TU
Technische Universität Wien

M A G I S T E R A R B E I T

Algorithmic Composition

ausgeführt am
Institut für Gestaltungs- und Wirkungsforschung - Multidisciplinary Design Group
der Technischen Universität Wien

unter Anleitung von
ao. Univ.-Prof. Dr. Gerald Steinhardt, Dipl.-Art. Thomas Gorbach, PD Dipl.-Ing. Dr. Hilda Tellioglu

durch

Daniel Aschauer Bakk.techn.
Stättermeyergasse 34/8
1150 Wien

Wien, Februar 2008 _____

Zusammenfassung

Diese Magisterarbeit befasst sich mit dem Einsatz von algorithmischen Methoden und Computern zur Erzeugung von Kompositionen. Eine Auseinandersetzung mit der geschichtlichen Entwicklung zeigt, wie Methode und Zufall schon vor Jahrhunderten eingesetzt wurden und wie technische Neuerungen und theoretische Ansätze den Kompositionsprozess revolutionierten. Ein wichtiger Arbeitsschritt in der Computerkomposition beinhaltet die Umsetzung der Algorithmen und Konzepte zu einem konkreten hörbaren Ergebnis. Verschiedenste Techniken, von einfachen Zufallsverteilungen bis zu komplexeren Systemen - wie genetischen Algorithmen - werden erklärt und deren Potential verdeutlicht. Praktisch wurden einige dieser Techniken implementiert und damit eine Komposition erstellt.

Abstract

This thesis deals with the use of computers and algorithmic methods for the production and composition of music pieces.
An examination of the historical development shows how change and method were applied already centuries ago and how technical innovations and theoretical approaches revolutionized the composition process. An important step in computer composition includes the translation of the algorithms and concepts into a concrete audible result. Various techniques from simple random distribution to complex system - such as genetic algorithms - are explained and their potential pointed out.
Finally, some of these techniques were implemented and used to create a musical composition.

Contents

Chapter 1

Introduction

Composing music is a process that requires creativity and imagination. As algorithms and computer techniques are primarily associated with determination, their relevance for such a creative task maybe may not be easily seen at first. However, the task of this thesis is to dissolve this ostensible contradiction and to suggest how algorithms can be applied to musical creation.

To the composer algorithmic composition means to work on a meta-level, as he doesn't compose music by e.g. describing a score setting note after note, but rather designs a system composed of rules and algorithms. He then conducts the concrete piece of music by deciding on the values of the parameters used in the system. One principal potential of algorithmic techniques lies in this ability of meta-level composition. Implemented in a computer in this way it is possible to explore lots of potential compositions by applying slight changes to the global parameters. An artist's work can also be seen as a task of exploration (for aesthetic), algorithmic composition enriches this exploration adventure with a powerful tool.

Another potential of the usage of computer and algorithms lies in its combination with sound synthesis techniques. Doing so enables a way of composing, working with the sound itself on a microscopic level. When the parameters of sound synthesis systems are that way, the evolution of the (synthesized) sound itself forms part of

the composition. Therefore sound synthesis is an important related issue, including a width range of digital synthesis techniques (e.g. *granular synthesis*, *FFT*). As the present paper does not cover a description on sound synthesis, interested readers may refer to [Roa04].

This thesis starts with a historical view in chapter 2, which shows that the use of formal composition methods and specially designed machines for composition looks back at a long and so was not any new invention of the twentieth century and the appearance of modern computer science. However computers enabled pioneer composers to apply formal composition techniques more effectively, and moreover provided them with plenty of new possibilities.

New algorithmic methods appeared which often were derived from other fields of scientific study. Chapter 4 points out selected techniques that have already found appliance in the field of music. Background and functional principle are explained and potential usefulness for composition purposes is analysed.

Chapter 3 discusses issues about different composition structures and the usage of parameters in the music creation process.

This thesis includes own implementations of some introduced techniques, which are described in detail in chapter 5 and then used within a composition by the author. The composition is explained in chapter 6 and can be found on the supplied CD.

Chapter 2

History of Algorithmic Composition

Recent approaches to algorithmic composition propose innovative methods, possible through the calculation power of modern computers. However various fundamental techniques and basic aesthetics were already suggested centuries before. This chapter briefly sets the historical background of algorithmic composition, explaining formal processes in music used for years, development of machinery, theoretic background and the pioneer approach in computer music.

2.1 Formal processes

Using formal processes for the composition of music has always attracted composers. In the early 11^{th} century Guido d'Arrezo developed a technique that assigned a pitch to each vowel in a given text and, thus, created a melody accompanying the text [Roa96].

In the 15^{th} century Guillaume Dufay followed a quite common approach, using number ratio to derive musical parameters. Some of his works are inspired by the proportions of a Florentine cathedral or the ratio known as the "golden selection" (the ratio of 1:1.618). Furthermore Dufay anticipated techniques that centuries later should form part of the "serial music" by applying procedures like inversion and ret-

rograde to sequences of tones.

Another common formal technique shows up in the isorhythmic motets, a form of motet of the Medieval and early Renaissance (14^{th} and 15^{th} century). An isorhytmic motet is based in the periodic repetition or recurrence of rhythmic patterns in one or more voices. Works of the French composer Guillaume de Machaut typify this type of composition, Dufay and Vitchy composed isorhythmic motets, too.

In many other traditional music styles like fugues, rounds and canons composition follows strict rules, which makes part of the composer's work a formalizable process.

Maybe the most famous example of an early algorithmic composition is W.A. Mozart's "Musikalisches Würfelspiel" (musical dice game), a method Mozart used to compose minuets by arranging prewritten melodies. In his technique the result of thrown dices determines the arrangement of the pieces in the musical composition. In doing so Mozart introduced elements of chance and randomness, long before the invention of computers. Furthermore his role as composer changed as he composed by selecting some base material and invented a technique for it's arrangement. Thus he worked on an abstract level and created a "metacompostion".

Mozart's approach was commercialised later in a set of playing cards that was sold in 1822, facilitating the composition of up to 214 million waltzes - as indicated by the advertisement of the game [Roa96].

2.2 Composition Machines

Thus, various formal techniques exist already for centuries. However in order to apply automated processes in a more advanced manner a proper machinery is needed. Numerous machines applicable for composition purposes were invented and opened new ways in music composition and reproduction.

In 1660 Athanasius Kirchner developed his "Arca Musirithmica", a machine that among other things could generate aleatoric compositions.

Another invention was a room-sized machine called "Componium" and built by Dietrich Winkel in 1821. It was able to produce variations on themes that were

programmed on it [Roa96].
In the early 19^{th} century the German inventor Johann Maelzel, commonly known as the inventor of the metronome, constructed the "Panharmonicon", an orchestra of forty-two robot musicians. Beethoven's work "Wellington's Victory" was composed for it [Cha97].
The mentioned examples demonstrate early attempts for machinery based composition. Afterwards in the 20^{th} century electronic inventions abounded. During the 1950s the electronic engineers Harry Olsen and Hebert Belar not only invented the RCA Synthesiser, but moreover an electromechanical composition machine based on probability [Roa96].
However, more than any specific composition automata, new electronic music and studio devices like the tape recorder and the (sequence-controlled) synthesiser promoted renewed composition experience by providing the composer with posibilities of recording and playback. Equipped with these new tools experimental approaches emerged that made use of these techniques in their composition. So the "musique concrte" movement employed magnetic tapes for composition and sound manipulation. In another example John Cage used variable-speed turntables for his "Imaginary Landscape No. 1" (1939).

2.3 Theoretic Background - Aesthetics

2.3.1 Serial music

While technical improvement prepared the ground for automated composition, new trends emerging from the musical world have to be seen as a huge inspiration for works in algorithmic composition.
With twelve-tone music and serialism, theories emerged that built the framework for systematic composition and strictly formalized music. One variant of twelve-tone technique was devised by the Austrian composer Josef Matthias Hauer in 1919. Hauer's tonal system is based on the idea of so called *tropes*. A *trope* consists in two complementary unordered hexachords. These two hexachords serve as building-

blocks for the composition [Hau25]. Compared to Schöenberg's later published theory of twelve-tone music his work stayed mainly insignificant.

Arnold Schönberg independently developed his twelve-tone technique in 1923 to a model that became known as *dodecaphony*. In contrast to his adversary Hauer, the order of the notes is of great importance in Schönberg's approach. Schönberg proposed that a composer should treat the twelve musical notes equally, meaning that they all have the same importance. This idea marked a break with traditional music theories that were based on tonal musical scales where certain notes were considered more important than others. He abolished the idea of using a particular tonal system as a basis for the composition and replaced it with a system based on a tone row, an arrangement made up by the twelve different notes in a certain order. Repetition of a single note is not permitted and every note has to be used. These basic rules ensure the absence of any kind of tonal centre. A basic tone row, or set, serves as starting point of the composition, from where it can be modified by some defined transformations. Permitted transformations include transposition, inversion (inverting the intervals of the series), retrogression (playing the series backwards), and retrograde inversion [Sin98].

Serialism is an extension to twelve-tone music and was introduced by the composers Anton Webern, Olivier Messian and others. They generalised the serial technique to cover other parameters, too, such as duration, dynamics, register, timbre and more. The rows that control the musical parameters do not have to be the same, so different rows and transformations may be used to derive the values of pitches, duration and dynamics [DJ85]. An important piece of work is Pierre Boulez's "Structures", an entirely serial composition for two pianos. For writing the piece Boulez defined a pitch, a duration, a dynamics and a modes of attack series and then applied serial processes that ensured that no pitch coincided with a duration, a dynamics or a mode of attack more than once [Mir01].

2.3.2 Aleatoric Music

In contrast to the very strict framework and extreme determinism of serial composition in the opposed movement of *Aleatoric Music* composition rules slightly disappear and elements of chance play an important role. The notion of *aleatoric music* emerged and was strongly coined by the American composer John Cage, although a few precedent applications can be found. For example, Marcel Duchamp invented a system made up by a vase (with a funnel), a toy train, and various balls each with number representing a note printed on it. A ball then had to be thrown in a vase from where it fell in a car of the train, which would pass it to the musician, who then played the indicated note [DJ85].
In aleatoric music certain details of the composition are left to chance or open to the interpreter. The term aleatoric derives from the Latin word *alea*, which means dice. In many cases John Cage arranged his pieces according to the results of a thrown dice or coin, or by playing old divination games. Along with Cage other composers such as Karlheinz Stockhausen, Pierre Boulez and Earle Brown employed aleatoric methods and elements of randomness in their works [Roa96].

Although serialism and aleatoric music describe two opposite ideas, both mark important trends in post-war music history, as they led to different new ways of thinking about music, breaking up with the traditional idea of tonality. Their ideas strongly influenced algorithmic composition and are reflected in applied methods and techniques.

2.4 Computer pioneers

Although algorithms and formal processes were used for composition purposes already before the 20th century, it was the computer that brought a new research field and new possibilities for the composers interested in automated composition.

2.4.1 Lejaren Hiller

One of the pioneers was Lejaren Hiller who started to use a computer for music composition in 1955. Hiller's point of view on composition was influenced by the concept of *information theory* as it was formulated and coined by Shannon in the field of communication around 1950. According to Shannon's ideas a message could be valuated by the meaning of its information content: the measure of content increases as the message's content decreases in predictability. Applying this theory to the field of composition, the piece of music takes the place of the messages and is therefore seen as rich in information as redundancy and degree of predictability is minimal. Randomness thus represents the highest grade of information content. Hiller's approach is based on the idea of a "musical message" generated by random processes. However the message is then filtered by a set of rules derived from a particular musical style, so that the selected values are more predictable than the random values. He describes this process as a message's movement from chaos to order, from indeterminacy to predictability [Har95].

Working on an Illiac computer at the University of Illinois in Urbana, he and Leonard Isaacson carried out a set of experiments in algorithmic score generation. Four of their experiments formed the four movements of "The Illiac Suite for String Quartet", which was the first piece of music created with computer-based algorithmic composition methods when it was published in 1957. In his work Hiller mainly borrowed traditional music forms to describe the rules of a *generate and test* procedure [Sup01]. In the first of the suite's movements ("Monody, Two-Part, and Four-Part Writing") he explicitly formulated sixteen different rules in three categories that described the events that are allowable, forbidden or required. For example it was not allowed for pitches to form tritons, and the starting pitch of a melody had to be the middle-C. Then a large amount of random numbers was created and tested against the given criteria. Accepted numbers - numbers that fitted in the described form - were added to the outputted note list [Cha97].
The second experiment ("Four-Part First-Species Counterpart") was an extension of

the first allowing the generation of different musical styles. In the third movement ("Experimental Music") Hiller applied serial techniques and structures. "Markov Chain Music", the fourth part of the suite, used stochastic methods like *Markov chains*, which strongly influenced and inspired later works in algorithmic composition [Sup01]. The program's output was a list of numbers which had to be converted by hand to musical notation to be readable by an interpreter.

Together with Robert Baker, Hiller also developed the utility MUSICOMP (Music Simulator Interpreter for Compositional Procedures). MUSICOMP formed a library including numerous subroutines that provided the user with useful functions for algorithmic composition. Among others the programmer may apply a selection based on a probability distribution to a list of items, shuffle them by random or enforce melodic rules [Roa96]. These subroutines and other programming modules have to be linked together forming the composer's individual program. The system's output can be printed out or used as input for sound synthesis programs. MUSICOMP was used in various later compositions by Hiller himself like in his "Algorithm 1" (1969), as well as by other composers, first of all Herbert Brün who composed "Sonoriferous Loops"(1964) and "Non Sequitur VI" (1966). Hiller's further works include the music piece "HPSCHD" (HarPSiCHorD), composed in collaboration with John Cage and finished in 1969. For this composition he wrote various procedures, among them a procedure called ICHING that chooses random number in the range of 1 to 64 in accordance with the rules in the I Ching. Another procedure, named "Dicegame", was inspired by Mozart's musical dice game [Cha97].

2.4.2 Iannis Xenakis - Stochastic music program

Along with Hiller, the Paris-based composer Iannis Xenakis provided one of the pioneering works in algorithmic composition. Already in 1955 he finished his composition "Metastasis" for orchestra, in which he used stochastic formulas but worked out the music piece by hand. In "Achorripsis" (1957) the occurrences of musical elements like timbre, pitch, loudness and duration were arranged on the basis of the

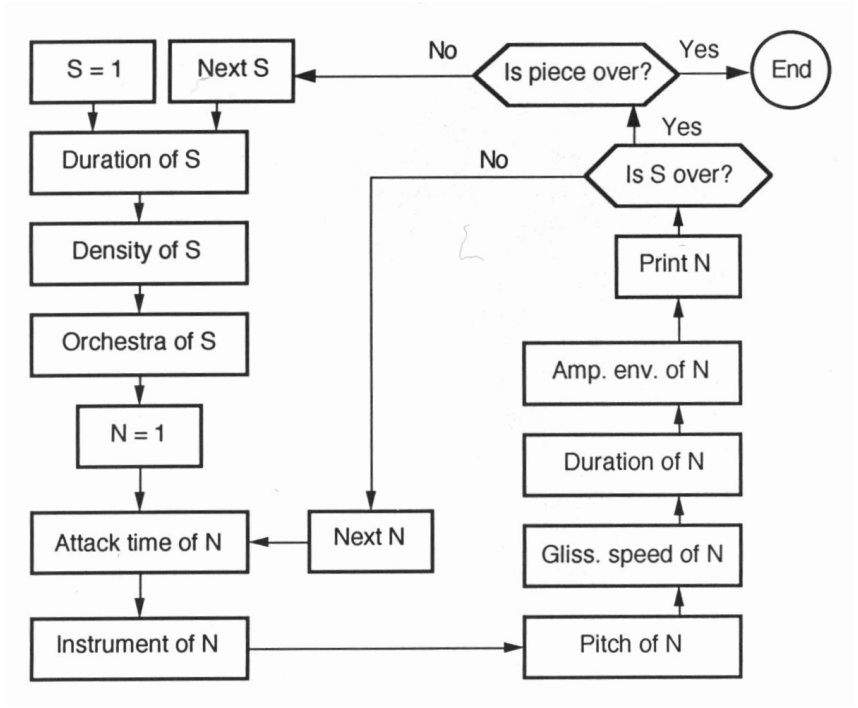


Figure 2.1: Stochastic music program (S=Section N=Note in section)

Poisson distribution. In 1961 he started working with computers as he moved to IBM-France. The following year he had his new computer-generated compositions "ST/48-1,240162", "Amorsima-Morsima", and "Atres"performed by the "Ensemble de Musique Contemporaine de Paris" [Cha97]. Xenakis coined the notion of stochastic music that he described in his book "Formalized Music" [Xen92]. His approach is neither completely based on randomness nor deterministic. He proposes the use of statistical principles for composing.

"Formalized Music" also contains a former version of SMP as "Free Stochastic Music program". In 1962 he applied his ideas in SMP (Stochastic Music Program), a piece of software that can be used for composition. SMP works with stochastic formulas that derive from the scientific field, as they were used originally to describe the behaviour of particles in gases. In Xenaki's metaphor the gas particles correspond with the individual notes in a composition that is formed by a sequence of clouds of sound. SMP creates a musical composition as a sequence of sections, each different in duration and density. The composer working with SMP has to agree on the global attributes in interaction with the program:

- Average duration of sections
- Minimum and maximum density of notes in a section
- Classification of instruments in timbre classes
- Distribution of timbre classes as a function of density
- Probability for each instrument in a timbre class to be played
- Longest duration playable by each instrument [Roa96]

The program then runs through the steps as indicated in Fig.2.1.

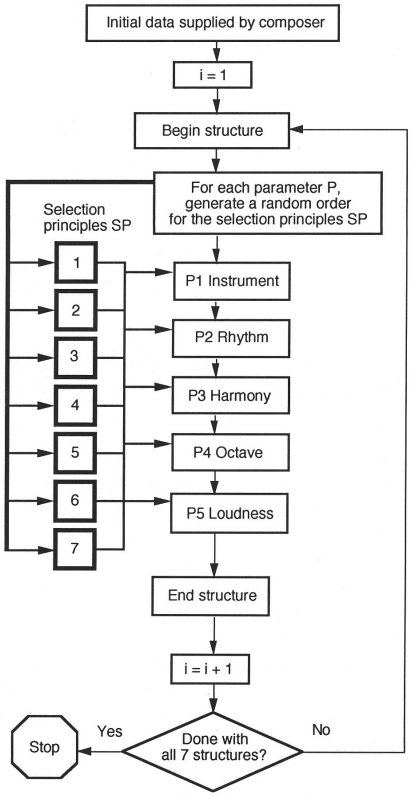


Figure 2.2: Structure of Project 1

2.4.3 Gottfried Michael König

Gottfried Michael König began to program his composition software *Project 1 (PR1)* in 1964 and finished its first version three years later in Utrecht. His work was based

on the theory of serial music but as he described "PR1 was meant to replace the very strict idea of series, a preset order of musical values, in a space in which the potential of any parameter was to change in any way" (König in [Cha97]).
The model of his program is derived from the opposition pair of regularity and irregularity, as non-repeatable serial sequences ("irregular") and group-forming multiplication rows (according to serial theory). Between these oppositions he defined five intermediate stages, where in the middle (step 4) some kind of bridging function brought the extremes together, resulting in a balanced mixture of regular and irregular events [Koe]. So the program is equipped with 7 "selection principles" that are applied to five basic parameters: instrument, rhythm, harmony, register, and dynamics, whereas different principles can be used simultaneously to control different parameters.
The composer working with PR1 has to provide the base material that is made up by:

- a set of thirteen entry delay (the time the attack of two notes)
- the total number of events to be generated
- a set of six tempi
- and a random number (as a seed for the program's procedures) [Las81]

The program then creates seven sections so that each parameter is determined by each selection principles once in a random order. The resulting output is a list of notes described by the mentioned parameters. The score may be transcribed to musical notation, whereas it is up to the composer (or interpreter) to interpret output values (as for example instrument number) [Roa96]. He continuously improved his "Project 1" in the following years with the aim to "give the composer more influence in the input data, and to give room to the way parameter values are selected before they are put into that score" (König in [Cha97]).

Meanwhile in 1968 König started building a second more elaborate program, which he called "Project 2". In PR2 control parameter are extended and cover

instrument, rhythm, harmony, dynamics and articulation. As in "Project 1" the composer provides a list of options for the described parameters and must specify the selection feature to be applied. A selection feature is a method used to compose the score out of the given list, and in PR2 various methods exist: SEQUENCE uses the user's own selection, ALEA is an arbitrary selection where repetition is permitted as already chosen elements stay in the list. Non-repetitive serial selection (SERIES) selects elements by random but doesn't allow repetition until there are no more elements in the list. RATIO implements weighted selection, TENDENCY selects within time-variable limits. GROUP is a higher-order selection feature that implements group repetition, whereas the repetition itself can be controlled by the ALEA or the SERIES selection methods. Higher-level selection principles ROW, CHORD and INTERVAL are used to create the score out of the list, whereby CHORD uses other lower-level methods (ALEA,SERIES) for a lookup of a user defined table of chords and INTERVAL implements a Markov chain described by the user-defined matrix [Ame87].

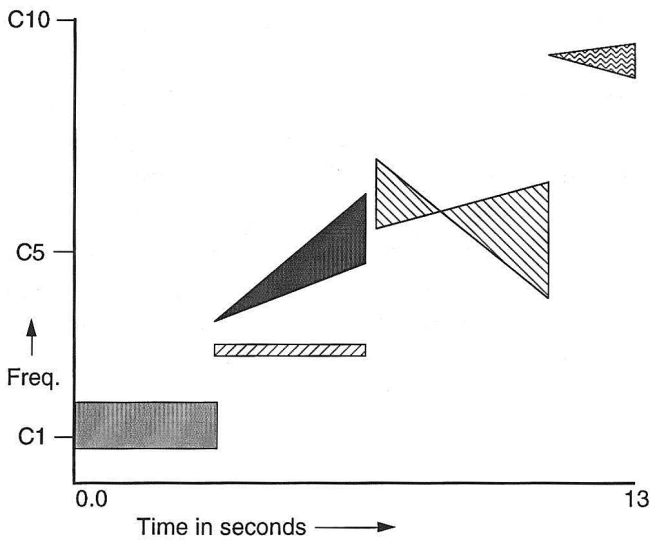


Figure 2.3: Tendency Masks in POD

2.4.4 Barry Truax - POD

The Canadian Barry Truax studied under König from 1971 and soon began to develop his own program POD (Poisson Distribution) at the Institute of Sonology in Utrecht. While the former systems - Xenakis's SMP and König's PR1 - were originally designed to output traditional score that had to be modified by the composer or played by an interpreter, POD introduced the concept of digital sound objects so that the system's output was directly connected to a sound synthesis device [Roa96]. The basic control elements in POD are the "tendency masks", that define frequency-versus-time regions (see Fig.2.3). Provided with these tendency masks, POD creates events inside the bounds of the regions. The arrangement of the generated events follows the Poisson Distribution. A density parameter is specified for a tendency mask that controls the number of events within a tendency mask. Similar to König's Project 1, Truax also applies selection principles that are used to choose the sound objects to be placed in the masks. In addition the composer may modify the program's output by changing parameters for tempo, direction, degree of event overlap, envelope and various more for the sound synthesis. The first version of POD was developed in 1972-73 and was applicable for (monophonic) real-time sound synthesis using amplitude modulation and frequency modulation. Over the years a lot of improved and more advanced versions of POD followed. In 1986-87 Barry Truax included real-time granular synthesis features to the program [Tru].

2.4.5 Other pioneer composers

Beside the mentioned examples of Hiller, Xenakis, Koenig and Truax, various other composers experimented with algorithmic composing.

In Los Angeles Martin Klein and Douglas Bolitho wrote a commercially orientated composition program using random processes and testing the generated values for melodic acceptability. The song "Push Button Berta", that was created this way, was first aired in 1956 [Ame87].

