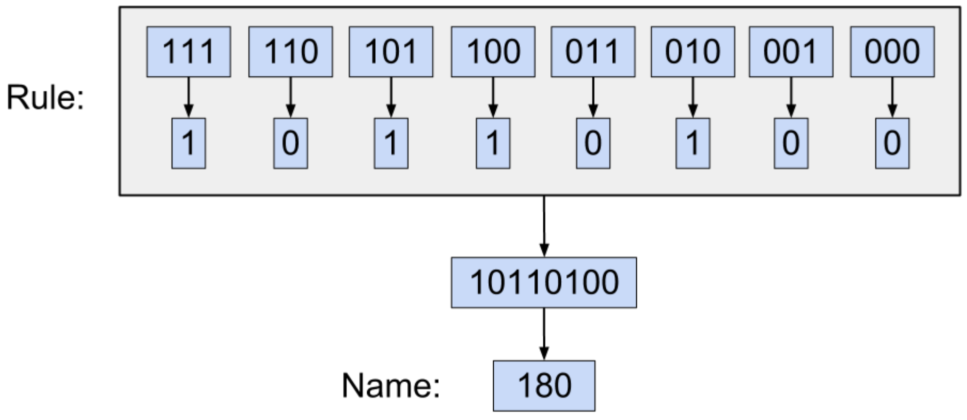


Figure 2.2: The Wolfram Code for an elementary CA with a specific rule.

2.1.2 State transition

The CA changes to its next state based on its previous state and its rule. For calculating the next state, a window of the size of the neighborhood of the CA is ran across the previous state. For each neighborhood the new state of the middle cell is calculated based on the rule. Figure 2.3 illustrates the transition from one state to the next in an elementary CA.

A lookup-table is often used to to aid the process of updating the cells. It consists of pairs of all unique neighborhood configurations and the signal states they translate to.

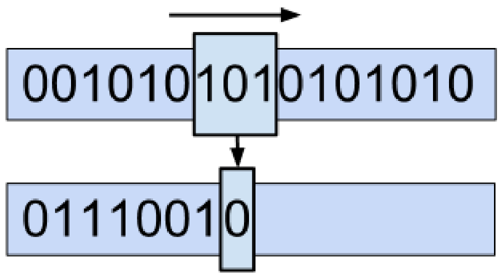


Figure 2.3: The transition from one state to the next.

The number of entries in the table is equal to the number of unique neighborhoods, which is:

$$unique\ neighborhoods = states\ per\ cell^{sizeofneighborhood}$$

2.1.3 Behavior

The states a CA visits can be thought of as its behavior. Stephen Wolfram created system for classifying behaviors of different complexities, called Wolfram Classes[4][9]:

- Class I: homogeneous fixed point
- Class II: periodic
- Class III: chaotic
- Class IV: complex

Figure 2.4 shows the behavior of Rule 110, an elementary CA with rule number (Wolfram Code) 110 with an initial state consisting of a single active cell. The behavior is neither periodic nor chaotic: it's complex. It is arguably the simplest known Turing complete system[9].

Periodicity

Because the state transition is deterministic and the number of states is finite, the CA will at some point reach a state it has already been in. At this point it will repeat itself over and over and all future states are determined. Figure 2.5 illustrates this: the figure shows the interpreted states of a CA, where it reaches a known state shown in red. The point of periodicity exists at some time-step between zero and the number of unique states. The higher the complexity of the CA, generally the longer the pre-periodic and periodic paths will be.

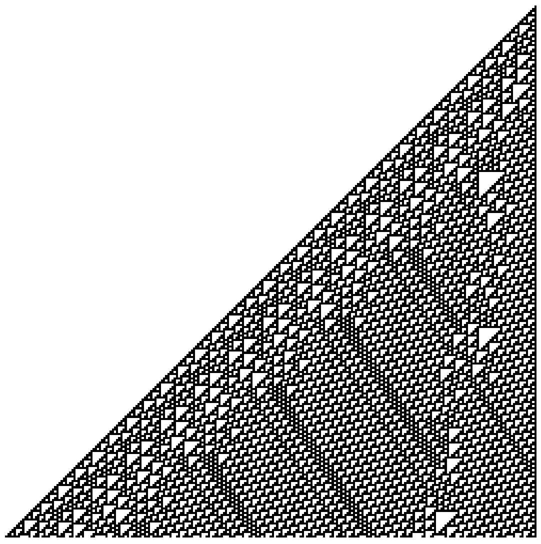


Figure 2.4: The behavior of an elementary CA with rule 110 and a single active cell in the initial state. The behavior is neither periodic nor chaotic: it's complex.

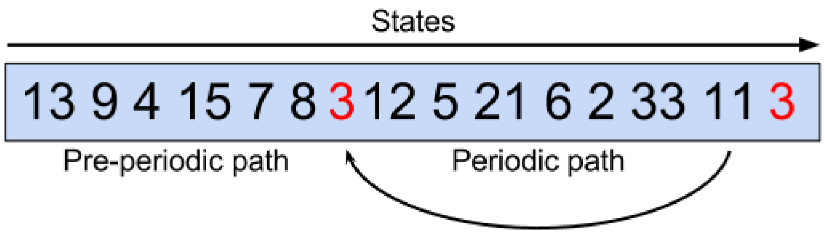


Figure 2.5: The periodicity of a CA. At some point it will reach a known state, at which point it will repeat it self over and over.

2.1.4 Size and boundary condition

Cellular automata are theoretically unlimited in size, but computers have limited memory. A finite size is therefore used when working with CA on computers. A problem arises when calculating the state of the bit on an edge of the machine. Figure 2.6 illustrates the problem: at least one of the bits needed to calculate the next state is outside of the machine and does not exist.

There are two common ways of dealing with this problem, called boundary conditions. With the cyclic boundary condition, the machine is considered circular. When reading from outside the state, one ends up at the other side of the machine. This is achieved using modular arithmetics. With non-cyclic boundary condition, when reading from the outside of the machine, a zero is returned.

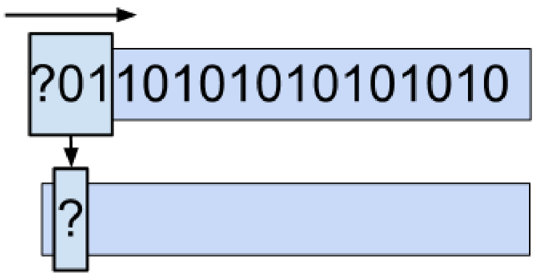


Figure 2.6: The boundary problem: what is outside the machine?

2.2 Artificial evolution

2.2.1 Evolutionary algorithm

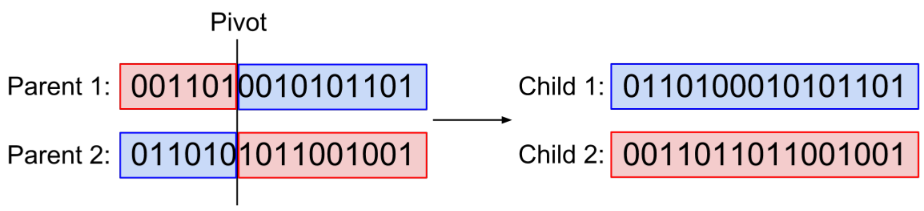
Evolutionary algorithms (EA) are search methods inspired by biological evolution[5].

The following is an example of a simple EA:

Initially a group of random solutions is created, called the population. Every time-step, the following occurs:

- The solutions are tested for their ability to solve the problem.
- The best ones are chosen, called parents.
- The parents make children with slight mutations.
- The children become the new population.

To evolve solutions one need the following. First, a way to represent the solutions. Second, a way to evaluate the solutions. Third, a way to combine and/or mutate the solutions. Figure 2.7 shows an example of sexual reproduction where two parents make two children: A pivot point is chosen, splitting the parents into two parts. The two children are the two cross-combinations of the parents around the pivot point. In asexual reproduction the children are clones of the parents.



Pivot
Parent 1: 0011010010101101
Parent 2: 0110101011001001
Child 1: 0110100010101101
Child 2: 0011011011001001

Figure 2.7: An example of sexual reproduction in artificial evolution.

2.2.2 Evolutionary strategy

An evolutionary strategy is an efficient approach to artificial evolution. In the most common version of the algorithm, a population of only one individual is maintained. Each step of the evolutionary algorithm the individual makes a few clones with mutations, and the best clone is chosen as the sole survivor and carried over to the next generation.

In variations of the algorithm, the number of individuals maintained can be more than one. The configuration of the algorithm is denoted as $(\mu + \lambda)$-ES, where μ denotes the number of parents and λ denotes the number of clones.

A downside with having just one individual in the population is that there is no memory of past solutions.

2.3 Discrete Fourier transformation

The discrete Fourier transform (DFT) is used to transform signals from the time domain to the frequency domain. It takes a sequence of complex numbers $x_0, ..., x_{N-1}$ in the time domain and transforms them into a sequence of complex numbers $X_0, ..., X_{N-1}$ in the frequency domain:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}}$$

The amplitude spectrum can be obtained by calculating the absolute value of X_k:

$$A_k = |X_k| = \sqrt{Re(X_k)^2 + Im(X_k)^2}$$

Fast Fourier transformation (FFT) is an effective implementation of DFT. It requires the length of the input to be a multiple of 2.

Chapter 3

Transformation of cellular automata

3.1 Introduction

This chapter consists of two parts. First, the process of transforming cellular automata such that multiple output variables emerge is explained. Then the process of evolving transformed CA is explained.

In the next chapter the transformation process and evolution of transformed CA will be tested in practice, experimenting with the choices and parameters described in this chapter.

3.2 Transformation of cellular automata

In problem solving, CA are usually treated as input-output systems and evolved to visit certain states given certain initial states. The n-th state of a CA is commonly treated as its output, such that one input gives one output. Figure 3.1 illustrates the way CA is used as a single-input, single-output system. The input is given in the form of an initial state. The CA is ran some time-steps and its ”final” state is its output.

The proposed transformation process gives CA multiple outputs. The transformation process consists of four main steps, illustrated in figure 3.2. First, the states of the CA are developed for some amount of time-steps. Second, the states are interpreted as numbers. Third, the numbers are Fourier transformed. Fourth, the spikes of the magnitude spectrum is interpreted as the outputs.

In each step there are settings and choices that affect the result of the transformation. The steps and the settings will be explained. In chapter 4, the effects of the settings will be determined experimentally.

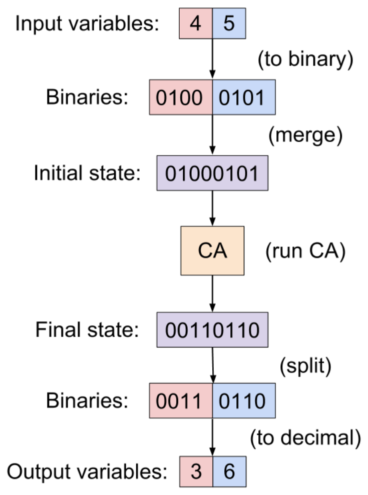


Figure 3.1: The single-input, single-output approach to CA as an input-output system.

