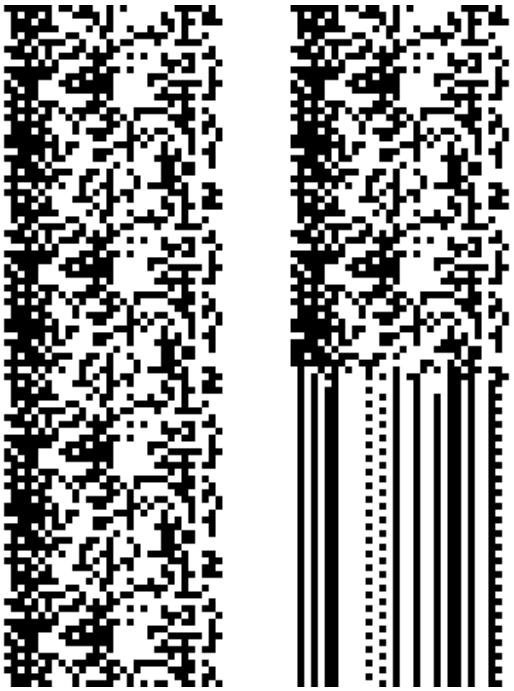
Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Katrine Roland

Audio Generation with Random Boolean Networks

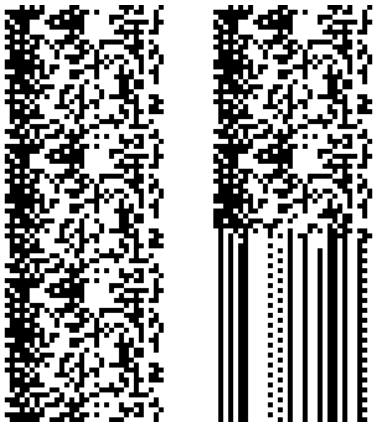
Master's thesis in Computer Science
Supervisor: Gunnar Tufte
June 2019



NTNU
Norwegian University of
Science and Technology

Katrine Roland

Audio Generation with Random Boolean Networks



Master's thesis in Computer Science
Supervisor: Gunnar Tufte
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Summary

Complex systems exist in various different forms, both in nature and artificially created by humans. While these systems can exhibit very complex behavior, this behavior can be based on very simple rules. In nature such rules may be governed by chemical or physical properties at different levels, of which examples are reactions, diffusion, surface tension, molecular interactions or cellular processes. In the artificial domain complex systems are often modeled by systems with no global control. The behavior emerge from local rules, which imply local control, and local interactions between basic units.

One such system is Random Boolean Networks, whose behaviors are based on such simple rules in randomly generated boolean nodes, which are in addition randomly connected. This project explores the properties of the behaviors of Random Boolean Networks based on different parameters. These networks were explored toward finding networks that can be used to control parameters of sound in audio generation.

An application that generates Random Boolean Networks based on different parameters, and iterates networks, has been developed, as well as scripts used for analyzing the iterating networks to detect networks that may be favorable for music generation. Promising networks are used to control parameters for audio generation. The goal of this is to generate audio which may sound somewhat musical, while being based on randomness. To accomplish this, an application which maps the states of the networks to various parameters of audio generation was developed. The Random Boolean Networks show behavior that has a degree of order, while still being random and nondeterministic. This makes it possible to generate audio which instead of being completely chaotic can possibly be recognized as something akin to music, without being overly repetitive.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

[Mit09] defines complex systems as ”an interdisciplinary field of research that seeks to explain how large numbers of relatively simple entities organize themselves, without the benefit of any central controller, into a collective whole that creates patterns, uses information, and, in some cases, evolves and learns”. [Wol02] posits that if science is to be at all possible, systems must follow rules. But the complexity of the ruleset does not define the complexity of the system. Systems with simple rules can also show complex behavior. It seems intuitive that a complex system would need complex rules. This is only true when we need to predict the behavior of the system, to be able to construct a system to behave in a certain way. When it isn't necessary to predict the behavior of the system, we can create systems based on simple rules that still show complex behavior [Wol02].

The world we live in is full of complex systems, and all it takes is that these systems behave in the same way as some simple programs that produce great complexity. For example, take the notion of phases and transitions between phases in matter. [Lan90] has found that this concept can also be found in computation, and that phases and phase transitions may be fundamental classes of dynamical behavior. There is also some evidence that other similar analogs may exist. In living cells, phase transitions can be found everywhere, which may suggest that living beings are the result of the same kind of ”computations” [Lan90]. Living cells are the basic building blocks of everything in nature, and they contain a lot of complexity which is inherently ordered. The way complexity in living organisms affect the behavior of cells may be a factor in biological evolution in addition to natural selection [Kau93].

Nature's complex systems can produce forms that humans find incredibly beautiful, but have struggled to replicate [Wol02]. Examples of such patterns are seashells and animal skins like zebra stripes or leopard spots. [BBC03] have attempted to replicate such patterns. While their results somewhat resembled what they tried to replicate, it wasn't completely successful. They also found that with their methods, the behavior of the movement of patterns were a lot more interesting than the final pattern [BBC03]. A more successful attempt at replicating complexity from nature was done by [Rey87]. He was able to quite realistically model the behavior of a flock of birds by giving each individual simulated

bird relatively simple behaviors to follow. The aggregation of these behaviors combined with the laws of physics created movement which intuitively correspond to "flock-like motion" [Rey87]. Since nature's complex systems can behave like simple programs, it can be possible to create similarly beautiful creations by applying simple rules. Systems in nature with quite different components can show very similar behavior. The same thing can be observed with simple programs: programs with quite different rules may show very similar behavior. This raises the notion that insight into the behavior of complex systems based on simple programs might at the same time provide insight into complex systems in nature [Wol02].

An important factor of complex systems is that they are irreducible, and thus the only way to find their behavior is to actually observe the behavior. This might even explain how humans, while at the most fundamental level following underlying rules, still can be unpredictable and show free will [Wol02].

There are many complex systems in the world, and they work despite being based on simple rules[Mit09]. Disordered and featureless systems might spontaneously organize themselves [Wol02]. This concept of systems that organize themselves without being controlled centrally, self-organization, will emerge in, among other phenomenons, network structures that are sufficiently complex [Mit09]. The simple rules of these complex systems can be applied to completely random initial conditions, but order and organization will still emerge with time [Wol02]. As previously mentioned, it is difficult to create complex systems from simple rules if it is necessary to predict the behavior. That is because even though order may emerge in complex systems, they need not be deterministic. Thus, the resulting behavior or form, will not be absolute [Wol02]. External factors may also perturb the system, contributing to the lack of determinism [Ger04]. This makes self-organizing systems, while interesting, unfavorable for critical applications where even a small failure may result in harmful consequences. However, in applications where nondeterminism and occasional unexpected behavior doesn't matter, or even is desired, self-organizing systems may have their place.

Random Boolean Networks is an example of a network structure which can be sufficiently complex for self-organization to emerge [Mit09]. Thus, Random Boolean Networks are an interesting topic for exploration. Looking at how Random Boolean Networks self-organize by reaching attractors, and how resilient various networks are to external perturbations may provide interesting results for applications that can handle nondeterminism and occasional unexpected behavior.

An example of an application where nondeterminism and unexpected behavior does not cause harmful consequences, is audio. The randomness of Random Boolean Networks can be used to create audio which is not repetitive, while the order which emerges will give the generated audio something that is not completely chaotic. Applying different Random Boolean Networks to different parameters while generating audio may provide results that might even sound somewhat musical.

The following chapter will provide background information on some complex structures, namely cellular automata, artificial neural networks and Random Boolean Networks. Random Boolean Networks will be covered more in-depth than the others. In addition, basic information on audio generation is covered. Chapter 3 describes how the Random Boolean Networks were generated and analyzed in this project, and also how they were

used to interact with audio generation. Chapter 4 contains descriptions on the different experiments which were done with the Random Boolean Networks, and what the results of each were. Chapter 5 presents the conclusion.

Chapter 2

Background

Network science makes it possible to achieve a greater understanding of complex systems [Bar16]. Cellular automata, Random Boolean Networks and artificial neural networks are all networks that are sparsely connected; there are relatively few connections between the nodes of the networks. A one-dimensional cellular automaton consists of a line of cells which are all either black or white. Belonging to the cellular automaton is a simple ruleset that determines a cells color at the next step based on the cell itself and its neighbours. With these simple rulesets, and with different initial states, both simple and complex patterns can emerge as a cellular automaton iterates. Figure 2.1 illustrates complex patterns in a one-dimensional cellular automaton. Cellular automatons can also be multi-dimensional; they then work in the same way, but the ruleset must support a multi-dimensional neighbourhood [Wol02].

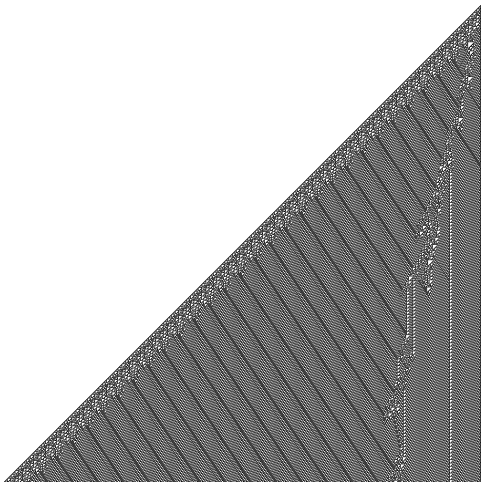


Figure 2.1: Iterations of a cellular automaton. Wolfram rule 110.

An artificial neural network is based on the idea of how brains work. The nodes in an artificial neural network are seen as artificial neurons. The neurons are connected in a network structure, and each connection has a weight which the incoming signal is multiplied by. Then an activation function calculates whether the neuron should activate based on the inputs. Another function calculates the output of the neuron, which is then an input of other neurons [Ger03].