The musician Pietro Grossi was a pioneer of computer music in Italy. Some of his

first pieces were performed in 1964 at the concert "Audizione di Musica Algorithmica" in Venice [Gio96].
Larry Austin, an American avant-garde musician, followed a quite different approach. His work was inspired by natural phenomena. In "Maroon Bells" he used the contours from a mountain range to derive the melodic lines of a live voice part and a tape, while in "Canadian Coastlines" a chart tracing actual coastlines within Canada determined limits for the stochastic choices that created the composition [DJ85].
Other composers include Herbert Brün, John Myhill (see [Ame87]), James Tenny, Pierre Barbaud and Clarenz Barlow (see [Sup01]).

2.5 Interactive Systems

While in the works of Hiller, Xenakis, König and others, the processes of composition and reproduction were strictly separated, another kind of systems includes parts of the composition in the act of the performance. Interactive Systems came up and gained importance allowing composition onstage in real-time.
Chabade offers a definition describing interactive composition as "a two-stage process that consists of (1) creating an interactive composing system and (2) simultaneously composing and performing by interacting with that system as it functions" [Cha84].

CEMS

In 1966 Joel Chadabe had the idea of building a completely automated synthesiser, that was built one year later by Robert Moog. The system called CEMS (Coordinated Electronic Music Studio) was installed at the State University of New York at Albany. It was assembled out of standard sequencers and synthesis components that were equipped with logic hardware enabling the user to program them. Chadabe described it as "the real-time equivalent of algorithmic composition" [Cha97]. By programming the CEMS system to play without intervention he realized his work

"Drift"(1970) and slightly later in "Ideas of a movement at Bolton Landing" he used joysticks to control oscillators, filters modulators, amplifiers and several sequencers.

SalMar Construction

Another important system was built by Salvatore Martirano and James Divilbiss between 1969 and 1972. The SalMar Construction was made up of two sections. The upper part consisted in numerous analogue cards for sound synthesis and the second part was a control console containing switches and lights as well as digital control circuits. A patching matrix connected these digital circuits to the sound-generating electronics.

Beside the CEMS, the SalMar Construction was one of "the first interactive composing instruments, which is to say that they made musical decisions, or at least seemed to make musical decisions, as they produced sound and as they responded to a performer" [Cha97].

Synclavier

The development of powerful commercial synthesizer systems facilitated activity of interactive "composers". The sophisticated Synclavier System, developed at Dartmouth College in the 1970s, was an integrated music synthesis and recording tool applicable for real-time performance and interaction.

So Chadabe supported by Robert Moog connected two proximity-sensitive antennas to communicate with the Synclavier synthesizer. By moving the hands away or towards the antennas he changed durations of the notes and the amplitude of different sounds. This way Chadabe composed "Solo" in 1978 and created the interactive installation "Scenes from Stevens", that motivated the public to play with the antennas [Cha97].

Other composers made extensions for the Synclavier System, among them composers from the French GAIV (Computer Arts Group of Vincennes). Whereas Didier Roncin created a control device consisting of joysticks and sliders, Guiseppe Englbert wrote his own software for the Synclavier. In the program he wrote for the composition "Juralpyroc" decisions regarding durations, timbre and pitches were

based on probabilities. During the performance Englbert was able to modify the probabilities in order to influence the result without really determining it [Cha97]. A lot of other approaches and systems showed up in following years, that can't be described in detail here. See "Chadabe, Electric sound" [Cha97] for more examples.

All interactive systems have in common that there is an interaction between the composer and a complex computer system that maintain a certain level of autonomy and take decisions that can't be foreseen but influenced and controlled in some way by the human composer (and performer).

The mentioned aesthetic movements and technical inventions not only serve as inspiration and as base for the design of modern electronic music devices, but did also change the role of a composer today, expanding the notion of composition. A modern composer usually has to deal with issues of performance, interaction, and electronic instrument and sound design. Moreover an algorithmic composer faces the challenge to translate a conceptional idea to an audible piece of art. The following chapters are going to deal with the questions and techniques involved in this process.

Chapter 3

Representation and Parameters

3.1 Musical Representation

Musical representation is an important challenge that any kind of composer has to meet. There are many different ways how music can be described and represented. In a "classical" composition a piece of music is normally described by a written notation and then interpreted and performed by a set of musicians, where furthermore the score may be complemented by written comments - instruction for performance. Other compositions of modern composers - as John Cage, Stockhausen, or Xenakis - are written down in its proper way, using instructions and explanations for the performance. In his composition Silence John Cage instructs the performer to sit down on the piano and maintain silence for 3 minutes, questioning what we expect as music.

In computer music and algorithmic composition, various file formats, protocols and programs for saving, manipulating and playing computer created compositions exist. Among all these solutions, the composer has to pick a solution that is most suitable for his purposes.

One approach consists in a score based representation, e.g. by using MIDI files. The MIDI protocol has the advantage of easy portability and further treatment, but comes up with many restrictions - as it is limited to twelve tone tonality and based on single events.

Composition programs like *csound* employ another score-like representation. In *csound* score and instrument definitions are separated and so the composer sets up his own instruments. Score and instruments are very extensible as their attributes can be chosen freely. Furthermore it isn't restricted to any tonality a priori.
In many other cases there won't exist any explicit representation, but the composition program replaces the score and produces an auditable output itself. Nevertheless such systems contain an internal parameter representation that conducts the generation process. The composition described in Chapter 6 follows this approach.

3.1.1 Musical Structure

Sound is basically defined as amplitude over time signal and is therefore perceived by human listeners in a temporal process. The task of composing music in this context consists in the organisation of sounds in space and time.

One approach in doing so is to deal with hierarchies and work with different *abstract structures*. These structures can be found in musical composition as well as when designing a composition algorithm, as the algorithm aims to work either on one, or maybe more of the different levels. The main levels of abstraction are the *microscopic level*, the *note level*, and the *building-block level* [Mir01].

- On the microscopic level the composer works with microscopic properties of sound itself, that means its work consists of changing numeric values of particular samples, or modifying the timbre of sound samples of instruments - e.g. by changing control parameters of sound synthesisers or conducting the envelope of voices. In this case the composition takes place in the range of fractions of seconds or milliseconds and deals with sound synthesis techniques and adapting their parameters.

- The basic element when working at the note level is a single note event. A note can be described by a set of attributes, where the most important ones are pitch, duration, timbre and dynamics. On the note level the individual notes are organized in a higher structure, forming motives or patterns, effectively

small parts of the composition.

- The building-block level works with patterns formed out of multiple notes, sounds or music material lasting at least several seconds. So here the composition is described at the highest-level abstract structure, determining the arrangement of used parts in the whole piece of music.

In analogy to "classic" composition, the microscopic level reflects the choice of the instrument determining the timbre. On the note level single melodies are composed. At the "building-block level" the composer puts together the whole composition.

3.2 Parameter Mapping

Differently to human composers and musicians, the computers implied in the composition process in the first place do not have any knowledge about music. Similar to a classical human composer who has to write down his composition in an understandable way to make it interpretable by the performing musician, in algorithmic composition, the computer has to represent music and the underlying composition in a reproducible way.

The use of algorithms and computers for musical compositions, forces us to formalize music by parametric descriptions. In the field of computer science the term parameter refers to the input variables of a system which influence its behaviour and its results. In a composition program a set of parameters describes and conducts events on different hierarchical levels, such as structural changes, note events or instrumental attributes. However, the values of the parameters used in the composition algorithms do not make any musical sense at first hand and thus a translation has to take place. When using algorithmic techniques for musical compositions, a decision on how the used parameters in the algorithm represent the musical attributes has to be made. We have to decide on which parameters will be used in the algorithmic processes and in which way they can describe the musical events of a composition. After this decision, we have to find a suitable mapping between

the musical attributes and the algorithmic parameters. There is certainly no general solution to the problem of representation and mapping, but the decision which attributes to use depends on the abstraction level, the goals we want to reach, the algorithm we use, etc. The chosen parametric representation and the mapping function applied are crucial in the algorithmic composition process. It is the part of the composition program where the underlying idea or the mathematical algorithms are translated to the concrete music.

Despite of the importance of mapping, up till now scientific and artistic discussions as well as investigation work about mapping techniques in algorithmic composition was quite rare compared to works about algorithmic techniques in general. This is because composer mostly doesn't treat it as a separate process in their composition but as integrated in the whole composition [Doo02].

However some computer compositions only consist in the idea and its mapping. This is the case in "Canadian Coastlines", a composition by Larry Austin where he mapped geographic information (the Canadian coastline) to music [DJ85].

3.2.1 Historical example: "Pithoprakta"

Another famous example by Xenakis is well described in [Xen92]. In his composition "Pithoprakta" he used the Brownian motion of gas particles and so statistically calculated a data set of about 1000 velocities of gas particles at given time instants. These velocities were then directly drawn as straight lines onto a two-dimensional plane. Xenakis then vertically divided the graph into fifteen pitched sections each of a major third. The lines were mapped to glissandi for forty-six instruments, whereby pitch values were directly mapped from the vertical direction, and a more complex mapping was used to obtain the temporal arrangement [Doo02].

Xenakis provides more interesting work related to individual mapping techniques, mainly connecting architectural structures and music composition [Xen92].

3.2.2 Mapping function

When a set of attributes that describe the musical events of the piece have been chosen, we have to think about which parameters refer to which attribute and how the variables of the algorithm can be mapped to musical events. In most cases algorithms will return results that simply consist of numerical values that at first do not make sense as attribute values. A *mapping function* is needed so that the out-coming values are scaled and shifted to cover the range of usable values of the musical attribute. Mapping functions are often implemented as simple linear functions, although basically every kind of function (e.g. exponential, probability, or sine function) or user-defined map is applicable [Tau04].

Linear mapping

In the most simply case values may be mapped linearly from the numeric outcome of the algorithms to values usable in the music domain. Linear mapping means to apply a scaling and an offsetting transformation. Using these functions does not alter the shape or the progressive form of the original values. Rather they are the equivalent of augmentation and diminution, as well as transposition of pitch intervals in a melodic domain. Linear mapping is described mathematically by a linear function $f(x) = kx + d$.

Non-linear mapping

Apart from linear mapping functions another group of functions exists that applies non-linear transformations to the results of the algorithms. These include exponential scaling functions, sine functions, or logarithmic scales. In some cases, for example when the values are mapped to tempo or most notably to amplitude, it may be more suitable to use exponential scaling instead of linear functions as it results more natural [Tau04].
Instead of using any user defined functions, an applicable approach in the amplitude domain is to work on the decibel scale itself, while pitch values can be mapped linearly to representation like MIDI or cent units and then converted to frequency.

In this way non-linear mapping is achieved by converting between alternative representation scales.

Envelopes

Basically an envelope consists of a set of curves or lines, and they are widely used in computer generated music, typically describing the progressive form of sound samples, partials, etc. in terms of time and amplitude values for attack, sustain, decay and release. Naturally, they are usable for the explored mapping tasks as well. The lines that make up the envelope determine different mapping functions for different regions. A proper envelope can be built by providing a set of coordinates (x, y) that are connected linearly or by defining a set functions $f(x)$ and assigning them to different intervals.

3.2.3 Mapping of concepts

The mapping process is an important step and crucial part of the composition task, as it is the mapping function that actually turns numerical data into music. However, mapping is not necessarily a techniques restricted to be applied to results of algorithmic methods, but rather more generally speaking it is a process that convert musical "gestures" to sound. In this context the term gesture is used to describe musical structures, an idea or concept. From these compositional structures musical parameters are to be extracted, by mapping one set of data to another [Doo02].

Chapter 4

Algorithms

4.1 Stochastic processes

Stochastic processes are a quite popular approach in algorithmic composition. Many composers and composition systems make use of them in some way. Furthermore, stochastic methods form an important class of algorithms from the historical viewpoint, for example they were already used in the first approaches of Hiller, Xenakis and Koenig [Roa96].
In statistics a stochastic process is used in order to find characteristic patterns and properties of a given distribution. However, the approach followed by a composer to create stochastic music is the other way round. His principal goal is to organise an amount of data according to a distribution with certain characteristics [Jon81].

4.1.1 Probability distribution

Basically a stochastic process is an algorithm that calculates its result by using random values, whereas the resulting quantity of elements follows a given statistical characterization. Consequently stochastic processes are suitable to organize a large amount of values according to a predescribed form or distribution [Roa96]. In order to compose a stochastic piece of music a stochastic structure and an event space for this structure have to be formulated.

A basic type of a stochastic structure is described by a simple probability distribution over a set of events. A probability distribution holds the quantity of events where a likelihood of occurrence is assigned to each event. The single probabilities are expressed as values between 0 and 1, and the sum of all probabilities must be 1. A probability value of zero means that the event will never occur, a probability value of one on the other hand implies that the event always occurs.

Discrete and continuous random variable

A probability distribution may be used to derive values of pitches as well as to determine durations and of course many other musical parameters. However there is a difference regarding the variable that should be calculated. In the case of a set of pitches - for example the twelve pitches of chromatic scale - we work with a *discrete random value*, i.e. a limited number of possible values.

In other cases the use of discrete values may not be possible or capable, as the result can be any value within a certain range. In this case, the variable is called a *continuous random variable*, and the probabilities of such a continuous random phenomenon can be described by the *probability density function*. From this function the probability that the resulting value falls within a certain range can be derived by calculating the area below the curve. The whole area that is formed by the function curve and the horizontal axis has to be 1.

Of course it is up to the composer how to deal with musical parameters. The duration of a note does not have to be expressed by a continuous random value as presumed above, but can be defined to assume a limited number of values. On the other hand definition of useful pitches can break with the concept of a scale and we might use any real number (and therefore continuous number) within a defined range.

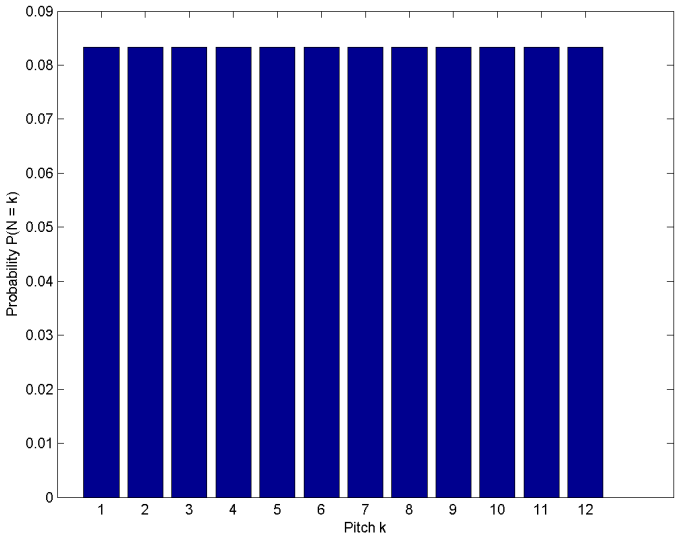


Figure 4.1: Uniform distribution

4.1.2 Basic distributions

When the composer chooses to use a probability distribution for a composition he might find it helpful to use a well-known distribution as starting point for further modification. A variety of different types of distribution and their typical characteristics are introduced below.

Uniform distribution

The simplest probability distribution is the uniform distribution, where the probabilities of all events are equal, so that the chance to be selected is the same for all events. This case is what we also refer to as aleatory, as it reflects complete randomness and no event has more preference than any other. Fig.4.1 shows an example of a uniform distribution. On the horizontal axis all possible events ?- the 12 pitches from C to B – are plotted. The vertical axis represents the probabilities of the corresponding pitches that are equal ($P = 1/12$) for every pitch.
The probability density function of a continuous random variable can be illustrated by a flat line (p(x)=1). The flatness of the function indicates a uniform distribution. Unlike when working with discrete variables there a single value cannot be assigned

to a probability (as it would be zero) but a probability can be calculated for the random variable to take a value within a certain range. This probability is found by calculating the area under the function curve.

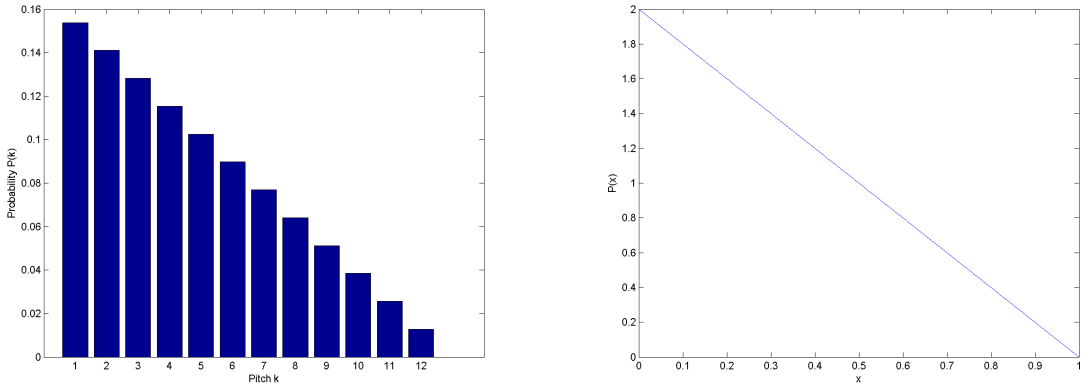


Figure 4.2: Linear distribution for discrete and continuous random variables

Linear distribution

An example of a linear distribution is shown in Fig. 4.2a for a discrete random variable. It indicates a higher probability for lower values as the note C has assigned a probability of $P = 0.15$ while note B only occurs with the probability of $P = 0.02$. Fig. 4.2b represents the density function of a linear distribution for a continuous random variable. In this case it is more likely to obtain a low value as result. The probability for the value to fall between 0 and 0.1 is 0.19 while the region between 0.9 and 1 has a probability of 0.01.

Triangular distribution

In a triangular distribution it is most likely to obtain a middle-valued result and there is a decreasing probability for both higher and lower values. Fig. 4.3 shows a simple example. A simple way to obtain a triangular distribution is to take the average of two uniformly distributed random variables.

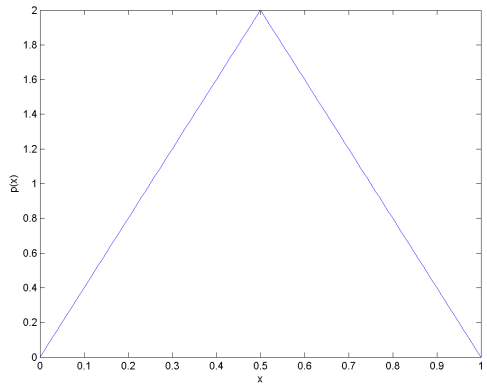


Figure 4.3: Triangular distribution

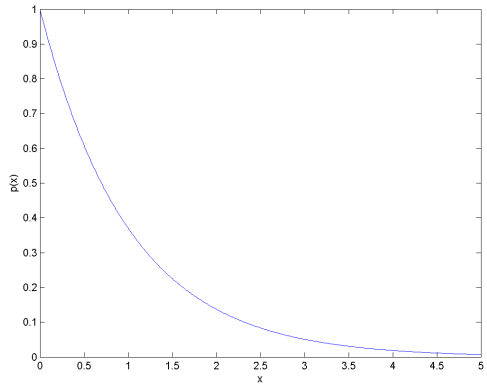


Figure 4.4: Exponential distribution

Exponential distribution

An Exponential distribution is drawn in Fig. 4.4 and its density function is given by

$$f(x) = \lambda e^{-\lambda x} \qquad \forall x \in \mathbf{R} : x \geq 0 \tag{4.1}$$

There is a greater probability to obtain lower values for an exponentially distributed random variable. The spread of the distribution is controlled by the parameter λ - whereby it extends with decreasing λ - and its mean value is defined by $0.69315/\lambda$. The function is only defined for $x > 0$ but has no upper limit, although higher values

are very unlikely to occur -? 99.9% of the results take values below $6.9078/\lambda$.

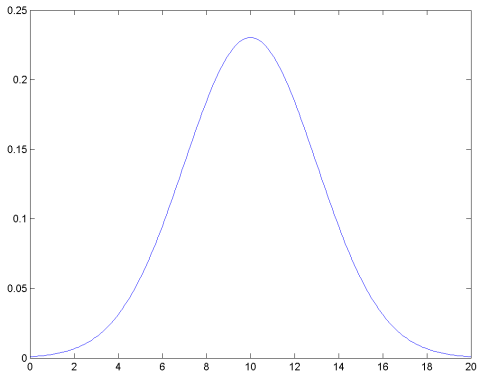


Figure 4.5: Gaussian distribution

Gaussian distribution

The Gaussian or normal distribution is a quite well-known distribution that prints the shape of a bell as shown in Fig. 4.5. It is expressed mathematically by the equation

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}}\ exp\Big[-\frac{(x-\mu)^2}{2\sigma^2}\Big] \tag{4.2}$$

where the parameter μ is the mean value of the function and therefore the centre peak of the bell and σ is the standard deviation that defines the spread of the distribution. 68.26% of all results will occur within $\mu \pm \sigma$ and although the interval $\mu \pm 3\sigma$ covers 99.74% of all results the variable is unbound [DJ85].
Many natural phenomena are known to be distributed according to the Gaussian distribution.

Cauchy distribution

The Cauchy distribution, shown in Fig.4.6, is bell-shaped similar to the Gaussian distribution, but fades out more slowly at its extremes and its mean is located at 0.

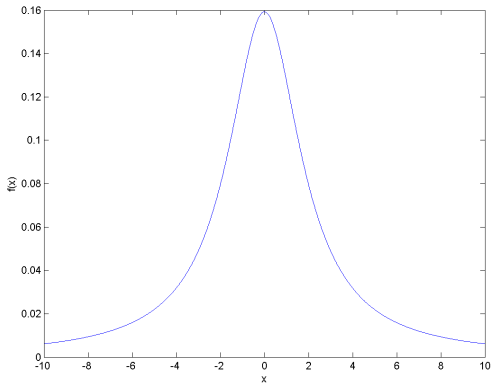


Figure 4.6: Cauchy distribution

It is defined by the mathematic equation

$$f(x) = \frac{\alpha}{\pi(\alpha^2 + x^2)} \tag{4.3}$$

The parameter α controls the density of the distribution, whereby a higher value of α will result in a more widely dispersion. As for certain applications negative values might be unusable, the left (negative) half can be folded onto the right half. This will result in a function that looks similar to the positive part of the former function with its mean centred in α. The equation has to be modified by changing π to $\frac{\pi}{2}$ [DJ85].

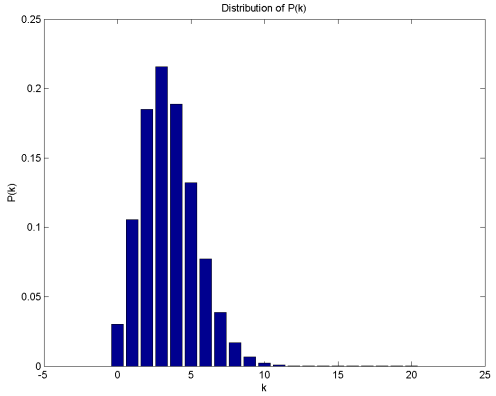


Figure 4.7: Poisson distribution with $\lambda = 3.5$

Poisson distribution

The Poisson distribution is a discrete asymmetric distribution. It assigns probabilities to the natural numbers 0,1,2,... These probabilities are given by the equation

$$P(N = k) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{4.4}$$

The Poisson distribution expresses the probability for a number of events to occur within a fixed period of time, whereby the average rate - or the expected number of occurrences - is known and expressed by λ. In our musical application naturally there is no reason why we should not use results in another context. For any chosen value of λ the distributed numbers can represent pitches, durations, and so on. Fig.4.7 shows an example of the Poisson distribution.

