UNIVERSITÄT DORTMUND

Komposition von Musik mit Methoden der Computational Intelligence

Roman Klinger

Algorithm Engineering Report
TR06-2-008
August 2006
ISSN 1864-4503

Universität Dortmund
Fachbereich Informatik
Algorithm Engineering (LS 11)
44221 Dortmund / Germany
http://ls11-www.cs.uni-dortmund.de/

UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

Roman Klinger

Komposition von Musik mit Methoden der Computational Intelligence

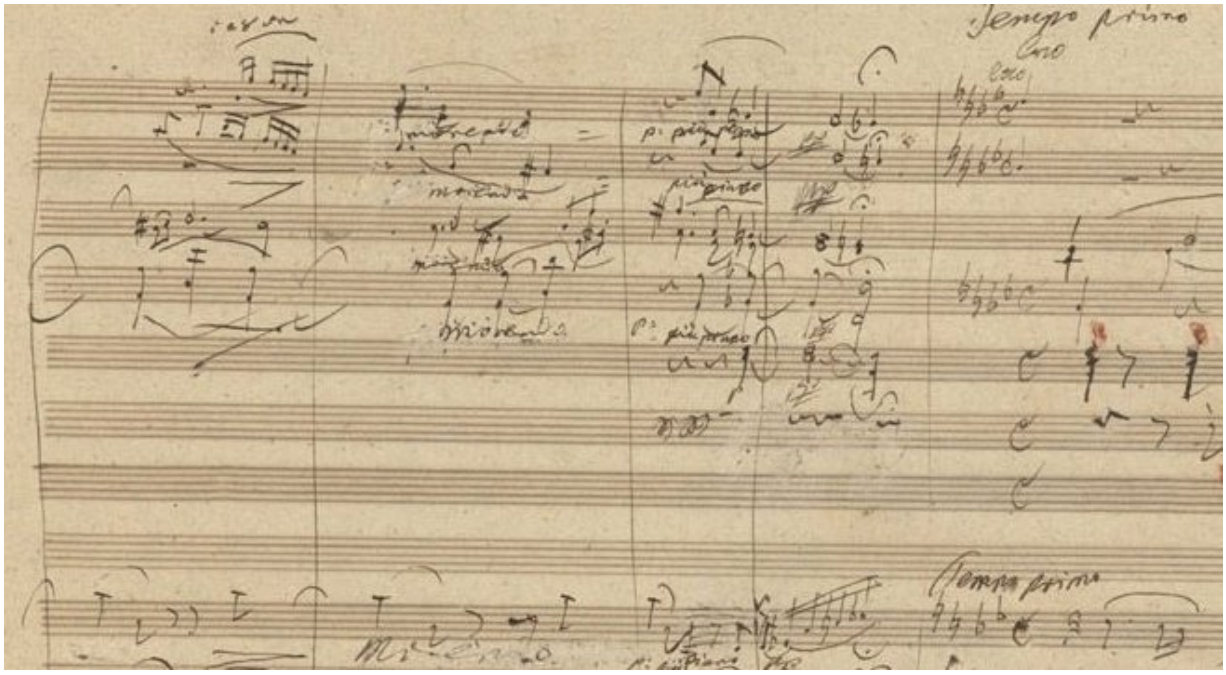
– Diplomarbeit –

1. Juni 2006

Lehrstuhl 11
Computational Intelligence
Fachbereich Informatik
Universität Dortmund

Gutachter:
Prof. Dr. G. Rudolph
Dr. L. Hildebrand

GERMANY · D-44221 DORTMUND



Ausschnitt aus Ludwig von Beethovens Sinfonie Nr. 9, d-moll, op.125
(Beethoven, 1824)

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmusverzeichnis

Kapitel 1

Einleitung

> *Why do we like certain tunes?*
> Because they have certain structural features.
> Because they resemble other tunes we like.

Dies bemerkt Marvin Minsky (1981) und trifft damit auf eine Frage, die bei der Komposition von Musik grundlegend ist, wenn sie auch nicht beantwortet werden muss, um erfolgreich Melodien zu entwerfen. Er stellt die These auf, dass Melodien gemocht werden, wenn sie an andere Melodien erinnern, die man mag. Des weiteren sollten seiner Aussage nach bestimmte strukturelle Merkmale gelten.

Robert Jourdain (2001) nennt einige solcher Merkmale. Er stellt aber auch fest:

> Regeln können zwar schlechte Melodien entlarven, aber schöne nicht vorhersagen.

Die vorliegende Arbeit basiert auf der Annahme (welche später auch untersucht wird), dass eine geschickte Kombination von Merkmalen von Melodien genutzt werden kann, um Musikstücke in ihrer Qualität einzuschätzen, insbesondere also, um schlechte Melodien zu „entlarven“. Diese Möglichkeit lässt auf die Existenz von Verfahren hoffen, welche automatisch Melodien erzeugen können. Warum aber sollte es erstrebenswert sein, solche Methoden zu nutzen?

