NTNU – Trondheim
Norwegian University of
Science and Technology

Emergent Behaviour in the Frequency-Power Spectrum of Discrete Dynamic Networks

Ole Henrik Jahren

Master of Science in Informatics
Submission date: June 2012
Supervisor: Gunnar Tufte, IDI

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

Abstract

In the fields of cellular automata and complex systems, emergence is often used as an interpretation of system behaviour. Computation and the resulting output are both products of the systems trajectory in the basin of attraction, where the output data is a point or cyclic attractor. As such, the system outputs only a single variable, the states of all units in the system. This work diverges from the norm on two aspect. Instead of exploring Cellular Automata as the computational architecture, Boolean Networks, a specialisation of the more generalised Discrete Dynamic Networks, will be used. Secondly, a different approach in interpreting the behaviour is taken. Instead of looking directly at the state of the system, the trajectory in the basin of attraction is instead transformed to a frequency-power spectrum representing the system output. This allows an easy interpretation of the output (peeks) to several output variables, were each variables can be given as the power at different frequencies in the frequency-power spectrum.

Because of the difficulty in programming, i.e. designing, Discrete Dynamic Networks with the desired characteristics, a Genetic Algorithm will be used to evolve the networks. This thesis takes an experimental approach, evolving Discrete Dynamic Networks capable of producing different number of peaks in the frequency-power spectrum. The results show that Discrete Dynamic Network exhibiting the desired emergent behaviour were successfully evolved.

Acknowledgements

I would like to extent huge thanks to my supervisor, Associate Professor Gunnar Tufte, for his valuable expertise and guidance, and my brother, Jon Emil Jahren, for letting me bounce ideas off him.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Emergent behaviour from a complex system perspective, is when a system exhibits behaviour that is more complex than the sum of its parts. This is typically done by using relatively simple building blocks, e.g. cells in the context of Cellular Automata[1] (CAs) or nodes in the context of Discrete Dynamic Networks[2] (DDNs), with very simple and deterministic behaviour. It is the interaction between these simple components that result in the emergent behaviour. A lot of work has been done on behaviour in the space-time spectrum, e.g. on CAs[3][4].

This thesis will look at emergent behaviour in a different spectrum than most have done before. Instead of looking at the space-time spectrum, the frequency-power spectrum will be investigated. An experimental approach in the form of three experiments, each designed to investigate the possibility of finding a solution that exhibits a predefined behaviour in the frequency-power spectrum.

The motivation for using the frequency-power spectrum instead of the arguably more traditional space-time spectrum, is brought on by the desire to interpret the emergent behaviour of the system as a function with multiple outputs. In contrast with what some have done[5], interpreting the state of the entire system as a single output.

The primitive building block used will be a boolean node. Multiple boolean nodes will be connected together to form a special case of a DDN, a Boolean Network[6] (BN). A BN is a little different from a CA, since it is capable of all-to-all connections.

Even though certain CA configurations have been proven to be Turing complete[7],

no easy way of programming them[8], in the traditional sense, has been published. To aid in the programming of the BNs, a Genetic Algorithm (GA) will be used in the search of emergent behaviour.

The GA will be used as an evolution guided search in solution space. The main metric used in the fitness function will be the correct number of peaks, and the power of those peaks compared to the noise in the frequency-power spectrum. To transform the time-space output of a network with a given initial configuration, Discrete Fourier Transform (DFT) is used.

It is intended as a proof of principle, i.e. the goal is to show that it is possible to evolve a network that exhibits the desired number of peaks in the frequency-power spectrum. To avoid having the fitness landscape have lots of jumps from zero fitness, when the correct amount of peaks are not present, to a potentially large value, when the correct amount of peaks are present, a fitness function that attempts to smooth the jumps was devised. The fitness function uses the logarithm of the fraction of the states that are repeated, when not enough peaks in the frequency-power spectrum are present, and takes the difference between the lowest peak and the noise, when enough peaks are present.

To explore the emergent behaviour of the evolved BNs, three experiments are designed. The goal of experiment 1, is to check that the experimental setup is capable to finding the correct number of peaks at all. Experiment 1 also aids in gauging the limitations of the implementation.

The goal of experiment 2 is find out how robust networks the experimental setup is capable of evolving, i.e. it is possible to evolve networks that are indifferent to their initial state, and still produce the correct number of frequency peaks. This is interesting because if it possible to evolve BNs that produce the same number of frequency peaks for all possible initial configuration, the network is capable of generating the same peaks no matter where it starts in the "life cycle".

Experiment 3 is designed to find a different number of frequency peaks for different initial values. E.g. for one initial value, it will produce two peaks in the frequency-power spectrum, for another initial value, it will produce one peak. This is interesting because, the output of the system can be interpreted as producing a different number output values as a function of the initial state.

All the experiments will be run in a pure software simulated environments.

This thesis is structured as follows; Chapter 2 contains background informa-

tion and references to earlier work built upon, Chapter 3 describes how the experiments are set up and how they are performed, Chapter 4 presents the result of the experiments, and Chapter 5 concludes the work and suggests a few ideas for further work in this area.

Chapter 2

Background

This work is based primarily on earlier work investigating emergent behaviour in CAs[4][3]. Although in the experiments, BNs will be used instead of CAs, the two are related through a more generalised network type known as DDNs or Sequential Dynamic Systems (SDSs). In this chapter, DDNs will be explained first, followed by CAs and BNs using DDNs as a basis.

GAs will be described to the extent used in the experiment, e.g. not every selection mechanism that exists will be explained, as there are far too many of them.

A brief description of how transformations can be used to look at state spaces in a different way will also be presented. This assumes familiarity of transforms in general and DFT in particular.

2.1 Discrete Dynamic Networks (DDN)

A DDN is a network of n nodes. Each nodes can have a state $S \in \{s_1, s_2, \cdots, s_m\}$ of m possible states. The directed connections between nodes can be connected to any other nodes in the network, including itself.

Each node has a next state function that maps the input connections, i.e. the state of the connected nodes, to a next state for the node. A next state is the state the node gets at the next step in time. The state of the nodes are updated according to an updating scheme as described in Section 2.1.1.

The simulation of a DDN can for example involve the following steps:

1. Assign initial state.
2. Update next state.
3. Next step in time.
4. State is updated from next state.
5. Goto step 2.

Figure 2.1 is an example of a DDN where each node is labeled *n0*, *n1*, et cetera. Each nodes has a next state function denoted by f_0, f_1, et cetera, that takes the state of all input nodes to the node as input, and assigns the result to the next state of the node.

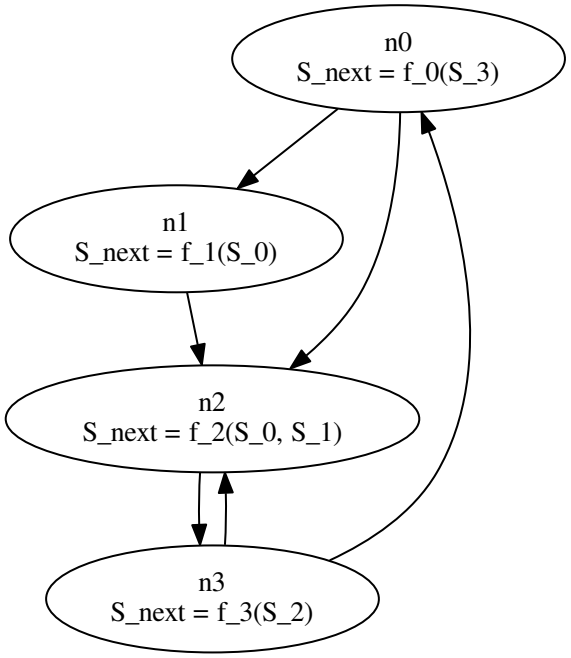

Figure 2.1: Example of DDN

2.1.1 Updating Schemes

The updating scheme on a given DDN, gives the rules for when the next state function of a node is called for updating the nodes output. The up-

dating scheme of a network is often classified according to two attributes, synchronous/asynchronous and deterministic/non-deterministic.

If the updating scheme is synchronous, all the nodes are updated at the same step in time. In an asynchronous updating scheme, only some nodes may be updated at a given step in time.

In a deterministic updating scheme, the network will always produce the same global state of the network at the same step in time. In a non-deterministic updating scheme, the global state may be different at the same step in time for consecutive simulations of the network.

Although there are many variation of the updating schemes, a non-exhaustive list of the most often encountered[9] schemes and a description of each.

- Classic i.e. synchronously updated.
- Asynchronous.
- Deterministic asynchronous (and variations, edges could e.g. have a propagation delay/length).
- Generalized asynchronous (non-deterministic).

Node state updating can be done using various state update schemes. The simplest, sometimes referred to as classical, updating scheme is synchronous. In a synchronous updating scheme, each node is updated using the nodes next state function at each time step. The update is not in-place so the update of a node is not visible until all nodes are updated. A synchronous updating scheme, is inherently deterministic since two runs using the same initial states and network configurations will always produce the same sequence of states.

In an asynchronous updating scheme, at each step in time, a set of nodes are stochastically selected and updated. This is a non-deterministic updating scheme since two simulations with same initial state and network configuration may produce different sequence of state.

Two other closely related updating schemes are; deterministic asynchronous and deterministic generalised asynchronous updating scheme[9]. In both of these, each node has parameters associated that determines how many steps in time are between updates of a given node. In the non-generalised scheme, the updates are actually asynchronous so you have to predefine which order the nodes are updated to not lose determinism. In the generalised form, the node updates synchronously so the determinism is inherent.

In a generalised asynchronous updated scheme, a set of nodes are randomly selected each time step, and updated synchronously. Because of the randomness, this update scheme is non-deterministic.

2.2 Boolean Networks (BNs)

A BN is a DDN, where each node only has one of $S \in \{0, 1\}$ two possible states. The first BNs were proposed by Kauffman[6], as genetic regulatory networks. The model used by Kauffman is often referred to as the NK-model. In the NK-model, you have N nodes in a network, and each node has K inputs. Note that the networks Kauffman used were Random Boolean Networks (RBNs). The networks used in this thesis are Boolean Networks, that does not follow the NK-model because the number of connections between nodes are evolved, not randomly assigned, and are not restricted to a fixed amount.

The main difference between BN and DDN is the number of states a node can have. While in a DDN a node can hold as many nodes as desired, a BN true to its name, can only hold two states $S \in \{0, 1\}$.

Figure 2.2 illustrates a simple BN where the nodes are labeled *n0* and *n1*. The next state function is very simple, the next state of each node, is the current state of the other node. With an network initial state of e.g. 01_2, the network will oscillate between 01_2 and 10_2.

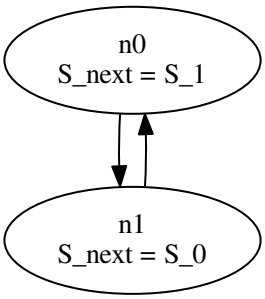

Figure 2.2: Example of BN

2.3 Cellular Automata (CAs)

Although CAs are not used here, a lot of earlier work looking at emergent behaviour have used CAs[10]. Hence it is prudent to give a brief background.

CAs are another special case of DDNs. A few minor differences exists; e.g. in a CA a node is called a cell. While CAs can have a different next state function for each cell (non-uniform CAs), it is often the case that all the cells have the same next state function[3][4] (uniform CAs).

The main difference between CAs and DDNs is that the cells of a CA has the concept of placement in space. E.g. for a one dimensional CA as shown in Figure 2.3, each cell has a position in space, relative to the other cells. The placement in space is important, because in a CA the cells are connected through the neighbourhood. The neighbourhood is defined by a neighbourhood template[4], and defines which cells are used as input to the next state function. E.g. for the one dimensional CA, a typical template size is 3, as shown in Figure 2.4. A cell is part of its own neighbourhood.

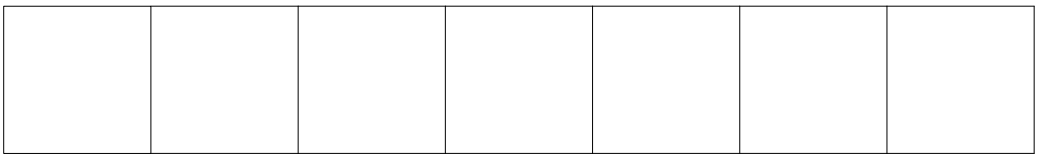

Figure 2.3: 1d CA

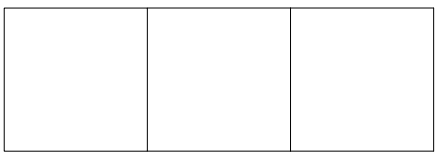

Figure 2.4: 1d CA - Neighbourhood of the middle cell

A lot of previous work with CAs is aimed at their emergent behaviour. Often referenced is the work done by Stephen Wolframs on classifying the behaviour of CAs[3]. Wolfram mapped out the entire behaviour in space-time of all the one dimensional CAs with the 3-cell neighbourhood template shown in Figure 2.4. The different behaviours were then classified into the four different classes seen Table 2.1.

Class I	Eventually transitions into homogeneous state.
Class II	Eventually enters periodically repeating states or transitions into homogeneous state.
Class III	Chaotic behaviour i.e. many time steps before repeating states.
Class IV	Seemingly chaotic behaviour, but localised order/structure.

Table 2.1: Wolframs classifications of CAs

2.4 Genetic Algorithms (GAs)

A GA[11] is a form of Evolutionary Algorithm[12] (EA). An EA is a guided search in a solution space, that attempts to mimic our understanding of biological evolution (survival of the fittest).

In a GA you have a genotype, which is the genetic representation of an individual. The genotype is developed to a phenotype, which is the physical representation of the individual. A fitness function evaluates the individual to determine how well the genotype is suited as a solution.

A population is made up of many individuals that competes against each other for reproduction. Which individuals that are allowed to reproduce is determined by parent selection. During reproduction, various genetic operation can affect the genotype of the offspring, e.g. crossover, mutation, etc. Adult selection is used to determine which children grow up and can become parents during next reproductive cycle and which dies off. Sometimes, a way for the n best individuals in each generation to bypass selection altogether is also added. These are called *elites*[13], and go straight from adults in generation g to adults in generation $g + 1$, bypassing all the different selection types.