Beta distribution

Under certain circumstances, the Beta distribution shows peaks at the extremes like in the example illustrated in Fig. 4.8. In these cases it can be categorised as U-shaped distribution. Its function is expressed by

$$f(x) = \frac{1}{B(a,b)} x^{a-1}(1-x)^{b-1} \tag{4.5}$$

The exact shape of the distribution depends on the values taken by the parameters a and b. When a and b are greater than 1 the function gets bell-shaped similar to the Gaussian distribution. A special case is reached when a and $b = 1$, which results in a uniform distribution. For values of a and b below one, the probability for values at the extremes of the function (near 0 and near 1) increases while a and b decrease, whereby the parameter a controls the function curve near 0 and b on the other hand close to 1. For any a and b taking the same value the function will be symmetric. The expression $B(a, b)$ in the function's equations is Euler's beta function that makes sure that the area below the function curve is equal to 1.

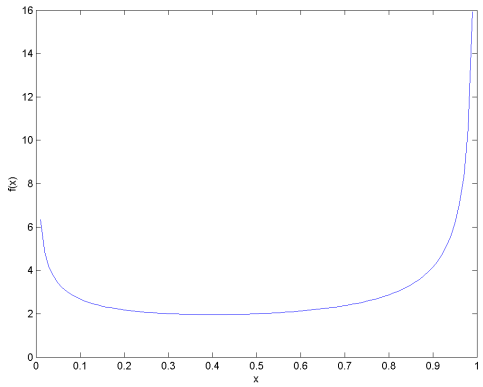


Figure 4.8: Beta distribution with a=0.6 and b=0.4

Other distributions

Of course the list of distributions given above is not complete and there are a lot more distribution that may be considered interesting for composition. Further examples include Bell-shaped distributions like hyperbolic cosine or the logistic distribution, the arcsine distribution (U-shaped) or the gamma distribution (asymmetric shape).

4.1.3 Cumulative distribution

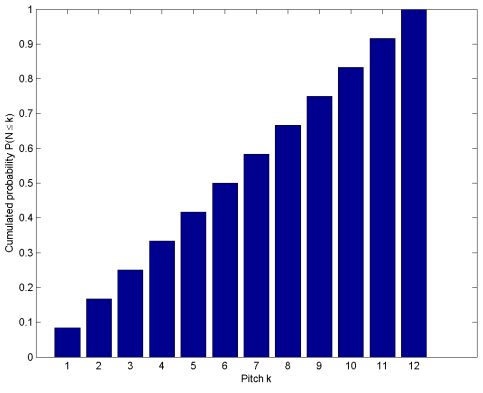


Figure 4.9: Cumulative distribution

The equations and graphic representations of the described distributions are good examples of how probabilities change over the defined interval. However when

implementing an algorithm that produces values according to the desired distribution another form of representation - the cumulative distribution - is slightly more usable. The cumulative distribution sums the probabilities successively. In case of a discrete random variable therefore the new value for an event will be the sum of its own probability and the probabilities of all former events. Fig 4.9 shows the cumulative distribution for the uniform distribution of the example shown in Fig 4.1.

Using this cumulative distribution a simple algorithm for selecting events can be built easily. A random number in the range between 0 and 1 is generated and then compared successively to the values in the cumulative distribution. As soon as the random number is smaller than the probability stored for the event, this event is returned as result.

For continuous random variables the integral of the density function represents what the cumulative distribution is for a discrete variable. Accordingly, a result value can be obtained by generating a random number z and then finding the corresponding x-value in the integrated density function, where $F(x) = z$.

4.1.4 Statistical Feedback

However, the use of such a simple algorithm based in a random number generator does not resolve a problem that the composer has to face. When a probability distribution is selected as shape for a composition it might be expected that the algorithm distributes the events as similar as possible to the original probability distribution. Yet the simple random process described above cannot assure that expectation and may on the contrary return quite distorted results especially when the number of generated events is small. That is because randomness itself depends on the *Laws of Large Numbers*.

In order to make sure that the result fits the requested distribution, even when just a few events have to be created, other more sophisticated algorithms have to be applied. The method named *statistical feedback* was developed by Charles Ames in the mid-1980s. The main idea in statistical feedback is to memorise each event's

occurrence in its statistic variable, so that it is less likely to occur when it has been selected already (too) many times [Ame90].

4.1.5 Extended concepts

To work with an algorithm that only uses one probability distribution implies that the probabilities of the events don't change over the time the piece lasts. However an interesting extension is to allow the probabilities to be changed during the composition. For example the piece can start with one distribution and end with a completely different one. This can be efficiently obtained by an interpolation of the two distributions. Of course splitting the composition in more sections allows us to apply more distributions assigned to the different sections or to the arrangement of the sections within the whole piece.

4.2 Markov Chains

The concepts discussed so far in the last chapter are restricted as they deal with *unconditional probability*. This means that all decisions are taken disregarding past events. *Markov Chains* however take the context of the event into account, i.e. the preceded state or even a sequence of last events. This concept is well-known as *conditional probability* which means that the probability of an event depends on the last state.

4.2.1 Transition matrix

Changes in the chain's state are referred to as transitions and a Markov Chain can be expressed by a matrix of transition probabilities. Table 4.1 shows an example of a transition matrix that can generate simple melodies composed from the four pitches C, D, F, and G. Each cell in the matrix contains the probability for a pitch to occur next, the transition's destination, given the last selected pitch, referred to as the transition's source or the chain's current state. The rows represent all possible values of the current pitch where the columns imply the pitches that can

occur next. For example if the last selected pitch was D, the next pitch may either be C, with a probability of 70%, or G, which has a lower probability of 30%. The cell for F contains the value 0, which indicates that the note D will never be followed by F. Neither will more than one D occur successively, because there is a probability of zero defined for that case as well. However when the current pitch is a G then the next event will generate a C for sure, as the corresponding cell contains the probability value of one.

Source state	Destination state			
	C	D	F	G
C	0.1	0.1	0.4	0.4
D	0.7	0	0	0.3
F	0.3	0.2	0	0.5
G	1	0	0	0

Table 4.1: Transition table

The composer designing his own transition matrix has to bear in mind two fundamental rules. First, the sum of all conditional probabilities for any selected pitch, so to say the sum of all rows, must be one. Second, dead ends should be avoided in the design. Dead ends occur when probabilities are set in that way that the process gets stuck in a loop, and it produces for example sequences like C-C-C-C.. or C-D-C-D-C-D.. and so on.

4.2.2 Stationary probability and waiting count

Two properties that provide us with information on the long-term behaviour of a Markov chain are the *average waiting counts* and the *stationary probabilities*.
In a transition-state matrix the so called waiting probabilities can be detected along the diagonal leading from upper-left to bottom-right. These values indicate the probability for an event to occur consecutively. The average waiting count W for an event, that is the expected number of repetitions, is then calculated by $W = \frac{1}{1-P}$ whereas P is the waiting probability (also referred to as fixed-state probability). The expected waiting count of a Markov chain gives information about the rate of

activity in the chain. An increasing value results in a chain that evolves slowly, producing a large amount of same events successively.

The stationary probabilities reflect the ratio between the stages in the result sequence over the long-term. Their exact values can be found by an iterative calculation. The transition probabilities for the next step are given by the matrix itself. Then the probabilities after 2 steps are calculated by $P \times P = P^2$ and after 3 steps by $P^2 \times P = P^3$, and so on. Usually, $\lim_{n \to P^n}$ exists, and shows the probabilities of being in a state after a really long period of time. Stationary probabilities are independent of the initial state and so the system will settle down in an equilibrium of states. This can be observed in P^n, where the stationary probability for a state is found by taking any value in the column below it [Ame89].

4.2.3 Higher-Order Markov Chains

The matrix discussed above explicitly describes the likelihood of a sequence composed by only two notes. The so defined Markov chain is called a first-order chain. The order of the chain expresses the number of prior states that influence the next state. Note that a first-order Markov chain already defines probabilities of longer sequence implicitly. Thus, the probability of the sequence C-D-G when C is the current state can be calculated by multiplying the probabilities of C-D and D-G. More generally we can formulate $p_{ijk} = p_{ij}p_{jk}$ [DJ85]. However when D is the last note, the probability that G is selected next always stays that same regardless of further predecessors. On the other hand in a higher order Markov chain there are different probabilities defined depending on the sequence of last selected events. Table 4.2 shows an example of a second-order Markov chain. Each sequence made up of two notes has its own row that expresses the probabilities of all possible notes to occur next. Theoretically a Markov chain of any order n would be possible, yet complexity and required memory grows quite quickly with the table's order.

Source state	Destination state			
	C	D	F	G
CC	0.1	0.1	0.4	0.4
CD	0.7	0	0	0.3
CF	0.3	0.2	0	0.5
CG	1	0	0	0
DC	0.1	0.1	0.4	0.4
DD	0	0.2	0.6	0.2
DF	0.1	0.1	0.2	0.6
DG	0.5	0.5	0	0
FC	0.1	0.2	0.4	0.3
FD	0.55	0.15	0.1	0.2
FF	0.8	0.2	0	0.1
FG	0.6	0.2	0.1	0.1
GC	0.02	0.18	0.7	0.1
GD	0.8	0.05	0.05	0.1
GF	1	0	0	0
GG	0.7	0.1	0.19	0.01

Table 4.2: Transition table for a 2^{nd} order Markov chain

Source state	Destination state				
	C	D	E	F	G
C	0	1	0	0	0
D	0.5	0	0.5	0	0
E	0	0.5	0	0.5	0
F	0	0	0.5	0	0.5
G	0	0	0	1	0

Table 4.3: Transition table for a random walk

4.2.4 Random Walk

The Random Walk is a special kind of Markov chain process. In a random walk the only events reachable from a selected event are its neighbours. So the process resembles a man who can either move one step forward or backward. The corresponding matrix shows non-zero values only in the cells that are immediately next to the matrix's diagonal while the rest is filled up with zeros [Jon81].

Furthermore a random walk can be categorised by its behaviour when it approaches the extremes - the first or last element – of the event space. A random walk that has reflecting boundaries causes the process to turn around every time a boundary is reached (Table 4.3 shows a corresponding transition table). Another type of boundary, the absorbing boundary, causes the walk to stop once one extreme was encountered. In a walk using elastic boundaries the probability to advance in a direction of a boundary decreases while the process gets closer to it.

4.2.5 Extension of Markov Chains

A common way to enrich Markov Chains with more complexity is to extend the concept of an event to a vector of more events. This is the case when the event instead of describing only one value, for example the pitch - contains values for several control parameters, like duration, amplitude, and so on [Jon81].

Another approach of extension is to use an event to pool a grouping of successive pitches, chords or rhythmic patterns. Then, the Markov Chains is not used to compose melodies out of individual notes but to arrange defined sequences in a superior structure. An already described famous example, W.A. Mozart's "Musikalisches Würfelspiel", implements this method [Roa96].

Extending this idea Baffioni, Guerra and Lalli proposed a hierarchical organization of Markov Chains. In this approach low-level matrices are used to select the musical details, whereas a low-level matrix that should be used is chosen by another medium-level matrix, which on its part is selected by a higher-level matrix, and so. Any number of levels can be chosen in order to compose a complex chain of chains [Ame89].

4.3 Chaotic systems

Chaos theory deals with *non-linear dynamical systems* and the term chaos is used to describe the output that these systems generate under certain conditions. Although one may associate the word "chaos" with randomness and indeterminacy, and the system's output may appear to be random, chaotic systems are not conducted by chance but are strictly deterministic. However they are most famously characterised by their high-sensitivity to initial conditions, so that slight changes in the parameters and initial variables lead to entirely different results. Therefore their behaviour seems chaotic in some way.
Mathematicians used chaos theory to describe many natural phenomena, and to

model systems like the atmosphere, the solar system, plate tectonics, turbulent fluids, economies, and population growth.
A chaotic system can basically be described either as *iterated map*, i.e. an iterative function in which the result of one iteration becomes an input value for the calculation of the next iteration, or as *continuous flows*, noted with a set of differential functions.

4.3.1 Orbits and attractors

In order to distinguish different systems and behaviours we can examine the *orbit* that they produce. The term orbit describes the sequence of values generated by the iterative process. Note that a particular orbit produced by a non-linear dynamical system highly depends to the system's initial state and so a single system is able to generate quite different orbits.
Basically there are a few types of resulting orbits:

- orbits whose points converge towards a stable value, which we refer to as the motion's attractor, in this special case a *fixed attractor*, as it is a single point that attracts the motion.

- orbits whose points fall into an alternation between limited numbers of values - a behaviour known as *periodic motion*. The term *periodic attractor* is used to describe these set of points through which the motion oscillates.

- orbits whose points indicate chaos. This behaviour is referred to as *chaotic motion*, a non periodic complex motion that gives rise to what is known as *strange* or *chaotic attractor* (see 4.3.4). In a chaotic orbit it is impossible to identify a single recurrent pattern, although motion is deterministic and similar points are visited.

Furthermore any small change in the initial values of the system's variables will result in a complete different sequence of output values.
This effect was named the *butterfly effect* by Edward Lorenz, a meteorologist who

built a computer model of a global weather system. In his work he realised that slight changes of one variable provoked huge differences in his weather model after just a few iterations. The *butterfly effect* suggests that the flap of a butterfly on one side of the globe may cause a hurricane on the other side [Mir01]. Note that for computer applications this property also means that rounding affects the obtained points and that results using different floating point precision or different processor types will diverge.

The logistic function

Let's take a look at a simple, but well-known example to illustrate the discussed properties of chaotic systems. The logistic function is described by the equation

$$x_{n+1} = ax_n(1 - x_n) \qquad 0 \leq a < 4 \tag{4.6}$$

As already mentioned the choice of the initial states, in this case the value of a, is crucial. However the initial state of x does not influence the function's long-term behaviour. Depending on the value taken by a the function develops in quite different ways. Let's examine the possible behaviours one by one:

- For $0 \leq a \leq 1$ the function converges towards a fixed point 0.
- For $1 < a < 3$ all iterations will also approximate a fixed value depending of the chosen a.
- For all $3 \leq a \leq 3.449499$ the system's behaviour becomes more complex and interesting. The fixed point still exists but instead of attracting the points of x, they are repelled by it. So the function will not converge towards a fixed attractor but will pass into a periodic alternation between two values, a so called *two-member limit cycle*.
- At the value $a \simeq 3.449499$ the two-member cycle bifurcates into a 4-member cycle, i.e.: the values will oscillate between four different attractions.

- In the small region $3.544090 < a < 3.569946$ the cycle splits further into a cycle of 8 members, then 16 and so on, creating a so called *harmonic cascade*.

- For $3.569946 < a \leq 4$ the system's behaviour gets even stranger. In some restricted areas, the iterations fall back to alternate between a few values. However outside these regions the system returns to generate chaos [Pre88].

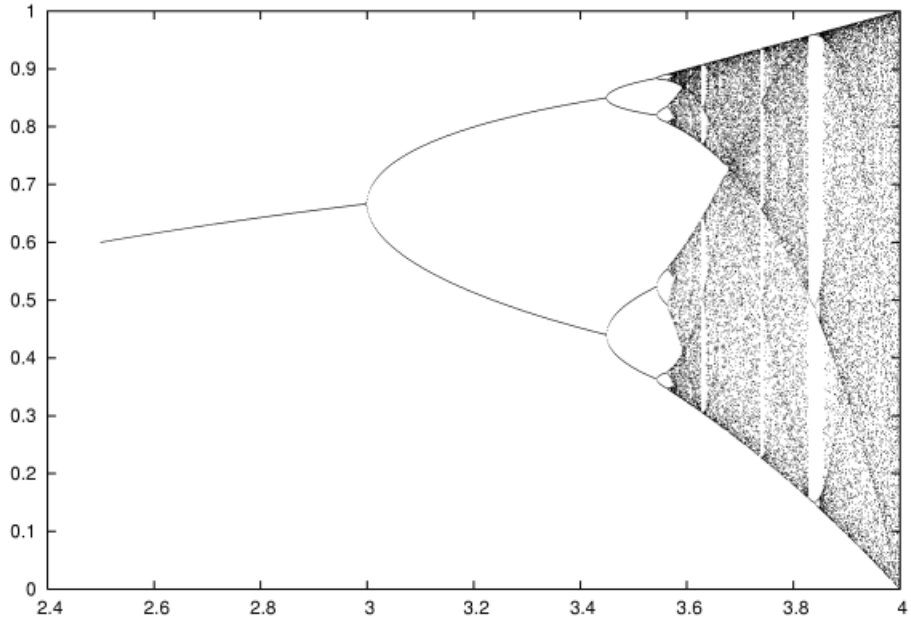


Figure 4.10: Bifurcation of the attractor of the logistic map: Values of y after 1000 iterations for r (x-axis) between 2.4 and 4. (Image source: www.wikipedia.org)

Fig 4.10 plots the phase diagram of the logistic function, that illustrates the behaviour of the logistic function, indicating the attracted values for any given a.

4.3.2 Principles of chaotic systems

The fact that an iterative function is high-sensitive to its initial conditions does not classify it as a chaotic system. Another two basic properties, which could be observed in the example of the logistic function already, determine whether a system is behaving chaotically or not. The property of *period doubling* refers to the doubling of points in the periodic attractor, while the principle of *sporadic settlement* describes

the sudden emergence of stable regions with fixed points or periodic attractors after unstable chaotic trajectories [Mir01].

4.3.3 Musical potential

For the application of chaotic systems in the field of composition we should consider the orbit produced by the system as a basis of raw material. As already mentioned the choice of initial values of variables and parameters determine the class of orbit we might obtain. So they have to be tuned in order to meet our demands. Generally a good balance between repetition and innovation seems to be desirable. A quick convergence towards stable values or small periodic attractors would be considered as quite boring and tedious.
Chaotic motion on the other hand is a more promising choice, whereby the regions just before the onset of a new cycle are of special interest. They produce an output that is chaotic in general, but falls into quasi-periodic behaviour only to return to chaos and so on [Pre88]. In other words, the system shows some kind of periodic movement by generating similar, but not identical, values within a fixed range. This way it produces new material, which can be called variations of previous themes.

4.3.4 Strange attractor

This effect of recurrence of similar pattern is also well known as "strange attractor" and is provoked by *chaotic motion*. Strange attractors are unique to chaos theory and can be described "as periodic cycles that are smeared out so that the orbits never repeat exactly" [Mad99].

Lorenz attractor

A well-known example of a strange attractor is the famous Lorenz system, a simplified three-dimensional model describing atmospheric turbulence. It is also called *Lorenz attractor* and can be described mathematically by three equations:

$$\frac{dx}{dt} = \sigma(y - x)$$

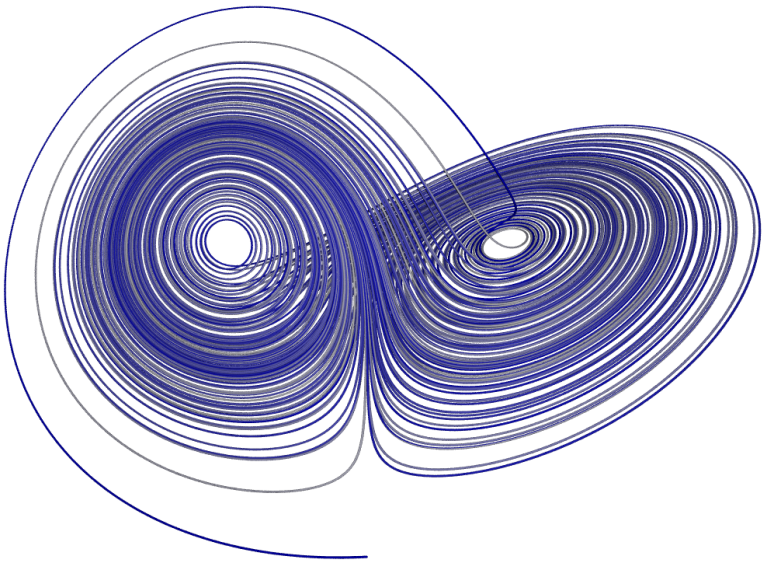


Figure 4.11: Lorenz attractor

$$\begin{aligned} \frac{dy}{dt} &= Rx - y - xz \\ \frac{dz}{dt} &= xy - Bz \end{aligned} \tag{4.7}$$

where σ, R and B are the constant parameters that have to be set to a positive value. We are now facing a system in three dimensions, so we have three interconnected variables that we might map to different musical parameters. Furthermore the description by differential equations also indicates that the Lorenz attractor, in contrast to the logistic map, is formed by a continuous flow. However in order to resolve the equation with a computer we will integrate the function by using a small dt, so that we obtain an orbit again that consists of a sequence of points.

The behaviour of the Lorenz system can be controlled by its parameter R. Values of $0 < R < 1$ result in an attraction towards the origin $(0, 0, 0)$. Up to the critical value of about 27.74 all orbits are attracted towards a stationary point - either $(\sqrt{(B(R-1)}, \sqrt{(B(R-1)}, (R-1))$ or $(-\sqrt{(B(R-1)}, -\sqrt{(B(R-1)}, (R-1))$. For almost all values $R > 27.24$ we obtain a chaotic orbit, whereby the former stationary points become repelling, and the points in the orbit print spirals that drift away from these points. This behaviour is plotted in Fig. 4.11. Nevertheless values of R exist,

as for example $R = 150$, where the orbit falls in periodic cycles [Bid92].

4.3.5 Hénon Map

The *Hénon Map* does not model any natural phenomena in particular, but was designed as a general two dimensional model for dissipative chaotic systems. It is expressed by the equations

$$\begin{aligned} x_{n+1} &= y_n + 1 - A{x_n}^2 \\ y_{n+1} &= Bx_n \end{aligned} \tag{4.8}$$

where the parameters A and B have to take positive values. The map produces an orbit with either a single-point, periodic and chaotic attractor, which primarily depends on the value of the parameter A. Fig. 4.12 illustrates two interesting cases of chaotic orbits.

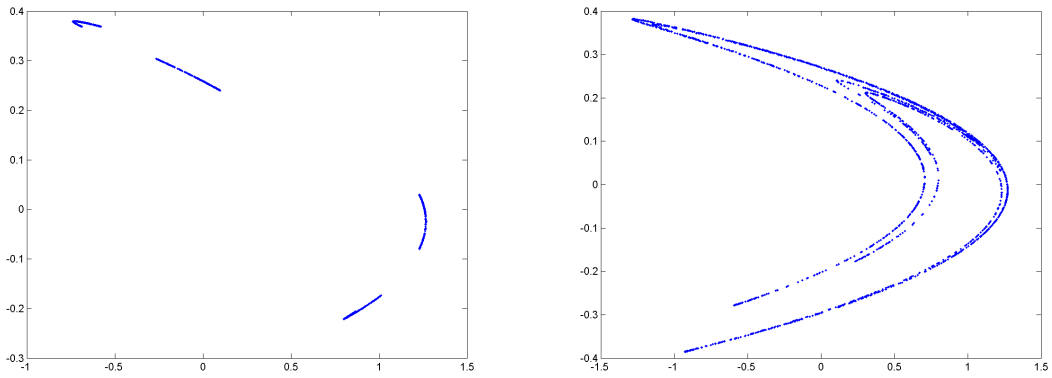


Figure 4.12: Henon Map: a) a chaotic attractor in four bands ($A = 1.064, B = 0.3$) b) chaotic attractor ($A = 1.4, B = 0.3$)

More chaotic systems

Apart from described examples, the logistic map and the Lorenz attractor, many other systems exist showing similar behaviour that makes them interesting for musical appliance. More details can be found in Bidlack[Bid92] and Pressing[Pre88].