Wolfgang Amadeus Mozart (1793) hat in seiner „Anleitung so viel Walzer oder Schleifer mit zwei Würfeln zu componiren so viel man will ohne musikalisch zu seyn noch etwas von der Composition zu verstehen“ bereits im Titel eine Begründung genannt: Automatische Kompositionsverfahren können musikalische Laien in der Erstellung von Musik unterstützen. Mozarts System besteht hierzu aus vorgefertigten kurzen Phrasen, welche durch Würfeln ausgewählt und zu einer zufälligen Sequenz zusammengestellt werden. Die Frage, ob der kreative Prozess das Würfeln oder das Vorkomponieren ist, scheint auf den ersten Blick leicht zu beantworten. Es ließe sich jedoch auch die Meinung vertreten, dass durch das Würfeln neue Musik geschaffen wird, an die Mozart möglicherweise gar nicht so gedacht hat.

Mir erscheint die Beschränkung auf vorgefertigte, notwendigerweise zusammenpassende Musikstückchen unvorteilhaft für die Wahrnehmung der geschaffenen Musik als kreativ und innovativ. Meine persönliche Motivation zum Entwurf eines Systems zur automatischen Komposition von Musik ist der Wunsch nach einer Unterstützung bei dem Entwurf der tragenden, monophonen Melodie eines Arrangements. Hierzu sollte der automatischen Generierung die Fähigkeit, etwas Neues zu schaffen, zugesprochen werden können.

Um dies zu erreichen, wird ein Optimierverfahren eingesetzt. Hier werden die grundlegenden Merkmale natürlicher Evolution in Form der evolutionären Algorithmen genutzt. Diese benötigen zunächst initiale Melodien, im Jargon der Evolutionstheorie auch Individuen genannt. Erzeugt werden sie unter anderem aus anderen Melodien mit Hilfe statistischer Methoden. Entsprechend der Vermehrung in der Natur vorkommender Lebewesen werden aus jeweils zwei Individuen Nachkommen generiert (Rekombination), wobei einige Eigenschaften (entsprechend der Gene in der Natur) verändert werden (Mutation). Dieser Ablauf kann als kreativ betrachtet werden, sowohl in dem hier beschriebenen Umfeld der automatischen Komposition, wie auch in der Natur. Eine schwierige Frage ist die nach der Umsetzung der Selektion, wie sie in der Natur vorkommt. Es muss die Qualität von Melodien beurteilt werden, um nur die Besten in einer Folgegeneration bestehen lassen zu können. Entsprechend

einer plakativ als „Züchtung von Melodien“ bezeichenbaren Methode kann eine interaktive Bewertung zum Einsatz kommen. Da hierzu jede Melodie gehört werden muss, ist dies recht ermüdend. Sinnvoll wäre ein automatisches Verfahren, welches in bisherigen Arbeiten üblicherweise durch die Nutzung von gewichteten Summen von Merkmalen der zu beurteilenden Melodie implementiert wurde. Zur Verbesserung der Beurteilung und der Möglichkeit der automatischen Erstellung von geschickten Verknüpfungen der Merkmale soll in dieser Arbeit der Einsatz von Methoden aus dem Bereich des maschinellen Lernens erprobt werden.

Wie selbstverständlich wurde bisher das Wort *Komposition* genutzt, wenn von der automatischen Erstellung von Musik die Rede war. Streng genommen stellt dies kein Problem dar, auch wenn zunächst die Assoziation mit einer menschlichen, kreativen Leistung naheliegt. Das Wort „compositio“ bedeutet jedoch übersetzt aus dem lateinischen schlicht „Zusammenstellung“ (Drosdowski, 1990). Und als solche ist jedes Musikstück zu betrachten.

Die Arbeit ist in zwei Hauptabschnitte unterteilt. In Teil I werden in den Kapiteln 2–3 die notwendigen Grundlagen gegeben. Diese sind neben den eingesetzten Klassifikationsverfahren der künstlichen neuronalen Netze und Entscheidungsbäume auch die evolutionären Algorithmen sowie die zur Initialisierung dieser genutzten Markovketten. Die verwendete übliche Notenschrift sowie notwendige musikalische Grundlagen werden ebenfalls eingeführt. Teil I wird mit Kapitel 4 durch eine Übersicht über die verwendete MIDI-Technologie sowie in Kapitel 5 über einen Teil der bisher in dem Bereich der automatischen Komposition geleisteten Arbeiten geschlossen.

In Teil II folgt die Erläuterung über die im Zuge dieser Arbeit durchgeführte Implementierung und daraus geschlossener Erkenntnisse. Begonnen wird mit einem Überblick über das System in Kapitel 6, in dem der evolutionäre Algorithmus mit seinen melodieverändernden Komponenten und der Initialisierung als kreativ betrachtet werden kann. Bewertend und damit einschränkend greifen dann die Klassifikationsmethoden ein. Zunächst ist in Kapitel 7 erläutert, in welcher Form Melodien repräsentiert und gespeichert werden. Als Ausgangspunkt werden initiale Musikstücke erzeugt, wobei die genutzten Techniken in Kapitel 8 eingeführt und an Beispielen erklärt werden. Die Möglichkeiten der Veränderung dieser Melodien, um gefällig zu wirken, werden in Kapitel 9 erläutert. Kapitel 10 beschreibt den Einsatz verschiedener Verfahren zur Bewertung von Melodien. Dies geschieht auf Basis der erwähnten Merkmale, welche hier erläutert sind. Darauf aufbauend setzen die bereits genannten Entscheidungsbäume und neuronalen Netze an, es werden aber auch interaktive, benutzergestützte Verfahren genannt. Hier findet sich ebenfalls eine genaue Beschreibung der Wirkung der unterschiedlichen Ansätze anhand von konkreten Beispielen. In Kapitel 11 werden Hinweise auf sinnvolle Parameterbelegungen gegeben, welche an ausführlichen Beispielen, die mit Blick auf die vorhergehenden Abschnitte erläutert sind, erklärt werden. Zum Abschluss findet sich in Kapitel 12 eine Einschätzung der gewonnenen Erkenntnisse sowie ein Ausblick auf ausstehende Arbeiten.