A GA can for example follow this series of steps[14]:

1. Generate initial children.
2. Map the genotype to a phenotype.
3. Evaluate phenotype fitness.
4. Perform adult selection for new generation.
5. If stop condition is met (e.g. max generations has been reached, or fitness has reach a desired threshold), stop.
6. Perform parent selection.

7. Perform mate selection.
8. Produce offspring.
9. goto step 2.

Many of these steps include a large number of sub steps, e.g. the process of producing offspring can include many genetic operators, like mutation and crossover. But the steps above, outlines what a GA can do at a high level.

2.4.1 Selection

A selection strategy is made up of two components, the selection protocol and the selection mechanism[15]. The selection protocol determines which individuals get to participate in the selection mechanism. The selection mechanism determine which individuals get selected for whatever task it selects for.

Various schemes exists for the different protocols and mechanisms. Since this is not intended as a exhaustive reference on the subject, only a few samples will be presented of each.

Adult Selection

The protocol determines which individuals gets to participate in the selection of the next adult population. Common protocols include:

- Full generational replacement
- Over population
- Generational mix

Full generational replacement kills all adults, and all children become adults. Hence there is no need for a selection mechanism.

Over population is when more children than fit in an adult generation is produced. Like in full generational replacement, all adults are killed. Since there are more children than can live on to adulthood, a selection mechanism is used to decide which becomes adults and which dies.

Generational mix means that both adults and children are thrown into the same pool, and a selection mechanism is used to decide which individuals lives/become adults and which dies.

To do the actual selection, if required, the following selection mechanism are commonly used:

- Best fitness
- Stochastic

Best fitness is simply picking the individuals with highest fitness to advance into adulthood.

Stochastic mechanism is simply picking individuals that advance into adulthood according to some stochastic measure. This can also involve the fitness of the individuals.

Parent Selection

The protocol determines which individual gets to be participate in the mating. A common protocol is also the most obvious; all individuals participate in the mating.

The mechanism selects which individuals become parents. Common mechanisms include:

- Fitness proportionate
- Sigma scaling
- Tournament
- Boltzman scaling

Fitness proportionate scales the fitness in the interval $[0, 1]$ based on the total sum of the fitness for the current generation. See equation 2.1. Treating $P(i, g)$ as the probability of individual i being selected.

$$P(i, g) = \frac{f(i)}{f(g)} \tag{2.1}$$

Where $P(i, g)$ is the proportion of the interval awarded to individual i in generation g, $f(i)$ is the fitness of the individual i, and $f(g)$ is the total fitness of generation g.

Sigma scaling uses a scaling factor based on the current generation's average and variance fitness, in an attempt to loosen the selection pressure inherent in the raw fitness value used in fitness proportional. The portion each individual get of the interval $[0, 1]$ is given in equation 2.2. Also here $P(i, g)$ is treated as the probability of individual i being selected.

$$\begin{aligned} ExpVal(i,g) &= \begin{cases} 1 + \dfrac{f(i) - \bar{f}(g)}{2\sigma(g)} & \sigma(g) \neq 0 \\ 1.0 & \sigma(g) = 0 \end{cases} \\ P(i,g) &= \frac{ExpVal(i,g)}{\sum_{n=1}^{i_{\max}} ExpVal(n,g)} \end{aligned} \tag{2.2}$$

Where $P(i, g)$ is the proportion of the interval awarded to individual i generation g, $f(i)$ is the fitness of the individual i, $i_{\max}$ is the number of individuals in a generation, $\bar{f}(g)$ is the average fitness of generation g, $\sigma(g)$ is the standard deviation of the fitness of generation g.

Boltzmann scaling is based on the physical principle that under higher temperature, a system exhibits more randomness than under low temperature. This is similar to how simulated annealing works. The equation used for Boltzmann scaling can be seen in 2.3. Again $P(i, g)$ is treated as the probability of individual i being selected.

$$\begin{aligned} f_{\exp}(i) &= e^{\frac{f(i)}{T(g)}} \\ ExpVal(i,g) &= \frac{f_{\exp}(i)}{\bar{f_{\exp}}(g)} \\ P(i,g) &= \frac{ExpVal(i,g)}{\sum_{n=1}^{i_{\max}} ExpVal(n,g)} \end{aligned} \tag{2.3}$$

Where $P(i, g)$ is the proportion of the interval awarded to individual i generation g, $f(i)$ is the fitness of the individual i, $i_{\max}$ is the number of individuals in a generation, $T(g)$ is the temperature in generation g, $\bar{f_{\exp}}(g)$ is the average fitness exponential in generation g.

Tournament selection stochastically, usually uniformly, picks k individuals out of the current generation. The one amongst the k picked out with the highest fitness is the winner of the tournament and is allowed to reproduce.

Mate Selection

The protocol determines which individuals can mate with each other. A common protocol is that any selected parent can mate with any other selected parent.

The mechanism selects who gets to which individual gets to mate with each other. One example mechanism is is stochastic uniform, in which all selected parent are randomly paired with each other, to produce monogamous pairs.

2.4.2 Genetic Operators

Genetic operators are usually applied during reproduction. The main purpose of the operators is to create greater variation in the genotypes of the children to explore more of the problem space.

While several operators exists, only a few will be explained here.

- Mutation
- Crossover

Mutation is when a small change is introduced into the genotype. The classic example is flipping a single bit in the genotype. Other variations exists, like adding or subtracting one, if the part of the genotype represent a number. The actual mechanism of the mutation is a minor detail, but the reason mutation is used is important. Mutation is there to create a *small* change in the genotype, to introduce diversity into the population. The reason diversity is important, is so the population does not get stuck in a local maxima.

Crossover is when whole chunks are exchanged between the two genotypes. This is a much larger change in the genotype than a small mutation, and can therefore potentially be more damage/rewarding. To keep crossover from damaging good solutions, the genotype is often divided into what is sometimes referred to as *features*. A feature is a set of bits in the genotype that logically belongs together. E.g. if a number is encoded in the genotype, that number could be regarded as a feature, so that crossover will never split the bits the number is made up of. It would always crossover at the boundary of the number.

Crossover can be done with one or more points of crossover, e.g. every second chunk of one genotype is exchanged with every second chunk of another

genotype.

2.5 Transformations

FIXME: fix state, distinction between node state and network state.

When looking at at the state space of a system, it can be interesting to represent the behaviour as a function of the state space. This can be done using various transforms, e.g. Fourier Transform, Wavelet Transform, etc, that translates the state space of state-time spectrum into another spectrum. E.g. using DFT, a state space can be translated from a time-amplitude spectrum into frequency-power spectrum for spectral analysis. For an illustration of looking at a transformed state space, imagine a BN consisting of two nodes, as illustrated in Figure 2.5. The next state function of both nodes are; node becomes 1 if the other node is 1. With an initial state of 01_2, the network will oscillate between the states 01_2 and 10_2 forever.

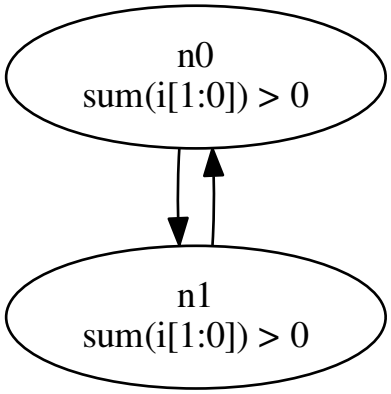

Figure 2.5: Transformation example - Network

If the state of the network is interpreted as a binary unsigned value, the value 01_2 become 1_{10} and the value 10_2 become 2_{10}. The values are then scaled linearly between -1_{10} and 1_{10}, in order to reduce the DC component of the DFT. I.e. 01_2 becomes $\frac{1-\frac{2^2-1}{2}}{\frac{2^2-1}{2}} = -\frac{1}{3}$ and 10_2 becomes $\frac{3-\frac{2^2-1}{2}}{\frac{2^2-1}{2}} = \frac{1}{3}$. Figure

2.6 plots the scaled amplitude of the time-state spectrum, as a function of time.

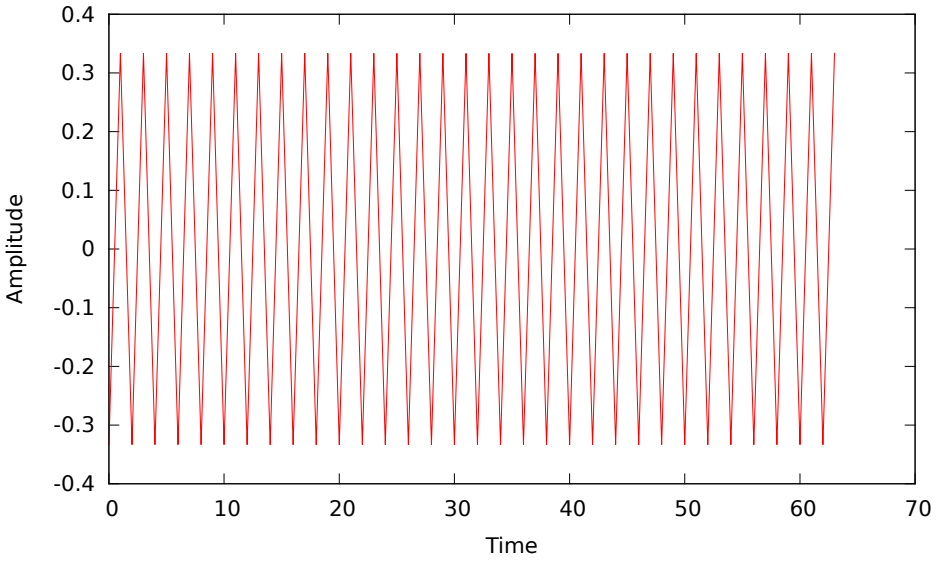

Figure 2.6: Transformation example - Time-amplitude plot

The DFT is taken over the time-amplitude spectrum, the result is a frequency-power spectrum representation of the state space, as shown in Figure 2.7. The DFT in Figure 2.7 is taken over 64 states, with a window size of 32. All the power ends up in the frequency bin for the frequency that is exactly half the size of the window used.

FIXME: can state space be used in this way? Is the meaning, given the context, clear?

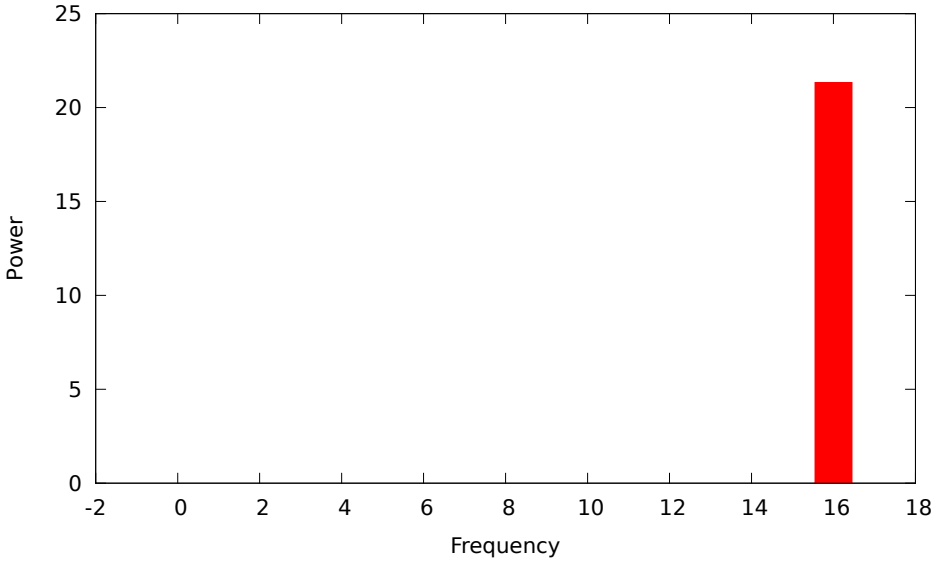

Figure 2.7: Transformation example - Frequency-power plot

Chapter 3

Experiment Setup

The purpose of these experiments is to explore the emergent behaviour in the frequency-power spectrum of BNs. It is intended as a proof of principle, and as such the various aspects of the GA are not analysed exhaustively nor optimised. The only tweaking done to the various parameters and algorithms is to get them working, nothing more.

To further the goal of exploring the emergent behaviour of BNs, a set of metrics have been devised. The main metric used is the number of peaks. This refers to the number of peaks seen in the frequency-power spectrum, of a DFT, e.g. in Figure 2.7, there is one peak. The peak count does not include the DC-component.

Another metric used is the coverage set. For the purpose of this thesis, the coverage set of a BN with a certain initial state, is the set of repeating states. E.g. if a two node network yields the sequence of states as follows; 01, 00, 11, 00, 11, ... With 01 being the initial state, the coverage set would be the set {00, 11}. Coverage is also sometimes used as a number. In this context, the coverage is the fraction of all possible states that is contained in the coverage set. E.g. for the previous example coverage set of {00, 11} the coverage fraction is $\frac{2}{4} = \frac{1}{2}$.

To explore some of the aspects of the emergent behaviour, a series of experiments are performed. Experiment 1, is to see if the algorithm is capable of evolving different frequency peeks at all, and to gauge some of the limitations of the implementation.

Experiment 2 is to see if evolved networks can be made a certain a certain level of robustness[16] with respect to the initial state, i.e. that the network

produce the same number frequency peaks regardless of the initial state. Note that the robustness used here, does not care about *where* in the frequency band the peaks are, only that the correct amount of peaks are present.

Experiment 3 attempts to generate different frequency-power spectrum, depending on the initial state. E.g. can one initial state generate two frequency peaks, while another initial state generates three frequency peaks, for the same network.

3.1 Discrete Fourier Transform

In the experiments each state of the network represents a sample. The objective is to find a network that produces samples with as high power of one or more frequencies as possible, compared to the other frequencies present.