Random Boolean Networks are explained more in-depth below. All of these structures consist of simple components, but the components together in the connected structures can show complex behavior. This complex behavior from simple components is known as emergent complexity [BY99].

2.1 Random Boolean Networks

A Random Boolean Network (also called an N - K network) is a collection of N nodes, with an average of K connections going into each node. At any point in time, each node has one of two possible values, usually represented by 1 and 0. Whenever a node is updated, its next value is decided by a boolean function whose inputs are the nodes K incoming connections. Both the connections and the boolean functions are generated randomly, and they are never changed [Ger04]. While Random Boolean Networks can be updated asynchronously, and many of the concepts they are used to model really are asynchronous, synchronous Random Boolean Networks usually model the concepts closely enough [Kau93]. As a Random Boolean Network contains N nodes, there are 2^N possible states for a network. As this is a finite number, the network will at some point reach a state which it has previously visited. After this has happened, as long as the network is not affected by any outside forces, it will only reach states it has already visited, and it has thus reached an attractor. Thus it will never reach any state outside the attractor [Ger04]. With K incoming connections to each node, there are 2^{2^K} different possible boolean functions for each node [Kau93]. Depending on the value of K, and if any requirements are imposed on the nodes boolean functions, the networks will behave in different fashions. A simple Random Boolean Network with N=3 and K=2 is shown in figure 2.2. Also shown is a trajectory of states which follow from a random initial state. An attractor is reached very quickly in this case.

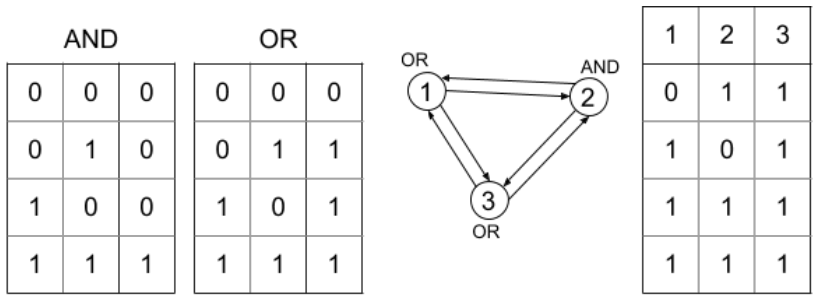


Figure 2.2: A simple Random Boolean Network and an example trajectory into an attractor.

With some of the boolean functions, the result can be known based on only one of the inputs, depending on the value of the input. These are called canalyzing functions. Simple examples of canalyzing functions are the boolean AND and OR. With AND, if any input is zero, the output will be zero independent of the value of the other input. With OR, if any input is one, the output will we one independent of the value of the other input. With a Random Boolean Network with randomly generated functions for each node, the fraction of functions which are canalyzing will decrease as K increases [Kau93].

When using randomly generated truth tables as functions, a bias P can be introduced. P represents the average fraction of the outputs in the truth table that should be ones, the internal homogeneity of the network. In addition to the value of K, the value of P and whether functions are canalyzing affects the behavior of a network. Lower values of K, and the use of canalyzing functions lead networks toward more ordered behavior. Values of P closer to 0 or 1 lead to more ordered behavior, while when P is equal to 0.5, the behavior is the most chaotic [Kau93].

The length of attractors will vary depending on the properties of a network. In K=N networks, the median length is $0.5(2^{N/2})$. Thus, attractor lengths increase as N increases. As this is the median, lengths will vary within one network. With long attractors, the number of attractors is a lot lower, N/e. For K=2 networks, both the median attractor length and the number of attractors is $\sqrt{N}$, and for K=1 networks, the attractor lengths are on the order of $\sqrt{2^N}$. Increasing or decreasing P away from 0.5 will create shorter attractors for K=N, but the attractor lengths will still increase exponentially: $0.5(\frac{1}{\sqrt{P}})^N$. This exponential increase actually hold for networks of $K \geq 5$. Though, with N much larger than K and sufficiently high internal homogeneity, networks with reasonably small K can show ordered behavior [Kau93].

The different ways the networks can behave are divided into three phases: ordered, on the edge of chaos and chaotic. These phases are also called solid, liquid and gas, like the phases of matter. Though, the liquid phase of a Random Boolean Network does not appear to be a quite distinct phase as the liquid phase of matter. As with matter, Random Boolean Networks may move between the different phase through phase-transitions. The chaotic phase is characterized by very long attractors with constant change at almost all the nodes, while in the ordered phase all, or almost all nodes stay at the same value. Between these two phases, at the edge of chaos, different numbers of nodes vary or stay constant. Varying K, P or the fraction of functions which are canalyzing may cause phase-transitions in a Random Boolean Network [Kau93].

Random Boolean Networks also show interesting behavior if at some point in time, the network is perturbed by inverting the value of a random node in the network. Before the perturbation, the network will have been in an attractor or a basin of attraction. Depending on the network and on what nodes value is inverted, the network will either end up somewhere else in the same attractor or basin of attraction, or it will jump to a different one. For example, for networks with K=2, there is a probability between 80 and 90 percent that the network will return to the same attractor aftor being perturbed [Kau93].

2.2 Audio generation

To generate audio, the shape of a sound wave has to be specified in some way. Synthesizers are a common tool used to generate audio, often of the musical kind. The most common type of synthesizer today is the subtractive synthesizer, of which a simple example is shown in figure 2.3. Subtractive synthesizers start with a (usually) complex waveform, generated by the left-most box in figure 2.3. Low-pass filters and other filters (middle box in figure 2.3) are then used to remove a number of frequencies from the waveform and thus creating many different waveforms. What is actually done here, is to start with a rather complex waveform and then remove frequencies from the waveform in various ways to achieve the desired sound. The pitch of a sound is decided by the lowest frequency it contains, and all the other frequencies are overtones that affect how a tone will sound.

Figure 2.3: The flow in a very simple subtractive synthesizer.

Two common waveforms to use are the saw wave and the square wave. They both get their names from their shape. The saw wave looks like the teeth of a saw, as can be seen i figure 2.4, which shows a saw wave in the time domain. The square wave consists of a lot of squares on either side of zero, as can be seen in figure 2.6, which shows a square wave in the time domain. While they both look simple when looked at in the time domain, they both contain many frequencies, as can be seen in figures 2.5 and 2.7 which show the same waves in the frequency domain. The square wave is a special case of the pulse wave, where the pulse width is 0.5. Changing the pulse width will make the amount of time the wave spends above and below zero uneven. Figure 2.8 shows a pulse wave with a pulse width different from 0.5, and the corresponding frequency spectrum can be seen in figure 2.9. Pulse width modulation is defined as changing the pulse-width over time [Rei99b].

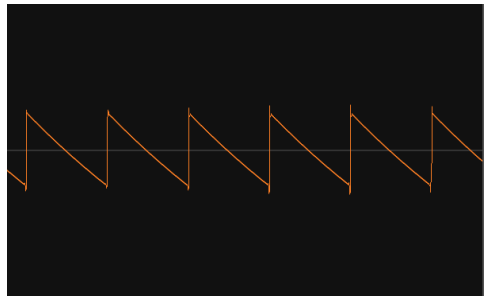


Figure 2.4: A saw wave in the time domain. Voltage as a function of time.

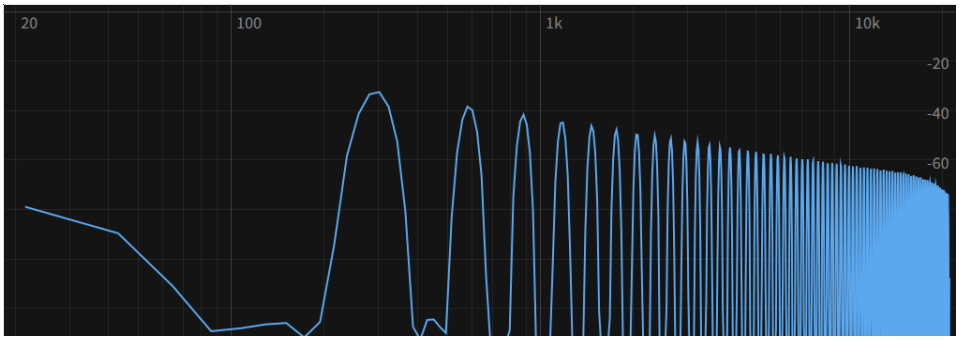


Figure 2.5: A saw wave in the frequency domain. Amplitude as a function of frequency.

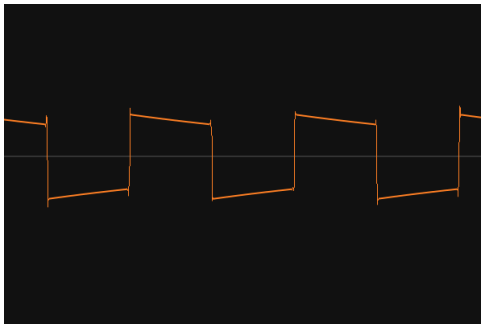


Figure 2.6: A square wave in the time domain. Voltage as a function of time.

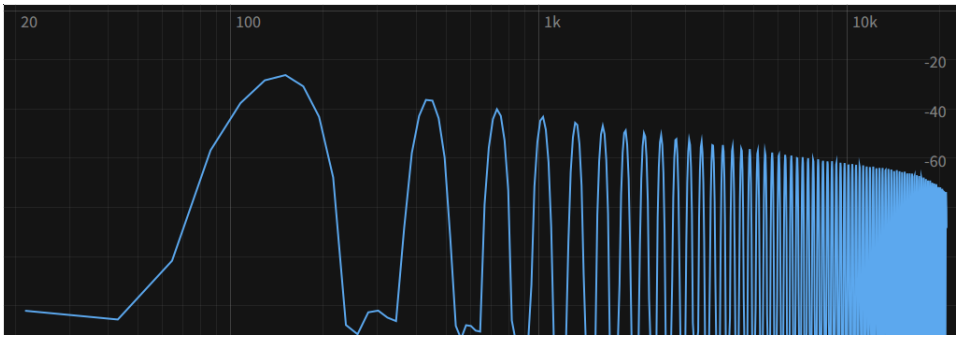


Figure 2.7: A square wave in the frequency domain. Amplitude as a function of frequency.