Die Implementierung des entworfenen Systems zur automatischen Generierung von Melodien und der dazugehörige Quelltext sowie hilfreiche zugehörige Programme finden sich auf der beiliegenden CD und auf der Internetseite `http://www.roman-klinger.de`.

Teil I

Grundlagen

Kapitel 2

Mathematische Grundlagen

2.1 Markovketten

Andrei A. Markov (1856–1922) entwickelte die nach ihm benannte Struktur von Sequenzen von Zufallsvariablen. Mit ihnen ist die Modellierung dynamischer Systeme möglich. Hierzu werden die Wahrscheinlichkeiten des Auftretens von Ereignissen in Abhängigkeit von vorherigen Ereignissen dargestellt. Die vorliegende Beschreibung basiert auf Huisinga und Meerbach (2005), Russell und Norvig (2003) sowie Miranda (2001).

Definition 1 *Eine Sequenz $X = \{X_k\}_{k\in\mathbb{N}}$ von (diskreten) Zufallsvariablen $X_k : \Omega \to S$ heißt* diskreter stochastischer Prozess *im Zustandsraum S.*

Der Index k lässt sich hierbei als Zeitangabe interpretieren.
Kommen wir zur Definition der Markovkette:

Definition 2 *Ein diskreter stochastischer Prozess $\{X_k\}_{k\in\mathbb{N}}$ in einem abzählbaren Zustandsraum heißt* Markovkette erster Ordnung, *wenn die Markovbedingung*

$$\begin{aligned} & P(X_{k+1} = z | X_k = y, X_{k-1} = x_{k-1}, \ldots, X_0 = x_0) \\ = \quad & P(X_{k+1} = z | X_k = y) \end{aligned}$$

für jedes $k \in \mathbb{N}$, und für $x_0, \ldots, x_{k-1}, y, z \in S$ gilt.

Dies bedeutet, dass der Zustand zu einem Zeitpunkt $k+1$ nur von dem vorherigen Zeitpunkt k, jedoch nicht von Zuständen zu Zeitpunkten $k' < k$ abhängt. Dies folgt aus der Definition der Unabhängigkeit von Ereignissen:

Definition 3 *Zwei Ereignisse A und B heißen* unabhängig, *wenn gilt:*

$$P(A|B) = P(A)$$

Definition 4 *Eine Markovkette heißt* homogen, *wenn die Wahrscheinlichkeiten nicht vom Zeitpunkt k abhängen.*

Definition 5 *Man spricht von einer* Markovkette n-ter Ordnung *wenn der Zustand von den n vorherigen Zuständen abhängt.*

Im folgenden werden homogene Markovketten erster Ordnung betrachtet.

2.1.1 Darstellung

Die Übergangsfunktionen von Markovketten werden üblicherweise in einer stochastischen Matrix oder durch einen Graphen dargestellt.

Definition 6 *Eine Matrix $P = p_{(x,y)}$ heißt* stochastisch , *wenn für alle $x, y \in \mathcal{S}$ gilt:*

$$p_{(x,y)} \geq 0 \text{ und } \sum_{y\in S} p_{(x,y)} = 1$$

Diese Matrix P wird durch die Funktion $\mathcal{P} : S \times S \rightarrow \mathbb{R}$ mit $\mathcal{P}(x,y) = P(X_{k+1} = y | X_k = x)$ gegeben. Bei gegebener stochastischer Matrix und gegebenem aktuellen Zustand ist es also möglich, die Wahrscheinlichkeitsverteilung für den nächsten Zustand an der Tabelle abzulesen.

Eine andere Möglichkeit ist die Darstellung als Graph $G = (V,E)$ mit Knoten $V = \{s_1, \ldots, s_n | s_1, \ldots, s_n \in S, n = |S|\}$ und gerichteten Kanten $E = \{(x,y) | \mathcal{P}(x,y) > 0\}$. Hierbei wird die Kante (x,y) mit $\mathcal{P}(x,y)$ beschriftet.

2.1.2 Beispiele

Beispiel 1: Wettervorhersage

In diesem sehr einfachen Beispiel möchten wir annehmen, dass das Wetter die Zustände $S = \{\text{Regen}, \text{Sonne}, \text{Nebel}\}$ annehmen kann. Die Wahrscheinlichkeiten für Wetteränderungen seien durch die stochastische Matrix in Tabelle 2.1 gegeben.