4.4 Fractals

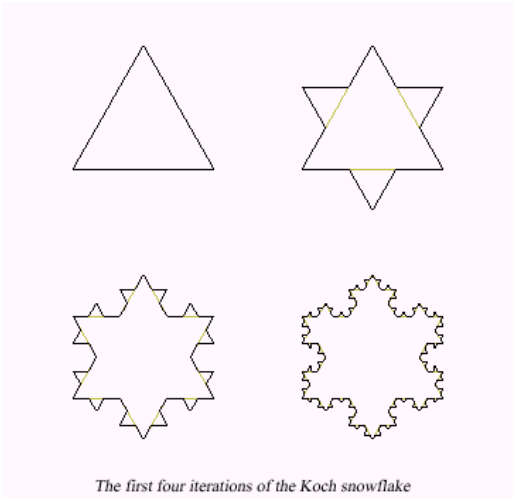
A fractal is basically a geometric object, similar to a rectangle or a circle, but with quite different properties. It can describe irregular shapes and possesses infinite details as it contains self-similarity structures. Fractal geometry was introduced by Benoit Mandelbrot in order to cover these phenomena that are inherent in many natural shapes like clouds, mountains and coastlines, but cannot be described by Euclidean geometry. The term *fractal* was coined by Mandelbrot in 1975 and derives from the Latin word *fractus* (broken). However fractal objects, for example the Koch snowflake, were already known earlier under the name *monster curves*.

4.4.1 Properties of fractals

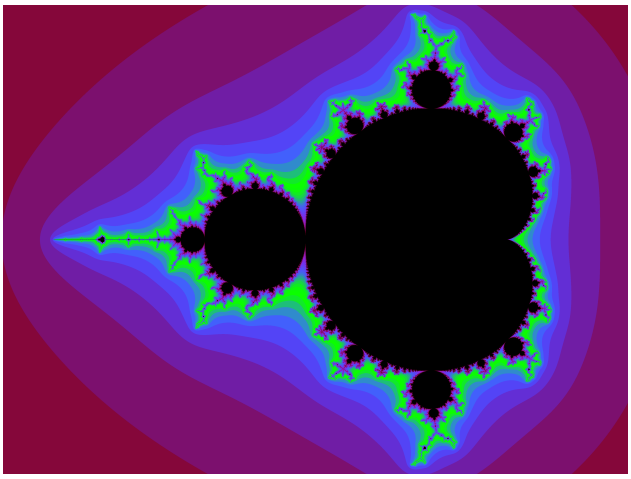
Naturally the main application of fractals lies in the field of computer graphics, roperties that are inherent in fractals may also be interesting for musical appliance. Fractals exhibit properties of *self-similarity* and *scaling invariance*. Furthermore they are of *fractal dimension,* a non-integer dimension, and can be described by a simple algorithm using iterative functions.

Self-similarity

Fractals are self-similar which means that they contain patterns, which repeat themselves at different sizes. A simple fractal object, the *Koch snowflake* (Fig 4.13(a)), illustrates this quality. In some cases the structures found at a smaller scale will be exact copies of the general shape, a property described as *exact self-similarity*. However, in nature we often find objects with recurring patterns that are quite similar, but not exactly equal. A case referred to as *statistical self-similarity*. Finally, *generalised self-similarity* can be observed when a transformation was applied to the copies of the shape.



(a) The Koch snowflake



(b) The Mandelbrot Set

Figure 4.13: Two well-known fractals

Algorithms

Basically, a fractal can be made up by a simple algorithm in an iterative process. The composition of the Koch snowflake is shown in Fig 4.13(a). It starts with a triangle, and in iterative steps another triangle is placed in the middle of every edge of the larger one.

Building the famous *Mandelbrot set* (shown in Fig 4.13(b)) is slightly more complicated but based on one single equation

$$z_{n+1} = z_n^2 + c \tag{4.9}$$

The variable c is a complex number and, therefore, composed by a real and an imaginary part: $x+iy$. In order to build a two dimensional image of the Mandelbrot set the iterative function is applied to every point(x, y) in the image. The point's colour is then derived from the necessary number of iterations to reach a given threshold. For some values of c the function stays bounded that results in assigning the colour black to the given point [Mir01].

Fractal music

So far we have seen that fractal geometry can create interesting images. Although these fractals are based on quite simple functions, applying these to the field of music is not so easy. Images like the *Mandelbrot set* contain two dimensions and are recognised at the first glance. Music on the other hand is described by a function of amplitude over time.

A way to create fractal musical content is based on the fractal's iterative function itself. This approach follows the concept of chaotic systems discussed in the previous chapter. Furthermore a certain type of fractals is constructed by *iterated function systems*, where transformations like rotations and translation are recursively applied to a set of points. However in other cases, this method is hardly applicable and the resulting music may not be as pleasing as the corresponding image representation suggests. Another approach consists of sticking important properties self-similarity and scaling invariance. *1/f noise* is such an approach.

4.4.2 Iterated Function Systems (IFS)

Iterated Function Systems (or IFS) are a group of fractals of two or more dimensions that are described as a solution to a recursive set of functions. These functions consist of simple geometric transformations or replacement rules, whereby the order in which they are used does not matter, but is chosen by random. The famous *chaos game* illustrates the principle of IFS. In the chaos game a polygon is set up as base for the algorithm and a random point in the plane is chosen. The game works by picking any point of the three vertices and move half the way towards it. Then the point is drawn and the process repeats. The output of the *chaos game* is the *Sierpinski triangle* (Fig. 4.14).

The main interest of Iterated Function Systems for composition consists in the possibility to carry the geometric transformations directly to the field of music. Serial music (chapter 2.3.1) has already introduced concepts about transforming tone sequences, as composers like Schönberg formulated rules about inverting, rotating

or translating a series. Moreover we may see any point (x,y) as a pair of tones, interpreting x as keynote and y as interval and, thus, using the IFS to generate musical content.

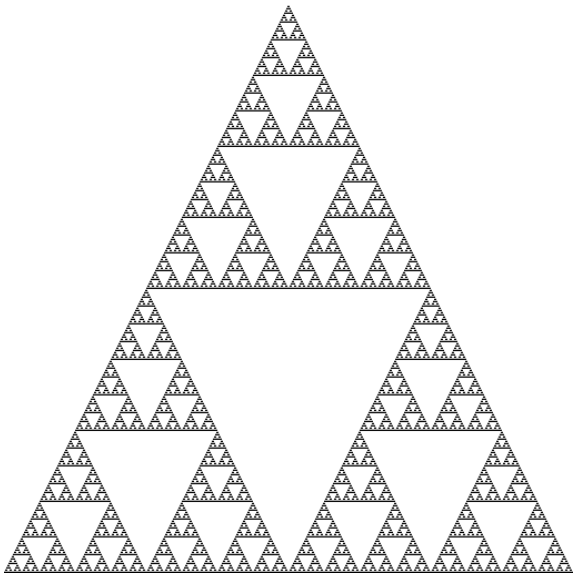


Figure 4.14: Sierpinski triangle

4.4.3 1/f noise

A way to translate the concept of fractal geometry into the field of music is *1/f noise*. We can view a sound signal, and therefore any sequence of numbers, in terms of its spectrum - the frequency domain. The *power spectrum* (or spectral density) of a signal gives us information about the "energy" of the frequencies it contains. Signals with a spectrum described by $1/f^g$ where $0 \leq g \leq 2$ are referred to as fractional noises.

Pure *white noise* contains all frequencies, therefore it has a flat spectrum defined by $1/f^0$ (=1). The other extreme is *Brownian noise*, which has a spectrum that is described by $1/f^2$, that indicates that low frequencies outweigh higher ones. In other words a value of the signal strongly depends on the previous one, as it can be observed in random walks (described in 4.2.4) [DJ85].

The far most interesting fractional noise however is the *1/f (one-over-f) noise*, or

pink noise. Sequences generated by 1/f noise are known to correlate logarithmically with the past. This means that past values influence the current value, whereby the last 10 numbers have as much influence as the last 100 or the last 1000 [Roa96].
Many natural phenomena were proved to have spectral density of 1/f, but music of different styles seems to produce similar spectra as well. This was derived by Richard F. Voss and John Clark from a set of observations. In addition they suggest 1/f noise to be a good choice for stochastic composition, as white noise sounded too random and Brownian noise too correlated [VC78].
Note that pink noise fully meets the property of self-similarity, as patterns of small details reflect patterns found at larger scales. Charles Dodge provides as with "Profile" - a musical work based in 1/f noise [Dod88].

4.5 Artificial Neural Networks

Artificial Neural Networks (ANN) offer an alternative approach to the traditional theory of computing mostly used in standard computers based on the *von Neumann architecture*. They break with the typical central processing unit (CPU) vs. memory paradigm by replacing it with a distributed system of many interconnected units. Neural Networks, also referred to as *Connectionist models* or *parallel distributed processing models*, do not resemble traditional systems that apply preprogrammed rules translated to a instruction set and executed in the processing unit, but rebuild a simplified model of the human brain. The main advantage of this approach is that these systems are able to learn from supplied examples, so that it's not necessary to analyse a problem in order to explicitly formulate functions and subroutines which produce the desired results. Instead neural networks are programmed by repeatedly presenting them existing examples. So they are taught the right behaviour for the inputs they get.

4.5.1 Structure of a neural network

Neural networks are inspired by the human brain which consists in a massive network of millions of neurons, each considered as an independent processing unit, whereas various different types of neurones exist. These units are interconnected via *synapse junctions* (or simply *synapses*). The separate neurons may stimulate other connected neurons by discharging an electrical pulse along its *axom* - a long slender fibre. An incoming pulse disturbs the internal voltage level of a neuron referred to as *equilibrium potential*. The neuron itself fires an impulse if the sum of the arriving stimulations provoke a sufficient depolarisation of its body [Mir01].

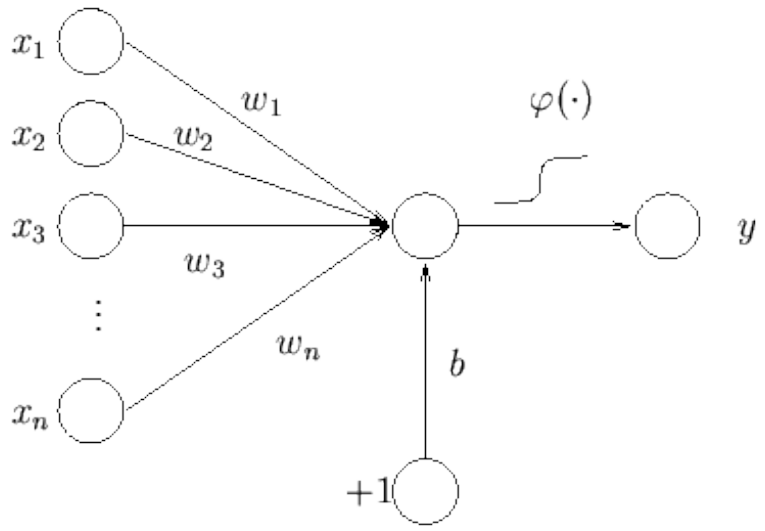


Figure 4.15: Perceptron: a model of an artificial neuron

Artificial neural networks use the same function principle. The first successful step in its development was the *perceptron*, a simple neural model introduced in 1958 by Frank Rosenblatt. In this model an artificial neuron (as illustrated in Fig.4.15) has a number of incoming connections that resemble the synapses of a real neuron. Each connection has a weight value assigned to, which serves as scaling factor for the input value, whereby positive and negative weights are possible. The neuron sums up the scaled inputs and passes the sum to a *activation function*. Additionally, a

neuron may also have a further incoming *bias* signal b, that raises or lowers the input to the activation function.

$$y_i = F(\sum_{i=1}^{N} w_{ij} x_j) \tag{4.10}$$

The activation function F defines the output value of the neuron according to its input. Basically two different types of functions are applied. *Hard limiting functions* (threshold functions) only return either 0 or 1 for input values that are beyond or above the defined threshold. *Soft limiting functions* also return values between 0 and 1 for inputs close to the threshold [Dol89].

First approaches consisting of simple single-unit networks (*single layer perceptron*) came up with quite fundamental limitations, but not until the early 1980s more sophisticated networks emerged - the *multilayer perceptron (MLP)*. Multilayer networks are composed of a set of neurons organised in various different layers. The *input layer* serves as the eyes and ears of the network and the *output layer* gives us the results. In between these two there may be one or more *hidden layers*, composed of nodes that do not have any external connections but are important for the internal information flow.

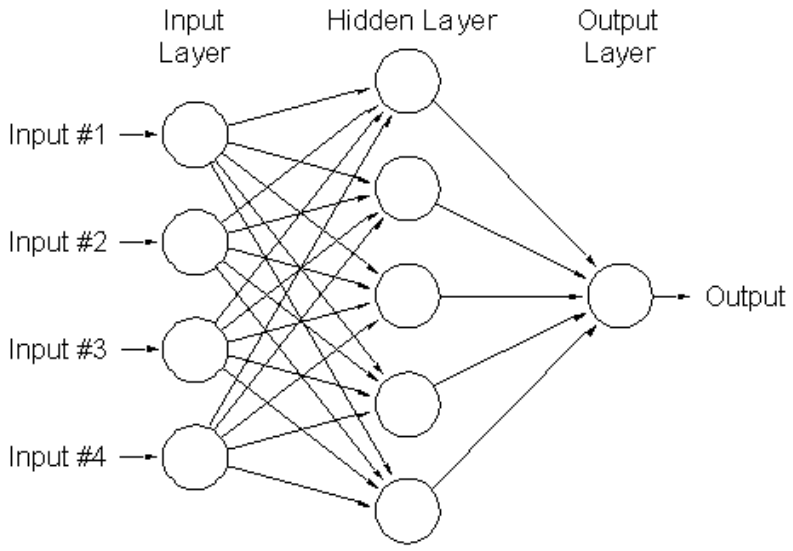


Figure 4.16: Feed-forward network with hidden layer

Today a range of variations exist within this architecture, mainly differing in the existing interconnections of the nodes. A *feed-forward* network (Fig.4.16) only contains connections in one direction, i.e. the way through the network is straightforward as there are no feed-back loops [Dol89]. Contrastingly in networks implementing a *feed-back* architecture, the output of a unit is connected back to its input. The feedback connection can either lead directly to the node itself, or indirectly, when the unit's output serves as input for units in a previous layer. In *fully interconnected* networks the output of each node is connected as input for all other nodes [Mir01].

4.5.2 Training of a neural network

A neural network has a general design and has no knowledge of the application domain in advance. Because of facing an unknown problem, it has no idea which setting of weights would result in the best solution. The learning process of a neural network consists in presenting the system a series of examples. These examples contain input values representing the given problem and output values describing desirable solutions. The weights of the connections in the network then will have to be modified in order to enable the system to approximate the example results.
Different learning schemes exist whereby the *supervised learning* is mostly applied, which seems to be practically in the music domain as well. It works as described above, while the samples have to be presented in a lot of iterations, maybe many thousands times [Mir01]. In order to adjust the weights the error of the actual configuration of the network has to be measured. Then a technique known as *back propagation* of error can be used to calculate the new weight setting. The *back propagation* algorithm measures the total error at the output by summing the squares of the individual error of each output unit, mathematically described as

$$\epsilon = \sum_{k=1}^{N} (y_k - \tau_k)^2 \tag{4.11}$$

where τ_k hold the expected output values and y_k the measured values. Then it goes back through the network's layers and adjusts each weight just a little so that the ϵ decreases [Dol89]. After a few iteration, during which the error could be minimised to an acceptable value the weights are fixed and the network is ready for use.

4.5.3 Music network

The above described MLP network is a usable approach for composition systems. However a general design of the network's architecture does not exist. Thus when constructing such a network various considerations have to be taken so that the system fits the faced task. Decision that have to be made include the number of layers and the number of units to use in each layer, the setup of the connections between the nodes, implementation of feedback loops and an appropriate representation of the musical structure for in- and output.

Modelling the time domain

Working in the field means to deal with a temporal process. Therefore the representation of time in the neural network is a fundamental design issue. For example when we want to build a system that produces melodic sequences that resemble the providing training set, the network has to get knowledge about pitch relations and intervals in the whole example pieces instead of only taking into account two successive notes. This will usually lead us to a feedback architecture which enables the network to include past events into their actual decision and, thus, builds up the network's memory.

An approach by Peter Todd[Tod89] demonstrates design issues and possible solutions for an ANN by producing simple monophonic melodies. Todd uses a modified, three-layer, feed-forward network that was first proposed by Jordan [M.I86]. The modifications mainly concern the input layer, which is composed of a set of *context units* and a set of *plan units* (see Fig.4.17). While the plan units identify the pattern that is being learned or produced, the context units constitute the memory as they have incoming connections from the output units. The weights of these connections

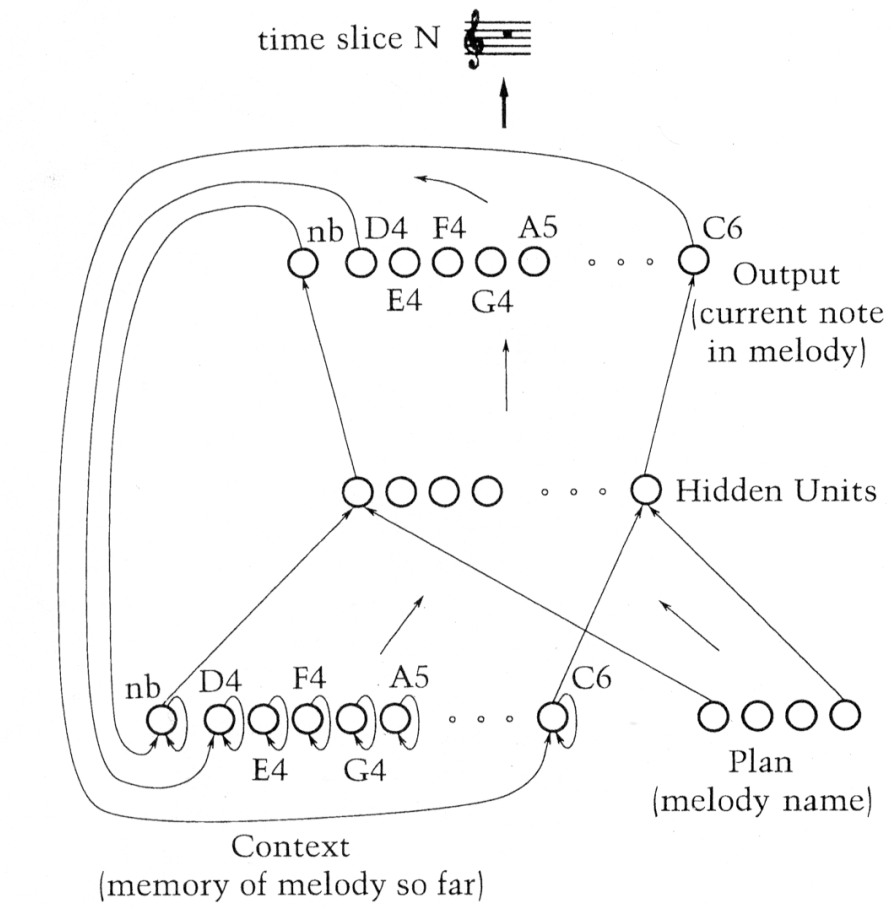


Figure 4.17: Musical network with feedback architecture (taken from[Tod89])

have a value of 1, thus output values are taken over unchanged to the context units. Each context unit has an additional self-feedback connection. That ensures that more than just the previous event can be remembered, whereby the weights of these self-connections are less than 1, so that the influence of past values exponentially decreases.

Representation of pitch and duration

A suitable representation to be used for describing pitch and duration of notes in the melody has to be chosen. Pitch information may either be codified as intervals between successive pitches or by their actual absolute values. Each approach comes with its own drawback. Absolute pitch values are more limited to a given range

and cannot be easily transposed. Relative information on the other hand is a key-independent representation, but an error in just one note will affect all successors. So the melody would shift to a totally different key within the sequence. Another issue is whether it is better to encode each pitch value (or interval) in an own unit (*localist representation*) or to use a binary coding scheme (*distributed representation*).

For the representation of durations Todd suggests the use of a separate note-begin unit. In this approach each time step of the network corresponds to a duration (e.g. an eight-note) and so successive same pitch values can be either interpreted as notes with longer duration (note-begin unit is set on) or more notes with the same pitch (note-begin is off).

Generalisation ability

The main attraction of neural networks is their ability to generalise. That means that in the learning process it is no worthwhile target that a network just memorises the presented examples. It should rather discover underlying structures and patterns inherent in the samples of the training set in order to be able to produce new material out of the generalised rules found by learning. This generalisation ability is fundamentally affected by the number of hidden units incorporated in the network architecture. By using a lot of hidden units and a few training examples the network will go the easy way, and as a result, tends to mere memorisation. Therefore a basic rule in the design of musical network is to keep the number of hidden units as small as possible while presenting many training examples [Dol89].

Learning issues

The network can be trained by the back propagation of error method that was introduced above. However in networks similar to Todd's system - where output units are connected back to input units - the input units have to be fed by the target values of the actual training example instead of the networks output values in order to ensure a proper learning process.

Another interesting point is that teaching two equal designed networks the same examples will not provoke the same result. As the weights in the network are initialised randomly before starting the learning process, they may not settle down on the same values at the end. This includes the problem that weights can get stuck in a *local minima*, i.e.: they settle down on a suboptimal solution. However the better solution cannot be reached by further training because the way to it passes a region of even less applicable values [Dol89].

4.5.4 Limitation

A major restriction of the neural approach to composition consists in the weakness of neural networks at a higher-level structure. While they perform quite well in learning and creating low-level details like short melodic sequences, they lack the ability to recognise higher-level organisation of compositions. Thus they wander from note to note without having a clear direction or destination [Tod91].

4.6 Cellular automata

With *cellular automata (CA)* techniques that are widely used in the field of *Alife* (Artificial Life) found appliance in algorithmic composition [Mir01]. Cellular automata are dynamical systems that are idealisations of natural systems and useful to investigate complexity and self organisation. Cellular automata were first introduced in the early 1950s, when John von Neumann's search for an abstract model of self-production in biology lead him to build a two-dimensional cellular automaton in 1951, following suggestions by Stanislaw Ulam [Wol02].

4.6.1 Introduction

Generally a CA is made up by a lattice of cells each representing a particular state, taken from a finite set of states (usually integer numbers). The states of the cells evolve in discrete time steps, whereas the new value of a particular cell is determined by a set of transition rules that take into account the previous states of the cell and its

neighbourhood [Wol83a]. Accordingly, CA could be compared to parallel-processing computers of simple construction.

The lattice can be of any finite number of dimensions, but usually automata of one, two or three dimensions are used. Normally, *periodic boundary conditions (PBC)* are applied, that means that the site on one extreme of the lattice and the site on the other extreme are treated as neighbours [BPT00].

A CA's actual state can be printed out in a coloured grid, whereby cells are coloured according to their states and every possible integer value has a different colour assigned to. Using this graphical representation the CA's evolution and thus the emergence of patterns can be easily viewed.

4.6.2 Classes of Cellular Automata

The long-term global behaviour of cellular automata is used to classify different types of CA. Stephen Wolfram[Wol84] distinguishes between four classes of CA:

- Class 1 (Homogeneous) Cellular Automata evolve to a unique homogeneous state from almost all initial states.
- Class 2 Cellular Automata generate simple structures, which are either stable or periodic, out of particular (usually short) initial value sequences.
- Class 3 Cellular Automata lead to aperiodic chaotic patters from almost all possible initial states.
- Class 4 Cellular Automata show a complex, essentially unpredictable behaviour. An import feature of these CA is the presence of propagating structures and their capability for universal computation [Wol84]. Conway's *Game of Life* CA (see 4.6.5) falls into this category.