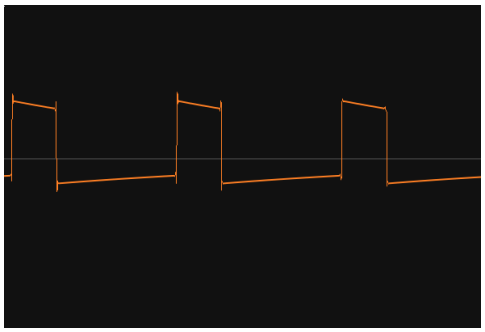


Figure 2.8: A pulse wave in the time domain. Voltage as a function of time.

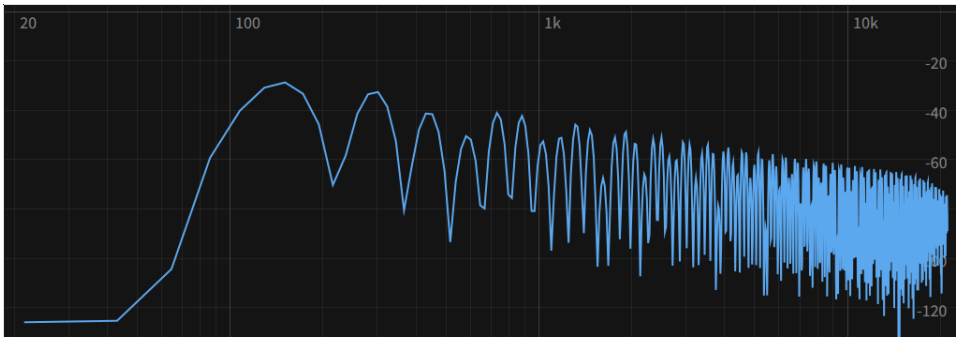


Figure 2.9: A pulse wave in the frequency domain. Amplitude as a function of frequency.

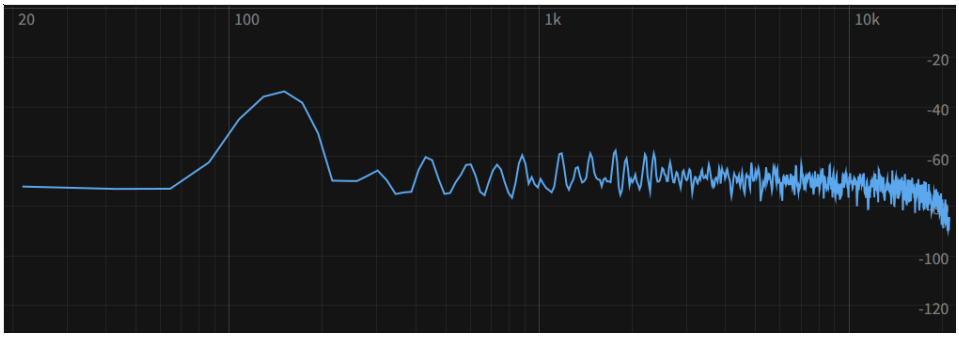


Figure 2.10: An unfiltered wave. Amplitude as a function of frequency.

A common way to remove frequencies from a waveform, is to use a low pass filter. With a low pass filter, all frequencies above a certain frequency is attenuated. Figures 2.10 and 2.11 show a wave without and with a low pass filter applied. How much each frequency is attenuated depends on the characteristics of the filter. In addition, most low pass filters have the ability to use resonance to further emphasize the frequencies at and around the cut-off point [Rei99a]. Exactly which frequencies are emphasized and what happens to the rest of the frequencies vary depending on the exact characteristics of the filter. Thus different filters applied to a waveform can give very different sounds when resonance is applied. The same wave as before is shown with a low pass filter with resonance applied in figure 2.12.

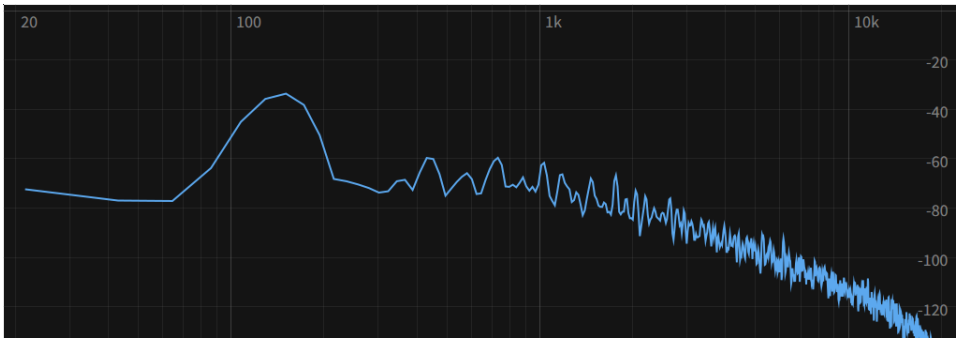


Figure 2.11: A wave after going through a low pass filter. Amplitude as a function of frequency. The highest frequencies have a lower amplitude than in figure 2.10.

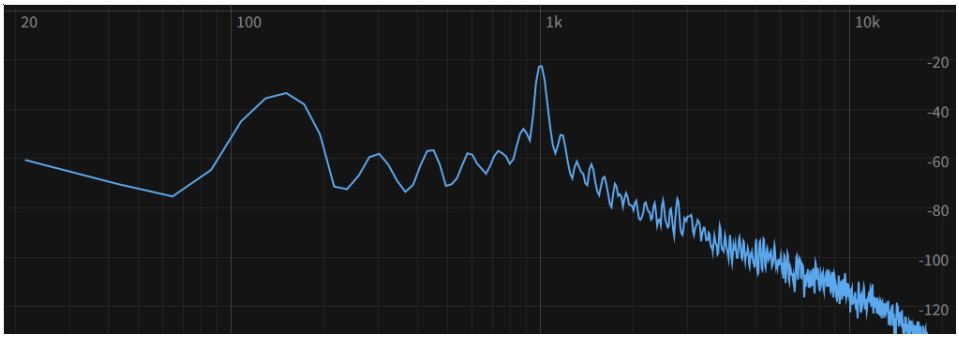


Figure 2.12: A wave after going through a low pass filter with resonance. Amplitude as a function of frequency. The resonance increases the amplitude at around 1kHz.

The waveforms and low pass filter presented in this section are examples of components commonly used in audio systems. The time and frequency spectrums shown are common ways of representing audio visually.

2.2.1 Csound

Csound is a sound and music computing system which generates audio based on files written with a specific syntax [Cso19]. In a Csound file, waveforms, various elements that change the sound in some way, like filters, and the path the signal takes through everything are specified. Figure 2.13 shows how some simple components from Csound can be put together to create sound. It is also possible to specify how different parameters should change at different times. In addition, there is an API which makes it possible to interact with Csound files from various programming languages. Csound can run in real-time, making it possible to affect the audio while it is being generated.

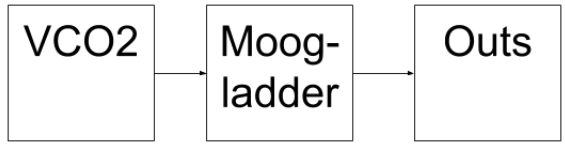


Figure 2.13: The path through some simple Csound components. VCO2 is a signal generator, moogladder is a low pass filter and outs outputs the resulting audio signal.

Chapter 3

Method and implementation

Random Boolean Networks are an interesting case to look at for use in controlling audio, as they are self-organizing, and thus can display ordered behavior even if they are randomly generated. This makes it possible to get somewhat predictable behavior, while still keeping it random. This can make it possible to generate sound which because of the ordered behavior can be somewhat controlled, while the randomness will make sure it is still nondeterministic. With audio, there will not be any dire consequences if unwanted behavior should occur. The generated sounds may not sound pleasing, but it will not have any lasting negative effects. Thus, using Random Boolean Networks to control audio exploits the controlled randomness the networks show in their behavior, while avoiding harmful consequences should the behavior go in unsuitable directions.

3.1 Random Boolean Network generation

A Random Boolean Network is represented as an array of nodes, where each node contains an index, the current value, its boolean function and information about the nodes which affect it. See the topmost box in figure 3.1. When a network is newly generated, each nodes starting value and all links between nodes are chosen using pseudo-random numbers generated based on the current time. The functions are also generated based on pseudo-random numbers, but they can be generated in a few different ways. One option only allows the OR and AND functions (or NOT and its inverse for K=1 networks). The other option allows any kind of boolean function, but with the possibility of specifying a bias which decides the probability that any one entry in the functions truth table should be a one rather than a zero (this represents the earlier mentioned P). The implementation only supports networks with up to and including 32 nodes, but more nodes could easily be supported using larger data types. In this case however, 32 nodes should be more than enough. There are 2^{2^K} possible boolean functions for each of the 32 nodes, which gives $2^{2^{K^{32}}}$ possible combinations of functions. 32^K possible combinations of links for each node, gives $32^{K^{32}}$ possible combinations of links between all

the nodes. $2^{2^{K^{32}}} \cdot 32^{K^{32}}$ possible networks lead to a very large search space. For example, 3614737867146518396094859318021923665089733007170019231594754471504248102862334079879518618873894396122749267837803515619997819988324340412961987953263291016231418997097876634332969052790660515486409420132908198868140687 36 or 2^{736} different networks exist for K=3. Each network also has 2^{32} unique states.