Similar to the explanation in Section 2.5, each network state is treated as a binary unsigned number. The binary number of scaled linearly on $[-1, 1]$. The scaled numbers are treated as samples sampled uniformly in time, as shown in Figure 2.6. The DFT of the samples is calculated. Followed by calculation of the absolute value of the complex result, to get the power, as shown in 2.7.

The frequencies that can be recreated from the samples are theoretically, limited by both the amount of nodes in the network, and the frequency the samples are sampled at.

In order to avoid confusion, a few things about the nomenclature used needs to be explained. f_s means sample frequency, and is the frequency the samples of the network are sampled at. Since the system does not really have a concept of time, f_s can be changed at will, it is just a definition of what the sampling frequency is defined to be. f_w is the frequency of the window used when calculating the DFT, i.e.

$$\begin{aligned} f_w &= \frac{1}{\frac{1}{f_s}w} \\ &= \frac{f_s}{w} \end{aligned}$$

where w is the number of samples in a window.

Because of the sampling theorem[17][18], the maximum frequency $f_{\max}$ that

can be reconstructed from the samples, are

$$f_{\max} = \frac{f_s}{2} \tag{3.1}$$

where f_s is the frequency the samples are sampled at.

The minimum frequency $f_{\min}$ that can be reconstructed from the samples, is

$$f_{\min} = \frac{f_s}{2^n} \tag{3.2}$$

where f_s is the frequency the samples are sampled at, and n is the number of nodes in the network. The reason for this is that, in a network of n nodes, there is only 2^n possible states. It follows that the longest period that is possible to have, is one that contains all possible states of the network.

Fitness is evaluated by using DFT on the samples generated by the network. Due to the computational complexity of DFT, a window is used to avoid having to look at all the samples at once. Use of a window limits what frequencies that can be detected. E.g. if the window is w samples, and the sampling frequency is defined as f_s

$$\begin{aligned} f_s &= \frac{1}{t_s} \\ t_s &= \frac{1}{f_s} \\ t_w &= t_s w \\ \hat{f}_{\min} &= \frac{1}{2t_w} \\ &= \frac{1}{2\frac{1}{f_s}w} \\ &= \frac{f_s}{2w} \\ &= \frac{f_w}{2} \end{aligned} \tag{3.3}$$

where t_s is the period of a sample, t_w is the period of a window, and $\hat{f}_{\min}$ is half the window frequency. The frequency in Equation 3.3 is the lowest frequency that can be detected using this scheme.

Another limitation that occurs because of windowed DFT, is how large a resolution the classification gets. Using the symbols from earlier.

$$f_{\text{res}} = \frac{f_{\text{s}}}{w} \tag{3.4}$$

is the resolution obtained on the result. If the resolution is too low, a too wide range of frequencies can hide in the same block in the result of the DFT. Which can make the classification impossible.

To avoid spectral leakage into other frequencies when applying windowed DFT, a windowing function other than the rectangular can be used. E.g. Hann and Hamming. In these experiments, a rectangular function is used.

3.2 The Boolean Network

In these experiments, a synchronously updated BN with a fixed number of nodes is used. The connections between nodes are evolved using a GA. To keep it simple, the next state function (see Equation 3.5) is parametrised. I.e. the parameter can be evolved, but the rest of the function is fixed.

$$S_{t+1}(n) = \left(\sum_{i=1}^{Neighbour_{\max}(n)} S_t(Neighbour(n,i)) \right) \geq param \tag{3.5}$$

Where $S_t(n)$ denotes state of node n at time step t, $Neighbour_{\max}(n)$ denotes the maximal number of neighbours of node n, $Neighbour(n,i)$ denotes the ith neighbour of node n, and $param$ which denotes the parameter encoded in the genotype.

The reason for having a parametrised next state function is to avoid the possibility of having wildly different phenotypes for genotypes only separated by a single mutation.

3.3 The Genetic Algorithm

This section details the specifics of the GA used in these experiments. To keep things consistent, this section will try to use and build upon the terminology

used in Section 2.4.

Most of the parameters to the GA remains the same for all the experiments, a description of them is shown in Table 3.1. The value of most of the parameters are also the same for all experiments. Table 3.2 shows their default value. If the value of a parameter is changed for an experiment, it will be noted in the description of that experiment. Additional experiment specific parameters are described in the experiment where they are used.

3.3.1 Genotype

The genotype is a simple direct encoding, a bit string that is an array of bit arrays. Each element in the array encodes the configuration of a node. And the bit array, encoding each node configuration, is laid out as in Table 3.3.

Bits	Description
$0 \ldots \lceil log_2(n) - 1 \rceil$	*param* for next state function
$\lceil log_2(n) \rceil \ldots \lceil log_2(2) + n \rceil$	bit vector denoting whether connection exists

Table 3.3: Encoding of a node configuration in a bit array. n is number of nodes.

In the genotype, the bit string is prefixed with the number of nodes in the network, this is simply to avoid having to solve the equation $b = n(\lceil log_2(n) \rceil + n)$, where b is the length of the bit-string, and n is the number of nodes in the network, to get the number of nodes in the network. The genotype phenotype mapping is easiest shown by example, e.g. the genotype 2_{10} 001010_2 specifies a network with two nodes, as denoted by 2_{10}. The 001010_2 contains two bit arrays, one for each node, where each array is partitioned as described in Table 3.3.

Picking the bit string apart, as shown in Figure 3.1, the first bit array 001_2 describes node 0. The first bit 0_2 encodes that the parameter is 0, and two following bits 01_2 encodes no input to node 0 from node 0, and an input to node 0 from node 1.

The second bit array 010_2 describes node 1. The first bit 0 encodes that the parameter is 0 here too, and the two following bits 10_2 encodes an input to node 1 from node 0, and no input to node 1 from node 1. The complete phenotype of the network is shown in Figure 3.2.

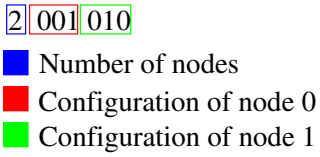

Figure 3.1: Genotype phenotype mapping example - Genotype

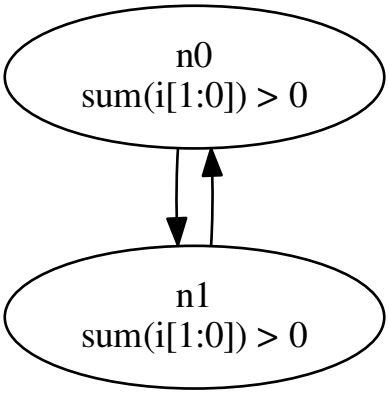

Figure 3.2: Genotype phenotype mapping example - Phenotype

3.3.2 Fitness

Fitness in this experiment involve several phases. The different phases will be explained in detail below. All emphasised words are explained in Table 3.1.

1. Generate *sim_statecount* states.
2. Calculate DFT using *dft_winsize* as the size of a window and *dft_winfunc* as windowing function.
3. Calculate the power in the real dimension from the DFT result.
4. Find local minimas and maximas in frequency-power spectrum.
5. Sort the local minimas and maximas in descending order.

6. (a) If enought minimas/maxinmas found. The fitness is the lowest of the number of peaks wanted after subtracting the item below it in the result ("noise").

   (b) If not enough minimas/maximas found. The fitness is calculated as $(log(coverage) - log(\frac{1}{2^{\text{nodes}}}))100$.

Phase 1 is simply going through the network and updating all the nodes synchronously. At each time step convert the state of the network into a number, and place it in the sample array. To give the opportunity to the algorithm to have as low as possible DC component in the signal, the number is normalised on the interval $[-1.0, 1.0]$.

In Phase 2 DFT is calculated applying *dft_winsize* as window size and *dft_winfunc* as windowing function. The result is placed in the DFT result array.

Phase 3 takes the complex result in the DFT result array and calculates absolute value to get the power. The power is saved in the power result array.

Phase 4 walks the power result array picking out all the local maxima and minima. The local maxima are placed in the sorted power result max array, and the local minima is placed in the sorted power result min array.

In Phase 5 the sorted power result arrays are sorted in descending order. This makes it easy to pick out the peaks of interest.

Phase 6a fitness is calculated. If p is the number of peaks sought, the highest p peaks from the sorted power result max array and subtracting the minima/maxima lower in the sorted power result arrays.

Phase 6b are for those networks that do not create enough peaks. In order to reward those individuals that may be closer than others in our search space the concept of coverage is used. The coverage of a simulation is here defined as the number of repeating states a given network reaches from a given initial state. To calculate the fitness, the fraction of the number of repeating states that are encountered of how large the state space is. Since this number easily falls into the noise range of the other fitness functions, it is calculated as $(log(coverage) - log(\frac{1}{2^{\text{nodes}}}))100$. This means the fitness gets a larger scaling for increasing coverage in the lower range, but the boost gets less the higher coverage that is attained, avoiding network with large fitness simply because they visited many states.

Parameter	Description
adults	The number of adults in the population after adult selection is done.
adult_sel_mech	What mechanism to use when performing adult selection. Possibilities: best fitness, and stochastic.
adult_sel_proto	What protocol to use when performing adult selection. Possibilities: full generation replacement, and generational mixing.
crossover_prob	The probability of crossover happening during child production.
crossover_points	How many crossover points to use when doing crossover
children	The number of children to produce.
elites	The number of elites to retain in the adult population when culling.
generations	The number of generations to run the GA for.
mutation_prob	The probability of a single gene mutating when producing children.
nodes	The number of nodes in the network.
dft_winsize	The window size of the DFT.
dft_winfunc	The windowing function used when doing DFT. Possibilities: hann, hamming, rectangular.
sim_statecount	How many states to simulate network for when doing fitness.
parent_sel_mech	Selection mechanism for parent selection. Possibilities: fitness proportionate, sigma scaling proportionate, and tournament.

Table 3.1: Parameters for the GA

Parameter	Value
adults	64
adult_sel_mech	Stochastic
adult_sel_proto	full generational replacement
crossover_prob	0.03
crossover_points	1
children	64
elites	1
generations	1024 / 10240
mutation_prob	0.18
nodes	8
dft_winsize	512
dft_winfunc	$f(x) = 1.0$ i.e. rectangular.
sim_statecount	2048
parent_sel_mech	sigma scaling proportionate

Table 3.2: Parameters for all experiments

Chapter 4

Experiments

The experiments performed are each designed to give an answer to a simple hypothesis. The first experiment, experiment 1 Section 4.1, is designed to test the hypothesis that by transforming the time-state output of a BN, it is possible to evolve networks that produces multiple output values from their emergent behaviour.

Experiment 2, which results can be viewed in Section 4.2, explores the state space by attempting to make a given network and initial state pair as robust as possible. This is an attempt to get some indications as to what the state space looks like.

In the third and last experiment, with results in Section 4.3, is designed to test the hypothesis that is it possible to evolve a network that is capable of exhibiting behavioural properties that, when transformed, gives a system with multiple output variables.

Section 4.4 is an overview of the results of all three experiments, with a short summary.

4.1 Experiment 1

The purpose of experiment 1, is to exercise the implementation, and see if the algorithm is capable of evolving the specified number of frequency peaks at all.

In this experiment, all simulations are run with the same initial value. The initial value used is the bit-string 00111100_2. Table 4.1 shows the results

from experiment 1. The *peaks* column gives how many peaks that are sought after with each initial state, the *run* column which instance of the experiment the row shows results for, and *generations* for how many generations each instance ran. The *success* column gives how many of the frequency peaks and initial state pairs found the sought after result.

Table 4.1 shows that only a single initial state were used per instance in experiment 1. Finding one, two, and three frequency peaks was fairly successful, four frequency peaks posed more of a challenge. It seems like the GA got stuck in a local maxima, unable to mutate/crossover out of it.

The highest number of peaks found in experiment 1, is three. Figure 4.1 shows the time-frequency plot of a typical high fitness individual. The three peaks are clearly visible in frequency "buckets" 85, 171 and 256, surrounded by minimal noise. The 64 first time steps of the samples the DFT is taken over, is shown in Figure 4.2. The reason for the relatively high DC-component is clearly visible in Figure 4.2; some of the frequencies have their entire period above 0, i.e. the samples within the period of the part of the signal that has the frequency, are all above 0.