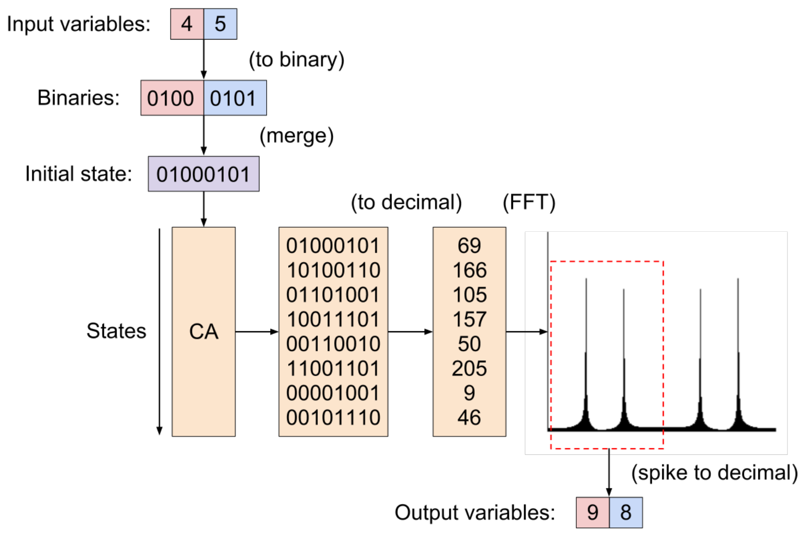


Figure 3.2: The transformation of CA such that multiple output variables emerge.

3.2.1 Development of states

The first step of the transformation process is to run the cellular automata for an amount of time-steps. There are some choices that can be made when developing CA which affects the behavior of CA. Some of the most prominent choices are explained: the rule space, the development target (what part of the CA to develop), the boundary condition and the size of CA.

Rule space

Before CA are developed, the rule space of CA must be determined. The scope limits the dimension of CA to the first dimension, which leaves the rule space. As described in chapter 2.1.1, the rule space defines the number of states per cell and the size of the neighborhood of CA, which in turn determines the amount of unique CA rules that exist. The most basic rule space is the elementary rule space, which has 2 states per cell and a neighborhood of 3. To increase the rule space one can either increase the amount of states per cell or the size of the neighborhood. The scope limits the size of the neighborhood to 3, which leaves increasing the amount of states per cell to 3. Summing up, the two rule spaces in focus are as follows.

- The elementary rule space (2 state per cell)
- The trinary rule space (3 states per cell)

Development target

The development target is the answer to the question: how many time-steps will the CA be developed? Since a CA has infinite states, there is no perfect answer to this. One answer is to develop the CA a set amount of time-steps. Another way is to develop the CA until it reaches a known state, which allows for some interesting development targets. Referring to chapter 2.1.3, the behavior of CA can be devided into two parts: the transient and the period. The transient is the states from the start of the CA until the first occurrence of a previously visited state. The period if the part that repeats itself infinitely after the transient is ended. Figure 2.5 illustrates this point. Combining the transient and period in various ways, one get the following development targets.

- N steps
- Transient
- 1 period
- Transient and 1 period
- Transient and N periods

Boundary condition

As discussed in chapter 2.1.4, the boundary condition determines what happens when reading a state outside of the CA. Figure 2.6 illustrates this. There are two common choices, both described in chapter 2.1.4.

- Cyclic boundary condition
- Non-cyclic boundary condition

Size of CA

As discussed in section 2.1.4, the size of CA is the platform of which behavior occurs. A big size allows for more unique states to occur, while a small size allows for fewer unique states to occur.

3.2.2 Interpretation of states

The states of CA need to be interpreted in a way that can be understood by a discrete Fourier transformation (DFT) algorithm. Normally DFT accepts a sequence of decimal numbers to be transformed. Two ways are proposed to interpret states into decimal numbers: decimal interpretation and summation interpretation.

Decimal interpretation

One way to translate a state into a decimal number is by first interpreting it as an n-ary number. In the example of elementary CA, the states are already represented as n-ary numbers, which makes converting them to decimal numbers straight forward. With a trinary CA, states can be interpreted as a trinary number and so on. Figure 3.3 shows an example of how the states of an elementary CA are interpreted to produce decimal numbers.

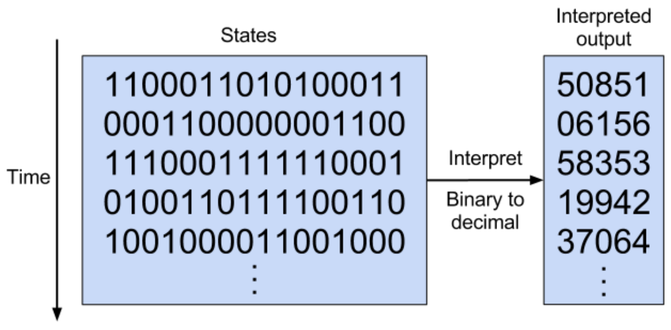


Figure 3.3: Interpretation of output. States are interpreted as binary numbers and converted to decimals.

The downside of treating states as n-ary numbers is that if the size of the CA is great, the interpreted decimal numbers will be too big to be efficiently represented by computers. To be able to translate the machine state represented by an n-ary number into a decimal number, the maximum size of the machine depends on the number of states per cell and how big numbers a computer can represent. The maximum number of states we can get when converting a state into a decimal number is:

$$\text{number of states}_{max} = \text{states per cell}^{size}$$

The highest number computers can represent by normal means is $2^{64} - 1$ or around 1.84×10^{19}, by using an unsigned 64 bit integer. Knowing this, the maximum size of the machine is given by the number of states per cell:

$$size_{max} = \left\lfloor \frac{log(2^{64})}{log(states)} \right\rfloor = \left\lfloor \frac{64}{log_2(states)} \right\rfloor$$

The trade-off between the number of states and the size of the CA is shown in figure 3.4.

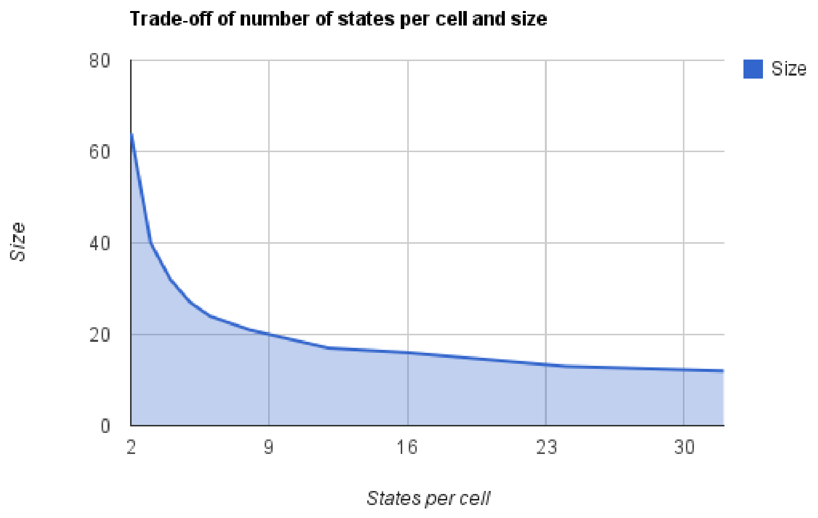


Figure 3.4: The trade-off between the number of states per cell and the size of the CA.

Summation interpretation

Another way to interpret a state into a decimal number is by calculating the sum of the values of the cells in the state. For example, the state "10203021" would be interpreted as 9. Using this method, a lot of states will be interpreted the same. This could be a great way to reduce the amount of unique CA in terms of behavior if the amount of CA is too large to be evolvable.

3.2.3 Fourier transformation of states

After interpreting the states as decimal numbers, they are transformed into frequencies by a discrete Fourier transform algorithm. Then the magnitude spectrum of the frequencies are calculated, which makes graphs with spikes. Figure 3.5 illustrates the transformation of a trinary CA into the magnitude spectrum. To the left is the behavior or a CA with a neighborhood of 3 and 2 states per cell. To the right is a graph showing the magnitudes of the transformation. In this example, the behavior consists of just a few repeated states. This results in a simple transformation consisting of a few spikes.

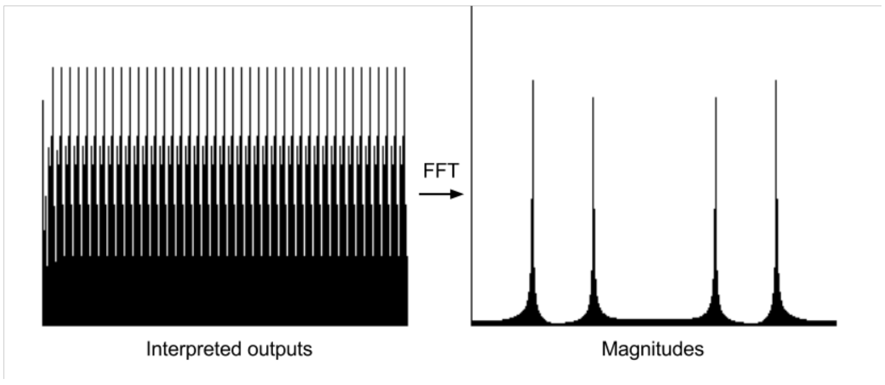


Figure 3.5: The transformation of a CA.

The fast Fourier algorithm is not used, as the length of the sequences can be unequal to powers of two.

3.2.4 Interpretation of spikes

After transforming the behavior of CA into spikes, the spikes are translated into variables. This is the final step, completing the transformation process of CA.

Figure 3.6 illustrates how the transformation of CA behavior is interpreted into output variables of CA. The first step of the translation process is to disregard the second half of the graph because of symmetry, see section 2.3. To eliminate noise a threshold is introduced. The spikes that are taller than the threshold will be translated into variables. The spikes translate into variables one to one. The values of the variables are equal to the heights of the spikes over the threshold. This way the variables are valued from zero and up.

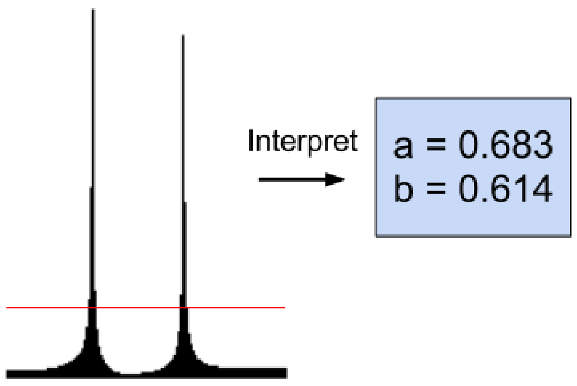


Figure 3.6: Interpretation of spikes into output variables. The variables are equal to the heights of the spikes over the threshold.

3.3 Evolution of transformed CA

Transformed CA are evolved using an evolutionary strategy (ES), see section 2.2.2. The 1+4-configuration of the algorithm will be used, meaning a population of one individual will be maintained, with four clones being created each generation.

3.3.1 Representation

The rules of CA are directly represented as strings. I.e. the elementary rule 1101110 (Wolfram code: 110) is represented as the string "1101110" and the trinary rule 120121002102100222210022201 is represented as "120121002102100222210022201".

3.3.2 Mutation

For each bit in the rule string there is a chance given by the mutation rate to flip the bit. In an elementary rule flipping a bit means turning 1 into 0 and vice versa. In a trinary rule space or higher, flipping a bit means randomly increasing or decreasing the bit by 1, modulus the number of states per cell.