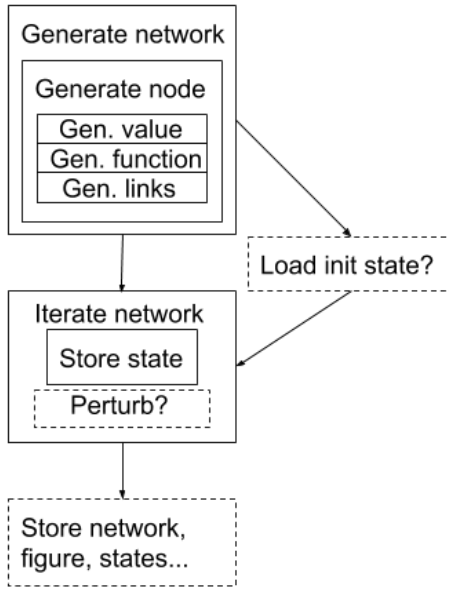


Figure 3.1: Generation of Random Boolean Networks.

A network is iterated by going through each node and calculating its new value according to its function. The new values for all the nodes are stored, and when all the nodes have been calculated, all the nodes are updated. It is also an option to perturb one or more random node(s) in a network, inverting its value, during an iteration of a network. See the middle box in figure 3.1. The states for all the iterations are stored to a file for further analysis.

Figure 3.2 shows a Random Boolean Network generated by the implementation using only AND and OR functions, with N=8 and K=3. Each node is labeled by its number, initial value and function. The network's states through ten iterations are shown in table 3.1. It can be seen that it reaches a point attractor after five iterations.

3.2 Automated analysis

The main program was written primarily with generating and iterating one network at a time in mind. Thus it proved simpler to write a script automating the execution of the program when the task was to run the same network many times. The main functionality

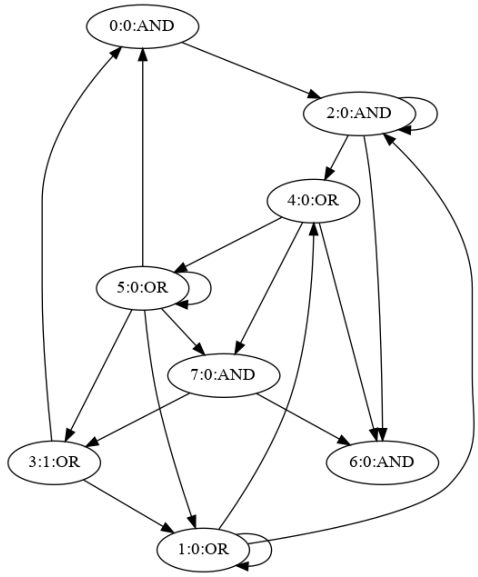


Figure 3.2: A generated Random Boolean Network with N=8 and K=3.

0	1	2	3	4	5	6	7
0	0	0	1	0	0	0	0
0	1	0	0	0	0	0	0
0	1	0	0	1	0	0	0
0	1	0	0	1	1	0	0
0	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1
1	1	0	1	1	1	0	1

Table 3.1: Trajectory for a generated Random Boolean Network.

of the script is shown in figure 3.3. The script simply calls the main program with the appropriate arguments and interprets the results. The script tells the program either to generate a new network or which existing network to use. It supplies the initial state for the network (the topmost box in figure 3.3), then it iterates the network (the middle box in figure 3.3) and it tells the program which node in the network to perturb, if any, in each iteration (the box on the right in figure 3.3). After the iterations are done, the script goes through the file where the states for the run are stored, and calculates the lengths of attractors and whether a network returned to the same attractor after perturbation (the bottom box in figure 3.3).

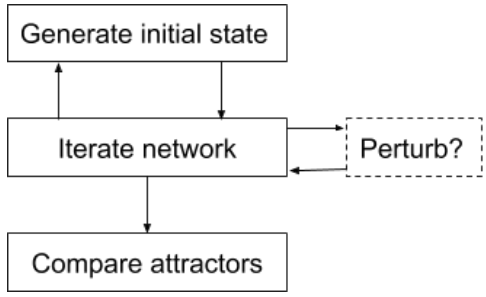


Figure 3.3: Analyzing network trajectories.

3.3 Audio generation

To actually generate audio, Csound was used in conjunction with the program. Various parameters like pitch, pulse-width modulation and low pass filter cutoff frequency are specified in a Csound file, and those parameters are accessed from the program. The program itself starts by loading the specified networks (the topmost box in figure 3.4). Then the initial states for those networks are set, often randomly (the next box in figure 3.4). After preparing the communication with Csound (the next box in figure 3.4), the program splits in to three threads. One is used for actually outputting the sound wave (the middle bottom box in figure 3.4), another is used to iterate the networks (the leftmost box in figure 3.4) and the last one waits for and handles user input (the two rightmost boxes in figure 3.4). The networks and their states are global, so all threads can access them.

The varying states of iterating networks are used to specify the values of the Csound parameters, that is, pitch, filter cutoff, filter resonance, and pulse width. The states are interpreted as values fitting the parameters in various ways: interpreting the collection of states as a binary number and take its value modulo an appropriate number, using the same number as the fractional part of a fraction when a number less than zero is required, and using the number of ones in a state as a value. When networks reach attractors, these parameters will cycle through the same values over time, or in the case of point attractors, the parameters will remain constant.

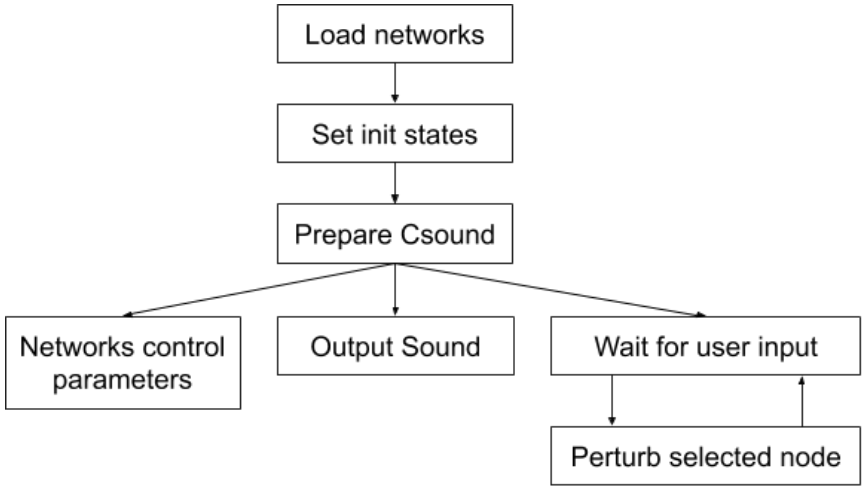


Figure 3.4: Generation of audio.

Chapter 4

Experiments and analysis

4.1 Networks and their initial states

To be able to look closer at the characteristics of specific networks, twelve networks were chosen, three each with K=1, K=2, K=3 and K=4. For each K, networks with P=0.25, P=0.50 and P=0.75 were chosen. The intention was to see how networks generated with different parameters behave differently, especially with respect to attractors. These networks were iterated with 3000 different random initial states each, where an equal amount of the initial states had 25%, 50% and 75% of the nodes values being one. For each initial state, the current network was iterated 100 times, and after half the iterations the network was perturbed by inverting the value of a random node. Then it was checked whether the network returned to the same attractor after the perturbation, or if it ended up in a different attractor. The fractions of runs where each network returns to the same attractor after perturbation were calculated for each network.

As mentioned, twelve specific networks were chosen, which the following results are based on. These networks may be outliers in their behavior, and thus these results are not general for all networks generated with the same parameters.

4.1.1 K=1

The fractions of runs where a network with K=1 returned to the same attractor after perturbation are shown in table 4.1. The top row shows the value of P for the generated networks while the leftmost column shows the fraction of nodes having a value of one in the initial state. With K as small as one, ordered behavior is expected. In this case, this is very true for two of the networks, the P=0.25 and P=0.5 networks. They always return to the same attractor after perturbation. The P=0.75 network behaves differently. This network seems to return to the same attractor more often than not when the initial state contains few values of one, and to return to the same attractor less often than not when the initial state contains few values of zero. When the amount of ones and zeroes are about the same, it returns to the same attractor about as often as not.

K=1	**0.25**	**0.50**	**0.75**
0.25	1.0	1.0	0.61
0.50	1.0	1.0	0.47
0.75	1.0	1.0	0.38

Table 4.1: K=1

Figure 4.1 shows the unbiased (P=0.5) network with K=1. Each node is labeled by its number and initial value. Figure 4.2 shows the trajectory for this network with a random initial state. The white squares represent a value of one, and the black squares represent a value of zero. One line represents one complete state for the network, with the far left square representing node 0 and the far right square representing node 31. The leftmost figure just shows the trajectory for 100 iterations, and it can be seen that an attractor is reached after few iterations. The rightmost figure shows the state time plot for the same network with perturbation of a single random node after half the iterations. It can be seen that in this case, the network returns to the same attractor right away.

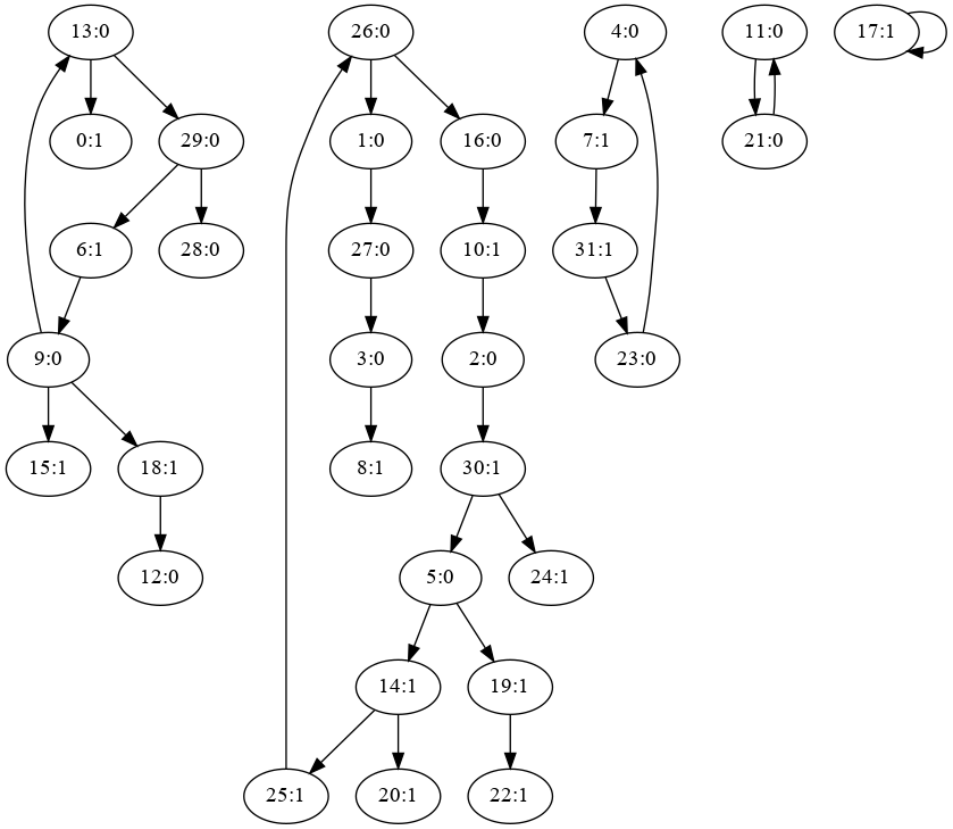


Figure 4.1: A generated Random Boolean Network with N=32 and K=1.