	Regen	Sonne	Nebel
Regen	0,5	0,1	0,4
Sonne	0,5	0,5	0,0
Nebel	0,5	0,2	0,3

Tabelle 2.1: Beispiel für Markovketten in Tabellendarstellung: Wettervorhersage

Diese Markovkette lässt sich auch wie in Abbildung 2.1 als Graph darstellen.

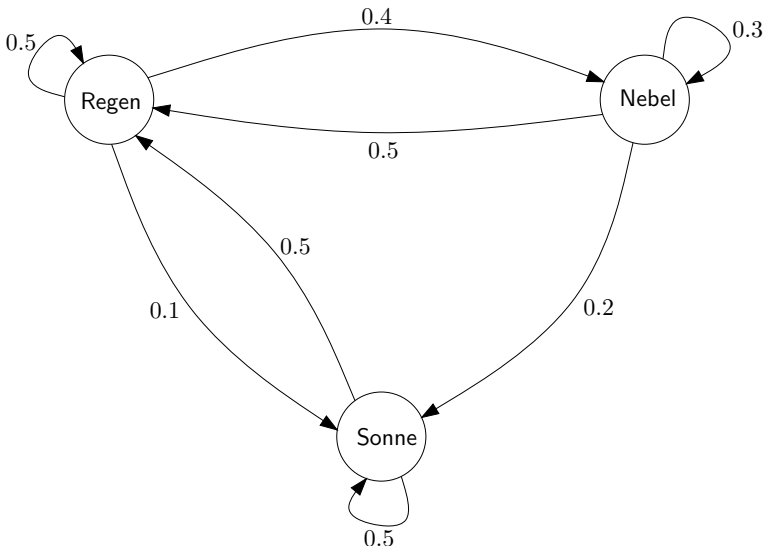


Abbildung 2.1: Beispiel für Markovketten in Graphdarstellung: Wettervorhersage

Beispiel 2: Random Walk

Ein geläufiges Beispiel für einen Random Walk ist die Vorstellung eines sich im Raum beweglichen Partikels (vergleiche Iosifescu, 1980; Feller, 1968).

Die Position dieses Partikels sei durch die Koordinaten $(x_1, \ldots, x_n)_t$ zum Zeitpunkt t ($x_1, \ldots, x_n \in \mathbb{Z}, t \in \mathbb{N}$) angegeben. Eine Bewegung ist möglich, in dem sich zum Folgezeitpunkt $t+1$ die Koordinaten

unabhängig voneinander durch Addition von gleichverteilten Zufallszahlen $z_i \in \{-1, 1\}$ ($i \in \{1, \ldots, n\}$) verändern.

Für $n = 1$ hat eine solche Markovkette die charakteristische Matrix in Tabelle 2.2 gegeben. Ein Beispiel für eine so entstehende Bewegung für $n = 2$ ist in Abbildung 2.2 zu sehen[1].

	$\cdots$	-3	-2	-1	0	1	2	3	$\cdots$
$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
3	$\cdots$	0	0	0	0	0	$0{,}5$	0	$\cdots$
2	$\cdots$	0	0	0	0	$0{,}5$	0	$0{,}5$	$\cdots$
1	$\cdots$	0	0	0	$0{,}5$	0	$0{,}5$	0	$\cdots$
0	$\cdots$	0	0	$0{,}5$	0	$0{,}5$	0	0	$\cdots$
-1	$\cdots$	0	$0{,}5$	0	$0{,}5$	0	0	0	$\cdots$
-2	$\cdots$	$0{,}5$	0	$0{,}5$	0	0	0	0	$\cdots$
-3	$\cdots$	0	$0{,}5$	0	0	0	0	0	$\cdots$
$\vdots$		$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	

Tabelle 2.2: Beispiel für die charakteristische Matrix eines Random Walks

t	x_1	x_2
0	0	0
1	-1	-1
2	0	0
3	-1	1
4	-2	0
5	-1	-1
6	0	0
7	1	1
8	2	0
9	1	-1
10	-2	2