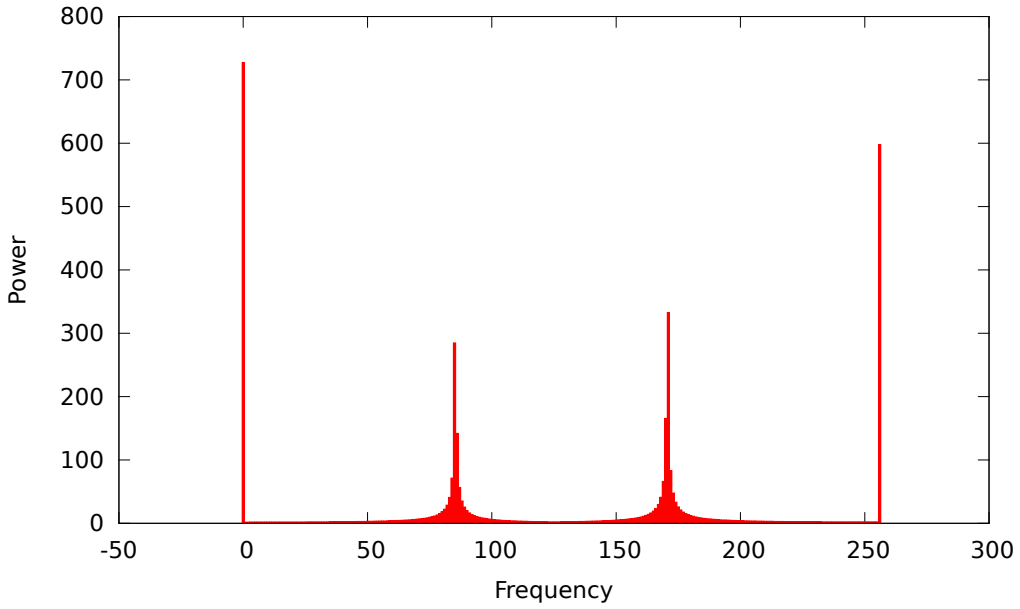

Figure 4.1: Experiment 1 - 3 peaks - run 7 - Frequency-power plot

Figure 4.3 shows the network, with labeling of nodes and next state functions. Node *n0* corresponds to bit zero, *n1* to bit one, et cetera, when the network state is interpreted as a binary unsigned number. Notice *n2*, which has the

Peaks	Run	Generations	Success
1	0	128	1/1
	1	128	1/1
	2	128	1/1
	3	128	1/1
	4	128	1/1
	5	128	1/1
	6	128	1/1
	7	128	1/1
2	0	1280	1/1
	1	1280	1/1
	2	1280	1/1
	3	1280	1/1
	4	1280	1/1
	5	1280	1/1
	6	1280	1/1
	7	1280	1/1
3	0	12800	1/1
	1	12800	1/1
	2	12800	1/1
	3	12800	0/1
	4	12800	1/1
	5	12800	0/1
	6	12800	1/1
	7	12800	1/1
4	0	128000	0/1
	1	128000	0/1
	2	128000	0/1
	3	128000	0/1
	4	128000	0/1
	5	128000	0/1
	6	128000	0/1
	7	128000	0/1

Table 4.1: Experiment 1 results

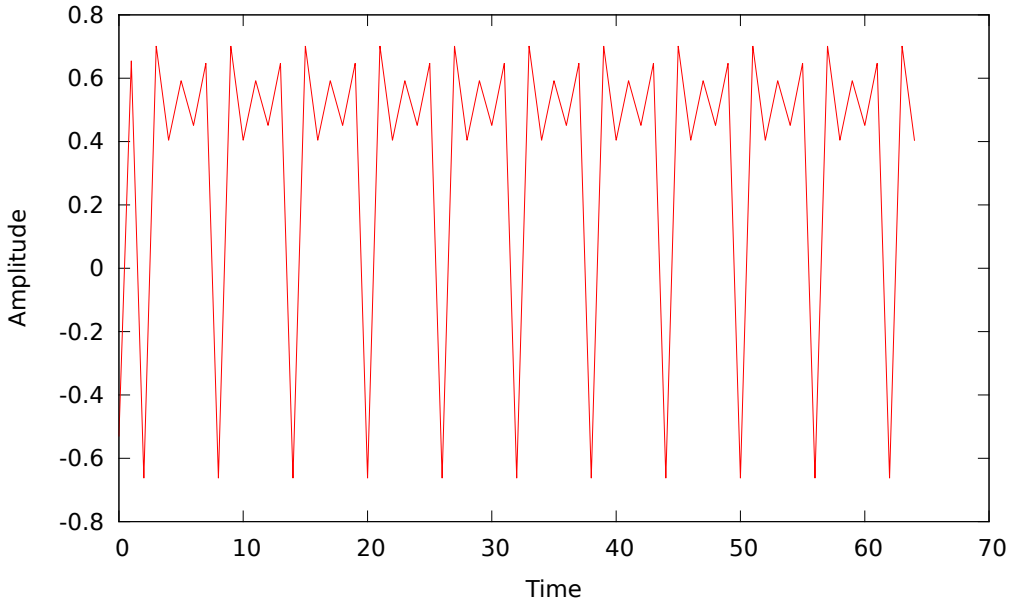

Figure 4.2: Experiment 1 - 3 peaks - run 7 - Time-amplitude plot

next state function $S_{\text{next}} = \sum_{j=0}^{7} input_j > 7$. This means that the next state function of *n2* will always evaluate to 0, since it only has 5 inputs. The only two possible outcomes from this is; either *n2* has 1 as initial node state and switches to 0 after first time step, or initial node state 0 and the node state stays 0 forever. The former case can be observed in Figure 4.4.

A trend seems to be that the fewer frequency peaks that are sought, the more "dead nodes" like *n2* in Figure 4.3 are observed. This could be because the choice of next state function restricts the state space, so the the amount of nodes needed could be proportional to the number of peaks sought.

Figure 4.5 plots the number of times each network state is visited. This is done by simply interpreting the network state as an unsigned boolean number, and counting how many times each number occurs. It follows that if you accumulate the count of how many times each state is visited, the number will be equal to the number of how steps in time that were done during simulation, including the time step of the initial state. The six network states contained in the coverage set stands out as large peaks, with the two states visited prior to oscillation, see Figure 4.3, barely showing as two small bumps.

The fitness of the instance can be seen in Figure 4.6. Red solid line, is the best fitness for that generation, the green crosses with error bars is the

average fitness of that generation with standard deviation. As seen in Figure 4.6 a solution was found before 2000 generations was reached. This is not typical for the three peak solution. Most of the three peak solution seems to be found around 8000 generations, where an even better solution was found in Figure 4.6.

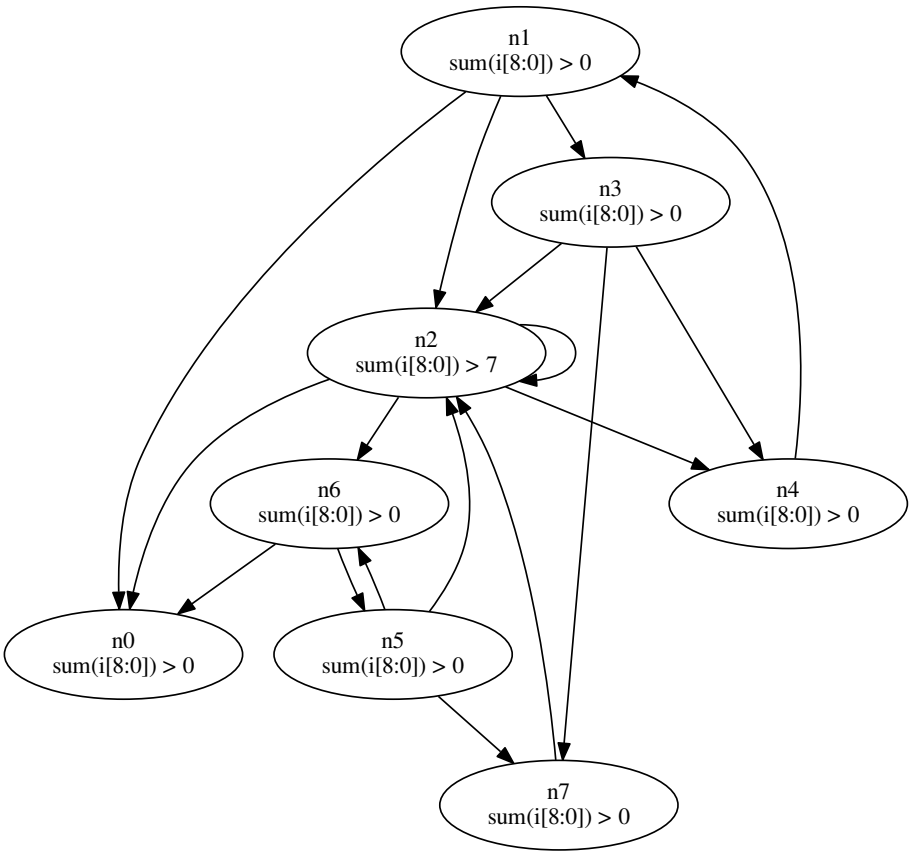

Figure 4.3: Experiment 1 - 3 peaks - run 7 - Network

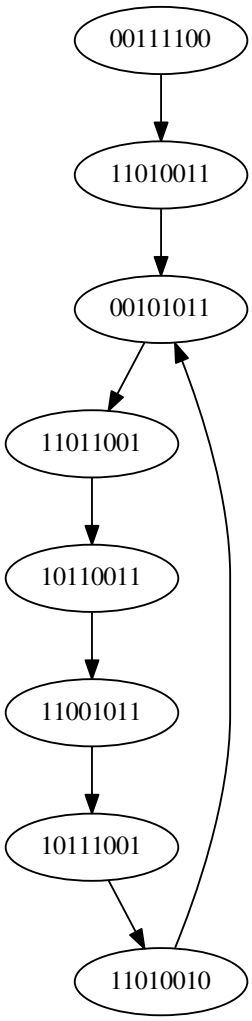

Figure 4.4: Experiment 1 - 3 peaks - run 7 - State space

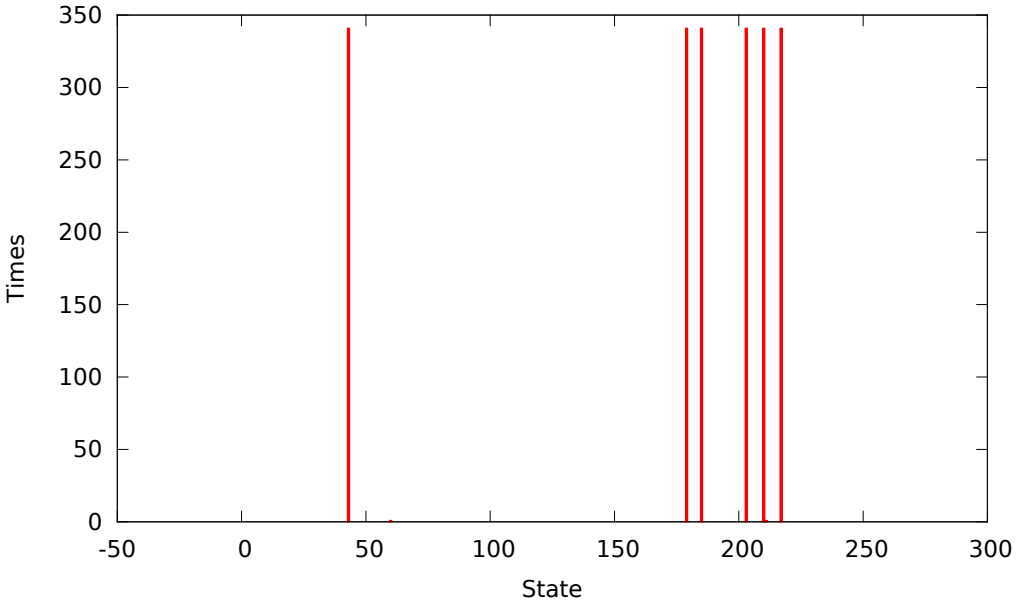

Figure 4.5: Experiment 1 - 3 peaks - run 7 - Coverage

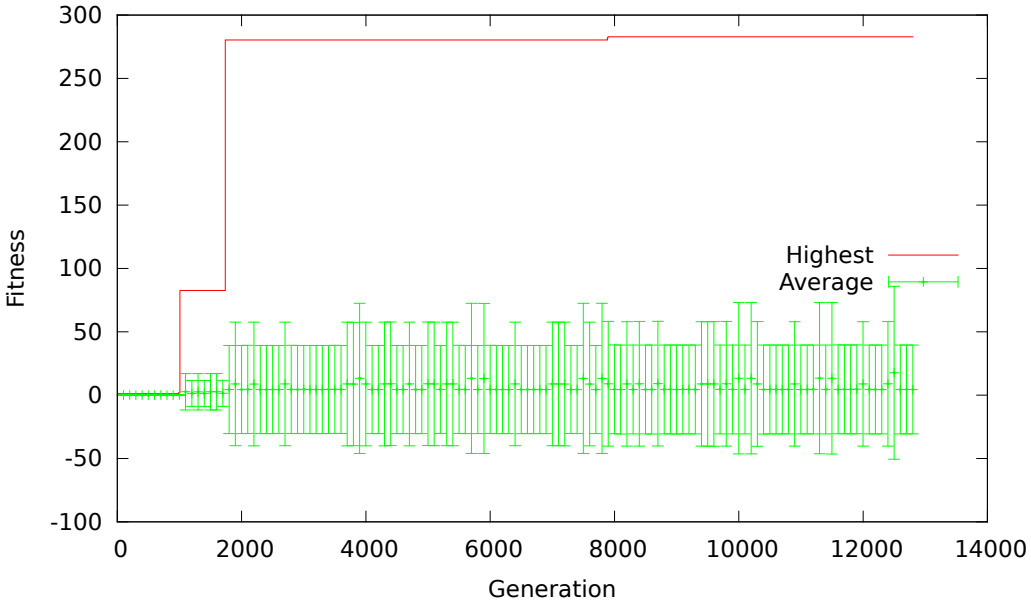

Figure 4.6: Experiment 1 - 3 peaks - run 7 - Fitness

4.2 Experiment 2

The purposed of experiment 2, is to evolve a network that is capable of generating the same number of frequency peaks for all initial states in a set of randomly generated initial states.

Eight randomly generated initial states, different between each instance of the experiment, were used. The results are given in Table 4.2. The *peaks* column gives how many peaks were sought after with each initial state, e.g. 1/1/1/1/1/1/1/1 means that one frequency peaks were searched for, one for each of the initial states. The *run* column gives which instance of the experiment the row is, the *generations* column how many generations the instance ran for, and the *success* column how many of the frequency peaks and initial state pairs found the sought after result. E.g. 7/8 means it found the correct number of frequency peaks for seven out of eight of the initial states.

As shown in Table 4.2, for all the different values of frequency peaks searched for, a network capable of producing the same number of frequency peaks for all initial values, where found. Since this experiments had eight initial values per run, only a few illustrative examples of the plots will be shown.

Figure 4.7 shows the frequency-power spectrum of the third initial state of run 0 with 10000001_2 as the initial value. The three frequency peaks are clearly visible in the frequency "buckets" 85, 171, and 256, with the large peak at 0 being the DC-component. Figure 4.8 shows the frequency-power plot from exact same network, see Figure 4.9, with the initial value 00011111_2, and having frequency peaks in the exact same places as the initial state form Figure 4.7. Note that the scale of power in the two figures is not the same.