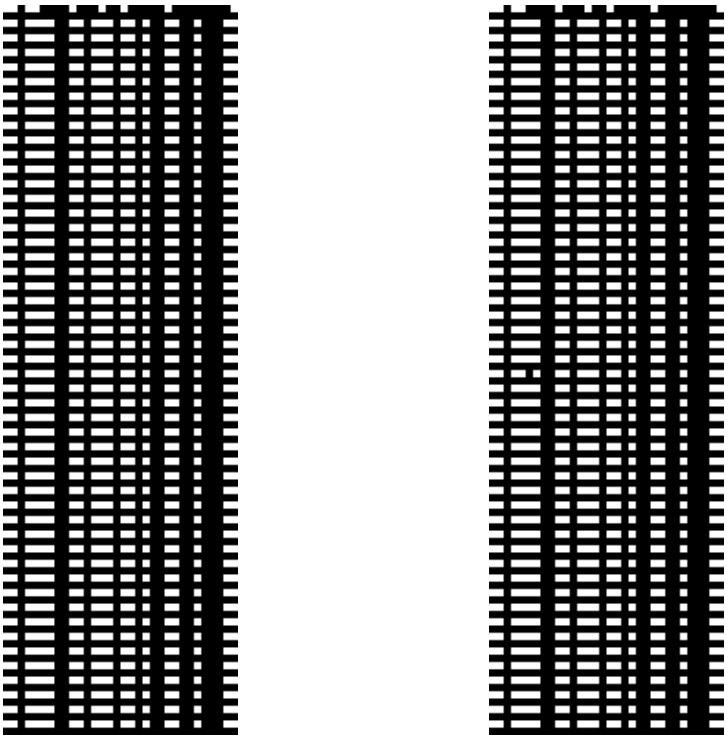


Figure 4.2: Trajectories for the unbiased K=1 network, the rightmost one being perturbed after half the iterations.

4.1.2 K=2

The fractions of runs where a network with K=2 returned to the same attractor after perturbation are shown in table 4.2. The top row shows the value of P for the generated networks while the leftmost column shows the fraction of nodes having a value of one in the initial state. K=2 is also a rather low value for K, so quite ordered behavior is still expected. This is especially the case with the P=0.25 and P=0.75 networks, which show extremely ordered behavior by always returning to the same attractor, that is, a strong attractor. The P=0.5 network is, as expected, not showing quite as ordered behavior as the other two; it returns to the same attractor about half the time.

K=2	**0.25**	**0.50**	**0.75**
0.25	1.0	0.57	1.0
0.50	1.0	0.51	1.0
0.75	1.0	0.53	1.0

Table 4.2: K=2

Figure 4.3 shows the unbiased (P=0.5) network with K=2. Each node is labeled by its number and initial value. Figure 4.4 shows the trajectory for this network with a random initial state. The white squares represent a value of one, and the black squares represent a value of zero, with the far left square representing node 0 and the far right square representing node 31. One line represents one complete state for the network. The leftmost figure just shows the trajectory for 100 iterations, and it can be seen that an attractor is reached after few iterations. The rightmost figure shows the state time plot for the same network with perturbation of a single random node after half the iterations. This results in the network reaching a different attractor after few iterations.

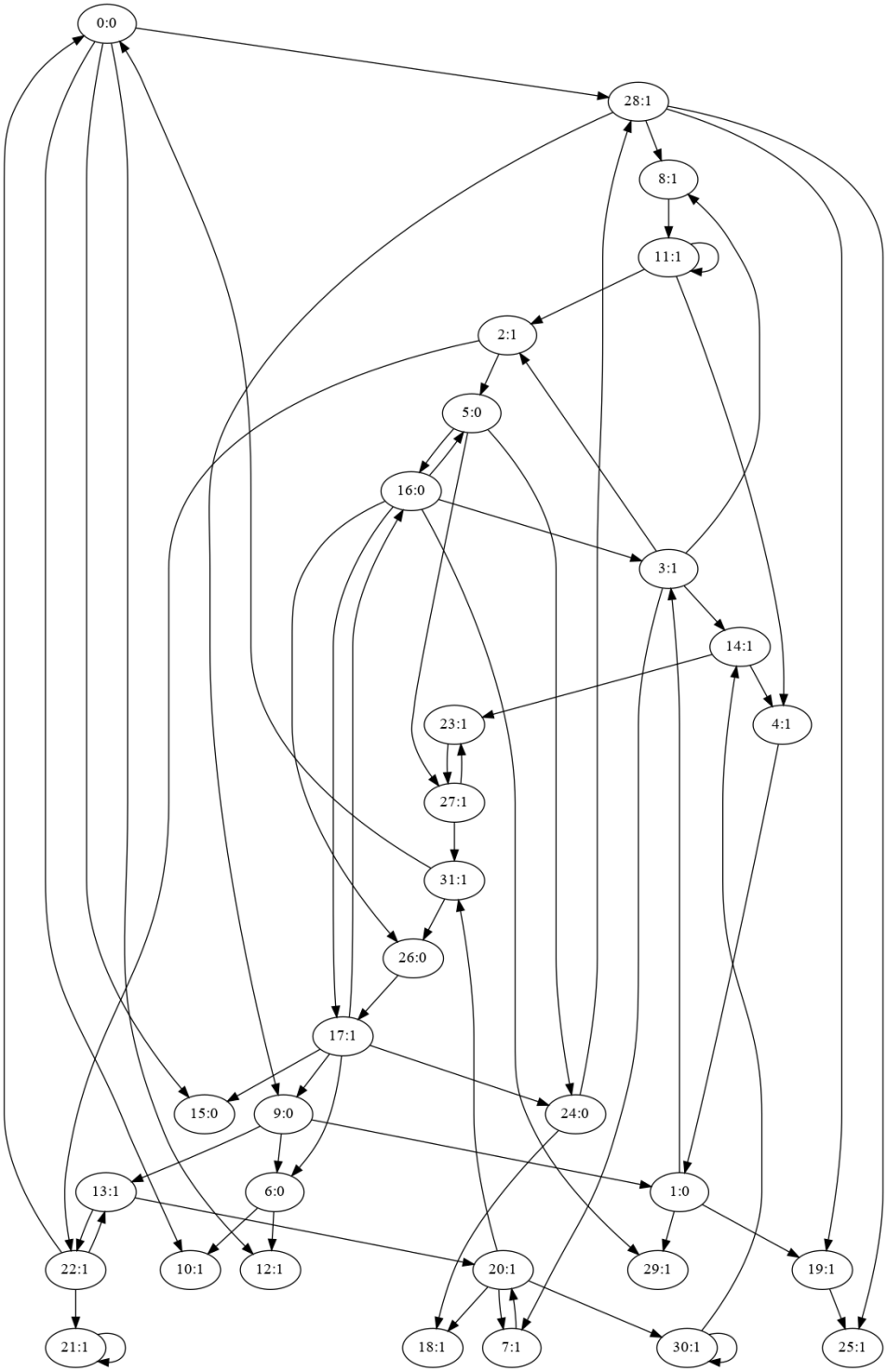


 Figure 4.3: A generated Random Boolean Network with N=32 and K=2.

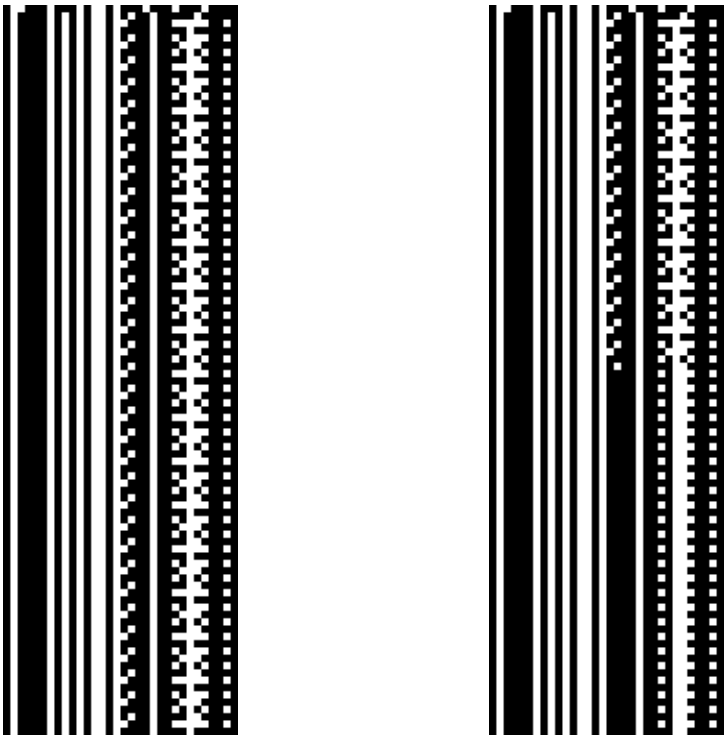


Figure 4.4: Trajectories for the unbiased K=2 network, the rightmost one being perturbed after half the iterations.