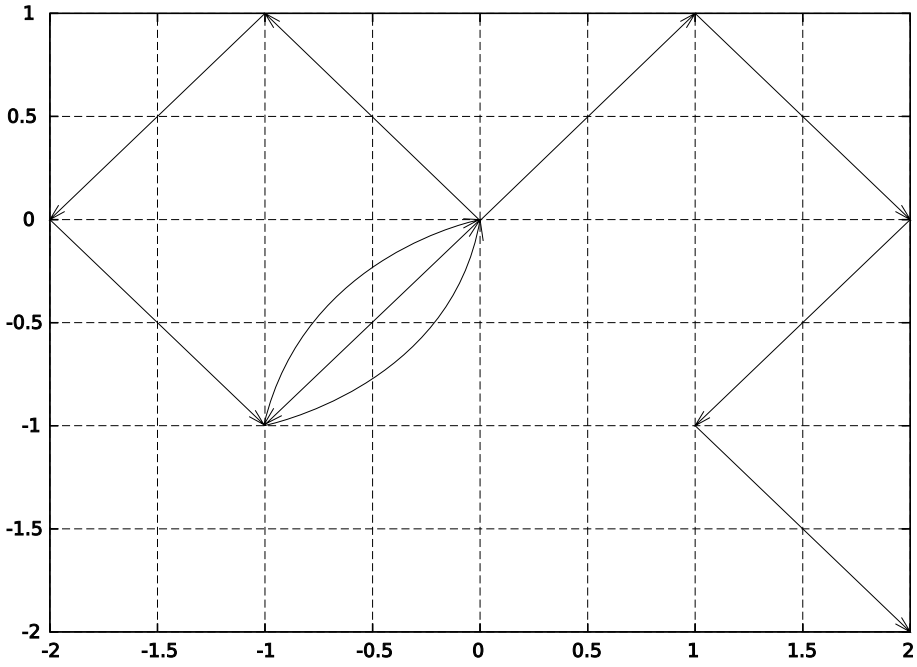


Abbildung 2.2: Beispiel für den Verlauf eines Random Walks.

2.1.3 Eigenschaften

Markovketten haben einige interessante Eigenschaften, welche helfen, sie zu charakterisieren.

Definition 7 *Seien X_k mit $k \in \mathbb{N}$ eine Markovkette mit Transitionsfunktion $\mathcal{P}$ und $(x, y) \in S$ beliebige Paare von Zuständen.*

[1] Ein Programm zur Simulation solcher Random Walks findet sich auf der CD im Ordner `SmallTools` in der Klasse `RandomWalk.java` oder auf `http://www.roman-klinger.de`

1. *Der Zustand y ist* erreichbar *von Zustand x aus (geschrieben als $x \to y$), wenn gilt*

$$P(X_m = y | X_0 = x) > 0,$$

für ein $m \in \mathbb{N}$, welches möglicherweise von x und y abhängt. Dies bedeutet, dass die Möglichkeit besteht, von Zustand x aus in m Schritten Zustand y zu erreichen.

2. *Die Zustände x und y* kommunizieren, *wenn sowohl*

$$x \to y \text{ als auch } y \to x,$$

also kurz

$$x \leftrightarrow y$$

gilt.

3. *Eine Markovkette heißt* irreduzibel, *wenn alle Paare von Zuständen kommunizieren.*

Die Relation $\leftrightarrow$ ist eine Äquivalenzrelation, wie leicht überprüfbar ist:
Es gilt:

1. Reflexivität[2]: $x \leftrightarrow x$
2. Symmetrie: Aus $x \leftrightarrow y$ folgt $y \leftrightarrow x$
3. Transitivität: Aus $x \leftrightarrow y$ und $y \leftrightarrow z$ folgt $x \leftrightarrow z$

Die durch die Äquivalenzrelation definierten Äquivalenzklassen nennt man *Kommunikationsklassen*.

2.1.4 Berechnungen

Verschiedene Fragestellungen bezüglich Markovketten können durch entsprechende Algorithmen beantwortet werden. Hier sei auf den Forward-Algorithmus zur Bestimmung der Wahrscheinlichkeit einer bestimmten Ausgabesequenz und auf den Baum-Welch-Algorithmus zur Bestimmung der Parameter einer Markovkette zur Maximierung der Wahrscheinlichkeit der Erzeugung einer bestimmten Sequenz verwiesen (Bilmes, 1998).

In dieser Arbeit werden homogene Markovketten betrachtet, welche durch die Maximum-Likelihood-Methode geschätzt werden.

2.1.4.1 Erstellung einer Markovkette

Eine Markovkette wird aus einer Menge $B = \{B_1, \ldots, B_n\}$ ($n \in \mathbb{N}$) von Beispielsequenzen erstellt. Hierzu wird die Häufigkeit $h(x, y)$ des Auftretens eines Zustands x zu einem Zeitpunkt k in Folge zu dem Zustand y zu einem Zeitpunkt $k-1$ für alle $x, y \in S$ und alle k bezüglich allen Beispielsequenzen gezählt. Diese Häufigkeiten definieren eine Matrix H. Aus dieser Matrix wird nun eine stochastische Matrix H^s erzeugt. Hierbei gilt für die einzelnen Einträge $h^s(x_a, x_b)$ an Position (a, b) mit $1 \leq a, b \leq |S|$:

$$h^s(x_a, x_b) = \frac{h(x_a, x_b)}{\sum_{i=1}^{|S|}(h(x_a, x_i))}$$

Als Beispiel sei die Menge $B = \{B_1, B_2, B_3\}$ mit

$$\begin{aligned} B_1 &= (a, b, d) \\ B_2 &= (c, d, e) \\ B_3 &= (a, b, e, c, d, e) \end{aligned}$$

[2] Für eine genauere Beschreibung sei auf Iosifescu (1980) verwiesen.

gegeben.

Hieraus ergibt sich eine Matrix

$$H = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Durch Normalisierung entsprechend oben genannter Formel ergibt sich die stochastische Matrix:

$$H^s = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0{,}5 & 0{,}5 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

2.1.4.2 Erstellung einer Sequenz aus einer gegebenen Markovkette

Bei der Erzeugung einer Sequenz wird angenommen, dass eine Anfangswahrscheinlichkeitsverteilung zu S gegeben ist. Entsprechend dieser Verteilung wird ein Startwert zufällig bestimmt. Die Folgewerte zu einem Zustand x' werden entsprechend der Wahrscheinlichkeitsverteilung $P_{x'} = p(x', \cdot)$ bestimmt.