4.6.3 Elementary one dimensional CA

An elementary one-dimensional cellular automaton is the simplest type of CA. It is composed by a set of n adjacent sites with possible values 0 and 1 (Number of states

$k = 2$). The radius r of the neighbourhood determines the number of neighbours that is the cells that are involved in the transition rules. Building a CA with $r = 1$ means that the cell's neighbours are defined as the cells on either site of it. These three cells can build 8 (2^3) different patterns and consequently there are 256 (2^8) evolution rules possible. These 256 different CA are named by a number - the decimal translation of its binary rule table. So the Rule 30 CA is defined by the rule table 00011110 (see Table 4.4): If three adjacent cells show the pattern 111 the middle cell will take the value 0, for pattern 100 the next time step will set the cell's value 1, and so on. Figure 4.18 shows the pattern generated by rule 30 CA starting

111	110	101	100	011	010	001	000
0	0	0	1	1	1	1	0

Table 4.4: Rule 30 cellular automaton transition table

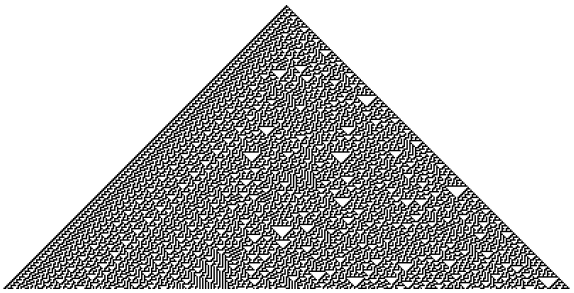
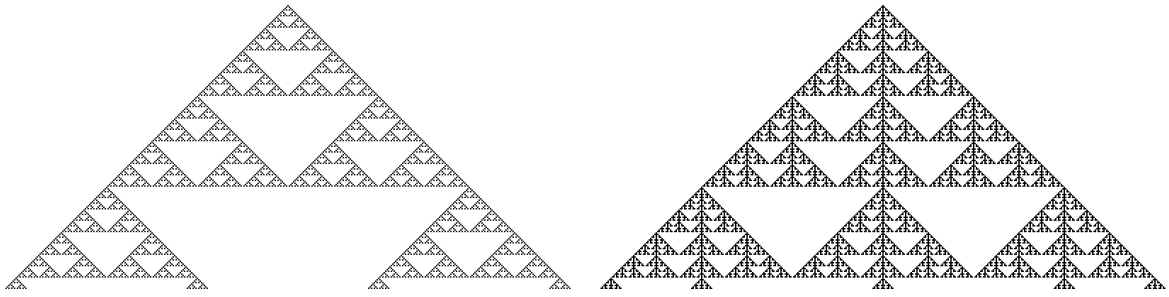


Figure 4.18: Patter generated by Rule 30 CA

from a simple initial condition (with only one site in state 1).

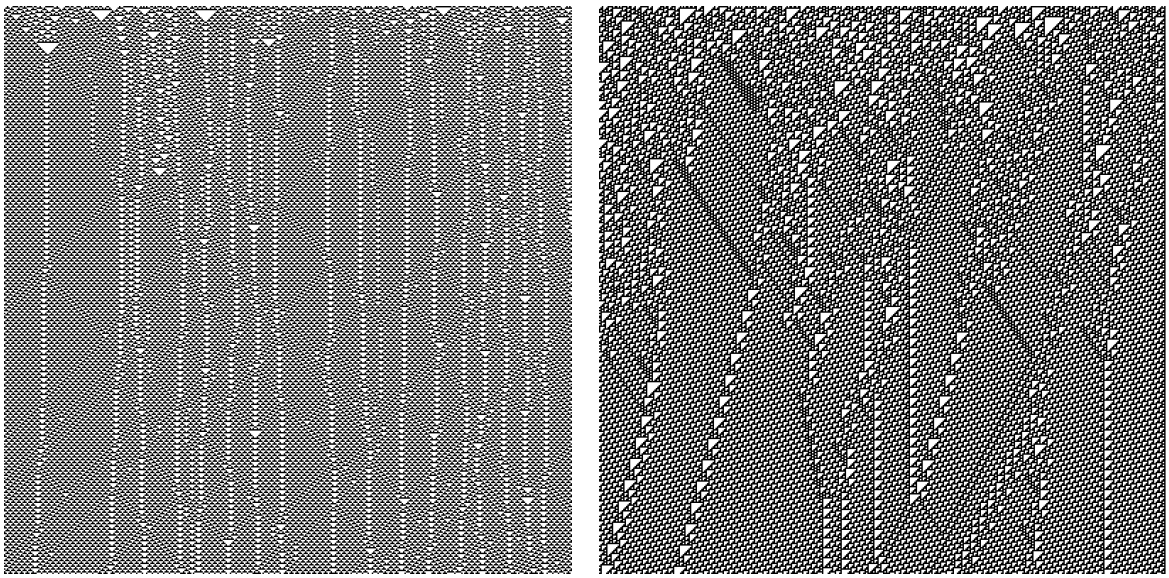


(a) Rule 146 CA (10010010) (b) Rule 150 CA(10010110)

Figure 4.19: Fractal patters generated by rule 146 and 150 CA

The 256 possible rules that can be formulated lead to very different types of cellular automata. By applying the rule 146 or 150 an elementary one dimensional

CA is also capable to produce fractal structures which are self similar and scale-invariant. The corresponding emerging patterns are illustrated in Fig. 4.19. Rule 18, 90, and 218 produce the same pattern as 146 - Pascal's triangle reduced by modulo 2. Rule 150 shows a more complex structure and thus a greater information content.



(a) Rule 54 CA (00110110) (b) Rule 110 CA (01101110)

Figure 4.20: Patters generated by rule 54 and 110 CA

Eleonora Bilotta[BPT00] stresses two rules to be the most appreciable for musical translation: rule 54 and rule 110 (Fig. 4.20). Elements in the patterns can be divided into two groups. There are those that make up the background structure and others that differ and print perceivable substructures. At first music produced by both patterns is perceived as dissonant and chaotic but it was experimented that improvements can be achieved by only taking into account the structures of the second group for musical composition, by filtering the background [BPT00].

4.6.4 Emerging patterns: Gliders, Beetles and Solitons

Complex structures produced by elementary one dimensional CA can be either stable or unstable depending on the initial conditions chosen. Extending the neighbourhood of the CA discussed above to $r = 2$ we obtain CA defined by 32 (2^5) local

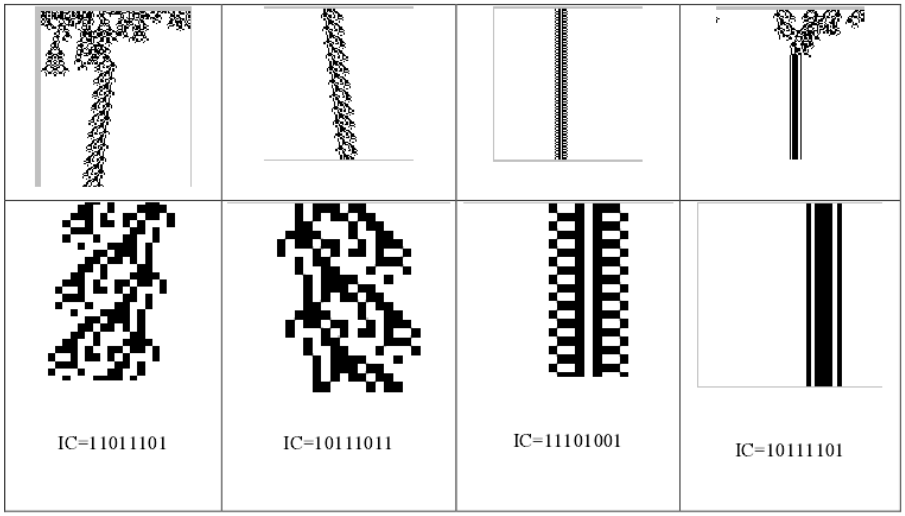


Figure 4.21: Gliders produced by a Rule 20 CA (taken from [BPT00])

transition rules.

For rule 20 CA - whose rules are described by its binary representation 01101001100101101001011001101000 - the emergence of such stable structures can be observed, when suitable initial conditions have been set. Fig. 4.21 shows examples of resulting patterns, known as *gliders*. Gliders preserve a sequence of states of a certain number of sites. Their structures can either move in space (as the first two gliders in Fig. 4.21) or be steady. However music produced out of these gliders was experimented to be quite monotonous [BPT00].

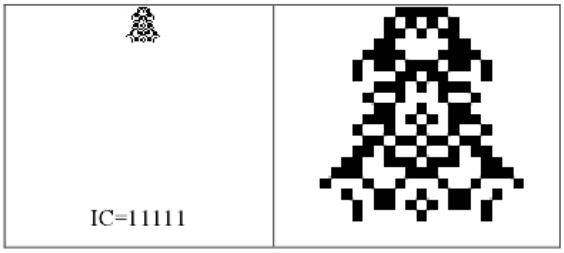


Figure 4.22: Beetle structure (taken from [BPT00])

With another initial configuration the same rule 20 CA produces different unstable structures known as *Beetle*. Fig. 4.22 illustrates such a pattern that is drawn in

twenty time steps.

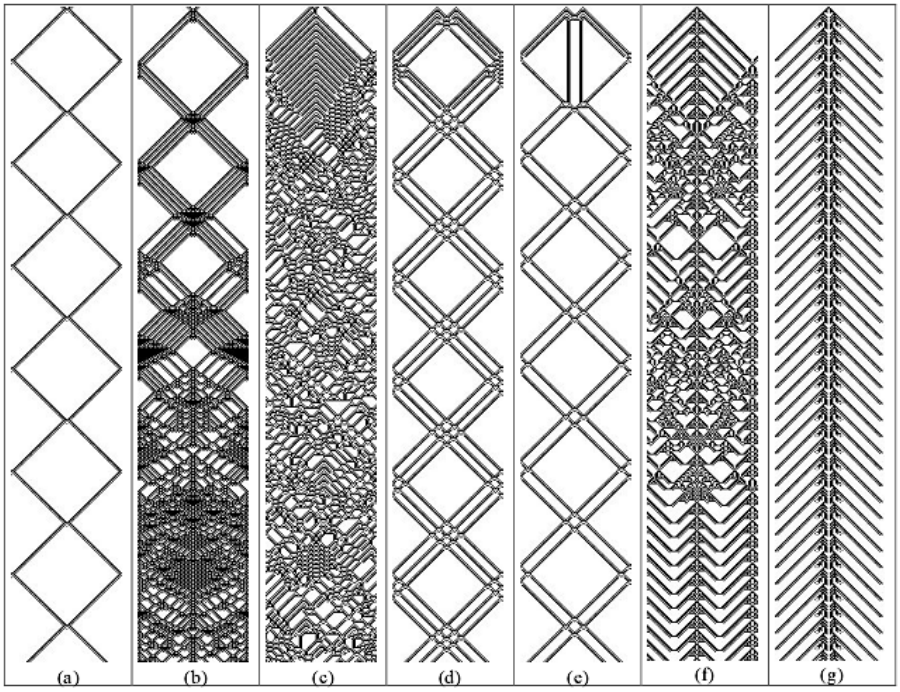


Figure 4.23: Various types of solitons (taken from [BPT00])

Solitons are another group of emerging patterns produced by CA with $r = 2$. They can lead to different complex structures, some of them illustrated in Fig.4.23. More detailed information on these emerging structure can be found in [BPT00].

4.6.5 Game of Life

A well-known example of a CA is the *Game of Life*, devised by John Horton Conway in 1970 to model a population of simple virtual organism [Mir01]. It is a two-dimensional CA, whose cells may take two possible states: alive or dead.
The automaton's rules are quite simple: Unless a cell has two or three neighbours that are alive, the cell will "die", two neighbours that are alive will leave the cell's state unchanged, while three neighbours cause the cell to be alive [Wol83b]. Fig. 4.24 illustrates the evolution of a Game of Life automaton.

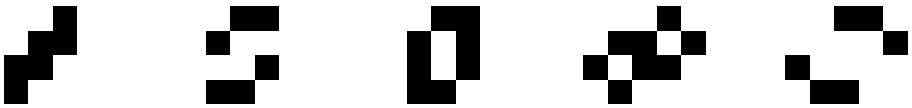


Figure 4.24: Evolution of a Game of Life CA

Conway's Game of Life forms part of a special class of CA called *totalistic cellular automata*. The rules describing a totalistic CA do not consider the single states of the cells in the neighbourhood but only take account of the sum of values of these cells, whereas an *outer totalistic CA* - as Game of life - includes the cell itself.

4.6.6 Demon Cyclic Space

The Demon Cyclic Space is a two-dimensional CA with n possible states. A cell in a state i at a time step $t-1$ will dominate any neighbour cells in state $i-1$, i.e.: their states will be set to the state i of the dominating cell at time step t. As the set of possible values is cyclic, the first value (0) dominates $n-1$. The Demon Cyclic Space CA is normally initialised with random values, whereby every states has a different colour assigned to, and then evolves into a self organised structure, a stable pattern reminding of crystalline growth (Fig.4.25) [Mir01].

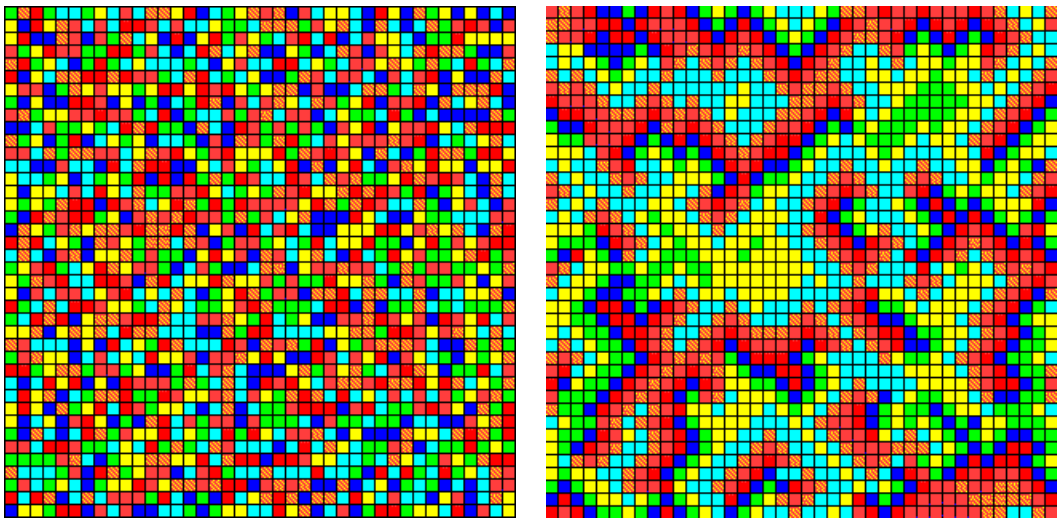


Figure 4.25: Evolution of Demon Cyclic Space CA (first and 20th pattern)

4.6.7 Example: Camus 3D

The above describing CA - *Game of Life* and *Demon Cyclic Space* - are suitable for composition purposes, as Eduardo Miranda[MMH99] demonstrates in two composition programs: *Camus* and *Camus 3D*.

Camus 3D uses three-dimensional versions of both automata in parallel to render music. At each time steps the living cells of the Game of Life automata are used to determine four-note chords, whereby the coordinates of a living cell determine the intervals of the notes to be played. A fundamental pitch is chosen by stochastic methods and then the x position of the living cell defines the interval to the next highest pitch, the y position gives the semitone interval to the next, and consequently the z coordinate makes up the last interval from the second highest to the highest pitch (Fig.4.26).

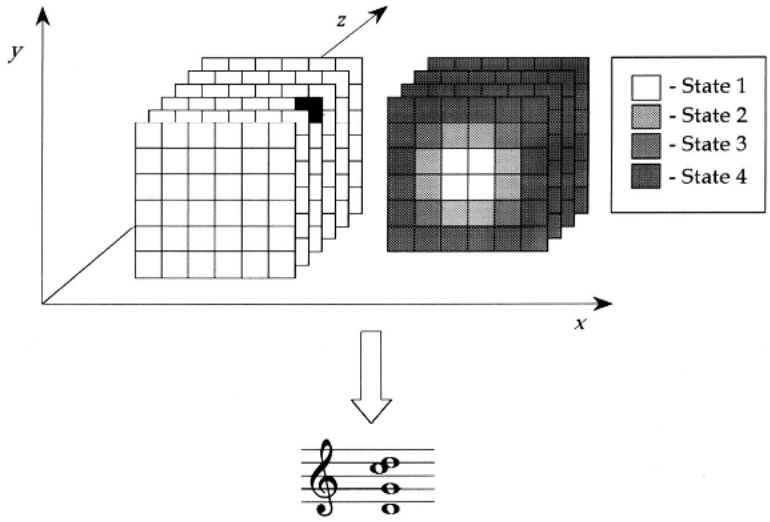


Figure 4.26: Matching of coordinate in *Camus 3D* (taken from [MMH99])

The Demon Cyclic Space determines the instrumentation. The state of the cell that is found on the coordinates of the chosen cell from the Game of Life CA is used to assign an instrument (for example by using a MIDI channel) that will play the events. The four notes that form the event may be played simultaneous or in a rhythmic order, a decision made by a stochastic selection routine choosing any of 24 predefined arrangements [MMH99].

4.6.8 Synthesis with CA

Already by 1990 Jacques Chareyron realized the usability of CA for sound synthesis. He describes a technique named *Linear Automata Synthesis (LASy)* and stresses two positive characteristics: The simplicity of the algorithm and the wealth of musical results achieved by it.
In fact a LASy can be built easily. A wave table stored in memory is taken as a linear automaton, so that every sample is seen as a cell of the CA. The transition rules modify the waveform every time step and the resulting sound is put out. Usually the transition rules are summing the values of the cell and its neighbours and then applying a function $F(x)$ or transition table on this sum. The huge amount of possible CA and therefore sounds that can be generated using this technique is a great potential, however is makes it quite difficult to filter out those that are actually capable to produce interesting sounds. Chareyron gives a detailed explanation in [Cha90].

Another more recent approach was implemented by Miranda in his software *Chaossynth*, that is described in detail in [Mir95].

4.7 Genetic algorithms

Genetic (evolutionary) algorithms (GA) together with neural networks (described in 4.5) are biologically inspired methods that have recently attracted researchers and composers in the field of computer music. As suggested by its name they are used to model the biological processes of evolution, i.e.: the further development of a population of heterogeneous individuals towards an optimal solution - a fitter population. Thus, procedures found in nature, like selection, reproduction and mutation are imitated in these artificial computer systems.

However genetic algorithms can be applied also to many research problems in other non-biological domains. GA may be typically used for search and optimi-

sation tasks, starting from any given non-optimal solutions of a problem and then continuously improving these solutions approximating an optimum in iterative steps. Therefore GA can be seen as directed search algorithms, which constitute their main advantage compared to other existing search algorithms applied in computer science. Usually exploration through genetic techniques is suitable for search problems characterised by an huge search space, where alternative solutions may exist but cannot be calculated easily.
The creative process of a composer - as of a designer - somehow resembles an exploration of many possible compositions in search of an appropriate solution, whereby the definition of such a fit solution is subjected to the composer's taste.
Although in this point of view lies the attraction and advantage of the usage of algorithms for composition in general, this analogy especially signs the technique of GA as an attractive exploration help for computer music composers [GJC03].
This is mainly because of the implication of explicitly or implicitly formulated rules that give the exploration a clear direction compared to the more experimental exploration of changes And effectively a lot of approaches and systems emerged recently that apply GA to artistic tasks.

4.7.1 Introduction

A genetic algorithm operates on a population of many individuals, in which one individual is a representation of a candidate solution. The individual may be composed of separate blocks (called *genes*) where each describes a different property, that together make up a (binary) string encoding the individual's genetic information (*genotype*). The algorithm normally starts with a population generated randomly and then runs iteratively through the stages of *evaluation*, *selection*, *reproduction* and *mutation*, as illustrated in Fig.4.27.

All individuals of the population are first evaluated for fitness, i.e.: their suitability as a solution is measured either according to a pre-described *fitness criteria* or alternatively by the user (composer) himself. The individuals will fulfil the require-

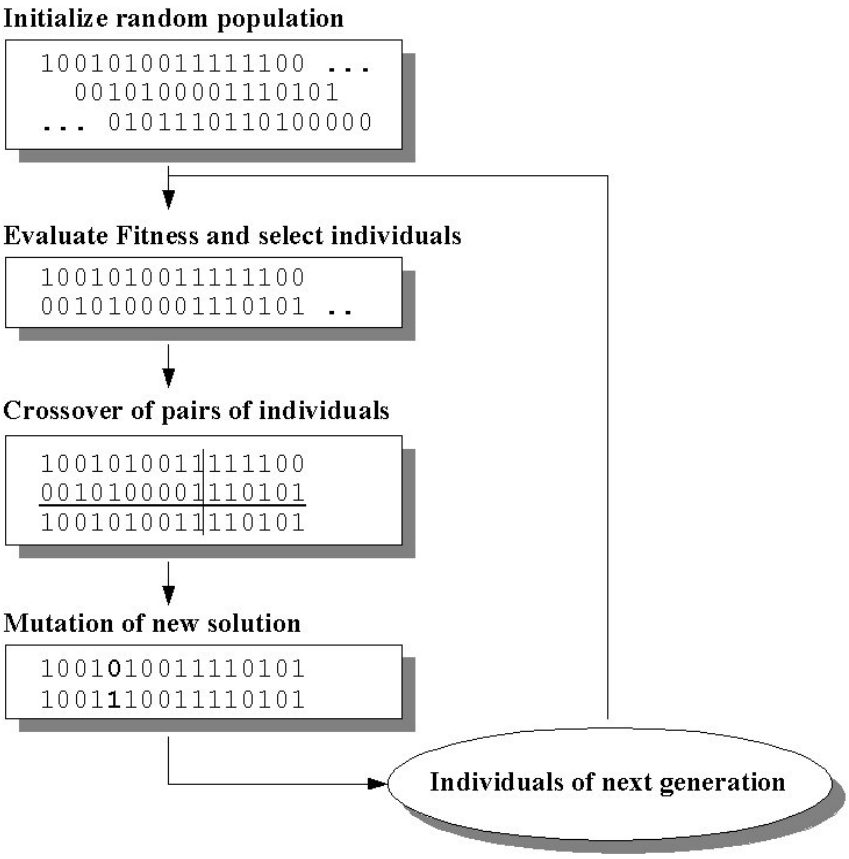


Figure 4.27: Working principle of a genetic algorithm

ments to a greater or lesser degree, which normally also indicates its probabilities for a stochastic selection method, although in other cases one might be interested in selecting the fittest individual directly. The selected individuals form the *mating pool* for the reproduction stage.

During the *reproduction* stage pairs of individuals are formed that give birth to the offspring, whereas these pairs are usually chosen at random out of the pool of candidates. Then a procedure called *crossover* is applied in order to breed a new individual. In a crossover operation the binary strings representing the parents' genetic information are cut at a random length and then mixed together forming the child's genotype.

In the last stage of the evolution cycle the newly created individuals undergo some kind of *mutation*. In the simplest case mutation means just changing a single bit in the individual's string. This final mutation operation is of great importance as it prevents the population from converging prematurely on a suboptimal solution.

The newly created individuals then form the population's new generation and the algorithm starts the next cycle. The fate of the individuals from the last cycle may vary but usually they die out without causing any further effects. In some cases however the fittest parents might stay alive for another cycle(s) [JC02].

4.7.2 Musical context

In order to take advantage of the efficiency and flexibility offered by genetic algorithms for the search and exploration of potentially interesting music material, a suitable implementation of the described stages has to be devised. The main areas that have to be considered to ensure that our algorithm works properly include *search space*, *codification* and *fitness evaluation*.

Search domain

The search space describes the amount of solutions to our composition task. As we might consider a vast quantity of possible candidates to be uninteresting or meaningless at first, it is advisable to restrict the exploration domain of our algorithm by imposing certain constraints upon the decision-making processes. For example when a GA is used to create a sequence of notes, the melody can be determined to contain only notes out of a particular range or within a given scale. In doing so the algorithm is prevented from producing undesirable solutions and its individuals will fulfil the chosen aesthetics [BV99].

Codification method

Another important decision to make concerns the codification method of the musical content as individuals of the population. Typically the codification results in binary

string representations, in which it is desirable to employ the smallest possible alphabet. A simple coding scheme for note events uses blocks of a few bit to describe pitch, dynamics and instruments of a note event, similar to the widely used MIDI protocol [Mir01].