3.3.3 Calculating fitness

Transformed CA are evaluated by calculating a fitness value based on how close its output variables are to a set of target variables. It is assumed that the CA has the correct amount of variables, otherwise it will be assigned a fitness of zero. The difference between the variables of the CA and the target variables are calculated using root mean squared deviation (RMSD), a statistical method similar to standard deviation. The RMSD of the output variables c and target variables t of lengths N is:

$$RMSD(c,t) = \sqrt{\frac{1}{N}\sum_{i=0}^{N}(c_i - t_i)^2}$$

The difference value is then translated into a fitness value between zero and one, where a high difference corresponds to a low fitness value near 0, and a low difference corresponds to a high fitness value near 1:

$$fitness = \frac{1}{1 + RMSD}$$

When evaluating CA in terms of how well it can produce multiple sets of output variables, the fitness of each of the sets of output variable are first calculated as described above. The final fitness is then the average of these fitnesses.

Chapter 4

Experimental approach

In this chapter the transformation process and evolution of transformed CA will be used in practice. First, the transformation process will be tested to determine what effect certain settings have on the transformation. Then, evolution of transformed CA will be tested to determine the evolvability of transformed CA with different settings.

A framework was developed in the programming language C# for running experiments. The framework supports CA of various rule spaces and settings. It handles transformation of CA as well as evolution of regular and transformed CA. All settings for the transformation process and evolution of CA presented in chapter 3 are supported in by framework.

The chapter is divided into 3 parts. In the first part it will be determined what effect different settings have on the transformation of CA. In the second part of it will be determined what effect different settings have on the evolvability of regular CA. In the third part transformed CA are evolved according to the findings in part 1 and 2. Evolved CA must be able to produce a set of output variables close to a set of target variables given a certain input. In the final experiment in part 3, transformed CA are evolved to produce multiple sets of output variables close to multiple sets of target variables given multiple inputs.

4.1 Transformation of CA

In this part the effects of certain parameters and settings on the shape of the magnitude transformation of CA will be determined. The goal is for the magnitude graph of transformed CA to contain clearly defined spikes that can easily be translated into the output variables of the transformed CA.

The following parameters and settings are identified as affecting the transformation process of CA. The parameters and settings will be explored to determine exactly what the effects are. The settings and parameters are explained in chapter 3, and listed below.

- The size of rule space. The complexity of CA.
- The size of CA. The number of cells in the CA.
- The boundary condition. Cyclic and non-cyclic boundary condition.
- State interpretation. The method of translating states into numbers.
- The development target. The part of the behavior to interpret and transform.

The settings affect different steps of the transformation process. The following affect the behavior of CA: the rule space, the size of CA and the boundary condition. One setting change the way the states of CA are interpreted: the state interpretation method. One setting change what part of the behavior that are interpreted: the development target. All the settings affect the final outcome of the transformation process.

The initial state in the experiments is a regular string in the form of "0101010..." that can be expanded to any size.

4.1.1 Experiment 1

In this experiment it will be determined how the two state interpretation methods affect the transformations of CA.

Six CA rules of different behavioral complexities will be tested using both state interpretation methods. The CA are as follows, listed by their Wolfram codes.

Rule 100	A simple static rule with a point attractor.
Rule 112	A rule with only localized change.
Rule 7	A rule with a short cyclic attractor.
Rule 90	A complex rule with long cyclic attractor nested pattern.
Rule 30	A chaotic rule.
Rule 110	The famous Turing complete (complex) rule.

The following settings will be used:

Rule space	Elementary (2 states per cell and neighborhood of 3)
Size of CA	63
Boundary condition	Cyclic
State interpretation	Both: summation and decimal interpretation
Development steps	500

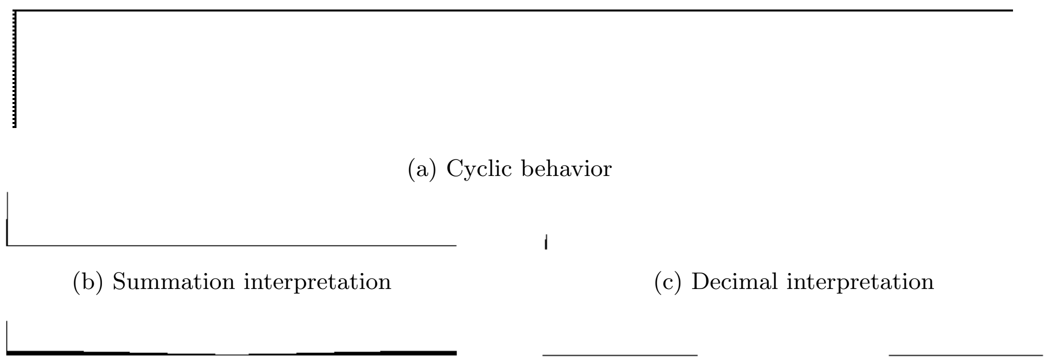
The following are the results of transforming the six rules with both summation and decimal state interpretation. For each rule, the behavior is shown in the first row. The interpreted states are shown in the second row and the magnitude graphs after transforming the interpreted states are in the third row. The interpreted states and magnitude graphs using summation interpretation and decimal interpretation are shown left and right, respectively. The three first transformations will be explained.

Figure 4.1 shows the transformation of rule 100. The behavior behavior consists of a single repeated active cell, shown in figure 4.1a. The interpreted states using both summation and decimal interpretation are flat lines, shown in figures 4.1b and 4.1c. The transformations of the flat lines also become flat lines, shown in figures 4.1d and 4.1e.

Figure 4.2 shows the transformation of rule 112. The behavior consists of a simple repeated pattern, shown in figure 4.2a. The sum of the pattern is constant, shown in figure 4.2b. The transformation of the constant states are blank, shown in figure 4.2d. The decimal interpreted states consists of repeating spikes of several different heights, shown in figure 4.2c. The transformation of the states consists of spikes all over the spectrum, mostly in the middle, shown in figure 4.2e.

Figure 4.3 shows the transformation of rule 7. The behavior consists of a simple repeated pattern, shown in figure 4.3a. The summation interpreted states consists of a pair of repeated signals, shown in figure 4.3b. The transformation of the repeated states consists of a single spike, shown in figure 4.3d. The decimal interpreted states consists of a simple repeated pattern, similar to the, shown in figure 4.3c. The transformation of the repeated consists of a spike, shown in figure 4.3e.

Figure 4.4 shows the transformation of rule 90, figure 4.5 shows the transformation of rule 30, and figure 4.6 shows the transformation of rule 110.



(a) Cyclic behavior

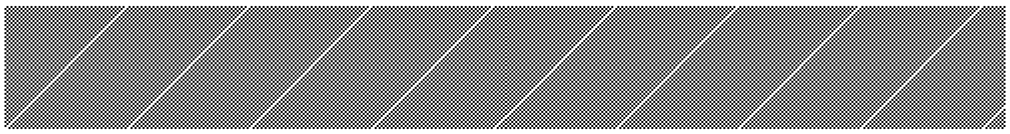
(b) Summation interpretation

(c) Decimal interpretation

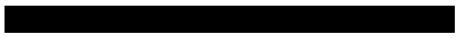
(d) Summation transformation

(e) Decimal transformation

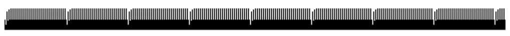
Figure 4.1: Transformation of rule 100 using both summation and decimal interpretation.



(a) Cyclic behavior

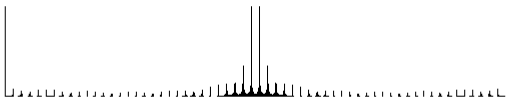


(b) Summation interpretation



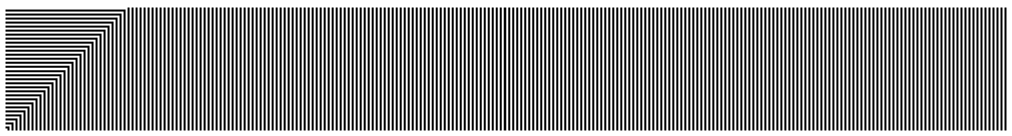
(c) Decimal interpretation

(d) Summation transformation

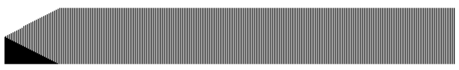


(e) Decimal transformation

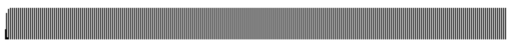
Figure 4.2: Transformation of rule 112 using both summation and decimal interpretation.



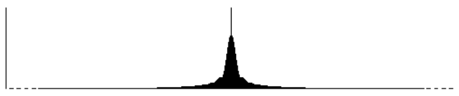
(a) Cyclic behavior



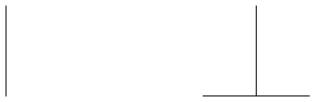
(b) Summation interpretation



(c) Decimal interpretation



(d) Summation transformation



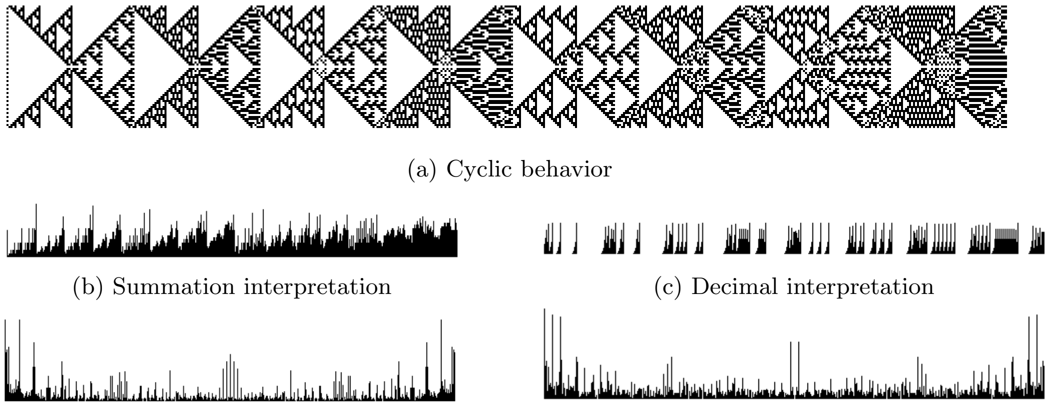
(e) Decimal transformation

Figure 4.3: Transformation of rule 7 using both summation and decimal interpretation.

Conclusions

At first glance, the both state interpretation methods seem to produce similar results in all the CA. However, if one take a close look at CA where the behavior appears random, the two interpretation methods produce very different results. When using summation interpretation on random states, the numbers become similar. With decimal interpretation, however, the numbers become very different. Summation interpretation could be a great way to decrease the number of unique CA in terms of behavior if the search space is too large for the search algorithm.