Sei als Beispiel die Markovkette in Tabelle 2.1 gegeben. Die Anfangswahrscheinlichkeiten seien auf (Regen; Sonne; Nebel) gleichverteilt.

Zufallswert	Vorgängerzustand	Folgezustand
0,5		Sonne
0,2	Sonne	Regen
0,7	Regen	Nebel
0,6	Nebel	Sonne

Tabelle 2.3: Beispiel 1 für die Erzeugung aus der Markovkette aus Tabelle 2.1

Nun könnte sich die in Tabelle 2.3 angegebene Sequenz ergeben. Hier gibt der initiale Zufallswert von 0.5 den Startzustand *Sonne* an. Entsprechend der durch den Wert *Sonne* gegebenen Wahrscheinlichkeitsverteilung (0,5; 0,5; 0,0) für (Regen; Sonne; Nebel) bestimmt der Zufallswert 0,2 den Wert *Regen* als neuen Zustand. Die Verteilung (0,5; 0,1; 0,4) und der Zufallswert 0,7 setzen *Nebel* als Folgezustand fest. Auf diesen folgt wegen der nun geltenden Verteilung (0,5; 0,2; 0,3) und dem Zufallswert 0,6 der Wert Sonne.

Auf dieselbe Weise ist die Sequenz in Tabelle 2.4 entstanden (wobei hier R für *Regen*, S für *Sonne* und N für *Nebel* steht).

Diese Reihung von Zuständen sei nun als Beobachtung angenommen, aus der die Markovkette entsprechend Abschnitt 2.1.4.1 geschätzt werden soll[3]. Zunächst werden die Häufigkeiten der Paare (x_a, x_b) ($x_a, x_b \in \{R, S, N\}$) von aufeinanderfolgenden Zuständen x_a und x_b gezählt:

$$H = \begin{array}{c} \\ R \\ S \\ N \end{array} \begin{array}{c} \begin{array}{ccc} R & S & N \end{array} \\ \begin{pmatrix} 76 & 22 & 56 \\ 36 & 35 & 0 \\ 42 & 14 & 19 \end{pmatrix} \end{array}$$

[3]Ein Programm zur Erstellung dieser Simulation und Abschätzung findet sich auf der CD im Ordner `SmallTools` in der Klasse `MarkovChainSimulation.java` oder auf `http://www.roman-klinger.de`

R S R N R N R R N N N R R N R S S S R S R S S R N S S S S R R R R R R R N R R
R S S S R N S R R N R N R N N R N R R R R R R R R N R S R S R R S S S S S S R
N N R R N S R R R N R S S R S S R R N N R R N N R R R N N N R N N S R R R R N
R R R R N R N R R R R N R N S R N S S S S S S S R N R R R R N N R N R N R N R R
N S R N S R R R R R S S R N R N S R R R N N R R R S S S R R R N R R R R R N R S R
S S R N N R R R R N R N N N R R S R S S R S S S R S R N S R R R R R N R S S R S S S
R R N R N N N S R N N N S S R S S R N N N R R N N N R R R S R R R N R N N N S S R N R R
R N R R R R R R N R R N N R N R N R N R N S S R

Tabelle 2.4: Beispiel 2 für die Erzeugung aus der Markovkette aus Tabelle 2.1

Durch Normalisieren der Zeilen auf 1 ergibt sich die charakteristische Matrix, welche die Markovkette in Tabelle 2.1 abschätzt.

$$H^s = \begin{array}{c} \\ R \\ S \\ N \end{array} \begin{array}{c} \begin{array}{ccc} R & S & N \end{array} \\ \begin{pmatrix} 0{,}494 & 0{,}143 & 0{,}364 \\ 0{,}507 & 0{,}493 & 0{,}000 \\ 0{,}560 & 0{,}187 & 0{,}253 \end{pmatrix} \end{array}$$

2.2 Evolutionäre Algorithmen

2.2.1 Einführung

Suchverfahren werden von Russell und Norvig (2003) in die Bereiche *uninformierte*, *informierte* und *lokale Suche* eingeordnet. Zu den *uninformierten* zählen die *Breitensuche* und die *Tiefensuche*, welche unabhängig von der Problemstruktur nach einer bestimmten Strategie alle möglichen Zustände besuchen, bis ein hinreichend guter Wert gefunden wird.

Zu den *informierten* Verfahren zählt die *Greedy-Suche*, welche grundsätzlich den nächstbesten Zustand besucht und dabei keine Verschlechterung zulässt, sowie die A^**-Suche*, welche den geschätzten Abstand zum Optimum, die sogenannte *Heuristik*, und den bereits zurückgelegten Weg kombiniert um den nächsten zu besuchenden Zustand auszuwählen.