Fitness evaluation

Fitness evaluation may be the most critical area when applying GA for evolving music. The process is responsible for assigning a measure of fitness, i.e.: each individual has to be judged according to its aesthetic value. It is easy to see that the design of such a function is not really straightforward. Different from other application domains of GA, in the field of composition we face a pure aesthetic situation, where a definition of what marks good music is needed. Unsurprisingly, recently much of the discussion and literature about evolutionary methods and art is focused on this question trying to find an appropriate fitness function. Traditionally two different approaches for evaluating an individual's fitness can be distinguished:

- *Automatic fitness assessment (AFA)*, where sufficient knowledge about desirable music is encoded in the system.
- *Interactive genetic algorithms (IGA)*, where a human critic rates the individuals and, thus, decides which ones are selected for reproduction in the next stage [GJC03].

4.7.3 Automatic fitness evaluation

Among systems working with AFA, several different approaches can be distinguished.

Deterministic fitness function

Determined fitness evaluation works by applying mathematical function to obtain a measure of the individual's fitness. In many cases this method is a kind of pattern-matching, where blocks in the candidates are compared to desirable values. An

early attempt to create GA music that way was the work of Horner and Goldberg in 1991 [Gol89]. They investigated the problem of *thematic bridging*, which is the modification of an initial musical sequence towards another defined final destination sequence. Therefore two fitness functions were designed: the first measures the degree of matching note patterns and the second the degree of identicalness between the two statements' durations [BV99].

Formalistic fitness function

Applications using formalistic fitness evaluation make use of a set of particular rules that may be inspired by traditional musical styles for building systems that imitate existing material. Generally these rules determine ranges for attributes measured in the candidate music statement, like duration of notes, pitch changes, spreading, harmonic content, and so on. An interesting approach was described by Thywissen in 1996, who built a composition tool called "Genotator" [Thy99]. In Genotator the composition is represented in a two-layer hierarchy, where the *phrase layer* contains a list of musical events and the *system layer* holds the arrangement of different phrases. The fitness of a phrase is evaluated by rules that define melodic structures, note durations, note densities and interval constraints (refer to [BV99] for a detailed description.)

Measuring fitness using Artificial Neural Networks

The use of *Artificial Neural networks (ANN)* to conduct music evolution offers the great advantage of measuring the fitness of individuals automatically without having to formulate explicit rules for what is good or worthwhile music. As a neural network is capable of learning its decisions are based on the provided training set. An early approach in applying a neural network as a critic for music created by a GA was the "neurogen" system, built by Gibson and Byrne[GB91] in 1991. They reduced the musical domain and therefore the search space of the algorithm to simplify the composition process. Their system *has been designed with a view to producing small diatonic, western-type, four-part harmony compositions using the knowledge*

extracted from a set of example musical fragments provided by the user. The aim has been to produce a piece of coherent music that resembles that typically found in traditional musical hymns [GB91]. Precisely the user provides a set of four bar rhythmic patterns that are translated into a fixed length binary representation and presented to the neural network. Once these examples have been taught to the system, a new random set of rhythmical phrases is created as a starting population for the evolution process. Fitness evaluation is made by two neural networks, in which one (the *Interval Assessor*) concentrates on the characteristic of the intervals between the pitches and the other one (the *Structure Assessor*) examines the overall structure.

4.7.4 Interactive genetic algorithms (IGA)

Interactive genetic algorithms (IGA) describe systems where a fitness value for new individuals is not assigned automatically, but a human being makes the aesthetic evaluation of the presented musical material. Thus user's taste directly conducts the further evolution of the population. However the rule of the critic can be played either by a single user or by a group of persons, like an audience in a interactive concert.

A situation one has to face when constructing an IGA is how much time it takes for a user to evaluate the amount of individuals of a population. This problem was called the *fitness bottleneck*[Bil94] as it slows down the evolution and thus may tire the user who has to listen to a huge number of musical examples. Besides, the systems have a high degree of subjectivity but on the other hand due to the direct interaction of human and computer systems an evolution towards the right aesthetic needs is assured. Furthermore the difficult task of finding rules that describe good music are omitted [JC02].
IGA seems to be the most common GA method in creative domains and systems that implement IGA abound. Examples include Biles' GenJam[Bil03], Jacob's Variations[Jac96], Vox Populi by Manzolli [GMMvZ99], Tasoui Unemi's SBEAT3

[Une], and many others.

4.7.5 Hybrid Systems

In many cases we might use more than one of the described possibilities for fitness evaluation. In doing so we build *hybrid systems* where problems faced in the different approaches can be compensated by integrating features from another approach [SAD+00]. Accordingly, a possible way to overcome the problem of the fitness bottleneck in IGA is to use another fitness evaluation first and, thus, reducing material that has to be assessed by the user himself.

4.7.6 Genetic Programming

The exploration of suitable compositions can be viewed somehow as a search for a computer program that is capable of producing such result. Techniques of *Genetic Programming (GP)* were introduced by Koza in the early 1990s and are based on the same concept as GA, whereas the search space of a GP consists of individual computer programs instead of single musical statements. Therefore GP techniques produce programs or functions that solve the given problem best [Koz90]. Another difference compared to GA is that the search domain is not known in advance and the individuals may differ in size.

An example for GP applied to music is the work of Spector and Alpern[SA95] that consists in a call-and-response system, where a genetic "bebop" artist generates a four measure melody in response to another four measure melody. This improvisation works by applying functions that alter the given input, as transposition, inversion or extension of the sequences. Individuals are formed by different combination of these functions and so produce different variations of the given theme. The fitness value for one individual is measured by comparing the properties of the sequences to a library of known artworks [BV99].

4.7.7 Adaptive games and interacting agents

Instead of applying evolutionary methods directly to create artistic objects, they can be used to evolve agents that themselves will interact with each other in order to work on musical pieces together [JC02]. Therefore they must be able to hear other agents' compositions without having access to their inner structure and musical knowledge. Furthermore no global supervision should be imposed on the direction of the cooperative creative process. This approach resembles the situation of a "jam session" where various artists meet and play, and will have to interact and evolve a performance together [Mir01].

Biles implemented this idea in "GenJam", where he first created and trained several artificial soloists and then let them join in a live performance where they played and improvised with the artist himself [Bil03].
Another approach is described by Gartland-Jones and Copley[GJC03]. Their "Indago-Sonus" System uses blocks, that are combined together, represented in a three-dimensional graphical model. Each block has their own musical home phrase and the ability to play and compose music and then listen to music from other blocks. The user can choose a block that presents its musical phrase to its neighbours that will then start their composition task, a genetic algorithm that uses the home phrase as starting point and evolving towards the passed motive. The block passes its music on to all its neighbours once it has finished the recomposition [GJC03].

4.7.8 Sound synthesis with GA

Like other algorithmic techniques genetic algorithms are also applicable for sound synthesis. Therefore the evolution can either consist in modifying a set of parameters that control the behaviour of known synthesis techniques, as frequency modulation or granular synthesis, or different waveformes themselves make up the population of the genetic algorithm.
Johnson describes a method for exploring sound-space using an interactive genetic algorithm [Joh03]. His software acts as an interface to a Fonde d'Onde Formantique

(FOF) algorithm (a granular synthesis technique). Individuals in the population represent lists of encoded parameters for the synthesis algorithm. Fitness is assessed by the user who can view and listen to the sounds in a graphical interface.

4.7.9 Conlcusion

In addition to the examples mentioned above, there can be found an abundance of other works employing genetic techniques. In the yearly hold workshop of evolutionary music and art (EvoMUSART) new approaches are presented and discussed. The lively interest in GA and art proves this technique to be an absorbing solution for algorithmic music composition. It offers great possibilities and advantage, while it is less restrictive than other methods. However the construction of a GA based system is a complicated and tricky task as it requires a proper representation of the population and its individual, methods for evolution, and the solution for the fitness evaluation. Due the difficulty to explicitly describe fitness function, a more promising concept lies in an interactive evaluation by a user (or an audience) or an evaluation in combination with a neural network. Once implemented such a system allows us to create a lot of varied pieces of music. By slightly modifying the evaluation, the result may be completely different.

Chapter 5

Implementation

Some of the techniques described in the previous chapter were implemented in a library for PD (Pure Data), a real-time graphical programming environment for audio, video, and graphical processing. This chapter describes how the selected techniques were implemented, how they can be used and then moves on to present the obtained results.

Programming languages and enviroments

In principle the implementation of the previously described techniques could be realized in many other environments.

After all any kind of programming language can be used, preferably one with a useful sound and music library - such as jMusic for Java - included when a direct audio output is desired. Special languages for sound programming might be far better options, although in some way more restrictive. Well known systems include Common Lisp Music (CLM), CSound, Haskore.

Furthermore with OpenMusic there exists a powerful visual composition enviroment based in CommonLisp. Like OpenMusic other programs as Pure Data, Max/MSP, jMax and SuperCollider use a visual programming environment, where the composition can be arranged. Of course, all of the listed options have their advantages and disadvantages, which will not be explored here.

Pure Data was chosen not just because of the great possibilities that it offers, but as

well due to the author's level of experience with the program and the implentation of extensions (in the C programming language).

Pure Data

PD was developed by Miller Puckette and company at IRCAM and is free software with existing versions for Win32, IRIX, GNU/Linux, BSD, and MacOS X. As its quite similar commercial successor *Max/MSP*, PD was designed to create interactive computer music and multimedia works.
In order to create his own system the user works on the software's graphical interface. In so called patches objects can be put together and interconnected determining the way of audio signal and control signal streams and how they are processed . Different types of objects are available to perform signal processing tasks, mathematical operations, or access hardware (Audio and MIDI in- and output), etc. For a full documentation about PD refer to [Puc].

A way to extend PD's functionality is to write new object classes (called *externals*), that then can be used just like any other PD-object. This is the way all implementations of this project were realised and so the written externals make up a library (a collection of objects) that can be loaded and used within PD. The decision to implement the algorithms rather as PD-objects than as separate standalone software was taken due to various reasons:

- The objects could be implemented in a quite general way so that they might be used in PD in various contexts.

- PD already offers a huge collection of objects for audio and MIDI processing, that can be used and so do not have to be rewritten.

- The graphical representation in the example patches helps to get a better idea at a glance (compared to e.g. pure source code).

- Interactive real-time composition systems can be easily designed.

- The implemented methods are usable in many different ways and on different abstraction levels.

- The user gets immediate feedback and so can directly observe effects of changes of parameters.

However this approach also comes up with a few disadvantages.

- Some algorithms (e.g. genetic algorithm) need a huge amount of parameters and variables. Numbers can be passed in PD via separate connections between in- and outlets or packed together in a list and sent via one connection. In both approaches however the patches get more complex, as the flow of parameters and its' values respectively can seen easily.

The following sections describe the implemented PD-objects in detail. Illustrated patches should help to understand concepts and usage of the "composition" objects. In many cases MIDI-output is used because results can be displayed easily this way. Nevertheless, the objects can be applied in many quite different ways - chapter 6 provides more insight into other techniques and parameter usage.

5.1 Mapping object - *map*

In order to map output values from the composition objects into a desired range, a mapping object *map* was created. The *map* object has to be initialised with four parameters, describing the range of the in- (a,b) and output values (c,d). Then it applies the linear mapping function described by

$$y = \frac{(x-a)(d-c)}{b-a} + c \tag{5.1}$$

and illustrated in figure 5.1 to the number(s) received on its inlet. Values outside the defined range will not be cut of but scaled accordingly. The rest of the chapter includes various figures of example patches that show how *map* can be used.

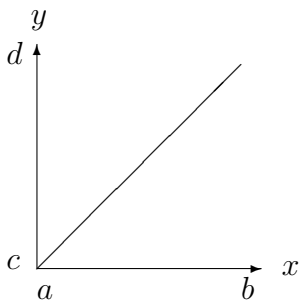


Figure 5.1: Linear mapping function

5.2 Chaos function objects

Implementations of chaotic functions include the *logistic map*, the *Hénon map* and the *Lorenz attractor*, each of them built as a separate PD-object.

5.2.1 Logistic Map - *logistic*

An object *logistic* was implemented in a straightforward way. *Logistic* iterates the logistic function as described in equation 4.6, and outputs its next value every time when a "bang" message was sent to its inlet. The value of the function's variable a has to be passed as argument when the object is created, but can be changed at runtime via the object's right inlet. The values obtained by *logistic* can be further used in different manners within PD likewise any other floating point number.

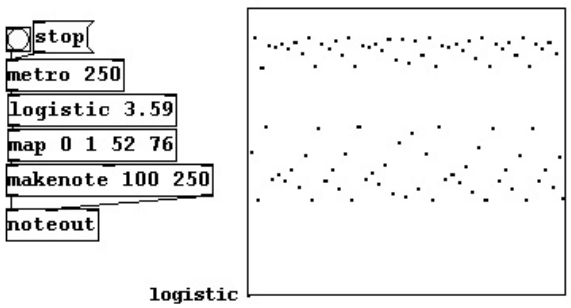


Figure 5.2: PD patch using the logistic map

In order to demonstrate how *logistic* can be applied, the example patch shown in Figure 5.2 was built, where *logistic* is used to create a sequence of notes. The output values of the logistic objects were mapped into a range of two octaves and then sent

to a MIDI out channel. The parameter of the *metronome*-object determines the tempo of the piece. The values visited by the function are drawn to the graph within the patch. The score created this way is printed in Figure 5.3.

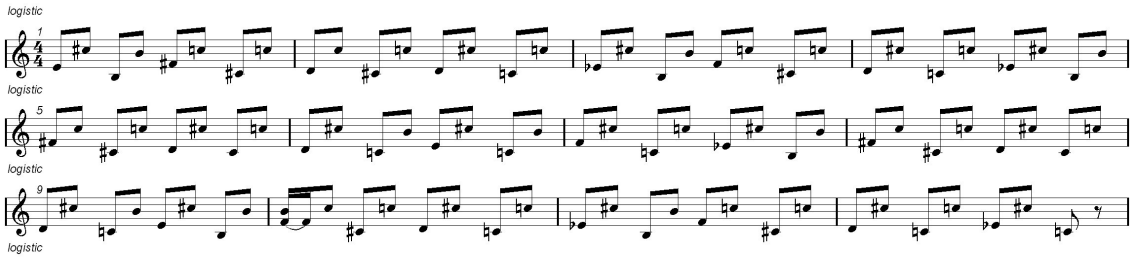


Figure 5.3: Score created by logistic map patch of figure 5.2

5.2.2 Hénon Map - *henon*

The Hénon map is implemented in the *henon* object and works just as *logistic*. When triggered via a "bang" to its inlet it sends the results of the function's next iteration to its outlets. However unlike the logistic function, the Hénon Map is a two-dimensional function. Of course the two output values can be used in a lot of different ways.

The patch illustrated in Figure 5.4 gives an example how *henon* can be applied for composing at the note level. The x-value of the resulting pair determines the pitch of a note to be played, while the y-value describes the interval within an octave for another note to add. In Figure 5.5 the resulting score is printed out.

5.2.3 Lorenz attractor - *lorenz*

The PD-object *lorenz* was built making use of the equation 4.7. It has to be initialised with its parameters σ, R and B set to the desired values. As already mentioned in chapter 4.3.4 the Lorenz attractor is a continuous flow rather than a discrete function but in its computational implementation it has to be evaluated in iterative steps, wherefore a step size δt will be used, which can be set as fourth creation argument (default $\delta t = 0.01$). The values of x,y and z are calculated and output by the three outlets every iteration.

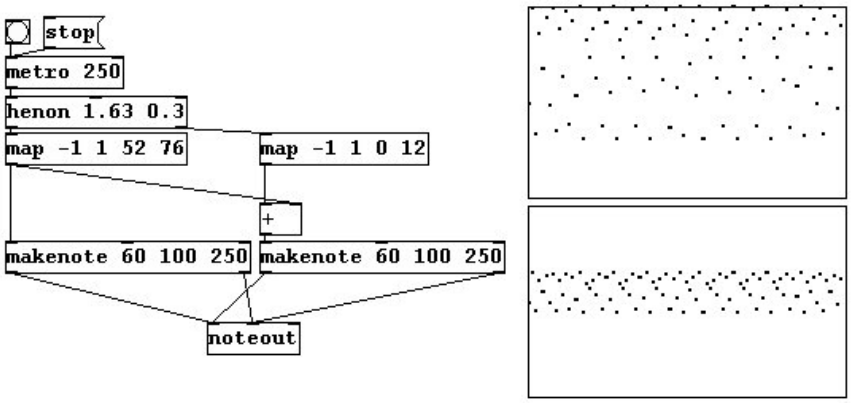


Figure 5.4: PD patch using the Henon map with parameters $a = 1.63$ and $b = 0.3$

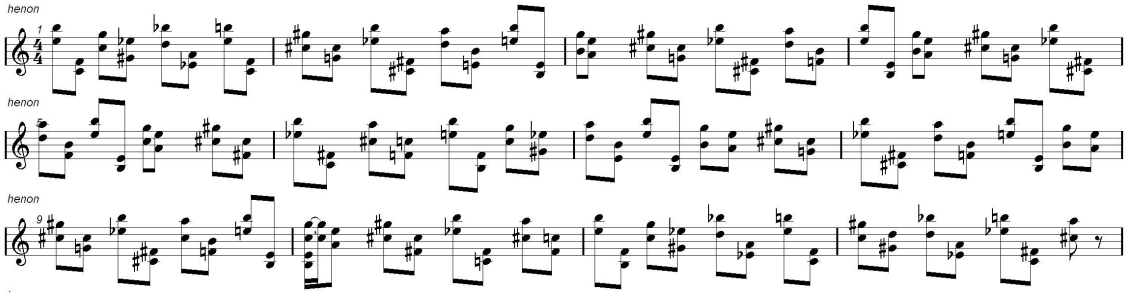


Figure 5.5: Score created by henon map patch of figure 5.4

The patch in figure 5.6 shows a usage example. This time instead of generating MIDI notes, the results of *lorenz* conduct a *frequency modulation (FM)* synthesis. Therefore x and y set the modulator and carrier frequencies, whereas the values z changes the signals amplitude.

5.3 Fractal objects

5.3.1 1/f noise - *oneoverf*

The *oneoverf* object implements a 1/f noise generator. Therefore the Voss-McCartney algorithm, an improved version of the Voss algorithm originally presented 1978 in [Gar78] was used. This algorithm uses N random number generators to create sequences of a length of 2^N. An output value is calculated as the sum of the N random

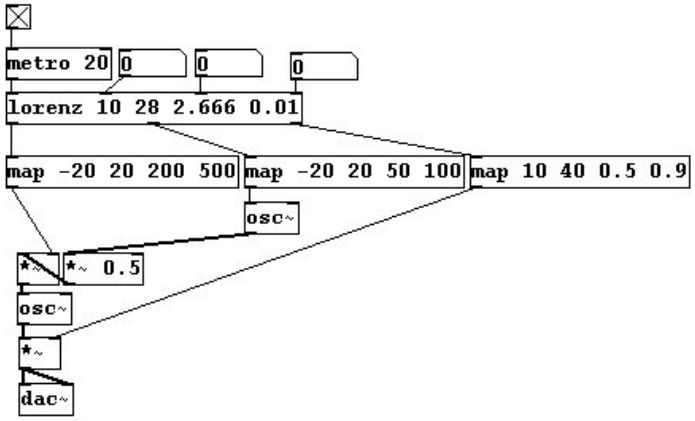


Figure 5.6: PD-patch implementing FM synthesis using *lorenz*

values. The context of past values is maintained as every output value is composed of old and new random numbers. In the original Voss algorithm a random generator $k(0 \ldots N-1)$ creates a new random number every 2^k time steps as illustrated in Table 5.1.

Time	Random number generator			
	0	1	2	3
0	•	•	•	•
1	•			
2	•	•		
3	•			
4	•	•	•	
5	•			
6	•	•		
7	•			
8	•	•	•	•

Table 5.1: Voss algorithm

Time	Random number generator			
	0	1	2	3
0	•	•	•	•
1	•			
2		•		
3	•			
4			•	
5	•			
6		•		
7	•			
8				•

Table 5.2: McCartney-Voss algorithm

The problem with this basic concept is that some values add up a lot more random values than others, which might provoke abrupt changes and thus discontinuities.

As an improvement in 1999 McCartney suggested using a different order for changing the random values, which is illustrated in Table 5.2. Applying this refinement

only one random value is changed every time step. McCartney's approach was implemented in *oneoverf*, in which the object outputs a new value every time it receives a "bang" message on its inlet. The object can be initialised using an argument that defines how many random number generators would be used. This way the appearance of the fractal noise is conducted. Using just one random number results in pure white noise, while a large number of random values generate more Brownian noise.

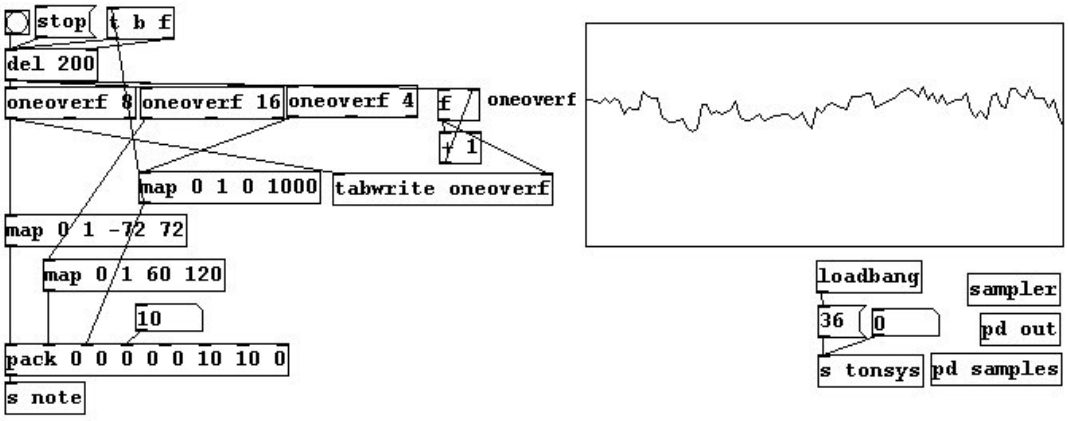


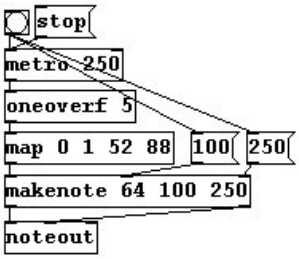
Figure 5.7: Fractal sampler composed with three *oneoverf*-objects

In the example patch shown in Figure 5.7 three fractional noise objects composed by 4 to 16 random number generators are used to conduct a sampler (that is built in a separate sub-patch not shown in the figure). The first *oneoverf 8* object generates the pitch of the played sample, the second and the third *oneoverf*-objects conduct loudness and duration. The graph within the patch shows the evolution of pitch values.

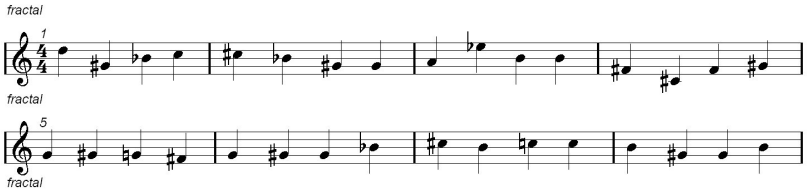
In figure 5.8(a) another PD-patch for *oneoverf* illustrates how to create a sequence of MIDI notes and figure 5.8(b) prints the resulting score.