The first three CA produced very uninteresting results for both interpretations. They have in common that the interpretations of their states are repeating. These three CA



(a) Cyclic behavior

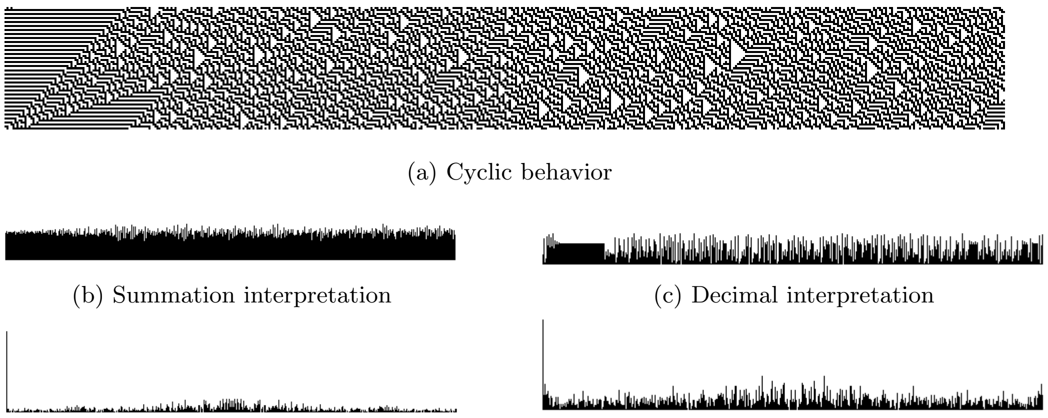
(b) Summation interpretation

(c) Decimal interpretation

(d) Summation transformation

(e) Decimal transformation

Figure 4.4: Transformation of rule 90 using both summation and decimal interpretation.



(a) Cyclic behavior

(b) Summation interpretation

(c) Decimal interpretation

(d) Summation transformation

(e) Decimal transformation

Figure 4.5: Transformation of rule 90 using both summation and decimal interpretation.

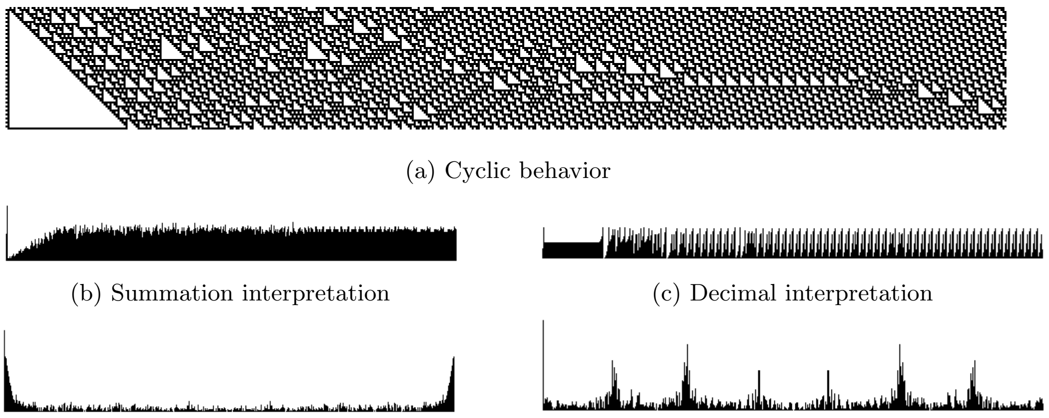
will be dropped from future experiments.

4.1.2 Experiment 2

In this experiment it will be determined how the boundary condition affect transformations of CA.

The three CA will be tested using the non-cyclic boundary condition and the results will be compared to the results of experiment 1.

The settings are:


(a) Cyclic behavior

(b) Summation interpretation

(c) Decimal interpretation

(d) Summation transformation

(e) Decimal transformation

Figure 4.6: Transformation of rule 110 using both summation and decimal interpretation.

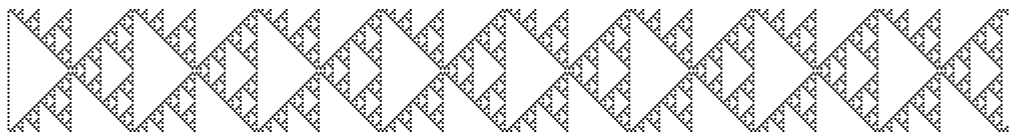
Rule space	Elementary (2 states per cell and neighborhood of 3)
Size of CA	63
Boundary condition	Non-cyclic
State interpretation	Decimal
Development steps	500

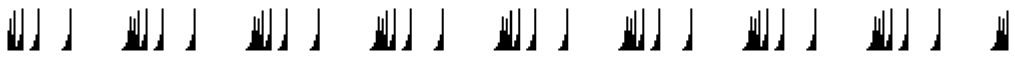
Results

The following are the results of transforming the three rules with the non-cyclic boundary condition and decimal state interpretation. Figure 4.7 shows the transformation of the first rule, figure 4.8 shows the transformation of the first second rule, and figure 4.9 shows the transformation of the third rule. For each rule, the behavior is shown in the first row, the interpreted states are shown in the second row and the magnitude graphs after transforming the interpreted states are in the third row.

Conclusions

The shapes of the transformation graphs change a little using non-cyclic compared to the results of experiment 1. The change is small enough that the choice of boundary condition can be deemed trivial. To make progress in the experiments, one of the two is kept and the other drops from future experiments. Cyclic boundary condition is kept because it is more relevant in relation to other work.


(a) Non-cyclic behavior


(b) Decimal interpretation

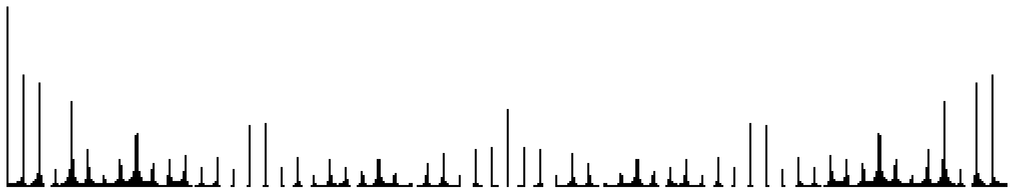
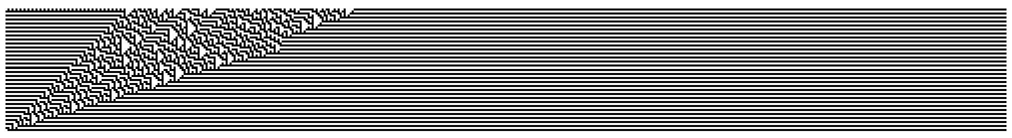
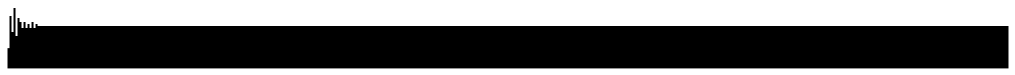

(c) Magnitude spectrum

Figure 4.7: Transformation of rule 90 using non-cyclic boundary condition.


(a) Non-cyclic behavior


(b) Decimal interpretation

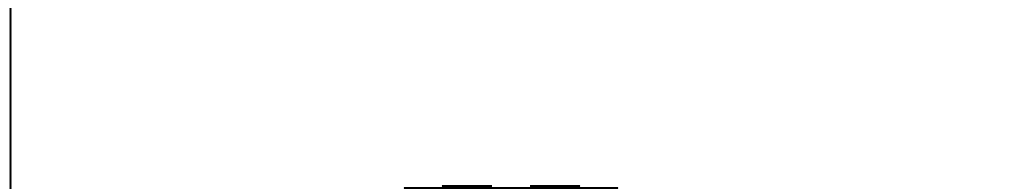
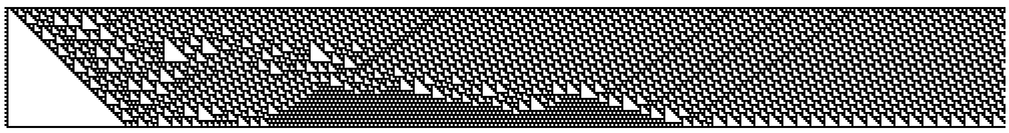
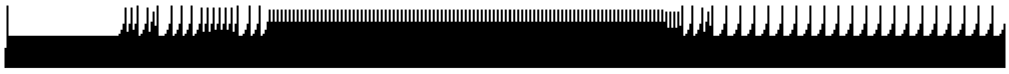

(c) Magnitude spectrum

Figure 4.8: Transformation of rule 30 using non-cyclic.

4.1.3 Experiment 3

In this experiment it will be determined how the size of CA affect the transformation process. A much smaller size for the CA will be used, to see how it impacts the trans-


(a) Non-cyclic behavior


(b) Decimal interpretation

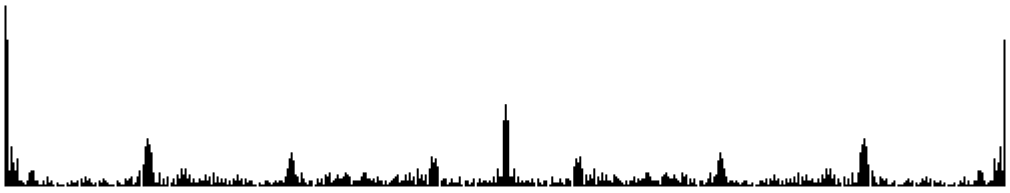

(c) Magnitude spectrum

Figure 4.9: Transformation of rule 30 using non-cyclic boundary condition.

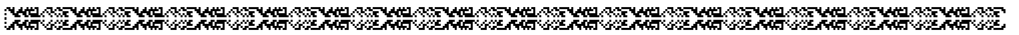
formation. The new size is 11.

The settings are as follows.

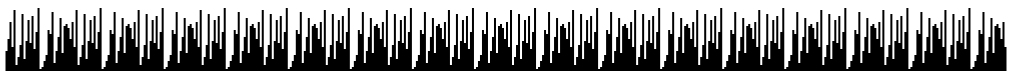
Rule space	Elementary (2 states per cell and neighborhood of 3)
Size of CA	11
Boundary condition	Cyclic
State interpretation	Decimal
Development steps	500

Results

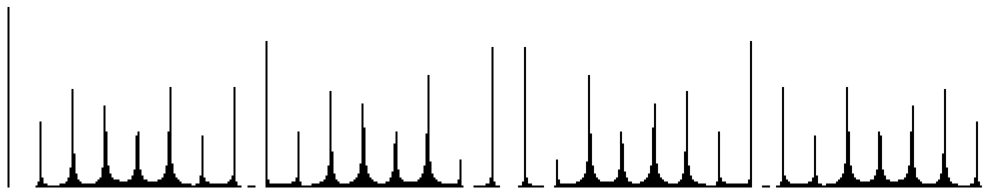
The following are the results of transforming the three rules using a small CA size and the cyclic boundary condition and decimal state interpretation. Figure 4.10 shows the transformation of the first rule, figure 4.11 shows the transformation of the first second rule, and figure 4.12 shows the transformation of the third rule. For each rule, the behavior is shown in the first row, the interpreted states are shown in the second row and the magnitude graphs after transforming the interpreted states are in the third row.



(a) Cyclic behavior

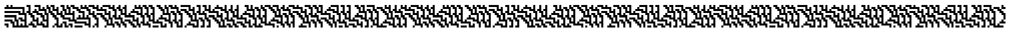


(b) Decimal interpretation

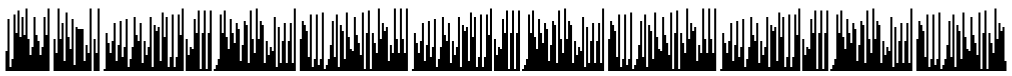


(c) Magnitude spectrum

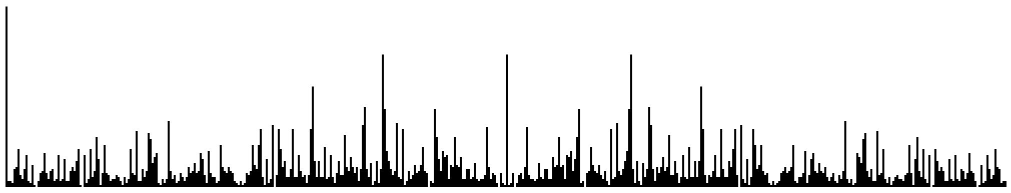
Figure 4.10: Transformation of rule 90 using a small CA size.