In Figure 4.9 the network is plotted, with labeling of nodes and next state functions. Node *n0* corresponds to bit zero, *n1* to bit one, et cetera when the network state is interpreted as a binary unsigned number. There are a few "dead nodes" in this network, witness *n3* and *n6*, which have the next state function $S_{\text{next}} = \sum_{j=0}^{7} input_j > 7$. It follows that the next state function of *n3* and *n6* will always evaluate to 0, since only 2 and 4 inputs are present, respectively. This means that either of the nodes can only follow two patterns; either the node has 1 as initial node state and switches to 0 after first time step, or initial node state 0 and the node state stays 0 forever.

In Figure 4.12 the former case can be observed for both node *n3* and *n6*, while in Figure 4.13 the former can be observed for *n6* and the latter for

Peaks	Run	Generations	Success
1/1/1/1/1/1/1/1	0	1280	8/8
	1	1280	7/8
	2	1280	8/8
	3	1280	7/8
	4	1280	7/8
	5	1280	7/8
	6	1280	8/8
	7	1280	7/8
2/2/2/2/2/2/2/2	0	1280	8/8
	1	1280	8/8
	2	1280	7/8
	3	1280	6/8
	4	1280	7/8
	5	1280	7/8
	6	1280	8/8
	7	1280	6/8
3/3/3/3/3/3/3/3	0	12800	8/8
	1	12800	4/8
	2	12800	4/8
	3	12800	7/8
	4	12800	5/8
	5	12800	5/8
	6	12800	0/8
	7	12800	0/8

Table 4.2: Experiment 2 results

n3.

Even though the behaviour in the frequency-power spectrum is similar, the behaviour in the time-amplitude spectrum is not that similar. This is illustrated by Figure 4.10 and 4.11, where the amplitude peaks are easily seen. Can be witnessed further in Figure 4.12 and 4.13, where the state spaces are clearly disjunct, with a coverage set of

$$\{10000001_2, 00100010_2, 10010000_2, 00000110_2, 00010001_2, 00100100_2\}$$

for initial state 10000001_2 versus

$$\{00110111_2, 10110101_2, 10100111_2, 10110011_2, 10110110_2, 10010111_2\}$$

for initial state 00011111_2.

Figure 4.14 and 4.15 plots the number of times each network state is visited, by simply interpreting the network state as an unsigned boolean number, and counting how many times each number occurs. For Figure 4.14 the six network states contained in the coverage set stands out as large peaks, with no smaller peaks since all the states in the state space is contained in the coverage set. In Figure 4.15, in addition to the six states in th coverage set, a tiny bump can be seen at 31, this is the initial state.

The fitness is plotted in Figure 4.16, with a red solid line indicating the best fitness for that generation. The green crosses are the average fitness for that generation, with standard deviation as error bars. A good solution was found a little after the 2000th generation, and seen by the steep rise in fitness.

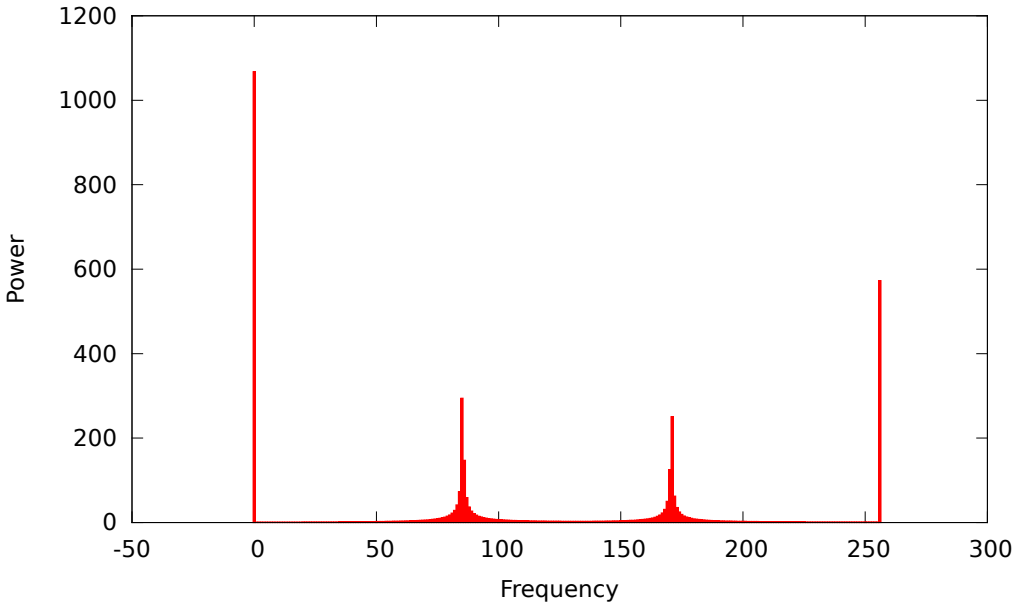

Figure 4.7: Experiment 2 - 3 peaks - run 0 - initial state 10000001_2 - Frequency-power plot

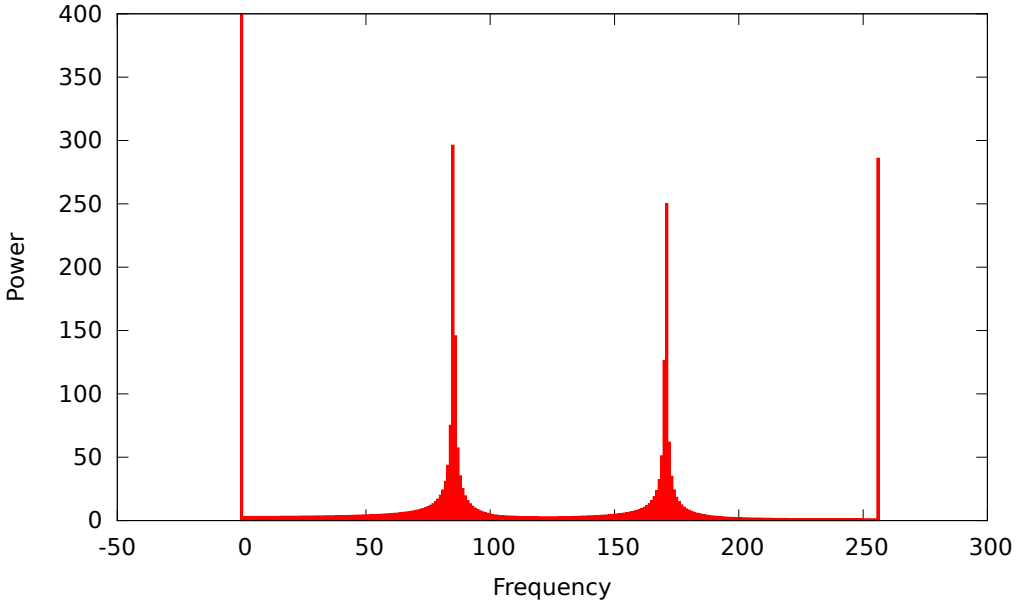

Figure 4.8: Experiment 2 - 3 peaks - run 0 - initial state 00011111_2 - Frequency-power plot

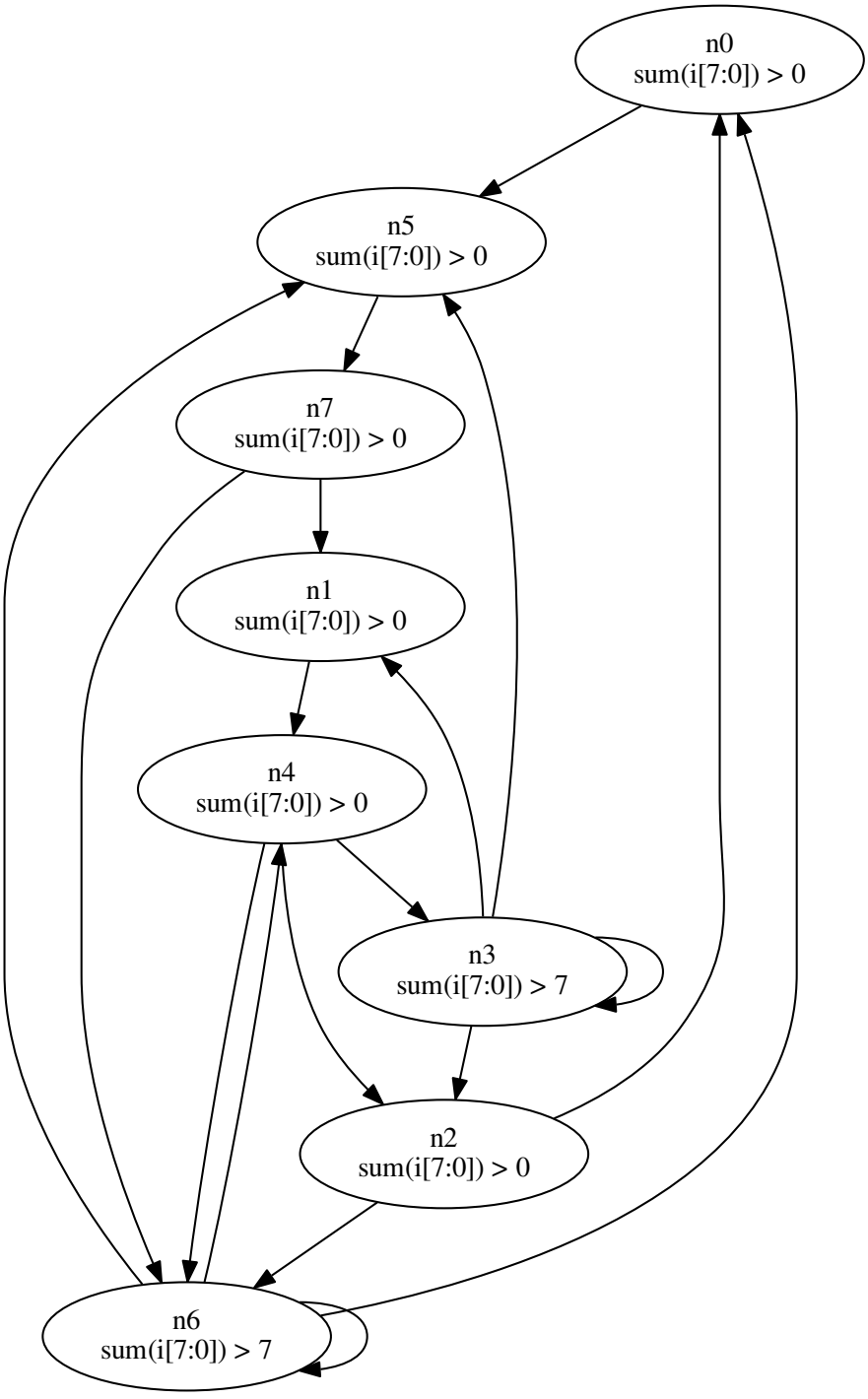

Figure 4.9: Experiment 2 - 3 peaks - run 0 - Network

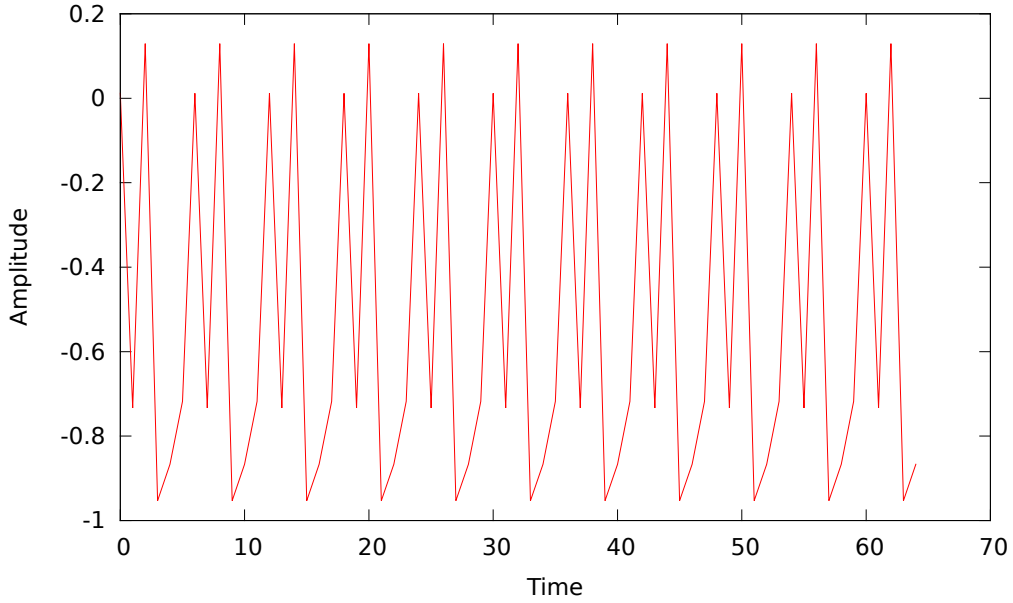

Figure 4.10: Experiment 2 - 3 peaks - run 0 - initial state 10000001_2 - Time-amplitude plot

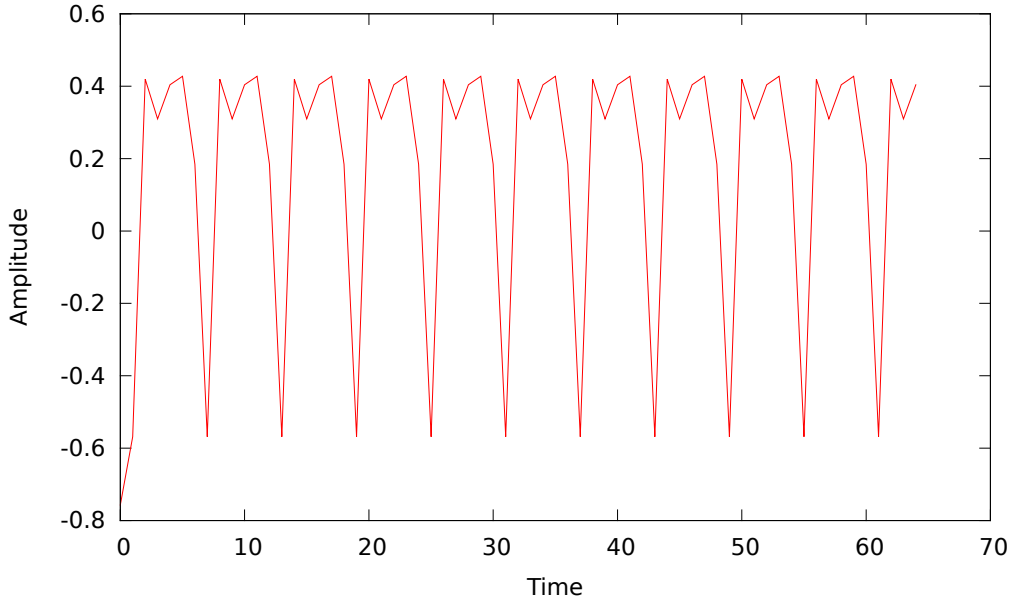

Figure 4.11: Experiment 2 - 3 peaks - run 0 - initial state 00011111_2 - Time-amplitude plot