5.3.2 Iterated function system - *ifs*

With the obje1ct *ifs* a general approach to iterated function systems was implemented. It iteratively applies a scaling and a translation function chosen by chance. The set of functions to be used can be changed and set at runtime. As a function is



(a) Pd patch



(b) Score created by 5.8(a)

Figure 5.8: A PD patch that uses *oneoverf* to generate MIDI notes and corresponding score.

described mathematically by

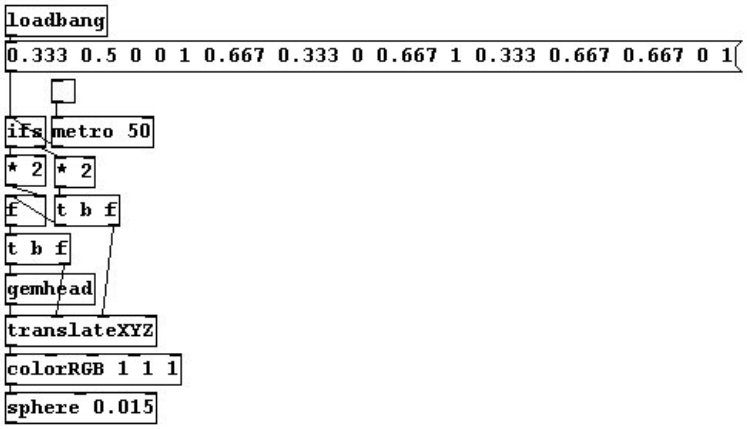
$$F_i \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} s_{x_i} x \\ s_{y_i} y \end{pmatrix} + \begin{pmatrix} t_{x_i} \\ t_{y_i} \end{pmatrix} \tag{5.2}$$

it can be determined by the four values s_x, s_y, t_x and t_y. A fifth value is used to describe the probability of the function to be chosen. So a set of N functions can be passed to *ifs* by sending it a list containing $5 \cdot N$ parameters of the functions. The IFS-algorithm is typically initialised at position $(0, 0)$ and every time a bang is sent to the object's inlet a new pair of coordinates (x, y) is calculated and output.

The example patch printed in Figure 5.9(a) uses a set of three functions, each having equal probability:

$$F_1 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x/3 \\ y/2 \end{pmatrix}, F_2 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 2x/3 \\ y/3 + 2/3 \end{pmatrix} and\ F_3 \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x/3\ + 2/3 \\ 2y/3 \end{pmatrix}$$

The coordinates (x, y) calculated by *ifs* are used to draw a fractal picture in a separate window using the GEM extension for PD (The subpatch that initialises



(a) PD patch that uses GEM to draw a fractal picture

(b) Painted fractal image

Figure 5.9: PD patch and the resulting fractal image painted after hundreds of iterations

GEM isn't shown in the figure. After banging *ifs* a few hundred times the picture of figure 5.9(b) was painted.

5.3.3 Iterated function system for music - *ifsmusic*

In *ifsmusic* the concept of IFS was adapted to work better with musical information itself instead of operating on two-dimensional coordinates. Like *ifs*, the *ifsmusic*-object uses scaling and translation operations. But as *ifsmusic* is provided by a sequence of N notes it applies one scaling factor to resize all intervals in the sequence and shift the notes by a translation value.

$$F_i \begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{N-1} \end{pmatrix} = \begin{pmatrix} x_0 \\ s_i \cdot (x_0 - x_1) + x_1 \\ \dots \\ s_i(x_0 - x_{N-1}) + x_{N-1} \end{pmatrix} + \begin{pmatrix} t_i \\ t_i \\ \dots \\ t_i \end{pmatrix}$$

As in *ifs* a set of functions can be set in the *ifsmusic*-object by sending it a list of the parameters. However in this case a function is described by just three variables, a scaling factor, a translation value and a probability

5.3.4 Self-similarity - *selfsimilar*

selfsimilar plays with selfsimilar melodies. Therefore it takes a melody and mulitplies it at different temporal scaling levels. That means that upon each note in the original sequence the whole melody is duplicated, resized to the length of one note in the base melody and translated by a given interval. The same procedure will be repeated upon the melody at this new level for creating another melodic line that is further rescaled and translated, and so on. The length of the note sequence at a level k may so be calculated as n^k, where n is the number of notes in the original melody.

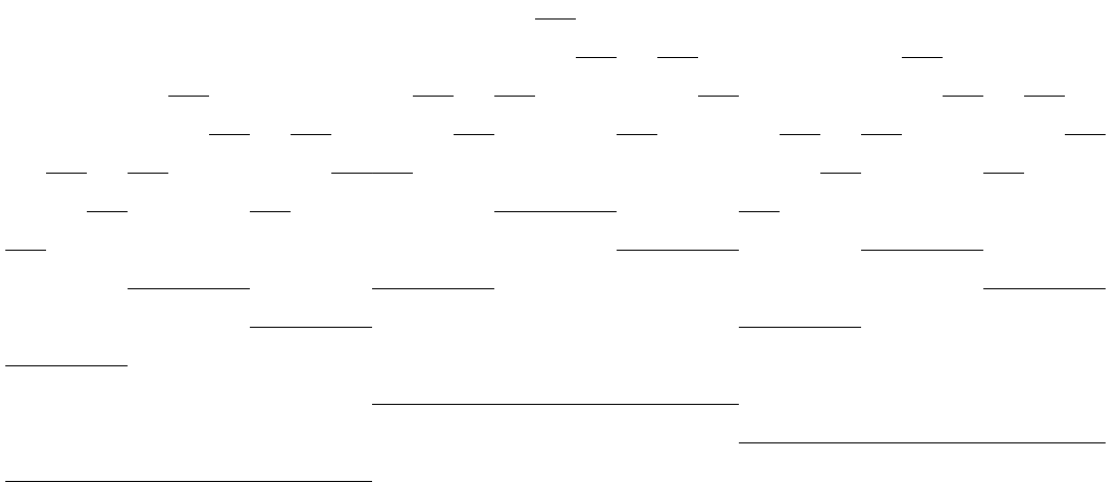


Figure 5.10: Selfsimilar melody composed at 3 layers

This principle (illustrated in Figure 5.10) is used by the implemented object *selfsimilar*. When initialised a number of parameters have to be set: the first argument determines the number of layers of the composition, the second one sets the interval used to shift a higher layer. The rest of values in the argument list describe the base melody. The resulting melody is sent to the object's outlets note by note, whereby each layer uses its own outlet.

Figure 5.11 shows a patch applying *selfsimilar* in order to create a 4-layered composition that is based on the sequence C-E-H-G and where each layer is shifted up by a quint.

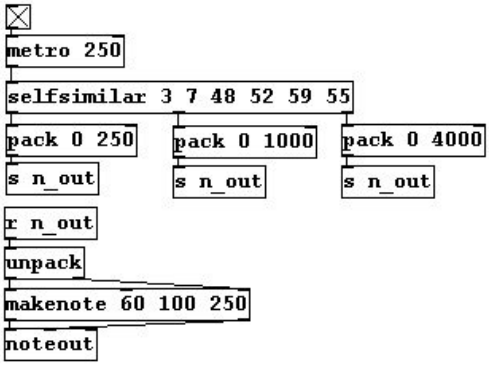


Figure 5.11: PD patch that creates a selfsimilar melody

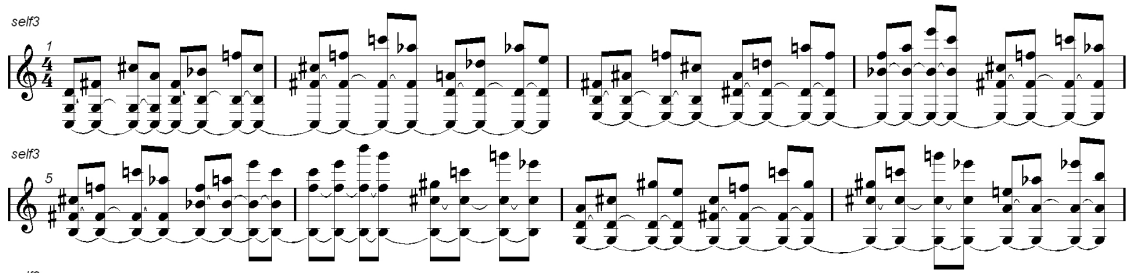


Figure 5.12: Selfsimilar score created by patch shown in figure 5.11

5.4 Elementary cellular automata - *eca*

The *eca* object implements an elementary cellular automaton (as described in Chapter 4.6). The CA used in *eca* has a neighbourhood of $r = 1$, which means that there are 256 possible evolution rules. The size of the CA has to be set by the user, whereby it ranges from 1 to 255 cells. These two configuration options have to be passed to the object as argument - the rule to be applied in decimal form $(0 - 255)$ and the size of the array of cells. Furthermore the object takes an additional argument that specifies the cells' initial states either to be calculated by chance (0) or just bring one cell (in the middle of the array) to life (1).

Once initialized every time step the object calculates the new states of the cells according to the supplied rule and sends the result - a list of the positions of all activated cells - to its outlet.

In the example patch in Figure 5.13 an elementary CA with an array size of 9 is

connected to the MIDI channel 10 (usually reserved for percussion instruments). Table 5.3 illustrates the so created sequences for the supplied rules 129 and 110. The points indicate when which instrument has to be triggered. The example demonstrates the known feature of a CA to produce repetitive and complex structures, a characteristic which is highly interesting for the composition of rhythmic patterns.

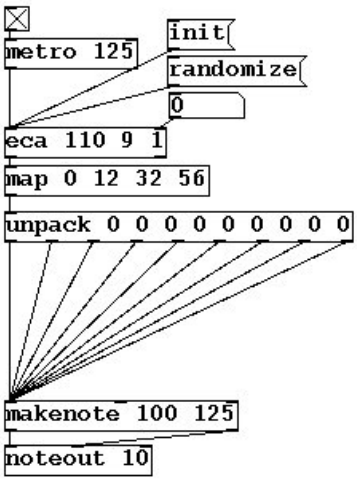


Figure 5.13: Cellular automata drummer using the *eca* object

(a) Rule 129 (one cell initialized)

Time	Instrument								
	1	2	3	4	5	6	7	8	9
1					●				
2	●	●	●				●	●	●
3	●	●			●			●	●
4	●								●
5			●	●	●	●	●		
6	●			●	●	●			●
7					●				
8	●	●	●				●	●	●
9	●	●			●			●	●
10	●								●
11			●	●	●	●	●		
12	●			●	●	●			●
13					●				
14	●	●	●				●	●	●
15	●	●			●			●	●

(b) Rule 110 (initialised randomly)

Time	Instrument								
	1	2	3	4	5	6	7	8	9
1	●	●	●			●			●
2			●		●	●		●	●
3		●	●	●	●	●	●	●	●
4	●	●							●
5		●						●	●
6	●	●					●	●	●
7		●				●	●		
8	●	●			●	●	●		
9	●	●		●	●		●		●
10		●	●	●	●	●	●	●	●
11	●	●							●
12		●						●	●
13	●	●					●	●	●
14		●				●	●		
15	●	●			●	●	●		

Table 5.3: Rhythmic patterns generated by *eca* for 9 percussion instruments

Chapter 6

Composition

This chapter discusses considerations that have to be taken when implementing a composition system or creating a musical piece. Furthermore, “ChaosfractalicCA” - a composition work realised in Pure Data applying the implemented “compositional” objects (introduced in chapter 5) - is described.

6.1 Design issues

Important issues for the composer working with algorithms are the decisions on the parametric representation and on the global structures of the music that he wants to create.
This means that an architecture composed of various composition levels (introduced in chapter 3) has to be designed, where a set of different parameters are involved.

Normally the results of algorithms that operate on a higher level conduct the setup and evolution on the lower level, and then the music is put out at the note or microscopic level. The composition may either create a score, generate an audio file or directly play the created music. In any case a proper representation is required, i.e.: a set of parameters that describe the resulting music. Typically parameters like pitch (frequency), note duration, loudness and timbre will play a role on this level. Then, when the implied parameters are chosen, a decision about which algorithmic technique(s) to use to conduct which parameter(s) has to be made. Furthermore

mapping concepts have to be found so that the algorithmic results fit the desired range of the music parameters. Events on a higher level will use other kind of parameters that determine a global behaviour by changing attributes of whole blocks. Of course, these parameters include musical attributes like tempo or register. Moreover other parameters may be implied that change the configuration of algorithms on lower levels - like variables for chaotic functions, probability distributions, rules for a CA, etc.
The next section describes the composition "ChaosfractalicCA" and in doing so provides suggestions on the usage and mapping of parameters, suitability of different algorithms and structured design.

6.2 "ChaosfractalicCA"

"ChaosfractalicCA" is a piece of music that was generated with a therefore designed PD-patch (illustrated in figure 6.5). The created patch can be seen as an exploration space for possible compositions, because as random methods are implied actually created pieces will diverge. Furthermore by changing mapping functions and setup parameters the composition can be modified easily.

The patch uses a division in *sections*, which are further composed by *blocks*, where note events are created, which then are used by a few synthesis instruments. The implied parameters and algorithms on the different levels are explained following a bottom-up approach (i.e.: starting with the details).

6.2.1 Synthesis instruments

The sound repertoire of the composition consists in four different synthesised instruments that as well serve as the four voices used in the piece. All of them are constructed by fractal or chaos objects and they slightly change their timbre every time they are played. Depending on the applied synthesis techniques different parameters are invoked.

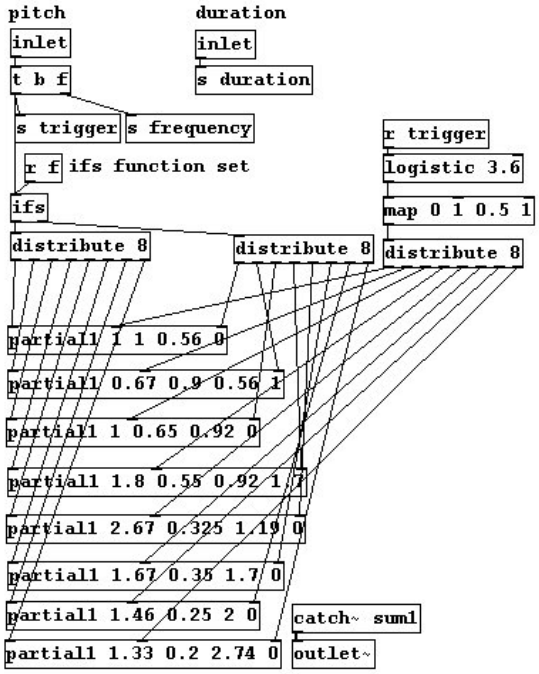


Figure 6.1: Abstraction patch for the ifspartial instrument

Additive synthesis with IFS - *ifspartial*

In the abstraction patch *ifspartial* an *ifs* object is used to build an additive synthesis instrument. The idea of this design is to use an IFS to create a sound space out of coordinates of the emerging two-dimensional fractal image. The x and y values returned by *ifs* determine relative frequency and relative duration of a partial (figure 6.1), which are derived by directly mapping the values.

An additional logistic function with its parameter $a = 3.6$ was added to implicate slight chaotic changes in the partials' amplitudes. The result values of *logistic* are mapped to amplitudes that lay between 0.8 and 1.

In total the additive synthesis is made up of eight partials, whereby the parametric setup for just one of them is renewed every time the instrument is played. So every new note changes one partial of the sound and thus every eight notes the timbre of the instrument is completely renewed - although depending on the used IFS function

set the actual timbre may sound similar to the old one.
Furthermore the *ifspartial*-instrument receives new function sets for *ifs* every once in a while (when the piece enters a new section, see section 6.2.5) and so changes its sound more significantly.

Amplitude modulation (AM) with the Hénon map - *henonam*

The *henonam* instrument is built by applying the amplitude modulation (AM) technique. Variables derived from the Hénon map (*henon*) are used to determine the *modulation frequency* as well as the *modulation index* for the AM. Therefore the *modulation frequency* is obtained by directly multiplying the resulting value of x of *henon* with the carrier frequency that is sent to the inlet of the patch, every time a note event for the instrument is generated. The modulation index is derived from the value of y that is directly mapped to an integer value between 1 and 5 (see figure 6.2).

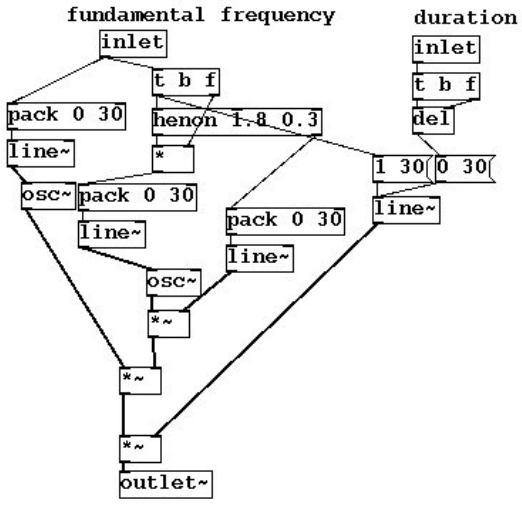


Figure 6.2: Abstraction patch for the henonam instrument

Frequency modulation (FM) by an IFS - *ifsfm*

The instrument *ifsfm* (illustrated in figure 6.3) is composed by an IFS that controls the parameters of a frequency modulation. Therefore the x coordinate of the IFS is

mapped from the range 0 to 1 to a modulation frequency in the range of 200 to 400 Hz, and the y coordinate is rescaled to a modulation index as an integer between 1 and 5. The carrier frequency for the FM is set from outside by the note events that are received by the inlet of the abstraction patch.

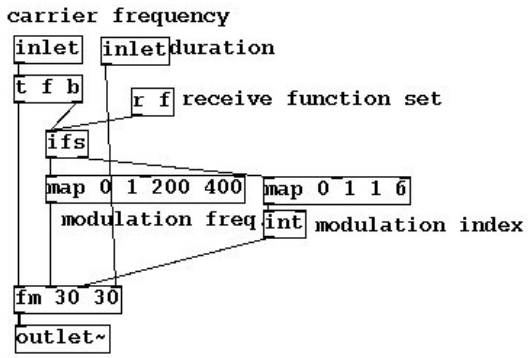


Figure 6.3: Abstraction patch for the *ifsfm* instrument

FM with the Lorenz Attractor - *lorenzfm*

The approach implemented in the *lorenzfm* instrument uses the frequency modulation technique (FM). All the parameters of the FM derive from the Lorenz Attractor, which is implemented in *lorenz*, where x describes the *carrier frequency*, y the *modulation frequency* and z the *modulation index* (see figure 6.4). All these coordinates are mapped linearly: x is mapped from the range -20 to 20 to the interval 220 to 440 hertz, y from -20 to 20 into 220 to 880 hertz. In order to get the modulation index the z-coordinate is rescaled from the range 10 to 50 to integer values between 1 and 5.

Every time the instrument is "banged" new values for these parameters are calculated. This way the configuration of the FM changes according to the curves drawn by the Lorenz attractor and the resulting sound tries to make this movement of the attractor hearable.

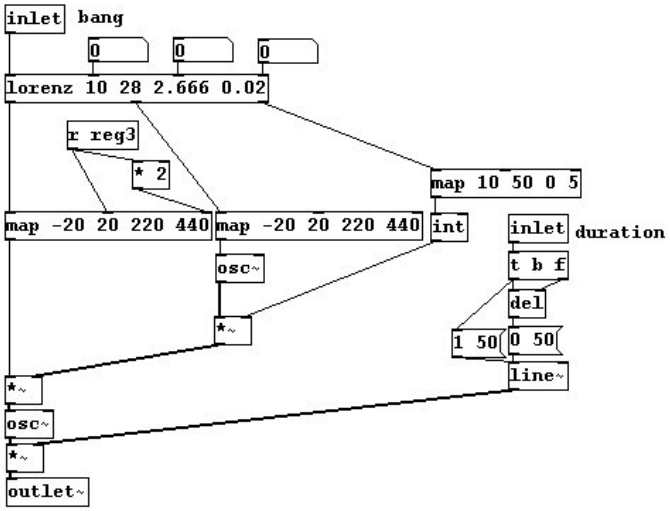


Figure 6.4: Abstraction patch for the lorenzfm instrument

6.2.2 Events on the note level

As long as one block lasts the activated instruments receive note events, whereby different algorithms are used to create the melodic and rhythmic information for each voice. Used parameters are pitch, loudness, duration and starting times. Furthermore other parameters within each instrument as described in the previous section (as modulation frequency, modulation index,..) are invoked in this level.

Pitches

Various algorithms are used for each instrument to obtain the pitches of the notes to be played.

- Pitches for *ifspartial* are conducted by a fractal noise generator *oneoverf* that was initialised with four random number generators. Values are mapped to a frequency interval of one octave, which changes every new section according to the received register values.

- The fundamental pitch (carrier frequency) for the *henonam* is set by the x coordinate of an IFS. The mapping process and the function set used by *ifs* are subject to the register value and IFS function set of the actual section.

- The idea of selfsimilar melodies is realised for *ifsfm*. Therefore a *selfsimilar* object is used, which sets up three layers and is based upon a melody of four notes. This base melody is expressed in form of MIDI notes: $42, 49, 44$, and 53.

  The transposition intervals between the different voices (layers) are three semitones (a minor third). The three outlets of *selfsimilar* are connected to three instances of *ifsfm* which in this way receive the pitch information.

- The frequency information for *lorenzfm* is included in the patch of the instrument itself, as it follows the curve drawn by the Lorenz attractor.

Loudness

The outlets of the patches of the four instruments are connected to *output* objects, which allow changes of their amplitudes in terms of decibel. These decibel values are obtained by linear mapping. However, as decibel itself is a logarithmic scale, this way the whole mapping process is non-linear.

- Loudness values for *ifspartial* are distributed according to the Gaussian distribution[1], which is described by its mean value 94 decibel and a standard deviation of 2.5 decibel.

- For *henonam* a logistic map is used for the creation of loudness values that are mapped to values between 87 and 96 decibels. The variable a of the logistic function is modified every block, moving the logistic function to a more chaotic behaviour and thus producing more changes in loudness with the progression of the composition.

- In order to obtain loudness for the *ifsfm* instruments another *selfsimilar* object is used, which likewise the one used for the pitch information is composed by three layers and uses a series of four loudness values: 86, 80, 84 and 82. The

[1] In order to get values that correspond to a Gaussian distribution the *gauss* object that is already implemented in PD was used.

values decrease by 1 for higher layers, i.e.: the voices playing lower notes will sound louder than higher voices.

- Loudness values for *lorenzfm* are conducted by a chaotic function (the logistic map). T hey will behave likewise describes above (for *henonam*), although they are mapped differently (to the range of 78 to 82 decibel) and the variable *a* in *logistic* is not exactly the same, but however following the same tendency - an increase in variation.

Duration

The abstraction patches for all instruments are constructed in a way that allows sending them durations (in milliseconds). The following techniques are used to obtain the corresponding values:

- Duration values for *ifspartial* are conducted by a fractal noise generator *oneoverf*, and will lie in the range of 1 to 4 seconds.

- The *y* coordinate of the *ifs* determines the duration of a note event. Linear mapping is applied to obtain duration between 400 and 800 milliseconds.

- Durations for the highest voice of the selfsimilar *ifsfm* instruments are mapped from the results of a logistic map to a range that depends on the setup of the corresponding section. The two other voices use the double and fourfold duration of the one set for the highest voice. Thus lower voices will sound longer.

- The *lorenzfm* instrument will play continuously, that is one note lasts until the next one is received. So the duration is the difference of the starting times of the actual and the next note.

Starting times

A particular logistic function (*logistic* object) is used for every voice to obtain the time-lags between the generated note events. The configuration of these logistic

maps, that is their *a* variables, changes every block, and thus, enforces rhythmic variation. The mapping process is linear but the actual range depends on the section setting, where a parameter *tempo* is used and is newly generated every section, provoking faster and slower sections. The tempo changes follow a tendency that is described 6.2.5.

6.2.3 Higher-level structures

The composition is made up of sections, which are further subdivided into several blocks. A section is described by a set of parameters that determine behaviour and appearance of its underlying blocks. The number of sections and the average number of blocks have to be set before starting the composition and, thus, affect the length of the created piece.