(a) Cyclic behavior



(b) Decimal interpretation



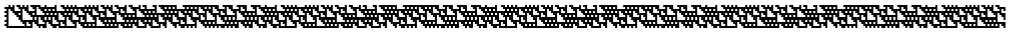
(c) Magnitude spectrum

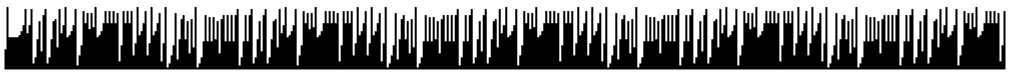
Figure 4.11: Transformation of rule 30 using a small CA size.

Conclusions

The magnitude spectra using the lower size are simpler than their larger counter-parts, having smoother graphs and fewer spikes. This happens because the number of unique states depends on the size. The method of state interpretation also affects the transformation. With decimal interpretation, the number of unique interpreted states is equal to the number of unique (behavior) states. With summation interpretation the number of unique interpreted states is equal to the size of the CA.

Using a small size decreases the number of unique interpreted states and therefore also the number of unique transformations. This could be a great way to increase the evolv-


(a) Cyclic behavior


(b) Decimal interpretation

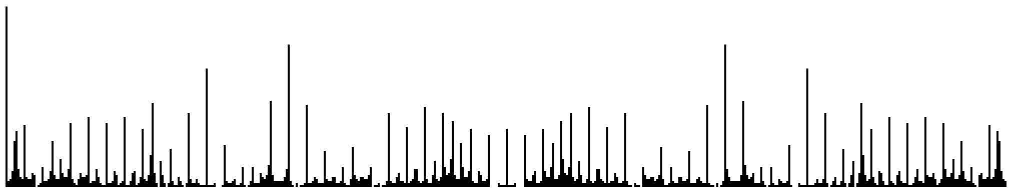

(c) Magnitude spectrum

Figure 4.12: Transformation of rule 110 using a small CA size.

ability of a rule space that is otherwise too large for the search algorithm to handle. This effect is the same as using the summation interpretation method, as found in experiment 1.

4.1.4 Experiment 4

In this experiment it will be determined how the development target, which part of the CA behavior to transform, affects the transformation process of CA.

Instead of transforming the first N states of CA, where N is 500 in the previous experiments, the CA will run until it reaches a known state, and various configurations of behavior will be interpreted and transformed.

The three CA will be tested with the following development targets:

- Transient only
- 1 period
- 3 periods
- Transient and 1 period
- Transient and 3 periods

The settings are as follows.

Rule space	Elementary (2 states per cell and neighborhood of 3)
Size of CA	11
Boundary condition	Cyclic
State interpretation	Summation
Development target	Various combinations of the transient and periods.

A size of 11 and the summation interpretation will be used, as they both simplify the transformation graphs. This will make it easy to observe the transformation target has on the graphs.

Results

The following are the results of transforming the three rules using different development. Figure 4.13 shows the transformation of the first rule, figure 4.14 shows the transformation of the first second rule, and figure 4.15 shows the transformation of the third rule. For each rule, the final transformation is shown using five different development targets. From top to bottom: only transient, only one period, transient and one period, three periods, transient and three periods.

Conclusions

The number of periods determines the resolution in the horizontal axis of the transformation graph. In some cases too few periods can make the spikes inseparable.

The transformations of CA without the transient appear sparse, with "holes" in them. Including the transient puts "meat on the legs".

4.1.5 Experiment 5

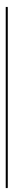
In this experiment it will be determined how changing the rule space affects the transformations process.

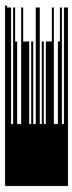
The rule space is increased to a trinary rule space. The size of the neighborhood is still 3.

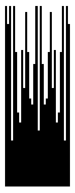
Three randomly chosen CA will be worked with:

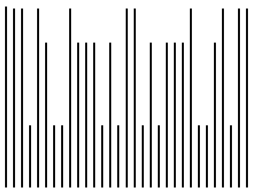
- Rule 212201220100121220020021111 (Wolfram code: 6704062261843)
- Rule 120102011120200000211210121 (Wolfram code: 435281430568)
- Rule 020022221201222021121102011 (Wolfram code: 1788232153180)

The settings are as follows.


(a) Transient


(b) Period


(c) Transient and period


(d) Three periods

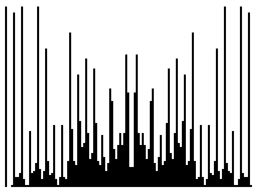
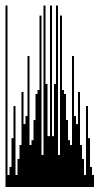
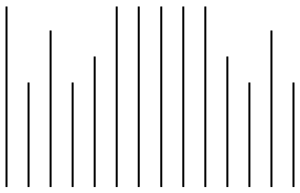

(e) Transient and three periods

Figure 4.13: Transformation of rule 90 different development targets.

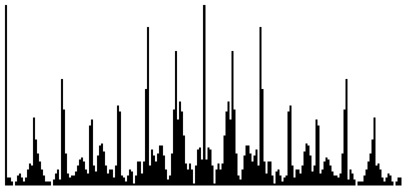
Rule space	trinary
Size of CA	31
Boundary condition	Cyclic
State interpretation	Decimal
Development steps	500



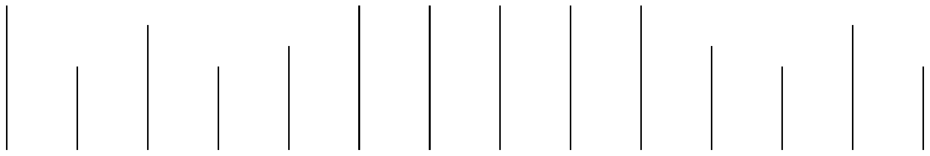
(a) Transient



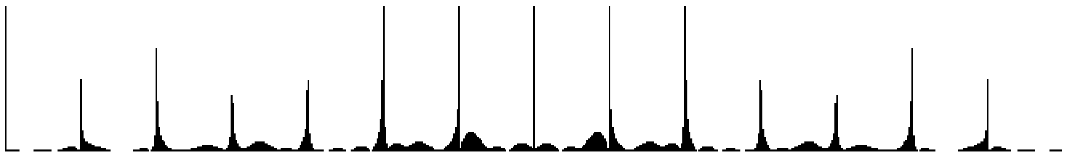
(b) Period



(c) Transient and period



(d) Three periods

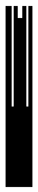


(e) Transient and three periods

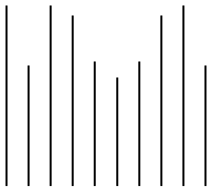
Figure 4.14: Transformation of rule 30 different development targets.

Results

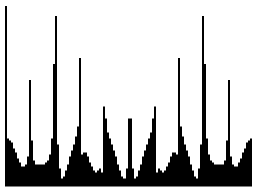
The following are the results of transforming the three trinary rules with cyclic boundary condition and decimal state interpretation. Figure 4.16 shows the transformation of the first rule, figure 4.17 shows the transformation of the first second rule, and figure 4.18 shows the transformation of the third rule. For each rule, the behavior is shown in the first row, the interpreted states are shown in the second row and the magnitude graphs after transforming the interpreted states are in the third row.



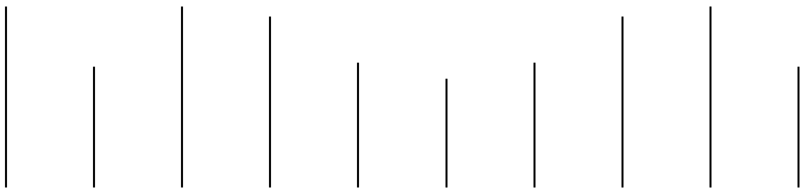
(a) Transient



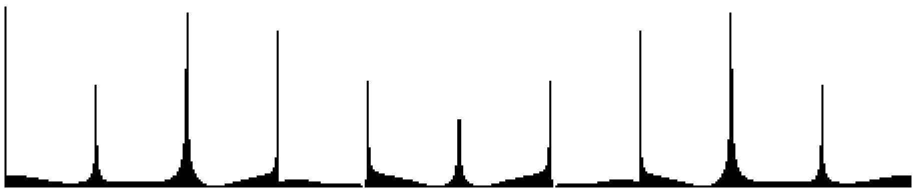
(b) Period



(c) Transient and period



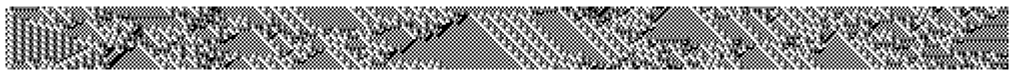
(d) Three periods

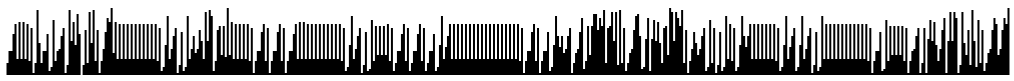


(e) Transient and three periods

Figure 4.15: Transformation of rule 110 different development targets.

The states in the behavior are shown as white (0), gray (1) and black (2).


(a) Cyclic behavior


(b) Decimal interpretation

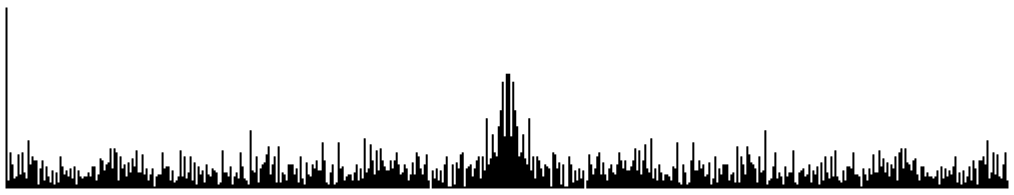
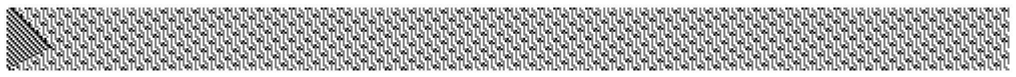
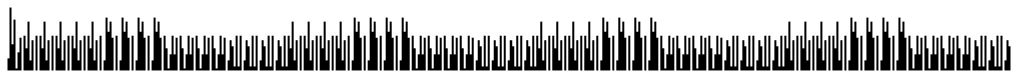

(c) Magnitude spectrum

Figure 4.16: Transformation of rule Rule 212201220100121220020021111 (Wolfram code: 6704062261843) using decimal interpretation.