4.1.3 K=3

The fractions of runs where a network with K=3 returned to the same attractor after perturbation are shown in table 4.3. The top row shows the value of P for the generated networks while the leftmost column shows the fraction of nodes having a value of one in the initial state. With K=3, more chaotic behavior is expected. Here we get the exact opposite. All three of the networks return to the same attractor more often than not, and the P=0.25 and P=0.75 networks rarely do anything else. This indicates that all of these networks have strong attractors. The P=0.5 network shows somewhat less ordered behavior than the other two, which would be expected from the unbiased network.

K=3	**0.25**	**0.50**	**0.75**
0.25	0.96	0.81	0.91
0.50	0.97	0.81	0.94
0.75	0.95	0.80	0.93

Table 4.3: K=3

Figure 4.5 shows the unbiased (P=0.5) network with K=3. Each node is labeled by its number and initial value. Figure 4.6 shows the trajectory for this network with a random initial state. The white squares represent a value of one, and the black squares represent a value of zero, with the far left square representing node 0 and the far right square representing node 31. One line represents one complete state for the network. The leftmost figure just shows the trajectory for 100 iterations. This network does not reach an attractor within 100 iterations for this particular initial state. The rightmost figure shows the state time plot for the same network with perturbation of a single random node after half the iterations. It can be seen that the bottom half of the rightmost figure shows a different trajectory than the leftmost figure. This shows that this network is capable of quite chaotic behavior given the right initial state, even though it generally behaves quite orderly, as previously mentioned.

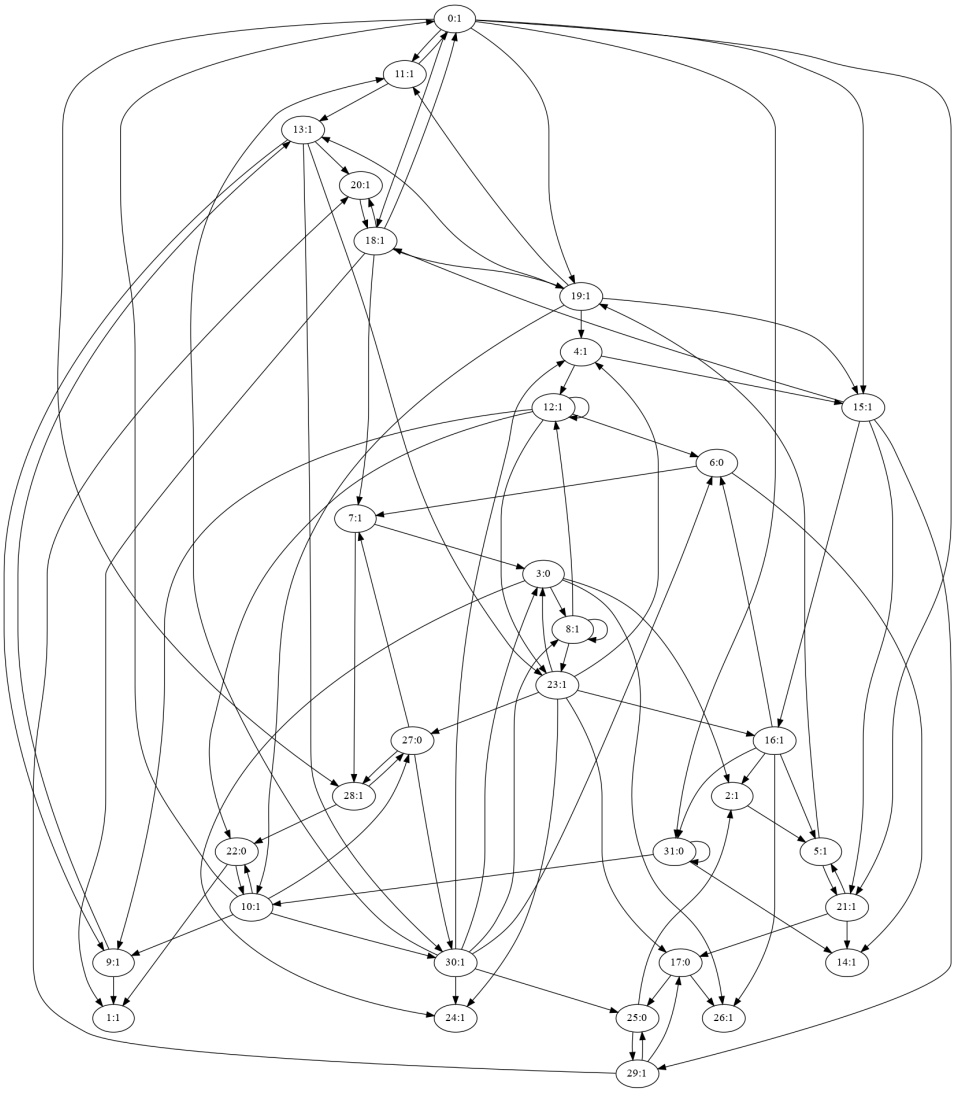


Figure 4.5: A generated Random Boolean Network with N=32 and K=3.

Figure 4.6: Trajectories for the unbiased K=3 network, the rightmost one being perturbed after half the iterations.

4.1.4 K=4

The fractions of runs where a network with K=4 returned to the same attractor after perturbation are shown in table 4.4. The top row shows the value of P for the generated networks while the leftmost column shows the fraction of nodes having a value of one in the initial state. With K=4, quite chaotic behavior is expected. The P=0.25 and P=0.75 networks behave as expected, by not returning to the same attractor very often. The P=0.5 network, on the other hand, behaves quite orderly, almost always returning to the same attractor. In general, the opposite would be expected.

K=4	**0.25**	**0.50**	**0.75**
0.25	0.19	1.0	0.14
0.50	0.16	1.0	0.18
0.75	0.19	1.0	0.19

Table 4.4: K=4

Figure 4.7 shows the unbiased (P=0.5) network with K=4. Each node is labeled by its number and initial value. Figure 4.8 shows the trajectory for this network with a random initial state. The white squares represent a value of one, and the black squares represent a value of zero, with the far left square representing node 0 and the far right square representing node 31. One line represents one complete state for the network. The leftmost figure just shows the trajectory for 100 iterations, and it can be seen that an attractor is reached, though it is a rather long attractor. The rightmost figure shows the state time plot for the same network with perturbation of a single random node after half the iterations. This results in the network reaching a completely different attractor which is much shorter and thus shows much more ordered behavior. As for the unbiased (P=0.5) network with K=3, this network can end up in a different, and much more ordered, attractor after perturbation, given the right initial state, even though it almost always returns to the same attractor.

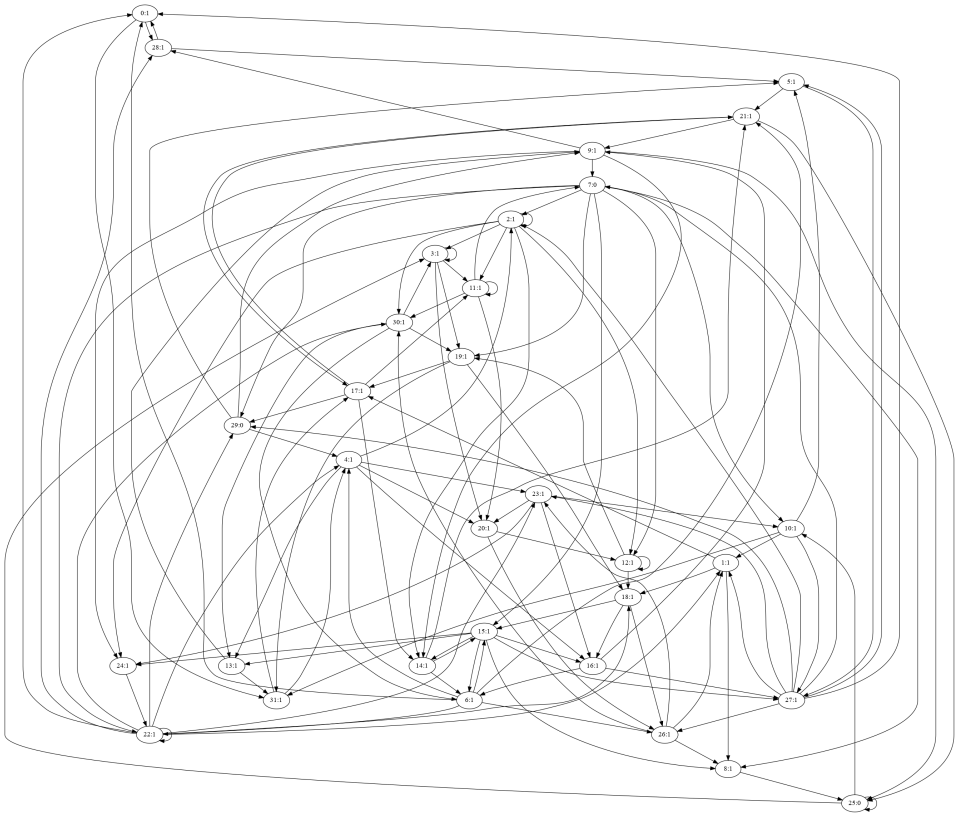


Figure 4.7: A generated Random Boolean Network with N=32 and K=4.