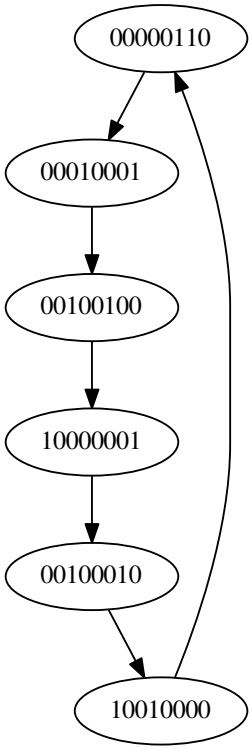

Figure 4.12: Experiment 2 - 3 peaks - run 0 - initial state 10000001_2 - State space

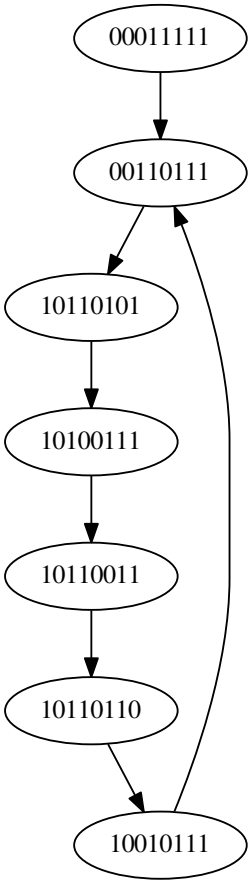

Figure 4.13: Experiment 2 - 3 peaks - run 0 - initial state 00011111_2 - State space

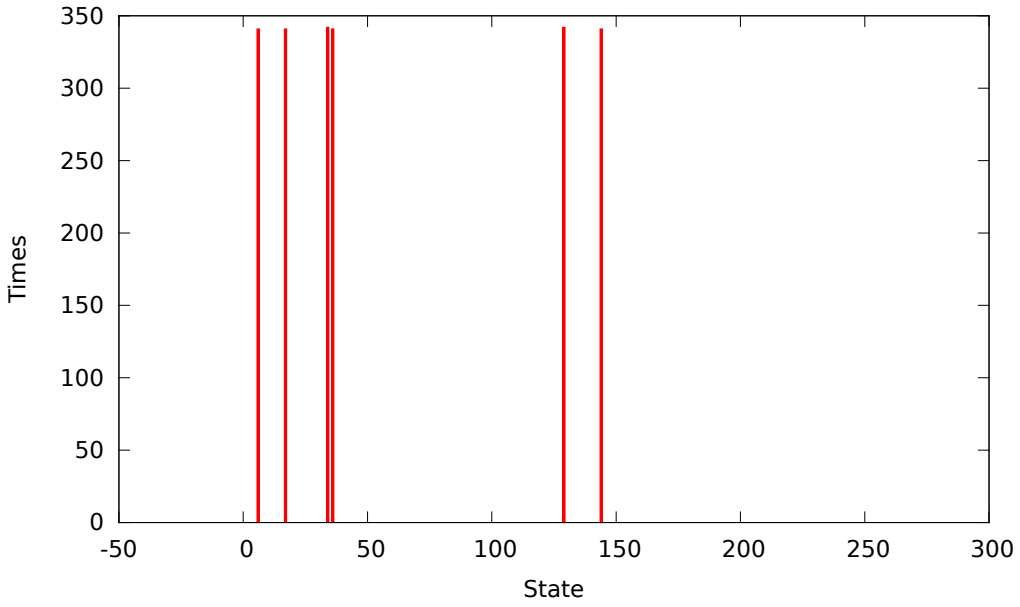

Figure 4.14: Experiment 2 - 3 peaks - run 0 - initial state 10000001_2 - Coverage

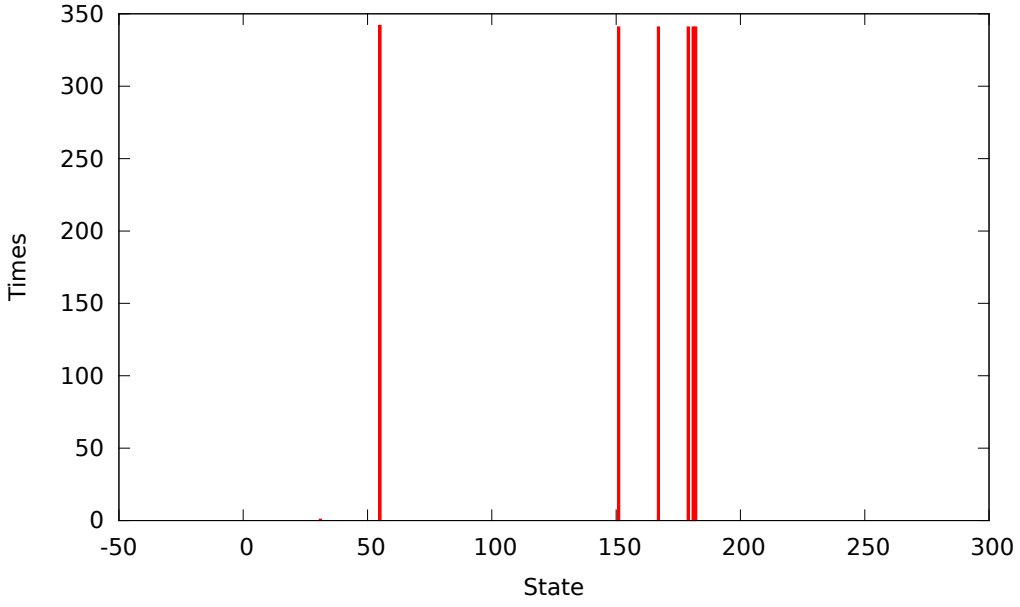

Figure 4.15: Experiment 2 - 3 peaks - run 0 - initial state 00011111_2 - Coverage

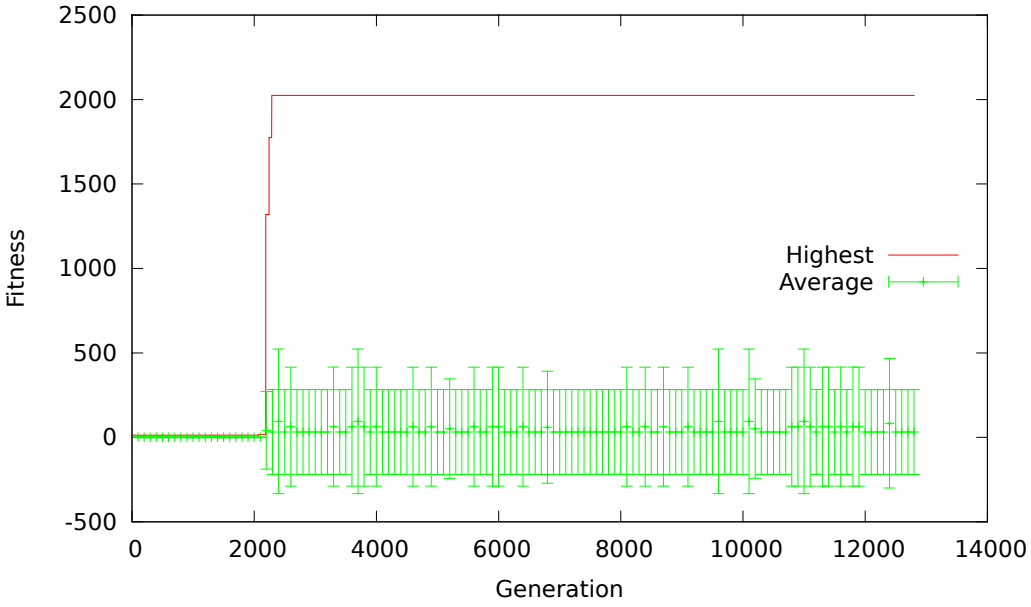

Figure 4.16: Experiment 2 - 3 peaks - run 0 - Fitness

4.3 Experiment 3

After establishing the possibility of evolving different number of frequencies, and establishing a baseline for what kind of behaviour the state space can support in experiment 1 and 2, experiment 3 attempts to establish that variable number of multivalued output in the frequency-power spectrum is possible.

In this experiment, two randomly generated initial states, that differed between each instance of the experiment, were used. Table 4.3 present the results, where the *peaks* column specifies how many peaks were sought after for each initial state, e.g. 1/2 specifies that one peak in frequency was sought after for the first initial state, and two frequency peaks for the second initial state. The *run* column gives which instance of the experiment the row provides results for, the *generations* column for how many generations the instance ran, and the *success* column how many of the frequency peaks and initial state pairs found the sought after result. E.g. 1/2 means it found the correct number of frequency peaks for one out of two of the initial states.

Peaks	Run	Generations	Success
1/2	0	12800	1/2
	1	12800	2/2
	2	12800	2/2
	3	12800	1/2
	4	12800	1/2
	5	12800	2/2
	6	12800	1/2
	7	12800	1/2
2/3	0	128000	1/2
	1	128000	2/2
	2	128000	1/2
	3	128000	1/2
	4	128000	2/2
	5	128000	0/2
	6	128000	2/2
	7	128000	1/2

Table 4.3: Experiment 3 results

As shown in Table 4.3, finding one and two frequency peaks for the same network was pretty successful. Finding two and three peaks in the same network proved to be harder, since the GA would find two peaks rather fast, and then spend all the time optimising the two peaks, never finding the three peaks. A countermeasure was employed, where finding two peaks were only attempted after first finding three peaks. I.e. an individual that had not yet found three peaks, would get the low fitness associated with not finding three peaks, even if it had found two. An individual that had found three peaks would get the fitness for the three peaks, in addition to whatever fitness it had for finding two peaks. This countermeasure proved to be successful, as shown in the results for 2/3 peaks in Table 4.3

In Figure 4.17, the frequency-power spectrum of of the result from instance three, with initial state 01110011_2 responsible for generating three frequency peaks, is shown. The three frequency peaks are visible in the frequency "buckets" 85, 171, and 256, with the DC-component at 0. In Figure 4.18 the frequency-power plot of output from the same network, illustrated in Figure 4.19, with the initial state 011110010_2, responsible for generating two frequency peaks.

Figure 4.19 illustrates the network, with labeling of nodes and next state functions. Node *n0* corresponds to bit zero, *n1* to bit one, et cetera, of the network state, when it is interpreted as a binary unsigned number. Note that node *n4* will always be the previous value of node *n1*.

Figure 4.20 and 4.21 shows the time-amplitude plot of the two different initial values, the one with three frequency peaks clearly stands out as more "busy" in the time-amplitude spectrum. Even though the three peak plot looks more busy in the time-amplitude spectrum, Figure 4.22 and 4.22 shows that the coverage set contains the same number of states for each of them Although the two coverage sets are the same size, the state spaces are clearly disjunct, as seen by

$$\{11111100_2, 00100111_2, 11011101_2, 00101011_2, 11011110_2, 11110011_2\}$$

for initial state 01110011_2 versus

$$\{11110100_2, 00100101_2, 00001101_2, 00001011_2, 11011010_2, 11110010_2\}$$

for initial state 01110010_2.

Figure 4.24 and 4.25 plots the number of times each network state is visited, by simply interpreting the network state as an unsigned boolean number, and counting how many times each number occurs. In Figure 4.24 the six

network states contained in the coverage set stands out as large peaks with with one tiny bump for the initial state at 115. For Figure 4.25, in addition to the six states in the coverage set, a tiny bump can be seen at 114, this is the initial state.

The fitness is plotted in Figure 4.26, with a red solid line indicating the best fitness for that generation. The green crosses are the average fitness for that generation, with standard deviation marked as bars. After a steep start, the best fitness in the population gradually rises until its peak is reached after 45000 generations or so.

One thing observed during this experiment, is sometimes the network evolved what seems to be "hybrid" networks, i.e. they can be interpreted as both three peaks or two peaks, as shown by the frequency-power spectrum plot of what is supposed to be a two peak solution in Figure 4.27. These "hybrid" solutions are not counted as success in this experiment.