Die uninformierte und die informierte Suche betrachten auch den Suchpfad, also den Weg zur Findung einer bestimmten Lösung. Im Gegensatz dazu ist bei der *lokalen Suche* (auch *Explorationsalgorithmus* genannt) nur der Zielzustand im Suchraum interessant. Zu diesen Verfahren zählt *Hill-Climbing*, bei dem wie bei der Greedy-Suche als Folgezustand nur der beste Zustand in Frage kommt. Allerdings wird hier eben nicht der Suchpfad betrachtet. Im Gegensatz zu Hill-Climbing lässt die randomisierte Variante, der *Metropolis-Algorithmus* (Metropolis u. a., 1953) mit einer festgelegten Wahrscheinlichkeit auch schlechtere Zustände als den aktuellen als Folgezustand in der Suche zu. Ähnlich arbeitet das Verfahren *Simulated Annealing* (Kirkpatrick u. a., 1983), welches die Idee verfolgt, im Verlauf der Suche die Wahrscheinlichkeit für das Verschlechtern zu senken.

Evolutionäre Algorithmen gehören in den Bereich der Explorationsalgorithmen. Dieses Optimierverfahren und seine verschiedenen Varianten werden im folgenden unter Betonung der für diese Arbeit besonders wichtigen Merkmale vorgestellt. Für eine Übersicht weiterer Suchmethoden sei hier auf Russell und Norvig (2003), Wegener (2002) und Schwefel (1995) verwiesen.

2.2.2 Grundlegende Elemente und Begriffe

In diesem Abschnitt werden die wichtigsten Begriffe zu und Elemente von evolutionären Algorithmen eingeführt, welche in allen Formen und Varianten vorhanden sind. Wir benötigen im weiteren einige Definitionen zur genaueren Erläuterung. Diese folgen Gottlieb (2000).

Definition 8 *Sei eine Menge von Variablen* $X = \{x_1, \ldots, x_n\}$ *mit den dazugehörigen Definitionsbereichen* D_i *zu* x_i *(*$i \in \{1, \ldots, n\}$*) gegeben. Dann ist der korrespondierende* Suchraum *die Menge* $S = \{D_1 \times \ldots \times D_n\}$.

Definition 9 *Sei ein Suchraum S gegeben. Dann ist eine* Zielfunktion *eine Funktion* $f : S \to \mathbb{R}$.
Eine Lösung $x \in S'$ *(*$S' \subset S$*) ist* optimal *bezüglich* f *und* S'*, wenn* $f(x) \geq f(y)$ *(*$y \in S'$*) gilt.*

Definition 10 *Eine* Nebenbedingung *ist eine Funktion* $c : S \to \mathbb{R}$. *Daraus ergibt sich der gültige Suchraum bezüglich der Nebenbedingung c durch* $F(c) = \{x \in S | c(x) \geq 0\}$. *Der ungültige Suchraum ist entsprechend* $U(c) = S \backslash F(c)$.

Mit diesen Definitionen kommen wir schließlich zum Begriff des Optimierungsproblems.

Definition 11 *Ein* Optimierungsproblem *ist gegeben durch*

$$\max_{x \in S} \{f(x) | C\}$$

wobei C eine Menge von Nebenbedingungen ist.

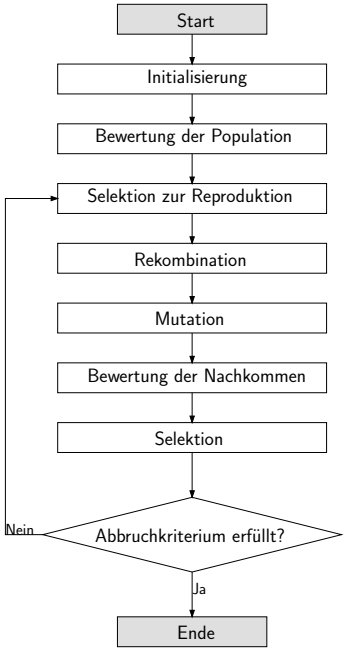
Hier wird von einer Maximierung von $f(x)$ ausgegangen, daher sprechen wir von einem Maximierungsproblem. Ein Minimierungsproblem wird gegeben durch $\min_{x \in S}\{f(x)|C\} = -\max_{x \in S}\{-f(x)|C\}$.

Wir sprechen von einem multikriteriellen Optimierungsproblem, wenn f die Form $f : S \to \mathbb{R}^k$ mit $k \in \mathbb{N}$ hat.

Wie bereits in der Einleitung erwähnt, sind evolutionäre Algorithmen Verfahren zur Lösung von Optimierungsproblemen. Im folgenden werden grundlegende Begrifflichkeiten eingeführt. Hierbei wird auch deutlich, dass eine Motivation der evolutionären Algorithmen in der Betrachtung der natürlichen Evolution liegt.