(a) Cyclic behavior


(b) Decimal interpretation

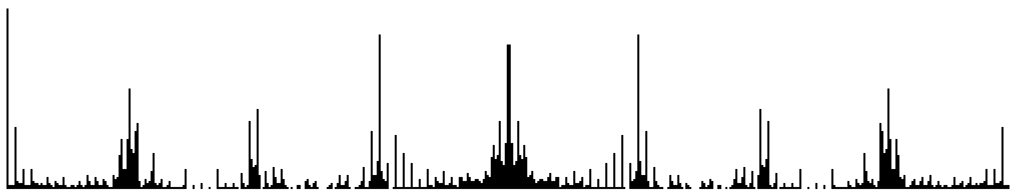
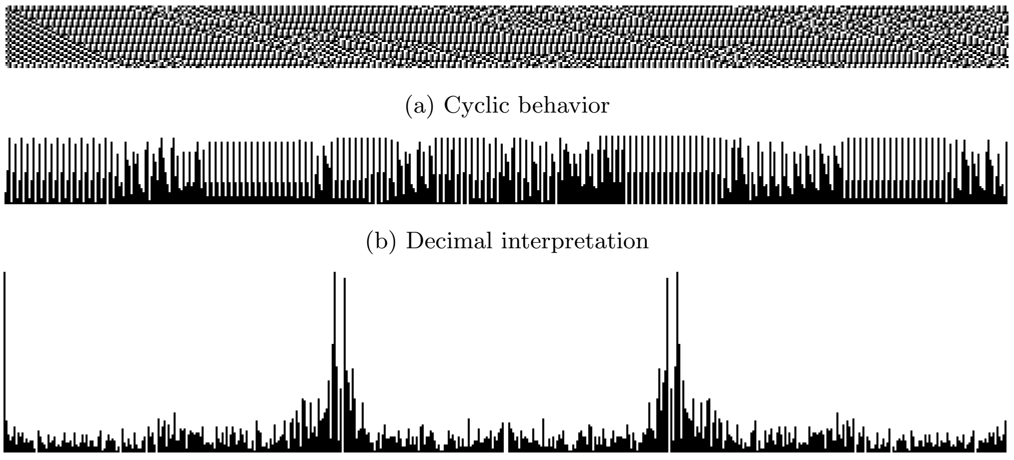

(c) Magnitude spectrum

Figure 4.17: Transformation of rule Rule 120102011120200000211210121 (Wolfram code: 435281430568) using decimal interpretation.

Conclusions

The increased rule space produce more complex transformations. This is because the number of unique states increases.


(a) Cyclic behavior

(b) Decimal interpretation

(c) Magnitude spectrum

Figure 4.18: Transformation of rule Rule 020022221201222021121102011 (Wolfram code: 1788232153180) using decimal interpretation.

4.1.6 Discussion

We have conducted experiments to determine what effect certain settings have on the transformation of CA.

The settings have different effects on the shape of the magnitude spectrum. The rule space changes the size of the search space, allowing more unique CA to exist. The size of CA and the state interpretation method affect the behavior of CA. A low size and summation interpretation have in common that they make many CA behave equally, effectively decreasing the number of unique CA. A large size and decimal interpretation have in common that they maintain the diversity of CA. The boundary condition changes the behavior of CA slightly. The behavior target changes the shape and resolution of the final transformed graph.

Since controlling a specific system requires a specific CA, the rule space should be larger than the elementary rule space, e.g. the trinary rule space. A low CA size and the summation interpretation method could be used to compensate for the enormous size. Hopefully a combination of one or both of these settings can make the trinary rule space evolvable while still including some CA that are complex enough to control a system.

With this knowledge transformed CA could be evolved, but first it must be determined that the evolutionary algorithm is able to evolve regular CA.

4.2 Evolution of CA

Before evolving transformed CA, it must be determined that the evolutionary algorithm (EA) is able to evolve regular CA. First CA in the elementary rule space are evolved to make sure the EA works properly. Then, the evolvability of a larger rule space is tested.

An evolution strategy (ES) with 1+4 configuration will be used, as explained in chapter 2.2.2.

4.2.1 Experiment 1

In this experiment it will be verified that the evolutionary strategy (ES) works as intended. The algorithm will be verified by reproducing the results of a known experiment, carried out by S. Nichele[6].

The goal of the experiment is to find the elementary CA that given the following initial state developed 64 steps gives the following final state.

Initial state	00000000000000000000000000000000100000000000000000000000000000000
Final state	11111111111111111111111111111111011111111111111111111111111111111

The CA settings are as follows.

Rule space	Elementary
Size of CA	65
Boundary condition	Cyclic
State interpretation	Decimal
Development steps	64

The ES settings are as follows.

Number of runs	1000
Maximum number of generations	100000
Mutation rate	0.15

Results

The results of the experiment are shown in table 4.1. The CA are listed by their Wolfram numbers.

Mean number of generations	10.421
Standard deviation	104.282
Found CA	Rules 51, 90, 105, 122, 123, 203, 217, 218, 219, 250

Table 4.1: The results of the evolution of elementary CA.

Conclusions

All CA that were identified in the original experiment was found, suggesting the ES works properly.

The number of possible CA is 256 and as found by Stefano 10 of them are solutions to this problem. In a random search the mean number of picks to find a solution would be $256/10 = 25.6$. The mean we found is considerably smaller, suggesting that the algorithm is effective and provides evolvability.

4.2.2 Experiment 2

In this experiment we are going to redo experiment 1 using various mutation rates to find one that maximizes evolvability.

The data for mutation rate of 0.15 is kept from experiment 1.

The settings for CA and ES are the same as in experiment 1.

Results

Table 4.2 shows the results given different mutation rates. They are illustrated in figure 4.19.

Mutation rate	Mean number of generations	Standard deviation	CA found
0.10	14.842	261.907	All 10
0.15	10.421	104.282	All 10
0.20	8.007	58.645	All 10
0.25	6.931	39.806	All 10
0.30	7.113	46.274	All 10
0.35	6.476	34.223	All 10
0.40	6.644	34.941	All 10
0.45	6.581	38.131	All 10
0.50	6.650	40.890	All 10
0.55	6.848	37.103	All 10
0.60	6.984	41.264	All 10
0.65	7.183	45.322	All 10
0.70	7.619	58.110	All 10
0.75	7.514	46.452	All 10
0.80	8.408	60.164	All 10
0.85	9.283	80.229	All 10
0.90	11.429	125.569	All 10

Table 4.2: The results of the evolution of elementary CA using different mutation rates.

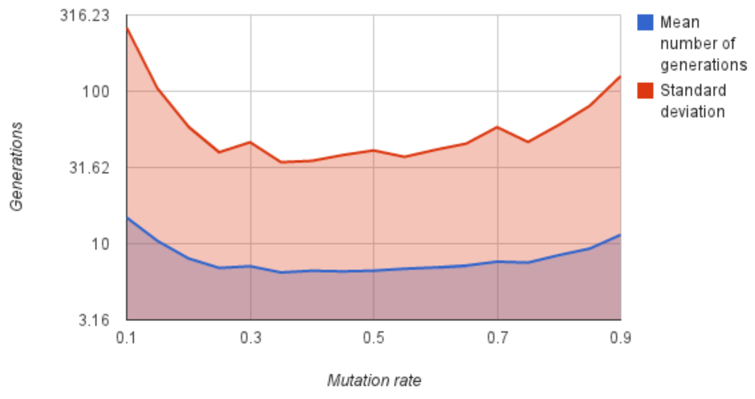


Figure 4.19: The effectiveness given different mutation rates.

Conclusions

Figure 4.19 shows that mutation rates between 0.25 and 0.75 produce good results, all of them considerably improving the effectiveness of the search. A mutation rate of 0.35 gave the lowest mean number of generations and standard deviation.

4.2.3 Experiment 3

In this experiment it will be determined how a larger rule space affects evolvability. The rule space is increased to a trinary rule space.

The number of possible CA is now $3^{3^3} \approx 7.6 \times 10^{12}$, many magnitudes larger than in the previous experiments.

A random CA (rule) is created and its final state (64 steps) is calculated. This way it is known that at least one CA with a correct solution exists.

Initial state	0000000000000001000000000000000
Target final state	1222112021221012220021100022110
Target CA	120121002102100222210022201

The CA settings are as follows.

Rule space	trinary
Size of CA	31
Boundary condition	Cyclic
State interpretation	Decimal
Development steps	64

The ES settings are as follows.

Number of runs	10
Maximum number of generations	10000
Mutation rate	0.35

Results

No solutions were found within 10000 generations. Figure 4.20 shows a fitness plot showing the fitness every 100th generation of one of the searches:

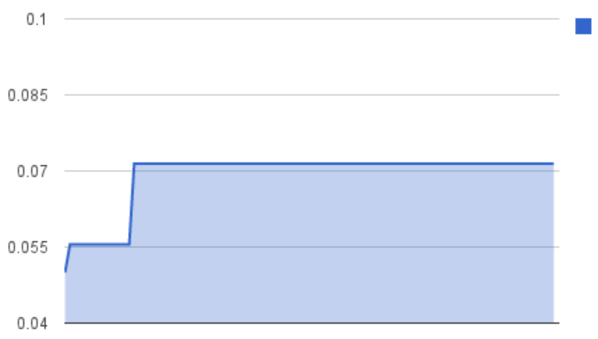


Figure 4.20: Fitness plot with a CA size of 31 and mutation rate of 0.35.

Conclusions

All ten searches get stuck early and don't improve notably within 10000 generations. In experiment 1 it was determined that the algorithm works properly, which suggests something goes wrong when moving to the larger rule space. Possible reasons for this include:

- The mutation rate is too high, causing the search to jump too far, unable to climb the landscape.
- The mutation rate is too low, making the search unable to jump out of a local maximum.
- The evolvability of the algorithm doesn't scale with the size of the rule space.

If the mutation rate is the problem, it is easily testable and fixable. If the scalability of the algorithm is the problem, there are a couple of ways to compensate for it. Part

1 and 2 of the experiments suggested two ways to compensate for a large rule space: a low CA size and summation interpretation instead of decimal interpretation. It will be tested whether changing the mutation rate or compensating for scalability can improve the evolvability.

4.2.4 Experiment 4

In this experiment it will be determined how changing the mutation rate affects the evolvability.

One lower and one higher mutation rate will be tested. Their fitness plots will be compared to the result of experiment 3.

The CA settings are as follows.

Rule space	trinary
Size of CA	31
Boundary condition	Cyclic
State interpretation	Decimal
Development steps	64

The ES settings are as follows.

Number of runs	10
Maximum number of generations	10000
Mutation rate	0.035 and 0.5

Results

Figure 4.21 shows the fitness plot of the evolution using a mutation rate of 0.035. Figure 4.22 shows the fitness plot of the evolution using a mutation rate of 0.5.

Conclusions

Both searches get stuck early, just like in experiment 3. This suggests the mutation rate isn't the reason for the low evolvability. Knowing this, the most probable reason for the low evolvability is that the algorithm doesn't scale with the size of the rule space.

4.2.5 Experiment 5

In this experiment it will be determined how reducing the size of CA affects evolvability. Reducing the size of CA could compensate for the algorithm not scaling with the size of the rule space.

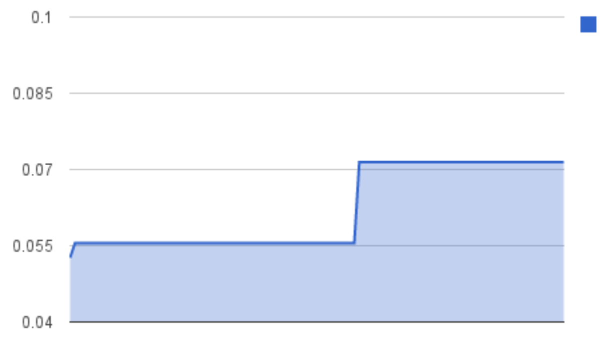


Figure 4.21: Fitness plot using a mutation rate of 0.035.