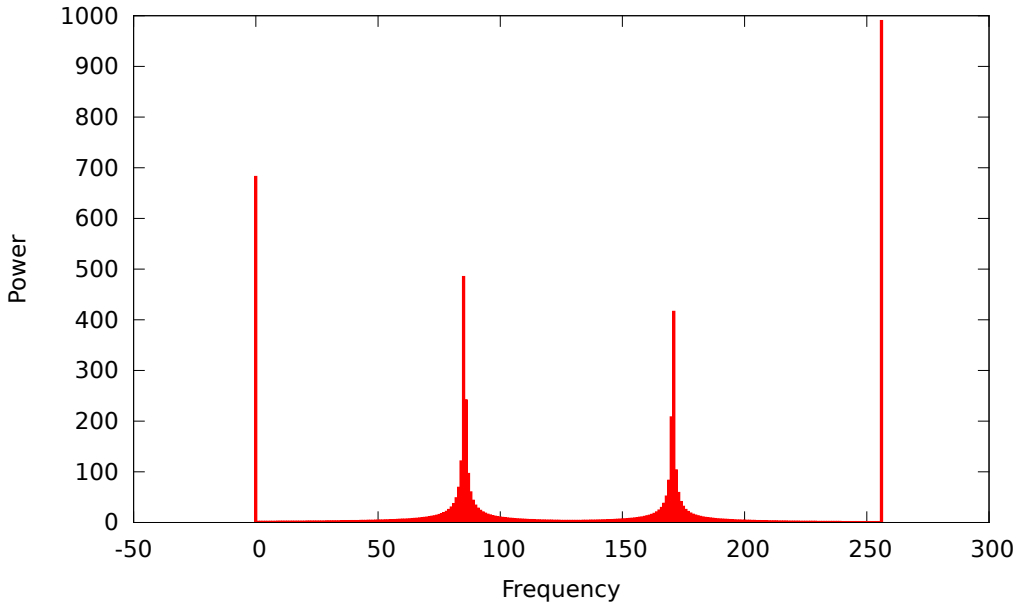

Figure 4.17: Experiment 3 - 3 peaks - run 6 - initial state 01110011_2 - Frequency-power plot

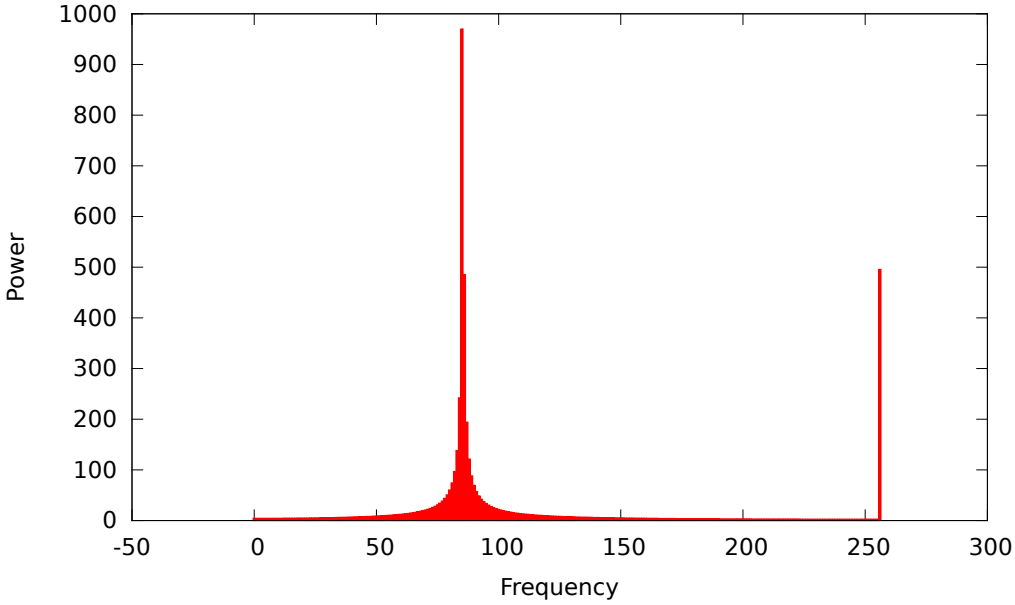

Figure 4.18: Experiment 3 - 2 peaks - run 6 - initial state 01110010_2 - Frequency-power plot

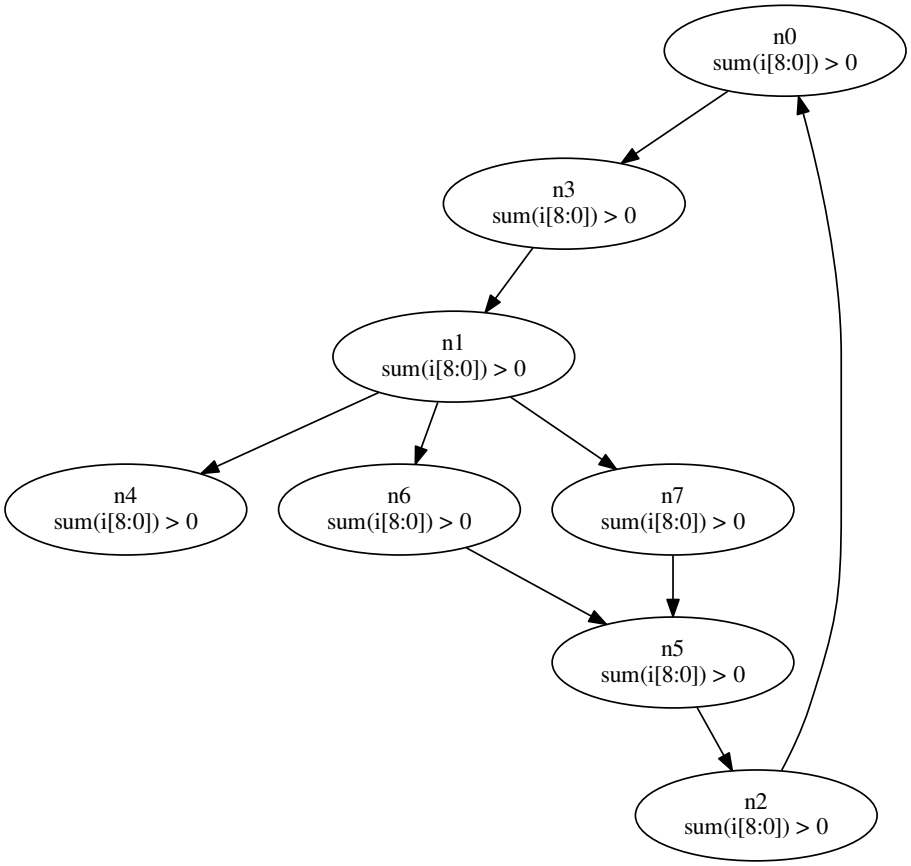

Figure 4.19: Experiment 3 - 2 and 3 peaks - run 6 - Network

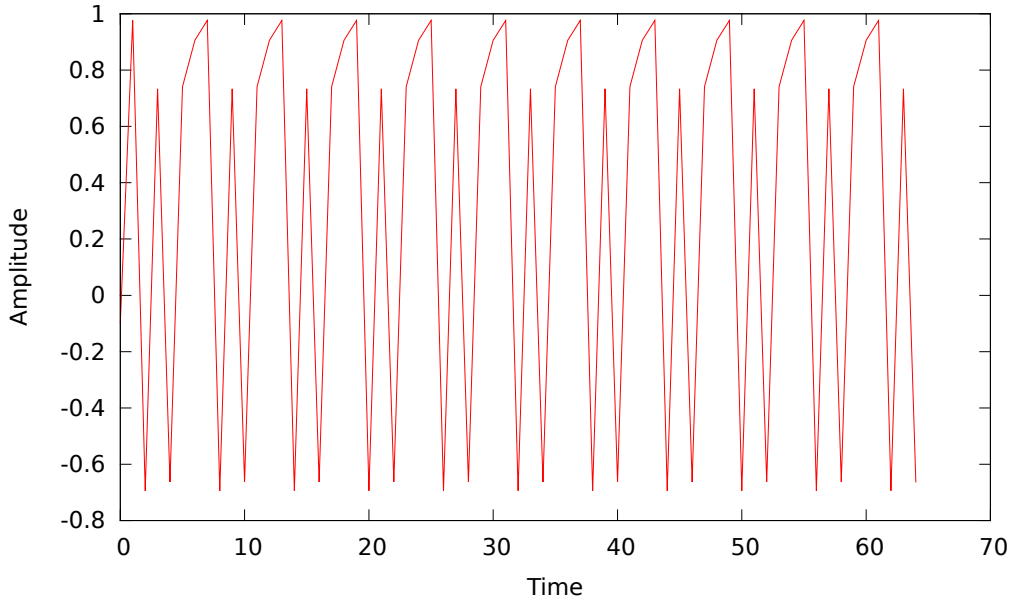

Figure 4.20: Experiment 3 - 3 peaks - run 6 - initial state 01110011_2 - Time-amplitude plot

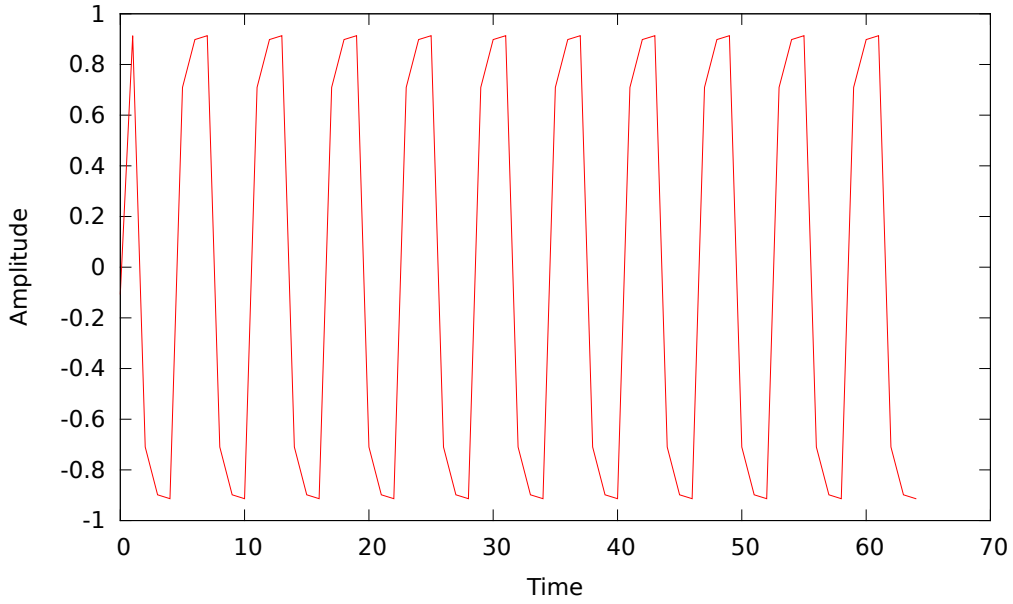

Figure 4.21: Experiment 3 - 2 peaks - run 6 - initial state 01110010_2 - Time-amplitude plot

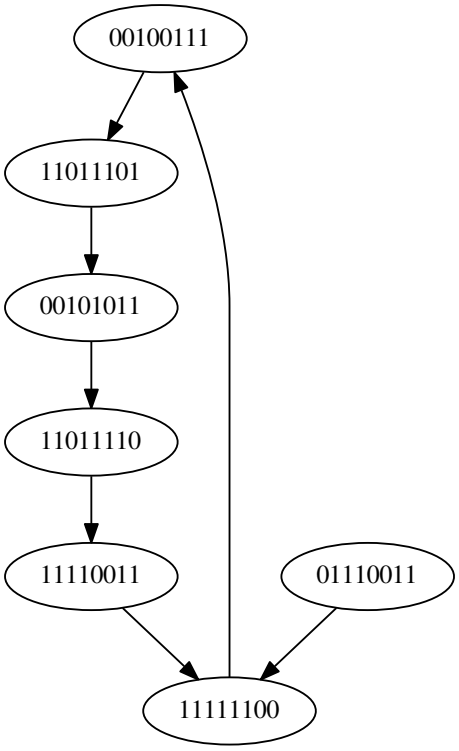

Figure 4.22: Experiment 3 - 3 peaks - run 6 - initial state 01110011_2 - State space

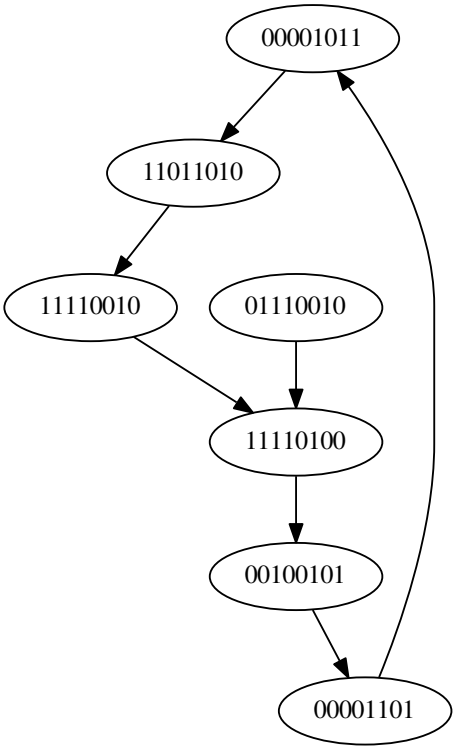

Figure 4.23: Experiment 3 - 2 peaks - run 6 - initial state 01110010_2 - State space

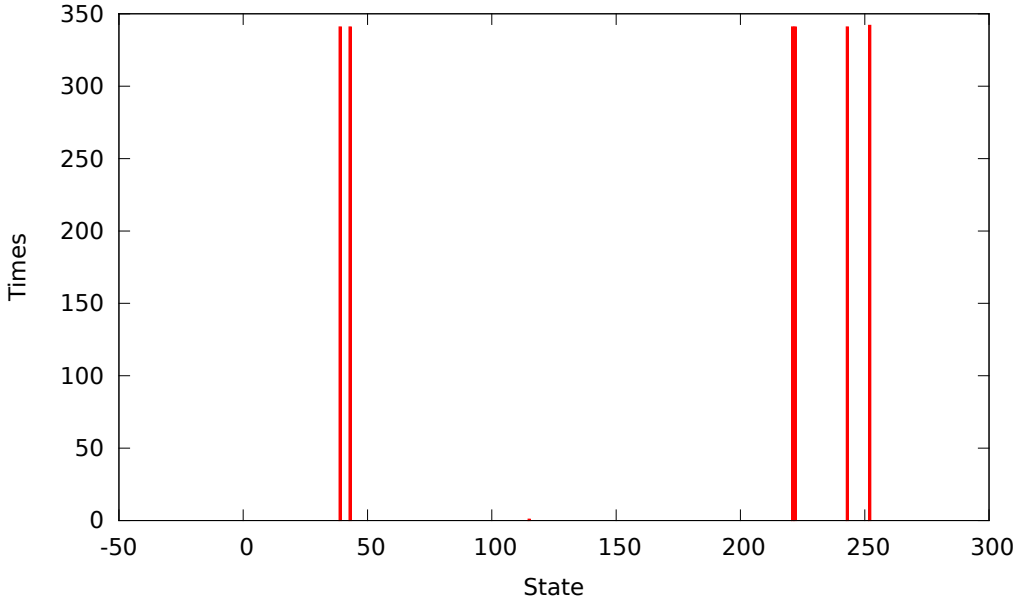

Figure 4.24: Experiment 3 - 3 peaks - run 6 - initial state 01110011_2 - Coverage