6.2.4 Events and parameters on a block level

A block is responsible for activating selected voices and for rhythmic changes. Therefore the following actions take place for each block:

Activation of voices

The voices/instruments that are active and thus will play and sound within the actual block are derived from the living cells of an elementary CA (using *eca*). Therefore the first four cells of the array are associated with the four instruments (in this order: *ifspartial*, *henonam*, *lorenzfm* and the selfsimilar *ifsfms*). The state of the corresponding cell determines the activation of the instrument. A living cell causes the instrument to play and a dead cell mutes the voice. State changes occur every new block according to the rule used within the section.

Block duration

The duration of a block is derived from a Gaussian distribution and is renewed for every block. The Gaussian distribution is described by its mean value of 3 seconds

and its standard deviation of 2 seconds. However values below 1 and above 5000 seconds are cut off.

Rhythm

The parameters a for the logistic functions implied in rhythm generation on the note level are renewed. Therefore two values have to be calculated, one to be used in the *logistic* objects for *ifspartial* and *henonam*, and the other one in *ifsfm* and *lorenzfm*. The new values are taken from a *beta* distribution[2], whose actual shape evolves and is renewed for each section. The beta distributions provide values between 0 and 1, which are mapped linearly to the range of 2.5 and 4.

6.2.5 Events and parameters on the section level

With the beginning of a new section a set of parameters are renewed. Several actions are taken that assign values to a number of parameters to be used by objects that belong to the block or note level.

Elementary CA rule

A *rule table number* for the elementary CA is set. Rule numbers for *eca* are defined in a series that after each iteration undergoes some transformation, which is realised with the *ifsmusic* object. At the beginning of the piece *ifsmusic* is initialised with a series of four values ($56, 111, 67$ and 75) - rules that have been tested to produce interesting complex patterns - and the transformation functions for *ifsmusic* are set. The series may either be inverted, shifted by 1, or played retrograde, whereby the operation to apply is selected by chance and all the transformations have the same probability. According to the principle of *ifsmusic* the transformed series serves as source for the following transformation.
When elementary CA receives a new rule number it changes its transition rules and moreover it is reinitialised randomly, i.e.: The states of its cells are reset by random.

[2] An object *beta* was used that is included in PD.

Function set for IFS

A *function set* for the IFS (used in instruments) is selected. Four defined function sets are available each consisting of three functions that combine a scaling and a translation operation and where each function has a probability of 1/3 to be selected. The following function sets were chosen as they were tested to produce pleasuring fractal images:

$f_{1,1}(x,y) = \frac{1}{2}(x,y)$, $f_{1,1}(x,y) = \frac{1}{2}(x,y) + (\frac{1}{2},0)$, and $f_{1,3}(x,y) = \frac{1}{2}(x,y) + (\frac{1}{4},\frac{1}{4})$.

$f_{2,1}(x,y) = \frac{1}{2}(x,y)$, $f_{2,2}(x,y) = (x,\frac{1}{2}y) + (\frac{1}{2},\frac{1}{2})$, and $f_{2,3}(x,y) = \frac{1}{2}(x,y) + (0,1)$.

$f_{3,1}(x,y) = (\frac{1}{3}x,\frac{1}{10}y)$, $f_{3,2}(x,y) = (\frac{2}{3}x,\frac{1}{3}y) + (0,\frac{2}{3})$, and $f_{3,3}(x,y) = (\frac{1}{3}x,\frac{2}{3}y) + (\frac{2}{3},0)$.

$f_{4,1}(x,y) = \frac{1}{2}(x,y) + (\frac{1}{4},0)$, $f_{4,2}(x,y) = (\frac{1}{2}x,\frac{1}{4}y) + (1,\frac{1}{2})$, and $f_{4,3}(x,y) = (x,\frac{1}{2}y) + (0,\frac{1}{2})$.

One of them is chosen by random and is sent to all *ifs* objects in the patch and its sub-patches.

Registers

Three *registers* values are chosen randomly using a uniform distribution. These values are used to set the frequency mapping function applied in the different voices and so determine the range of the played pitches, which will lay between the register value (in Hz) and one octave higher. Furthermore different uniform distributions are used for the instruments, resulting in register frequencies between 20 and 1200 Hz for *ifspartial*, between 20 and 400 Hz for *henonam* and between 20 and 800 Hz for *lorenzfm*.

Beta distribution for rhythm

As mentioned above in the block calculation a beta distribution is used for the setting of the parameters a of two logistic maps that determine the rhythmic behaviour. This beta distribution is modified every section by changing its parameters a and b. Changes are made so that *beta* is more likely to return low values at the beginning of the composition and then with progressive duration of the piece the distribution is modified in order to contain more higher values. Therefore the functions $a = 2i/N$

and $b = N/2i$ are used (N - Number of section, i - current section).
This modification of the beta distribution results in more order for the first sections of the piece and an increasing chaotic behaviour when reaching the end.
This movement is caused by the beta distribution's increasing tendency towards high values. The values of *beta* are mapped to values between 2.95 and 4 for the parameter a in the *logistic* objects of one block. As discussed in section 4.10 the results of a logistic map approximate a fixed-point attractor for low values, and generally turn to cyclic or chaotic behaviour for higher values (with the exception of a few regions). As the logistic map conducts rhythm, this tends to become more and more complex with ongoing duration of the piece.

Tempo

A *tempo* parameter is used and renewed for every new section. Its value determines the range of the mapping processes that scale the results of the logistic functions implied in rhythm generation. Therefore a beta distribution is used, which changes its shape during the composition - likewise described above. However, in this case the distribution tends to return high values (low tempo) at the beginning of the piece, changes toward a distribution dominated by low values until the middle of the piece and then evolves back again. The parameters a and b are calculated by $a = \frac{4|N-i/2|+1}{N}$ and $b = 1/a$.

Section duration

The duration of a section is not described in seconds but rather as a length in number of blocks in the section. In order to calculate this number a Poisson distribution (section 4.7) is used. Its mean value is the average number blocks per section introduced for the composition before starting the generation process.

6.2.6 Motivation

The composition is mainly inspired by chaos theory and fractal geometry. A change in the parametric configuration of the chaos functions results in movements from

order to disorder. The described functions that change the shape of the beta distribution invoked in rhythm generation set up a tendency towards increasing values of a for the *logistic* objects and this way assure the direction of the movement.
Fractal-like structures are introduced by melodies generated by fractal noise and iterated function systems for sound synthesis. Furthermore the concept of self-similar structure is brought to the music domain with the implemented self-similar melodies (using self-similar pitches and loudness values).
The elementary cellular automaton builds up the structure of active voices, whereby the changing rule set causes structures that range from simple repetitive to complex, expanding, or disappearing patterns.

The idea and motivation of the piece is to set to music fractals and chaos and to use these techniques to establish structures (in rhythm, melody and arrangement of voice) just to overthrow them instants later and replace them with others. The implied fractal, chaos and serial objects lead to a composition, where the actual outcome is not clearly predictable, while it will always stay within the known behaviour and range.
Therein lays the great potential of algorithmic composition in general. The composer may set the rules and the framework of a composition. Then possible pieces of music within these structure can be explored.

The created piece music "ChaosfractalicCA" can be found on the accompanying CD.

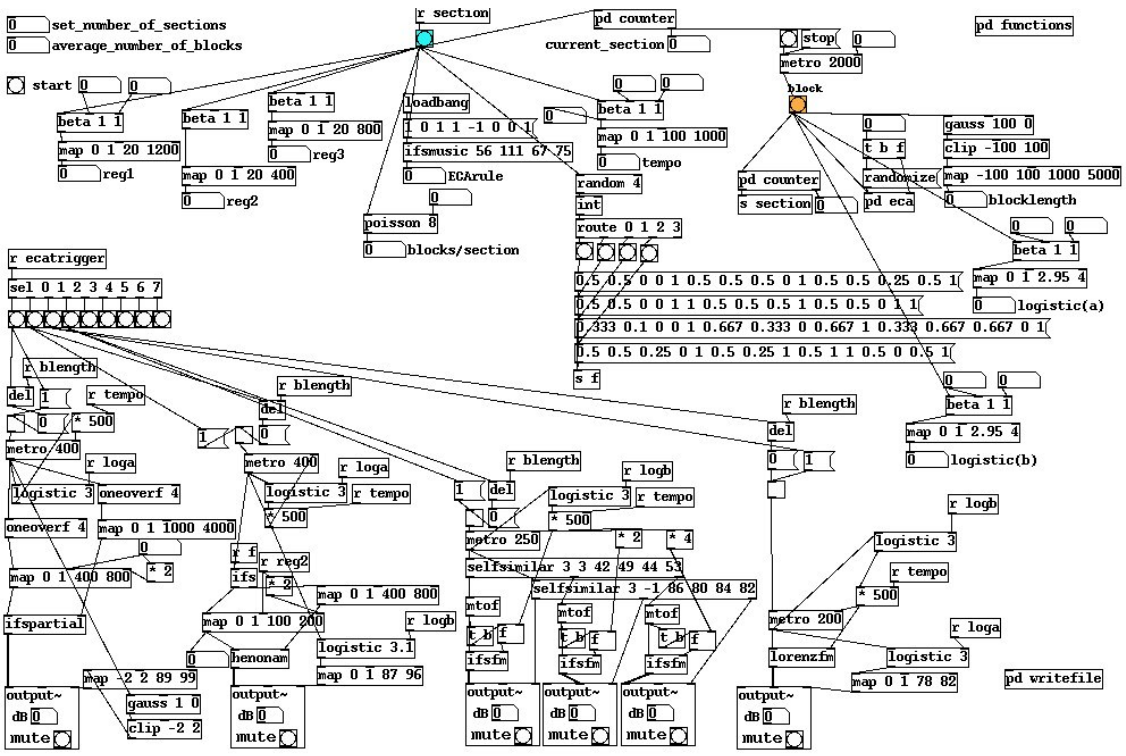


Figure 6.5: The composition patch.

Chapter 7

Conclusion

Musical composition - the process of creating a piece of music - is a task that requires creativity and imagination. Algorithms on the other hand are defined methods that normally do not have any straight musical relevance.
During the last century science and art began to influence each another. Especially Vienna was the starting point of a development that uses strict mathematical rules to control various musical parameters which lead to the terminus: Algorithmic Composition.
The present work asks the question how composers can benefit from algorithms by the exploration of their characteristics and potentials and faces the problem how the described methods can be implemented in a real-time environment.

The use of formal composition methods and specially designed machines for composition looks back at a long tradition, but electronic music and studio devices like the tape recorder and the (sequence-controlled) synthesiser promoted renewed composition experience by providing the composer with new possibilities of recording and playback. Modern computer technology empowers the composers with a lot of new options to explore and test compositions fast and efficiently using sample material, synthesiser and, of course, algorithmic techniques.

An important step in algorithmic composition is the mapping between results of the algorithm and the final musical output. The followed approach distinguishes three different levels. On the microscopic level with a range of fractions and microseconds the composer works with microscopic properties of sound itself. The note level works with note events as basic elements. On the building-block level patterns formed out of multiple notes as well as global behaviour can be arranged.

Algorithm results are numeric values that have to be translated into musical parameter afterwards. Depending on the level where they are used, these implied parameter consist of variables for synthesis techniques on the microscopic level or basic attributes of single notes like pitch, duration, timbre, etc. on the note level. On a higher level algorithms can be used to select patters or instruments or conduct global attributes like tempo, density, etc.

Mapping is done by applying mapping functions that fit the numeric values into a range where they are useful for the composer's purpose. Another important decision is the representation of the final composition. It can be represented in many ways, like a score-like form of representation, direct sound output, information coded for specific music software, or in form of the own program. Thus in many cases the composition program replaces the score and produces its own auditable output. This is applied in the author's composition work, where the composition consists in a PD-patch, alterable by a set of parameters.

The research work and the composition show potentials and possibilities of selected algorithms. Of course in the field of arts there are no restrictions how to use the described techniques or different algorithmic methods, and there is an abundance of application options not explored in this paper.

The relevance of the appliance of algorithms in composition lies in the composer's desire to formalize the composition process. Moreover the presented techniques offer a way for the search and exploration of possible compositions - in a more efficient manner than when composing "by hand".

However, a general approach how to benefit from the described algorithms in music cannot be offered, rather the composer needs to have the behaviour and potential

of a technique in mind in order to make use of it in composition or synthesis.
On the one hand, the described mathematical techniques as probability, chaos functions, and fractals provide a possibility to conduct the behaviour on various levels for a composition. In doing so, the outcome strongly depends on chosen parameters for the applied functions and how results were represented musically.
Biologically inspired methods like neural networks and genetic algorithms, on the other hand, imply a more general potential, in which composition can take place without explicitly formulated rules and without actually knowing the direction of the composition's evolution.
Consequently, these techniques are powerful, more complex and, therefore, considered a promising field, which still attract lots of research and experimental work.
The studied examples, together with the implementation and their application, point out the possible benefits of the different techniques. These potentials described below should be seen as suggestions rather than restrictions.
While Probability distributions offer a simple but useful way to conduct the range of parameters or note events, Markov chains extend the concept by implementing a conditional probability and thus a kind of small memory.
Chaos functions have a known behaviour, so that by applying changes to the implied variables chaotic or predictable cyclic behaviour con be provoked.
Self-similarity and scaling invariance are characteristic properties of fractals and can be brought to music, e.g. by creating self-similar melodies. The attractiveness of Iterated function Systems for composition lies in the possibility to carry the geometric transformations directly into the field of music.
Elementary one dimensional CA are useful for the creation of complex structures for rhythmic patterns or the control of global blocks. Another appliance is sound synthesis.
The main advantage of neural networks is their ability to learn. Thus they are able to learn from given music and copy different styles. Furthermore interpolation between different trained compositions is possible.
Genetic Algorithms were designed as a powerful search algorithm for optimal solu-

tions, in the field of music it is a useful exploration method. User Interaction or neural networks are a powerful addition that may be added to evaluate and conduct the algorithm.

With the *algocomp* external a widely usable library for the graphical programming environment PD was created. The decision to implement the methods within this environment offers the advantage that the programmed objects can be used in various ways. So they can be employed in different levels of a composition, like sound synthesis, note generation or higher composition structures. However in this way, a lot of decisions are left to the user of the program. The supplied help-patches provide examples of appliances.
Furthermore the composition shows how the implemented methods can be used in order to compose music and how the advantages and potentials of the particular techniques can be exploited. Together with the theory, the patches and the composition provide the user with the knowledge necessary to create their own musical works.
The path for composition "ChaosfractalicCA" however cannot be seen as an explicit composition, but it builds up a program, that generates an endless number of pieces of music that vary because of the implied random and probability function and changes in parameter values.

Generally many present approaches deal with genetic algorithms or techniques implying artificial intelligence. They offer great possibilities and advantage, while they are less restrictive than other methods. However the construction of a GA based system is a complicated and tricky task as it requires a proper representation of the population and its individuals, methods for evolution, and a solution for the fitness evaluation. Due to the difficulty to explicitly describe fitness functions, a more promising concept lies in an interactive evaluation by a user (or an audience) or an evaluation in combination with a neural network.
As these methods are by far not fully explored and exploited in the field music, future scientific and artistic works might follow this research approach.

Future work in the *algocomp* library should include, among other things, its extension by the implementation of the more powerful and more complex genetic techniques. The problem of the fitness evaluation can be resolved by validations made by the user. This way interaction is added and thus composition programs can be built, which can be influenced by the user in real-time. Another (possible) addition is the creation of a collection of abstract patches that can be used easily.

Bibliography

[Ame87] Charles Ames. Automated composition in retrospect: 1956-1986. *Leornardo*, 20(2), 1987, pp. 169–185.

[Ame89] Charles Ames. The markov process as a compositional model: A survey and tutorial. *Leonardo*, 22(2), 1989, pp. 175–188.

[Ame90] Charles Ames. Statistic and compositional balance. *Perspectives of New Music*, 28(1), 1990, pp. 80–111.

[Bid92] Rick Bidlack. Chaotic systems as simple (but complex) compositional algorithms. *Computer Music Journal*, 16(3), 1992, pp. 33–47.

[Bil94] J. A. Biles. GenJam: a genetic algorithm for generating jazz solos. In *Proceedings of the 1994 International Computer Music Conference*. San Francisco: ICMA, 1994, pp. 131–137.

[Bil03] J. A. Biles. Genjam in perspective: A tentative taxonomy for ga music and art systems. *Leonardo*, 36(1), 2003, pp. 43–45.

[BPT00] Eleonora Bilotta, Pietro Pantano, and Valerio Talarico. Music generation through cellular automata: How to give life to strange creatures. In *Paper of Generative Art GA2000*, Milano, 2000. http://galileo.cincom.unical.it/pubblicazioni/papers/2000/milano.pdf (March 2008).

[BV99] Anthony R. Burton and Tanya Vladimirova. Generation of musical sequences with genetic techniques. *Computer Music Journal*, 23(4), 1999, pp. 59 – 73.

[Cha84] Joel Chadabe. Interactive composing: An overview. *Computer Music Journal*, 8(1), 1984, pp. 22–7.

[Cha90] Jacques Chareyon. Digital synthesis of self-modifying waveforms by means of cellular automata. *Computer Music Journal*, 14(4), 1990, pp. 25–41.

[Cha97] Joel Chadabe. *Electric sound: the past and promise of electronic music.* Upper Saddle River, NJ: Prentice Hall, 1997.

[DJ85] Charles Dodge and Thomas A. Jerse. *Computer Music: Synthesis, Composition and Performance*. New York, NY: Shirmer Books, 1985.

[Dod88] Charles Dodge. Profile: A musical fractal. *Computer Music Journal*, 12(3), 1988, pp. 10–14.

[Dol89] M. Dolson. Machine tongues XII: Neural networks. *Computer Music Journal*, 13(3), 1989, pp. 28–40.

[Doo02] Paul Doornbusch. Composers' views on mapping in algorithmic composition. *Organised Sound*, 7(2), 2002, pp. 145–156.

[Gar78] M. Gardner. White and brown music, fractal curves and one-over-f fluctuations. *Scientific American*, 238(4), 1978, pp. 16–27.

[GB91] P. M. Gibson and J. A. Byrne. NEUROGEN: musical composition using genetic algorithms and cooperating neural networks. In *Second International Conference on Artificial Neural Networks*. London: IEE Conference Publication, 1991, pp. 309–313.

[Gio96] Francesco Giomi. An early case of algorithmic composition in italy. *Organised Sound*, 1(3), 1996, pp. 179–182.

[GJC03] Andrew Gartland-Jones and Peter Copley. The suitability of genetic algorithms for musical composition. *Contemporary Music Review*, 22(3), 2003, pp. 43 – 55.

[GMMvZ99] R. Gudwin, J. Manzolli, A. Moroni, and F. von Zuben. An evolutionary approach to algorithmic composition. *Organised Sound*, 4(2), 1999, pp. 121–125.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[Har95] James Harley. Generative processes in algorithmic composition: Chaos and music. *Leonardo Music Journal*, 28(3), 1995, p. 221.

[Hau25] Josef Matthias Hauer. *Vom Melos zur Pauke. Eine Einführung in die Zwölftonmusik*. Wien/New York: Universal-Edition A.G., 1925.

[Jac96] Bruce L. Jacob. Algorithmic composition as a model of creativity. *Organised Sound*, 1(3), 1996, pp. 157–165.

[JC02] Colin G. Johnson and Juan Jesus Romero Cardalda. Introduction: Genetic algorithms in visual art and music. *Leonardo*, 35(2), 2002, pp. 175 – 184.

[Joh03] Colin G. Johnson. Exploring sound-space with interactive genetic algorithms. *Leonardo*, 36(1), 2003, pp. 51 – 54.

[Jon81] Kevin Jones. Compositional applications of stochastic processes. *Computer Music Journal*, 5(2), 1981, pp. 381–396.

[Koe] Gottfried Michael Koenig. *Homepage of Gottfried Michael Koenig*. http://home.planet.nl/∼gkoenig/indexe.htm (September 2005).

[Koz90] John R. Koza. Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems.

Technical Report STAN-CS-90-1314, Stanford University, Stanford, CA, 1990.

[Las81] Otto Laske. Composing theory in koenig's project one and project two. *Computer Music Journal*, 5(4), 1981, pp. 54–65.

[Mad99] Charles B. Madden. *Fractals in music: Introductory mathematics for musical analysis.* Salt Lake City, UT, United States: High Art Press, 1999.

[M.I86] Jordan M.I. Serial order: A parallel distributed processing approach. Technical Report ICS-8604, La Jolla: University of California, Institute for Cognitive Science, 1986.

[Mir95] Eduardo Reck Miranda. Granular synthesis of sounds by means of a cellular automaton. *Leonardo*, 28(4), 1995, pp. 297 – 300.

[Mir01] Eduardo Reck Miranda. *Composing Music with Computers with Cdrom.* Newton, MA, USA: Butterworth-Heinemann, 2001.

[MMH99] Kenneth McAlpine, Eduardo Miranda, and Stuart Hoggar. Making music with algorithms: A case-study system. *Computer Music Journal*, 23(2), 1999, pp. 19–30.

[Pre88] Jeff Pressing. Nonlinear maps as generators of musical design. *Computer Music Journal*, 12(2), 1988, pp. 35–46.

[Puc] Miller Puckette. Pd documentation. http://crca.ucsd.edu/∼msp/Pd_documentation/ (December 2005).

[Roa96] Curtis Roads. *The Computer Music Tutorial.* Cambridge, MA: MIT Press, 1996.

[Roa04] Curtis Roads. *Microsound.* Cambridge, MA: MIT Press, 2004.

[SA95] Lee Spector and Adam Alpern. Induction and recapitulation of deep musical structure. In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI'95 Workshop on Music and AI*, Montreal, Quebec, Canada, 1995, pp. 20–25.

[SAD+00] Antonino Santos, Bernardino Arcay, Julin Dorado, Juan Romero, and Jose Rodrguez. Evolutionary computation systems for musical composition. In N. Mastorakis (Ed.), *Mathematics and Computers in Modern Science*. Athens: World Scientific and Engineering Society Press, 2000, pp. 97–102.

[Sin98] Wilhelm Sinkovicz. *Mehr als zwölf Töne. Arnold Schönberg.* Wien: Paul Zsolnay Verlag, 1998.

[Sup01] Martin Supper. A few remarks on algorithmic composition. *Computer Music Journal*, 25(1), 2001, pp. 48–53.

[Tau04] Heinrich K. Taube. *Notes from the Metalevel*. New York: Taylor & Francis, 2004.

[Thy99] Kurt Thywissen. Genotator: an environment for exploring the application of evolutionary techniques in computer-assisted composition. *Organised Sound*, 4(2), 1999, pp. 127 – 133.

[Tod89] P. M. Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4), 1989, pp. 27–43.

[Tod91] P. M. Todd. Neural networks for applications in the arts. In M. Scott (Ed.), *Proceedings of the Eleventh Annual Symposium on Small Computers in the Arts*. Philadelphia, PA: Small Computers in the Arts Network, Inc, 1991, pp. 3–8.

[Tru] Barry Truax. *Homepage of Barry Truax*. http://www.sfu.ca/∼truax/pod.html (September 2005).

[Une] Tatsuo Unemi. A design of genetic encoding for breeding short musical pieces. http://www.alife.org/alife8/workshops/5.pdf (September 2005).

[VC78] Richard F. Voss and John Clarke. 1/f noise in music: Music from 1/f noise. *Journal of the Acoustical Society of America*, 63(1), 1978, pp. 258–263.

[Wol83a] Stephen Wolfram. Cellular automata. *Los Alamos Science*, 9, 1983, pp. 2–21.

[Wol83b] Stephen Wolfram. Statistical mechanics of cellular automata. *Reviews of Modern Physics*, 55, 1983, pp. 601–644.

[Wol84] Stephen Wolfram. Universality and complexity in cellular automata. *Physica D*, 10, 1984, pp. 1–35.

[Wol02] Stephen Wolfram. *A new kind of science.* Champaign, IL: Wolfram Media, 2002.

[Xen92] Iannis Xenakis. *Formalized music - thought and mathematics in composition.* Stuyvesant, NY: Pendragon Press, 1992.

List of Figures

List of Tables