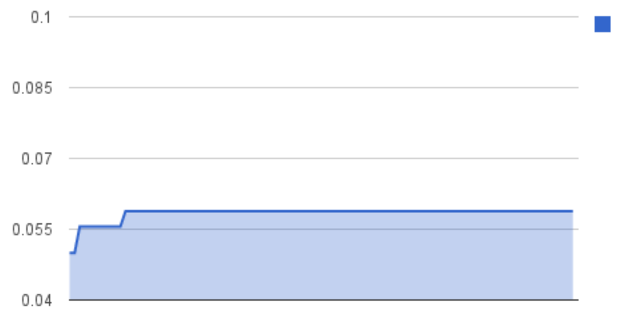


Figure 4.22: Fitness plot using a mutation rate of 0.5.

Two lower sizes will be tested: 11 and 7. The CA settings are as follows.

Rule space	trinary
Size of CA	11 and 7
Boundary condition	Cyclic
State interpretation	Decimal
Development steps	64

The ES settings are as follows.

Number of runs	100
Maximum generations	20000
Mutation rate	0.35

Results

With a CA size of 11, a solution is found within 20000 generations 21 out of 100 times. Figure 4.23 shows the fitness plot for one of the successful searches.

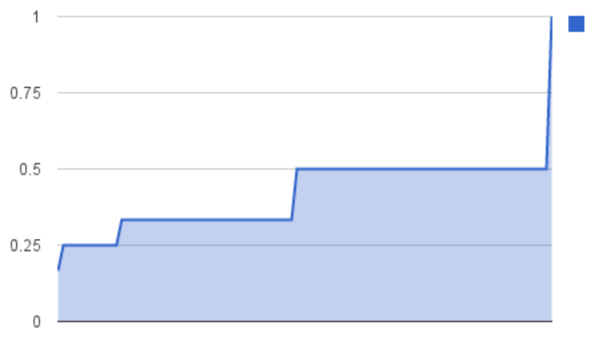


Figure 4.23: Fitness plot using a CA size of 11.

With a size of 7, a solution is found within 20000 generations every time. Table 4.3 shows the result of the experiment.

Mean number of generations	212.17
Standard deviation	40983.341
Minimum number of generations	2
Maximum number of generations	1227
Unique CA found	100

Table 4.3: The results of the evolution, using a size of 11.

Figure 4.24 shows the fitness plot of one of the searches:

Conclusions

Decreasing the size of CA considerably improved the evolvability of the system. In the previous experiments, the system was unevolvable with a size of 31. Lowering the size to 11 made the algorithm capable of finding solutions in some of the searches. Lowering it again to 7 made the algorithm capable of finding solutions in all the searches.

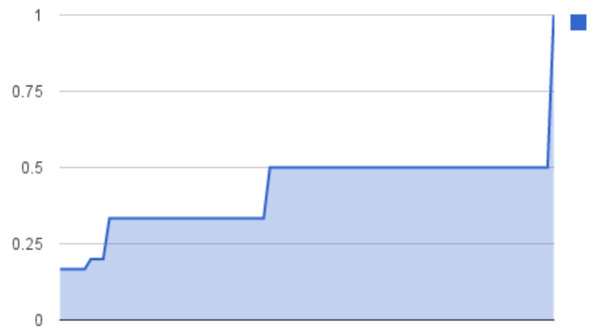


Figure 4.24: Fitness plot using a CA size of 7.

4.2.6 Discussion

We have experimented with the evolvability of CA in two different rule spaces: the elementary (2-state) rule space and the trinary rule space. The elementary rule space is highly evolvable. The trinary rule space is not evolvable. Changing the mutation rate did not make the trinary evolvable, suggesting the algorithm doesn't scale with the size of the rule space. A technique for compensating for the large rule space was tested: lowering the size of CA. It successfully made the trinary rule space evolvable.

4.3 Evolution of transformed CA

In this part of the experiments chapter transformed CA will be evolved using the knowledge gained from parts 1 and 2.

The following settings for CA were found in part 1 to produce good transformations of CA:

Rule space	Elementary (2-state) or trinary
Size	31 with elementary or 7 with trinary
Boundary condition	Cyclic
State interpretation	Decimal or summation
Behavior target	The transient and one period.

The following mutation rate were found in part 2 to work for evolving CA in both the 2-state and trinary rule spaces:

Mutation rate	0.35

Initially the elementary rule space will be used, then the trinary rule space will be used.

4.3.1 Experiment 1

In this experiment elementary CA will be evolved to determine the evolvability of transformed CA in the elementary rule space.

A target CA is developed and transformed using decimal interpretation with the development target being the transient and one period. CA will be evaluated by comparing their transformation graph to the transformation graph of the target.

Target CA	Rule 97
Initial state	0101010101010101010101010101010

The target magnitude spectrum is shown in figure 4.25.

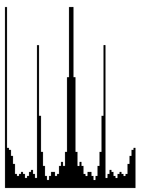


Figure 4.25: The target magnitude spectrum.

The CA settings are as follows.

Rule space	Elementary
Size of CA	31
Boundary condition	Cyclic
State interpretation	Decimal
Development target	Transient and period

The ES settings are as follows.

Number of runs	100
Maximum generations	10000
Mutation rate	0.35

Results

Table 4.4 shows the results of the experiment.

Mean number of generations	82.860
Standard deviation	10560.800
Min number of generations	1
Max number of generations	434
CA found	Rule 97 and 41

Table 4.4: The results of the evolution of transformed elementary CA, evaluated by the shape of their transformed output.

Conclusions

Two CA was found, one of which was the target. The transformation of the CA that was not the target, rule 41, is shown in figure 4.26.

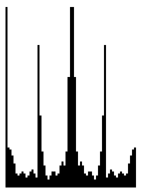


Figure 4.26: The magnitude spectrum of rule 41.

The transformation of rule 41 is identical to rule 97, suggesting that the algorithm works properly.

4.3.2 Experiment 2

In this experiment evolved transformed elementary CA will be evaluated by their output variables which are translated from the spikes of their transformation graphs.

The target CA is rule 97. Figure 4.27 shows how the output variables of rule 97 is calculated: first, the second half of the graph is dropped because of symmetry. Second, a threshold is chosen, and spikes over this threshold is analyzed. The height of the spikes over the threshold becomes the output variables of the CA. The process of translating spikes into variables is described in section 3.2.4.

According to the conversion rules described in chapter 3.2.4, the output variables of rule 97 are:

Target variables	7.38, 0.46 and 0.30

The CA settings are as follows.

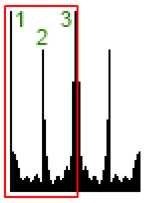


Figure 4.27: The interpretation of spikes into variables.

Rule space	Elementary
Size of CA	31
Boundary condition	Cyclic
State interpretation	Decimal
Development target	Transient and period

The ES settings are as follows.

Number of runs	100
Maximum generations	10000
Mutation rate	0.35

Results

Table 4.5 shows the results of the experiment.

Mean number of generations	18.560
Standard deviation	344.446
Min number of generations	1
Max number of generations	87
CA found	Rule 97 and 41

Table 4.5: The results of the evolution of transformed elementary CA, evaluated by the spikes in their transformed output.

Conclusions

The results are the same as in experiment 1, suggesting that the spike translation works and is evolvable.

4.3.3 Experiment 3

In this experiment evolved transformed elementary CA will be evaluated by how close they get to producing three randomly generated target variables. This means no perfect solution should exist. Different fitness thresholds will be used to see how many solutions exist with fitnesses higher than each fitness threshold. Both state interpretation methods will be tested: summation interpretation and decimal interpretation.

Target variables	0.21, 0.73, 0.78

The CA settings are as follows.

Rule space	Elementary
Size of CA	31
Boundary condition	Cyclic
State interpretation	Decimal
Development target	Transient and one period

The ES settings are as follows.

Number of runs per threshold	100
Maximum generations	10000
Mutation rate	0.35

Results

Table 4.6 shows the results of the experiment. For each fitness threshold the number of solutions with a higher fitness than the threshold is shown, for both summation and decimal interpretation.

	Unique solutions found in 100 runs	
Fitness threshold	Summation interpretation	Decimal interpretation
0.9	0	0
0.8	0	0
0.7	0	0
0.6	0	2
0.5	0	2
0.4	0	2
0.3	0	5
0.2	2	38
0.1	6	38

Table 4.6: The results of the evolution of trinary CA, using different fitness thresholds and both state interpretation methods.

Conclusions

Decimal interpretation produced better results than summation interpretation in the elementary rule space. The best solutions using summation interpretation had a fitness between 0.2 and 0.3. The best solutions using decimal interpretation had a fitness between 0.6 and 0.7. As discussed in part 1 and 2 of the experiments, summation interpretation makes many CA behave the same, practically making the search space smaller. In this case the rule space is already small, making it unnecessary to decrease it further.

A fitness between 0.6 and 0.7 is not very good. This illustrates well the problem of finding specific CA in a small rule space: there aren't enough CA that very specific ones exist. Luckily we have experimented with a larger rule space: the trinary rule space.

4.3.4 Experiment 4

In this experiment the rule space is increased to the trinary rule space. It will be determined how the larger rule space affects evolvability of transformed CA. Different fitness thresholds will be used to see how many solutions exist with fitnesses higher than each fitness threshold. Both state interpretation methods will be tested.

One search will be conducted for a maximum 100000 generations for each CA size and state interpretation method. Because they are ran only one time each, the results may not be totally representative, but it should give some general idea about which size and state interpretation method works best in the trinary rule space.

The target variables and settings of CA and ES are the same as in the previous experiment.

Target variables	0.21, 0.73, 0.78

The CA settings are as follows.

Rule space	trinary
Size of CA	31
Boundary condition	Cyclic
State interpretation	Decimal
Development target	Transient and one period

The ES settings are as follows.

Number of runs per threshold	100
Maximum generations	10000
Mutation rate	0.35

Results

Table 4.7 shows the results of the experiment. For each fitness threshold the number of solutions with a higher fitness than the threshold is shown, for both summation and decimal interpretation. Figure 4.28 show the results shown graphically.

Figure 4.29 shows the fitness plot of one of the runs, the one with a size of 9 and decimal interpretation.

	Highest fitness found after 100000 generations	
Size	Summation interpretation	Decimal interpretation
5	0.670	0.644
7	0.687	0.681
9	0.673	0.776
11	0.682	0.704
13	0.681	0.765
15	0.688	0.756
17	0.696	0.696

Table 4.7: The results of the evolution of trinary CA, using different sizes and both state interpretation methods.