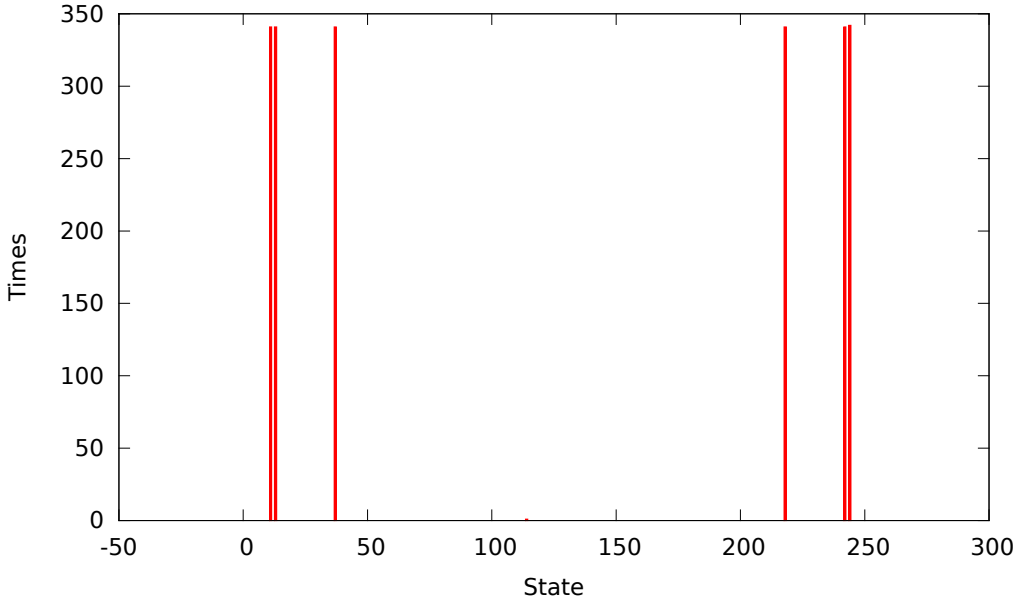

Figure 4.25: Experiment 3 - 2 peaks - run 6 - initial state 01110010_2 - Coverage

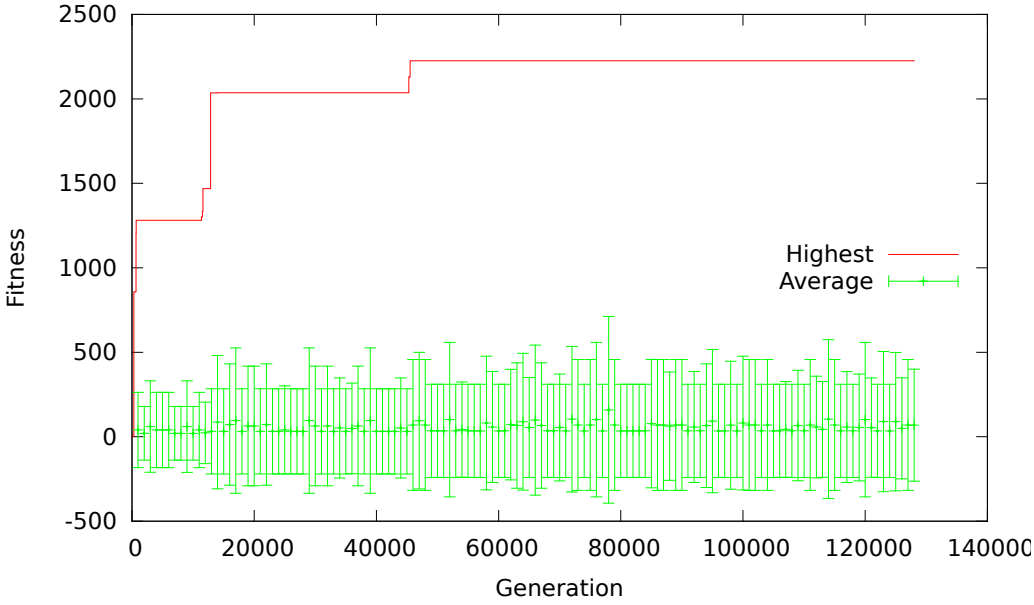

Figure 4.26: Experiment 3 - 2 and 3 peaks - run 6 - Fitness

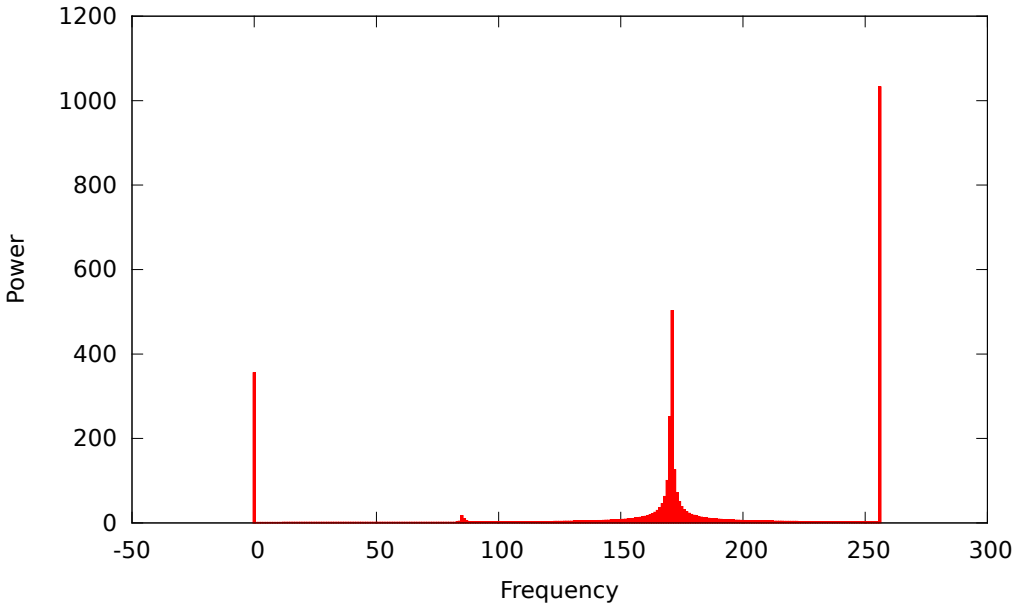

Figure 4.27: Experiment 3 - 2 peaks - run 7 - initial state 01110011_2 - Frequency-power plot

4.4 Experiments summary

The focus of this proof of principle is to show that evolving specific emergent behaviour in the frequency-power spectrum is actually possible. The experiments and their success rate is summarised in Table 4.4. As shown in Table 4.4 every experiment was a success, and the only limitation that was hit, was evolving more than three peaks. Theories about this limitation will be discussed in Chapter 5.

Experiment	Peaks	Success rate
1	1	100%
	2	100%
	3	75%
	4	0%
2	1/1/1/1/1/1/1/1	37.5%
	2/2/2/2/2/2/2/2	37.5%
	3/3/3/3/3/3/3/3	12.5%
3	1/2	37.5%
	2/3	37.5%

Table 4.4: Experiments summary

Chapter 5

Conclusion

The experiments were a complete success, the results show that using the emergent behaviour of a system in frequency-power spectrum opposed to the more traditional space-time spectrum is possible. The results also show that creating systems with emergent behaviour of variable number of output values (peaks), depending on initial state of the system is possible.

Through experiment, it has been shown that generating a specific number of peaks in the frequency-power spectrum is possible. One interesting result, is that four peaks proved so hard to find. In the experiment with four nodes, all instances ran into a local maxima and never got out of it. This could be due to several reasons, but two theories have been devised. The first theory, is one touched upon briefly; that it needs to be more nodes in the network. This is supported by the observation that a network producing smaller amount of frequency peaks tend to have more dead nodes, like *n2* in Figure 4.5. So simply having a larger amount of nodes could make the GA able to find networks producing more than three peaks in the frequency-power spectrum.

Another theory is that the trouble with finding more than three peaks, is caused by a combination of next state function, and where the solution lays in the solution space. I.e. it is very hard to find a path traversing the solution space with the genotype used, where an individual can gradually increase fitness and eventually stumble upon a solution with four peaks through mutation and crossover. Figure 5.1 attempts to illustrate this point. A reason for the solution space to look like this, could be the encoding of, or the next state function it self.

The reason a so simple next state function was chosen is two fold; small

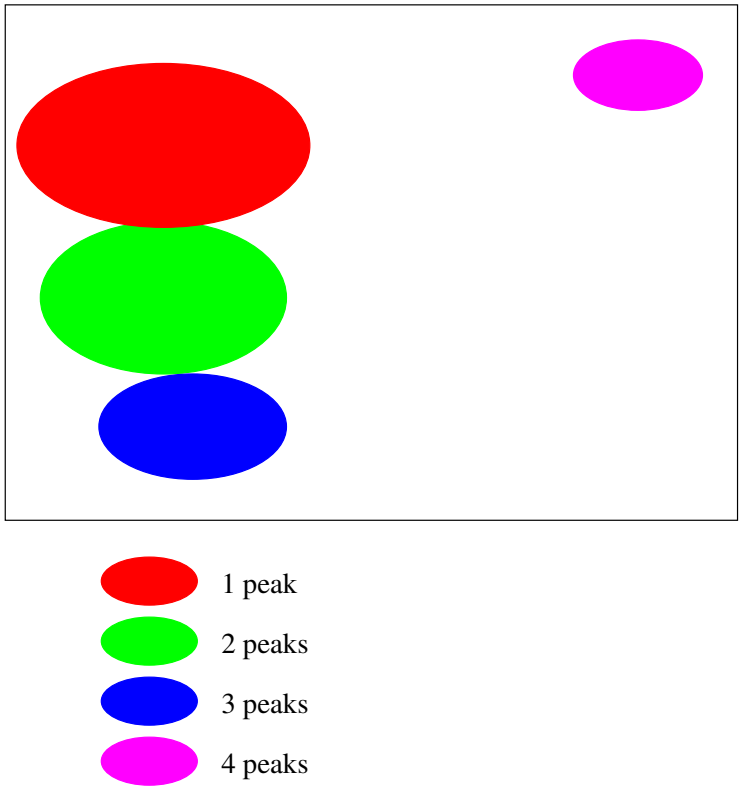

Figure 5.1: Solution space with large distance to four peaks

number of bits used to encode the function in the genotype to get as small solution space as possible. This minimisation of solution space, while it can make it easier to find solutions in it, could also potentially remove the solutions that are sought after from the solution space.

The other reason for the choice, was to avoid wildly different function as the next state function because of a single mutation. E.g. if a boolean AND, and boolean OR functions selected by a value of some bit, had been used, the next state function would be a wildly different function dependent of the value of that bit. It is an attempt to make the GA a fitness guided search in solution space, not an arbitrary search.

In these experiments, only BNs were, used. Since BNs are simply a specialisation of DDNs, more node states could be used, a la what Langton did with CAs[4]. Another thing to explore is more complex combinations of frequency peaks, e.g. finding one, two, and three peaks with the same network.

Bibliography

[1] John Von Neumann. *Theory of Self-Reproducing Automata*. University of Illinois Press, Champaign, IL, USA, 1966.

[2] Andrew Wuensche. Discrete dynamics lab: Tools for investigating cellular automata and discrete dynamical networks. In Andrew Adamatzky and Maciej Komosinski, editors, *Artificial Life Models in Software*, pages 263–297. Springer London, 2005.

[3] S. Wolfram. Universality and complexity in cellular automata. *Physica D: Nonlinear Phenomena*, 10(1-2):1–35, January 1984.

[4] Chris G. Langton. *Emergent computation*, chapter Computation at the edge of Chaos: phase transitions and emergent computation, pages 12–37. MIT Press, Cambridge, MA, USA, 1991.

[5] Melanie Mitchell. Life and evolution in computers. *History and philosophy of the life sciences*, 23, 2001.

[6] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *J Theor Biol*, 22(3):437–467, 1969.

[7] Matthew Cook. Universality in elementary cellular automata. *Complex Systems*, 15(1), 2004.

[8] Moshe Sipper. The emergence of cellular computing. *Computer*, 32(7):18–26, July 1999.

[9] Carlos Gershenson. Introduction to random boolean networks. In Bedau, M., P. Husbands, T. Hutton, S. Kumar, and H. Suzuki (eds.) Workshop and Tutorial Proceedings, Ninth International Conference on the Simulation and Synthesis of Living Systems (ALife IX). pp. 160-173. 2004, 2004.

[10] Melanie Mitchell. *Complexity: A Guided Tour*. Oxford University Press, Inc., New York, NY, USA, 2009.

[11] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[12] J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, USA, 1975.

[13] Dario Floreano and Claudio Mattiussi. *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*, chapter Evolutionary Systems. The MIT Press, 2008.

[14] Keith Downing. Introduction to evolutionary algorithms, 2010. `http://www.idi.ntnu.no/emner/it3708/lectures/notes/evolalgs.pdf`.

[15] Keith Downing. Natural and artificial selection, 2011. `http://www.idi.ntnu.no/emner/it3708/lectures/notes/ea-selection.pdf`.

[16] Gunnar Tufte. *Evolutionary Computation*, chapter From Evo to EvoDevo: Mapping and Adaptation in Artificial Development, pages 219–238. InTech, 2009.

[17] C. E. Shannon. Communication in the Presence of Noise. *Proceedings of the IRE*, 37(1):10–21, January 1949.

[18] H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47:617–644, April 1928.