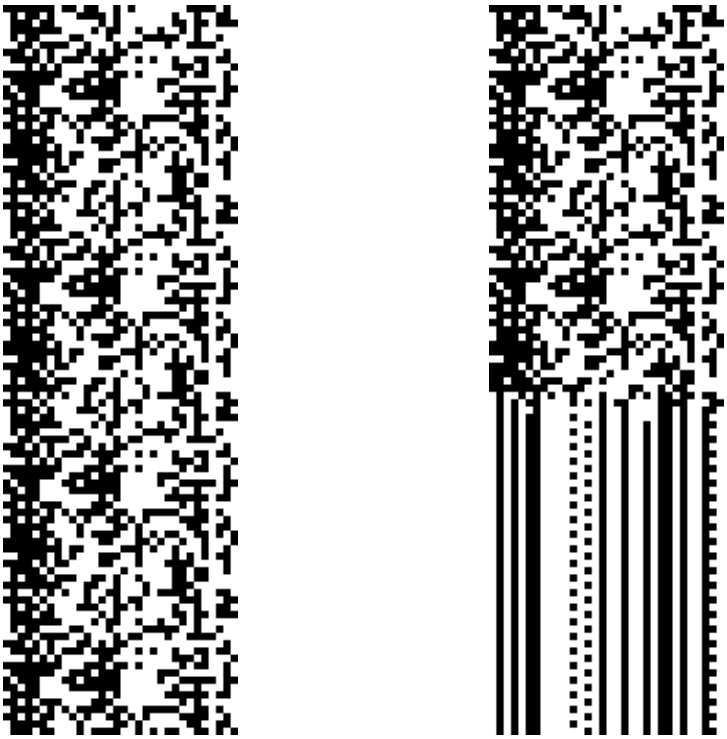


Figure 4.8: Trajectories for the unbiased K=4 network, the rightmost one being perturbed after half the iterations.

4.2 Audio generation

As described in the previous chapter, the chosen Random Boolean Networks were used to control various parameters of generated audio. The initial states of the networks were chosen randomly. Resulting sound files can be found at `folk.ntnu.no/katrinro/rbnsounds`.

4.2.1 Pitch

The network with K=2 and P=0.5 was used to control the pitch of a generated sound based on a saw wave. This network was chosen as it varies whether it returns to the same attractor or not, thus making it likely that the same pitches will not be repeated over and over if the network is perturbed. The number of nodes with a value of one were used to choose between 32 pre-defined pitches at every iteration. The result can be heard in the file 421pitch.wav. Figure 4.9 shows how the sound wave looks, and figure 4.10 shows a short slice of the sound wave, where the actual shape can be seen. It can be seen after that after six steps an attractor is reached, and it goes back and forth between the two same pitches. The file 421pitchperturb.wav and figure 4.11 show the same thing happening, but this time the network is perturbed after being in the attractor for a while, and it can be seen that it reaches a different attractor after that, which is also two steps long. Figure 4.12 shows a short slice of the same sound wave, where the actual shape can be seen.

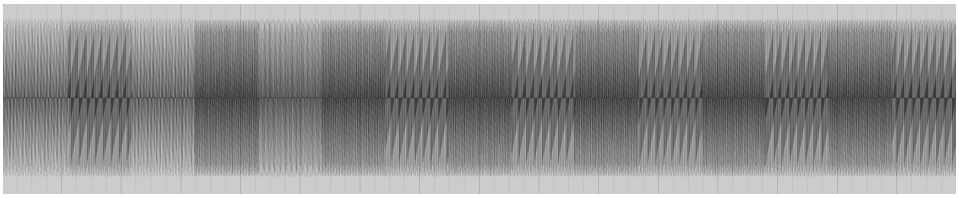


Figure 4.9: Saw wave with changing pitch.

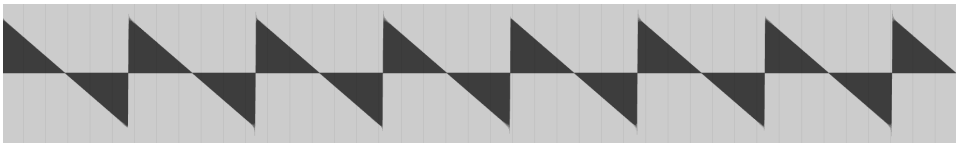


Figure 4.10: A short slice of the saw wave with changing pitch.

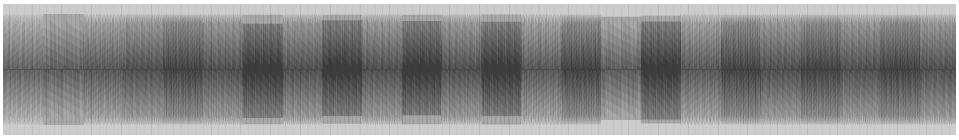


Figure 4.11: Saw wave with changing pitch which is then perturbed.

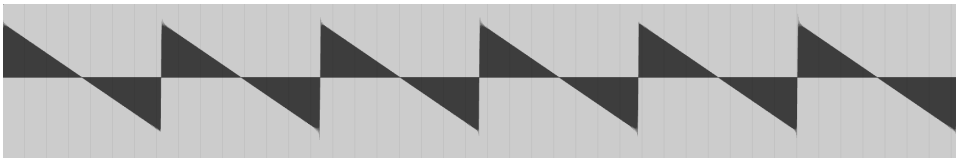


Figure 4.12: A short slice of the saw wave with changing pitch which is then perturbed.

4.2.2 Low pass filter

The network with K=3 and P=0.5 was used to control the frequency of a low pass filter. Since this network quite often returns to the same attractor, it is likely that the cutoff frequency of the low pass filter mostly will cycle between the same values. The state of the network is read as a binary number. Modulo 16000 of this number is then used as the cut-off frequency of the low pass filter, as higher frequencies are mostly outside the spectrum of human hearing and thus cutoff frequencies higher than this would not make a difference to what is heard. Also to keep the sound inside the range of human hearing, 200 is added to the cutoff frequency. Every time the network iterates, a step value is calculated, such that by changing the cut-ff frequency with the step value every microsecond, the calculated cutoff frequency will be reached when it is time for the next iteration of the network. This gradual change of the cut-off frequency can be heard in 422lpfilter.wav and is shown in figure 4.13. Figure 4.14 shows a short slice of the audio wave shown in figure 4.13, where the actual shape can be seen.

Then, the K=4 and P=0.5 network was used to control the resonance of the same low pass filter. As this network almost always returns to the same attractor, the resonance likely will continue to cycle between the same values, even if the network is perturbed. To get a value between zero and one for the resonance, the state of the network was interpreted as a binary number, which was then used as the fractional part of a number between zero and one. The result of applying resonance to the low pass filter can be heard in 422lpreso.wav. Figure 4.15 shows how the sound wave looks, and 4.16 shows a short slice of the same wave, where the actual shape can be seen.

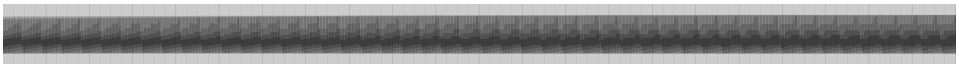


Figure 4.13: Saw wave affected by a low pass filter.

Figure 4.14: A short slice of the saw wave affected by a low pass filter.

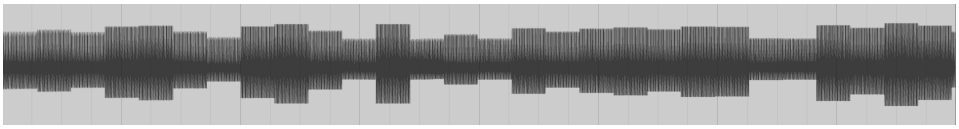


Figure 4.15: Saw wave affected by a low pass filter with changing resonance.

Figure 4.16: A short slice of the saw wave affected by a low pass filter with changing resonance.

4.2.3 Pulse width

The network with K=4 and P=0.25 was used to control the pulse width of a pulse wave. This quite chaotic behaving network was chosen so that the pulse width may behave in many different ways after being perturbed. This was changed gradually in the same way as the cut-off frequency of the low pass filter. The result of this can be heard in 423pwm.wav and figure 4.17 shows how the pulse width modulated sound wave looks. Figure 4.18 shows a short slice of the same sound wave, where the actual shape can be seen.

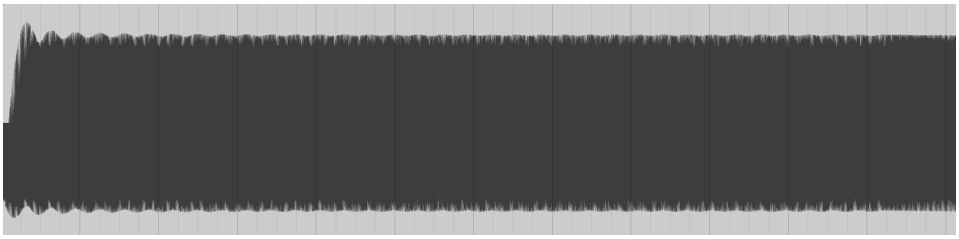


Figure 4.17: A pulse wave with changing pulse width.

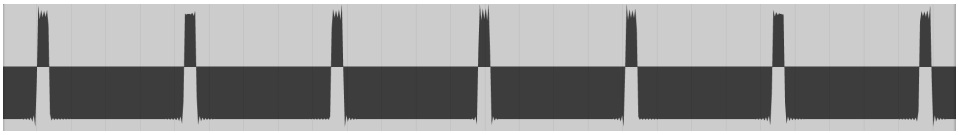


Figure 4.18: A short slice of the pulse wave with changing pulse width.

4.2.4 Combining the different parameters

The rest of the networks were used for deciding how often the networks controlling the parameters should iterate, and which networks should be perturbed when. Using both the saw wave and the pulse width modulated pulse wave with different pitches at the same time and applying the low pass filter on top gives a quite complex result which may sound somewhat musical. One example of how this ended up sounding can be heard in

everything.wav. The sound wave is shown in figure 4.19, and figure 4.20 shows a short slice of the same sound wave, where the actual shape can be seen.

Figure 4.21 illustrates which networks affect what and the flow of the sound signal through the application. The green arrows represent the sound signal, the blue arrows represent networks being told to iterate, the red arrows represent perturbations and the black arrows show which sound parameters are affected.