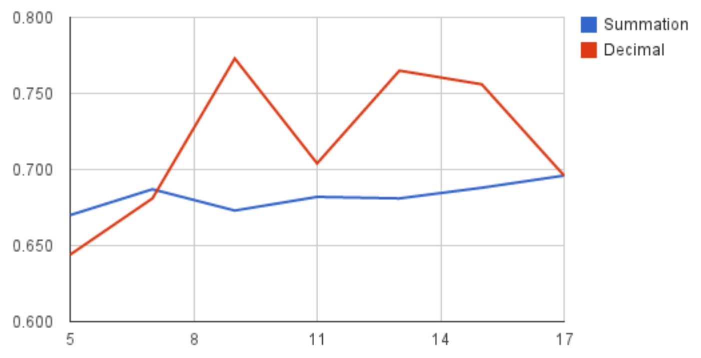


Figure 4.28: The best fitness of various sizes of CA after 100000 generations.

Conclusions

It was hypothesized that summation interpretation might improve the evolvability in a large rule space, but this has now been shown to be false. Decimal interpretation works better than summation interpretation in almost all cases.

The size of 9 produces the best fitness of 0.776 using decimal interpretation. For controlling a system where the CA needs to match multiple inputs with outputs, a fitness of 0.776 for only one set of input and outputs is not promising.

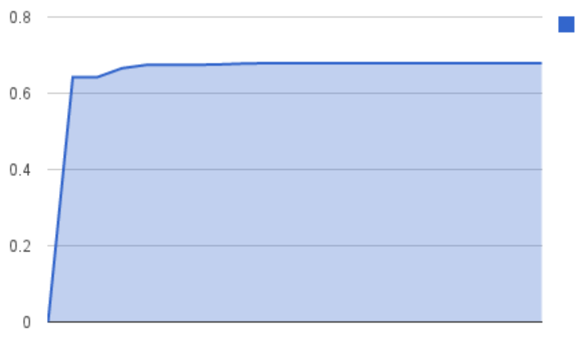


Figure 4.29: The best fitness of the run with size 9 and decimal interpretation.

4.3.5 Experiment 5

In this experiment it will be determined if running the evolution for a long time improves the result of the evolution. If it doesn't improve the result, it suggests that the algorithm doesn't scale with the size of the rule space, and that lowering the size of CA is not enough to compensate for this.

The search will be ran one time for a total of 1000000 generations.

The CA settings are as follows.

Rule space	trinary
Size of CA	9
Boundary condition	Cyclic
State interpretation	Decimal
Development target	Transient and one period

The ES settings are as follows.

Number of runs	1
Maximum generations	1000000
Mutation rate	0.35

Results

Table 4.8 shows the final fitness value after 1000000 generations.

Final fitness	0.738

Table 4.8: The fitness of the evolved trinary CA, after 1000000 generations.

Conclusions

The result was not better than in the previous experiment where the evolution ran for a shorter amount of generations. This suggests the algorithm does not scale with the size of the rule space. It also suggests that the two ways that could compensate for this don't work.

4.3.6 Experiment 6

In this experiment, transformed CA will be evolved to simulate a multi-variable system. Transformed CA must be able to produce multiple sets of output variables that are close to multiple sets of target variables, given multiple sets of inputs.

The system to be simulated is an imaginary one, consisting of one or more sensors that can measure the state of the system, a control unit, and one or more motors. At every time-step the sensors send the state of the system to the control unit, which calculates what power the motors should have to stabilize the system, and send it to the motors.

We are going to find a CA that can act as the control unit. It should be able to, given states of the system as initial states, produce output variables when transformed, that are good power values for the motors.

A set of states and the appropriate power values for the motors in these states are known for the system. A transformed CA will be evolved that can match those states to their corresponding power values as well as possible.

Five sets of inputs and outputs will be randomly generated. How close can a transformed CA get to matching the inputs with the outputs using the knowledge gained from part 3 of the experiments?

The sets of input states and their corresponding target output variables are as follows.

Set	Input (initial state)	Target output variables
1	"102001001"	0.45, 1.11, 0.86
2	"120010201"	0.11, 0.99, 1.55
3	"222010020"	0.73, 0.39, 0.91
4	"100220110"	1.21, 0.23, 0.41
5	"201121101"	0.87, 1.04, 0.76

The CA settings are as follows.

Rule space	trinary
Size of CA	9
Boundary condition	Cyclic
State interpretation	Decimal
Development target	Transient and one period

The ES settings are as follows.

Number of runs	1
Maximum generations	1000000
Mutation rate	0.35

Results

Table 4.9 shows the fitness of the final transformed CA. Table 4.10 shows the sets of output variables of the final transformed CA.

Final fitness	0.620

Table 4.9: The fitness of the evolved trinary CA, after 1000000 generations.

Set 1	1,62 0,34 0,21
Set 2	1,64 0,01 0,36
Set 3	1,36 0,4 0,49
Set 4	1,46 0,45 0,22
Set 5	1,55 0,34 0,31

Table 4.10: The fitness of the evolved trinary CA, after 1000000 generations.

Conclusions

The fitness is low, suggesting that the sets of output variables found by evolution are not close to the target sets of variables. Comparing the sets of variables confirms this.

4.3.7 Discussion

Transformed CA was evolved in the elementary and trinary rule space. The elementary rule space was found to be very evolvable, but did not contain CA specific enough to produce good output variables when transformed. A larger rule space was used to include more CA: the trinary rule space. It was found that the trinary rule space was unevolvable. Two techniques were tested to try to compensate for the unevolvability: low size of CA and summation interpretation of states. While low size of CA seemed to improve evolvability, it was not enough to produce good output variables.

A final experiment was conducted to try to evolved transformed CA capable of producing multiple sets of target variables in the trinary rule space. The results were not promising. The conclusion is that the algorithm does not scale with the size of the rule space, and there is no easy way to fix this.

Chapter 5

Discussion

This chapter contains discussions regarding several aspects of the report. First, the results of the experiments will be discussed. Then the consequences of the limitations determined in the scope will be discussed. Finally, some ways to improve the results are discussed.

5.1 Results

The transformation process was tested in the elementary and trinary rule spaces. It was determined what effect certain settings had on the transformation process. Evolution of transformed CA was carried out by an evolution strategy (ES) in the elementary and trinary rule spaces. Evolution of transformed CA was successful in the elementary rule space and unsuccessful in the trinary rule space.

The successful evolution in the elementary rule space suggests there is a correlation between rules of CA and their transformed output. This suggests the evolvability using the ES algorithm doesn't scale with the size of the rule space[8]. Two techniques were tested to compensate for the unevolvability. Decreasing the size of CA improved evolvability in the trinary rule space, but the results were still not satisfactory.

5.2 Scope

Several steps of the transformation process and evolution of transformed CA were limited by the scope. In this section, the consequences of the limitations determined by the scope will be discussed.

Two limitations in the transformation process were determined by the scope: only one-dimensional CA would be investigated, and only discrete Fourier transform would be used to transform states.

5.2.1 Dimension of CA

In academia, the interest in CA is not limited to the first dimension. Two- and three-dimensional CA are also very popular. In fact, the most well-known CA rule of all is probably Conway's Game of Life[4], a two-dimensional CA which simulates life-like properties such as reproduction and overpopulation. While three-dimensional CA requires immense computing power to compute, two-dimensional CA don't require that much more computing power than one-dimensional CA. The behavior of two-dimensional CA is more complex than the behavior of one-dimensional CA. This suggests transforming two-dimensional CA could be a great way to increase the efficiency of creating and finding multivariable transformed CA.

5.2.2 Size of neighborhood

The size of the neighborhood was limited to three, in line with the principle of locality. A larger neighborhood means the information would be less local.

The limitation had the consequence that increasing the rule space was done by increasing the number of states per cell. Without this limitation, one could get a rule space larger than the elementary rule space and smaller than the trinary rule space. This could severely increase the evolvability of regular and transformed CA.

5.2.3 Discrete Fourier transformation

The translation of states into spikes we could be done using many different methods. Other functions that could be used include wavelet transform and various filters. While DFT does exactly what we want, it is hard to know whether another technique would be better. In future work, it could be a good idea to experiment with other transformation techniques.

5.2.4 Evolution strategy

In the scope it was determined that an evolutionary strategy (ES) will be used for evolution of transformed CA. While ES has been used successfully with CA in the past, evolutionary algorithms that simulate biological life to a greater extent is more common. They use large populations and advanced techniques for tasks such as selecting individuals to reproduce and calculate the genetics of offspring. The results of the experiments suggests it was not the best choice. A genetic algorithm could scale better with the size of the rule space and improve evolvability.

5.3 Future work

Referring to the discussion about the scope, there are some alternative approaches in the transformation process that could improve the result. Using two-dimensional CA, instead of one-dimensional CA, could create more solutions while still maintaining evolvability. Increasing the size of the neighborhood of CA could create a more evolvable rule space. Using a transformation technique other than Fourier transform could reduce the number of spikes which would make more solutions with few variables. Finding a way to positively evaluate transformed CA with more spikes than target variables could increase the number of solutions.

The scope limited some approaches to the evolution of transformed CA that could improve the results. Using a biology-inspired evolutionary algorithm to find transformed CA could increase evolvability. Programming the experiments in a fast programming language like C could greatly increase the efficiency of the algorithm. Running the experiments on one or more fast computers for a longer time, could improve the results of the evolution experiments.

The evolutionary algorithm itself is not the only thing that could be improved. First, the experiments were programmed in C#, a high level and slow programming language. Second, some well-known optimization techniques for working with CA exist that were not used in the experiments. Third, the experiments were conducted on a laptop for no more than 30 minutes per experiment. Assuming we can get better results by running the evolution experiments for more generations, the following could improve the results: first, the experiments could be programmed in a faster programming language like C. Second, optimization techniques for working with CA could be used. Third, the experiments could be conducted on one or more fast computers for a longer time. These points could greatly improve the results of the evolution experiments.

When evaluating the fitness of transformed CA, there are a couple of things that could be done differently. Currently a transformed CA is evaluated positively only if it has the correct number of spikes. Most transformed CA have a very large number of spikes. When looking for a small number of variables, most CA are disregarded because of too many spikes. A quick workaround that could be looked into is disregarding the spikes after the first N spikes, where N is the number of target variables. Another way could be to filter the spikes in a way that reduces spikes. A third way could be to only regard spikes in certain positions, disregarding the rest. A number of these approaches could greatly increase the number of possible solutions.

Bibliography

[1] Edgar F. Codd. *Cellular Automata.* Academic Press, Inc., June 1968.

[2] Philippe Collard, Marco Tomassini, and Leonardo Vanneschi. Neutral Fitness Landscape in the Cellular Automata Majority Problem. *arXiv.org*, cs.NE, March 2008.

[3] Stuart A. Kauffman. Emergent properties in random complex automata. *Physica D: Nonlinear Phenomena*, 10(1-2):145–156, January 1984.

[4] Wentian Li, Norman H Packard, and Chris G Langton. Transition phenomena in cellular automata rule space. *Physica D: Nonlinear Phenomena*, 45(1-3):77–94, September 1990.

[5] Melanie Mitchell, James P. Crutchfield, and Rajarshi Das. The Evolutionary Design of Collective Computation in Cellular Automata. 1998.

[6] Stefano Nichele. Trajectories and attractor basins as a behavioral description and evaluation criteria for artificial EvoDevo systems . pages 1–111, September 2009.

[7] Moshe Sipper. The Emergence of Cellular Computing , April 1999.

[8] G. Tufte. Phenotypic, developmental and computational resources: scaling in artificial development. In *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 859–866, New York, NY, USA, 2008. ACM.

[9] Stephen Wolfram. A New Kind of Science. *Applied Mechanics Reviews*, 56(2):B18, 2003.