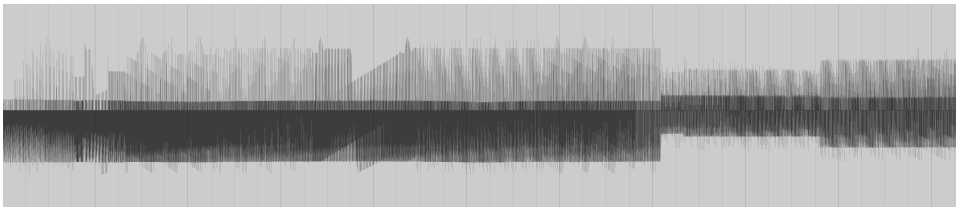


Figure 4.19: All the above-stated concepts combined.

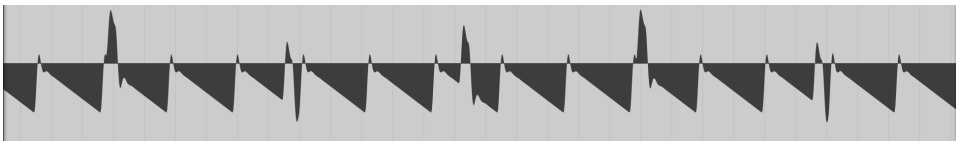


Figure 4.20: A short slice of the audio wave with all the above-stated concepts combined.

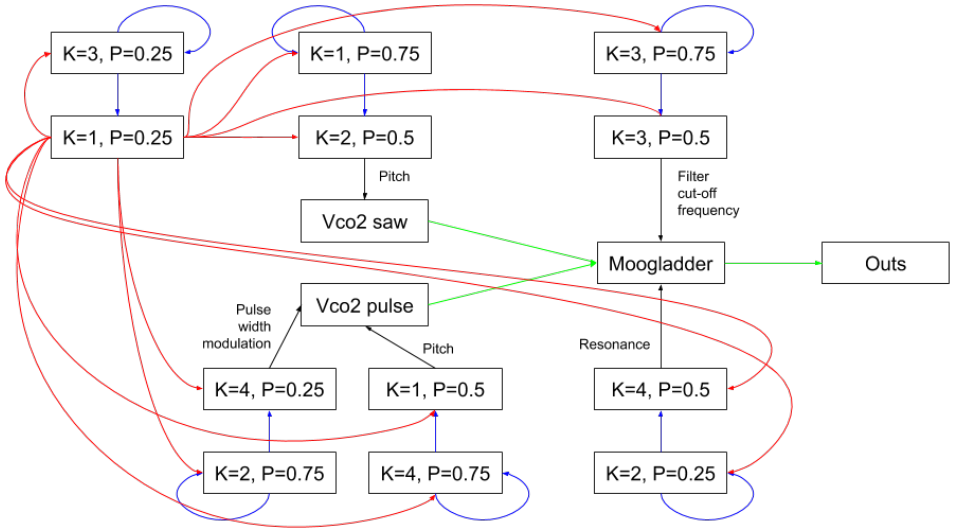


Figure 4.21: Which networks affect what while generating the audio. Green arrows represent the sound signal, blue arrows represent networks being told to iterate, red arrows represent perturbation and black arrows represent sound parameters being affected.

Chapter 5

Conclusion

Random Boolean Networks and their properties have been explored. Twelve networks with different parameters were chosen, and their traits were looked at more in depth. These networks were then used to control parameters of sound in audio generation. Networks with different characters of behavior were chosen for the different parameters, so the parameters would exhibit different behavior based on the underlying networks. This is interesting, as the resulting audio will be different every time, while still sounding similar, as long as the same networks are used to control the same parameters. It is possible to generate new types of audio by changing which parameters are controlled by the networks, or by generating new networks. All the networks used in this case were between K=1 and K=4, and using networks with higher K, while likely more chaotic, might produce unforeseen and interesting effects. A balance between higher K and different values for P might also produce networks of a suitable amount of order. Also, as previously mentioned, the search space for networks with each K is very large, so just using different networks of the same K may produce interesting effects which have not been seen with the networks used in this case. In music created by humans, there is usually some amount of repetition, which may give a feeling of familiarity, while there is also enough variety that the music keeps being interesting over time. Using networks which exhibit behavior at different points along the scale between order and chaos, might make it possible to mimic this combination of repetition and variety.

An application which integrates the use of Random Boolean Networks with audio generated by Csound has been developed. This application makes it possible to generate Random Boolean Networks or load previously generated networks. It includes mappings between the states of networks and Csound parameters. It is readily possible to expand the application to control more or other parameters than those mentioned previously.

By using methods based on complex systems, a creative process can be partly controlled. Combining different types of systems based on the desired behavior, automatic generation of audio, or other applications that can manage nondeterministic behavior, can be achieved.

Bibliography

[Bar16] A. Barabasi. *Network Science*. Cambridge University Press, 2016.

[BBC03] R. Behravan, P. J. Bentley, and R. Carlisle. Exploring reaction-diffusioon and pattern formation. In *First Australian Conference on Artificial Life*, London, England, 2003.

[BY99] Y. Bar-Yam. *Dynamics of Complex Systems*. CRC Press, 1999.

[Cso19] Csound. Csound. `https://www.csound.com`, 2019.

[Ger03] C. Gershenson. Artificial neural networks for beginners. 2003.

[Ger04] C. Gershenson. Introduction to random boolean networks. 2004.

[Kau93] S. A. Kauffman. *The Origins of Order*. Oxford University Press, 1993.

[Lan90] C. G. Langton. Computation at the edge of chaos: Phase transitions and emergent computation. *Physica D*, 42(1):12–37, 1990.

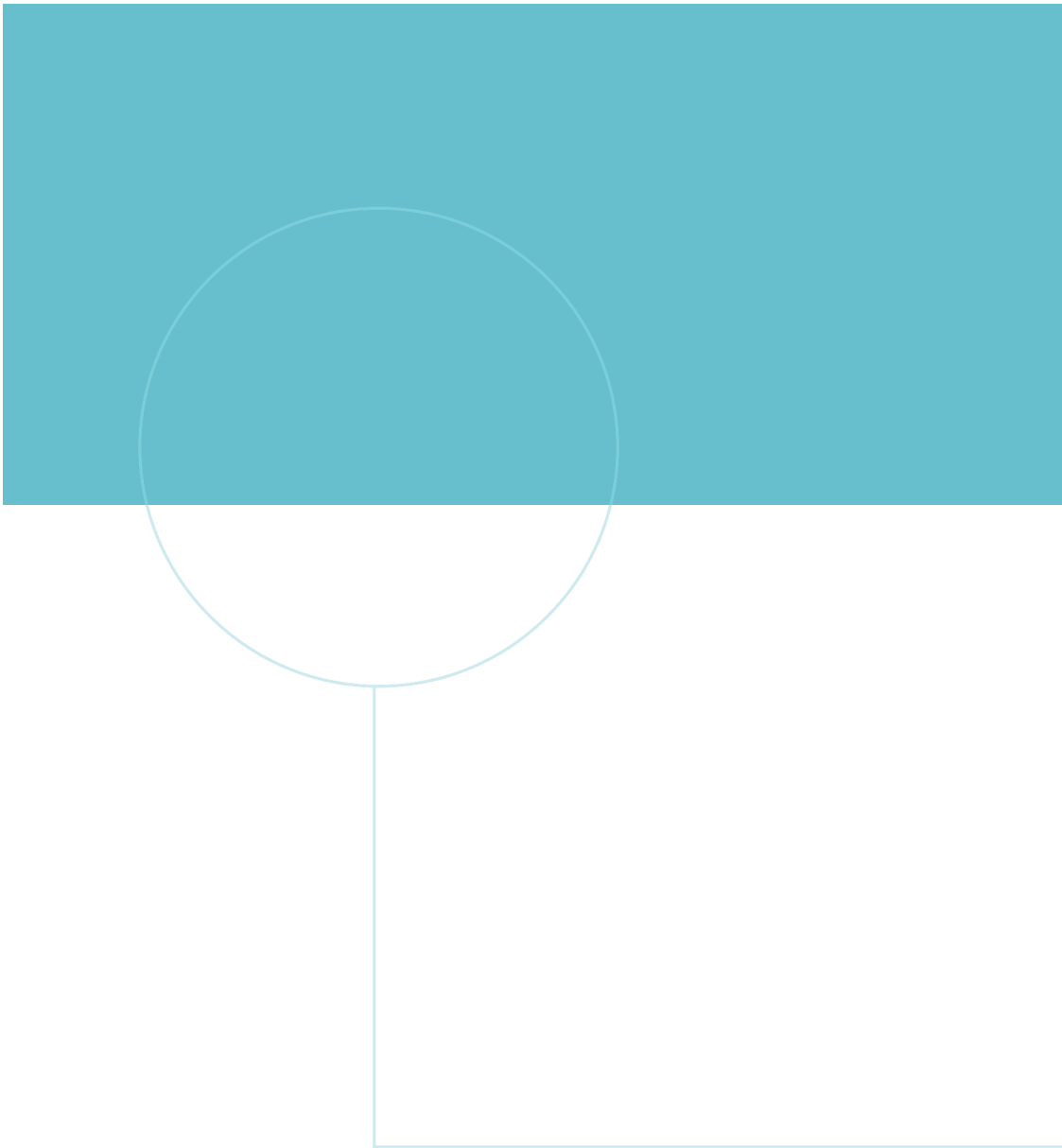
[Mit09] M. Mitchell. *Complexity*. Oxford University Press, 2009.

[Rei99a] G. Reid. Of responses & resonance. `https://www.soundonsound.com/techniques/responses-resonance`, 1999.

[Rei99b] G. Reid. Synthesizing strings: String machines. `https://www.soundonsound.com/techniques/synthesizing-strings-string-machines`, 1999.

[Rey87] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[Wol02] S. Wolfram. *A New Kind of Science*. Wolfram Media, 2002.



NTNU
Norwegian University of
Science and Technology