Ein *Individuum* repräsentiert einen bestimmten Punkt im Suchraum. Hierbei ist der in der Realität geltende Zustand in Anlehnung an die Biologie als *Phänotyp* bezeichnet, die Kodierung im Individuum wird *Genotyp* genannt. Neben dieser Repräsentation können noch die Suche beeinflussende Parameter im Individuum gespeichert sein. Eine *Population* ist eine Menge von gleichzeitig gespeicherten Individuen. Es ist praktisch nicht möglich, alle möglichen Suchpunkte gleichzeitig zu betrachten, so dass hier eine beschränkte Auswahl getroffen wird. Eine *Generation* bezeichnet eine zu einem Zeitpunkt aktuelle Population. Durch die Anwendung von Operatoren auf die Individuen wird aus dieser Generation die Folgegeneration. Zu den Operatoren zählen *Rekombination*, *Mutation* und *Selektion*. Durch die Rekombination wird aus mehreren Individuen (*Eltern*) ein (oder mehrere) weiteres (*Kind*) erzeugt, welches Merkmale der Eltern enthält. Dies kann also zum Beispiel ein Punkt im Suchraum sein, welcher zwischen den durch die Eltern repräsentierten Punkten liegt. Die Mutation erzeugt aus einem Individuum ein neues nach bestimmten Regeln. Die Selektion schließlich wählt bestimmte Individuen zur Bildung der Folgepopulation mit Bezug zur Zielfunktion, also der Fitness des Individuums, aus.

In Algorithmus 2.1 ist der Ablauf in einem Diagramm dargestellt.



Algorithmus 2.1: Ablauf eines evolutionären Algorithmus (nach Jansen, 2004)

Bevor in Kapitel 2.2.3 genauer auf den Ablauf und die einzelnen Elemente evolutionärer Algorithmen eingegangen wird, sollen hier die Ursprünge der wichtigsten Strömungen dargestellt werden. Dieser Abschnitt orientiert sich an Wegener (2002) sowie Bäck, Fogel und Michalewicz (1997).

2.2.2.1 Genetische Algorithmen

Als Erfinder der *genetischen Algorithmen* (*GA*) gilt John H. Holland (Holland, 1975). Als Repräsentation nutzt er binäre, diskrete Suchräume ($\{0,1\}^n$). Die Individuen einer Folgegeneration werden fitness-

proportional aus der aktuellen Generation selektiert. Der Schwerpunkt bei der Variation liegt auf der Rekombination.

2.2.2.2 Evolutionsstrategien

Den Ursprung haben die *Evolutionsstrategien* (ES), welche auf H.-P. Schwefel, Ingo Rechenberg und Peter Bienert (Schwefel, 1965; Rechenberg, 1965; Bienert, 1967) zurückgehen, bei der experimentellen Optimierung.

Zunächst werden nur stetige Suchräume wie $\mathbb{R}^n$ betrachtet. Die Repräsentation ist eine Menge von Komponenten $x_i \in \mathbb{R}$. Die ersten Anwendungen werden mit nur einem Individuum durchgeführt. Daher wird hier natürlich keine Rekombination, sondern nur Mutation durchgeführt. Diese besteht aus der komponentenweisen Addition einer mit der Standardabweichung der Zufallsänderung gewichteten normalverteilten Zufallszahl.

2.2.2.3 Evolutionäres Programmieren

Das ursprüngliche Ziel von Lawrence J. Fogel, Alvin J. Owen und Michael J. Walsh bei dem Entwurf des Verfahrens der *evolutionären Programmierung* (GP) ist das Erzeugen von deterministischen endlichen Automaten, welche sich an ihre Umwelt anpassen (Fogel, 1962; Fogel u. a., 1966). Es soll also zum Beispiel ein Mealy-Automat evolviert werden, der sich auf einer Trainingsbeispielmenge gut verhält und möglichst klein ist.

Es besteht eine gewisse Nähe zu den Evolutionsstrategien; wie auch bei diesen wird zunächst auf Rekombination verzichtet. Zur Mutation stehen die Operatoren „Ändern eines Ausgabesymbols“, „Ändern einer Zustandsüberführung“, „Hinzufügen eines Zustands“, „Löschen eines Zustands“ und „Ändern des Initialzustands“ zur Verfügung.

2.2.2.4 Genetisches Programmieren

Während genetische Algorithmen, Evolutionsstrategien und evolutionäres Programmieren Wurzeln der evolutionären Algorithmen darstellen, welche zwar zeitnah, jedoch unabhängig voneinander entwickelt wurden, stellt sich die Situation bei dem *genetischen Programmieren* anders dar. Dieses Verfahren wurde von John R. Koza popularisiert (Koza, 1992). Hierbei wird die Längenbeschränkung der Individuen aufgehoben, in dem eine variable Repräsentation gewählt wird. Diese Erweiterung ermöglichst die Evolvierung von Programmcode. Dieser kann in LISP-ähnlicher Syntax und damit in Form von Bäumen gegeben sein.

Als Operatoren kommen Rekombination und Mutation zum Einsatz. Der Schwerpunkt liegt jedoch, ebenso wie bei den genetischen Algorithmen, auf der Bedeutung der Rekombination.

2.2.3 Elemente der evolutionären Algorithmen

Trotz der höheren Bedeutung der Ansätze der genetischen Algorithmen für diese Arbeit soll im Weiteren die Trennung nach den historischen Ansätzen nicht weiter verfolgt werden. So werden nun die einzelnen Elemente der evolutionären Algorithmen (entsprechend Algorithmus 2.1) dargestellt, wobei als Ausgangspunkt die Ansätze der genetischen Algorithmen und Evolutionsstrategien verwendet werden. Evolutionäres Programmieren und genetisches Programmieren lassen sich durch geeignete Kodierungen und Operatoren auf diese zurückführen.

Zur Beschreibung sei in diesem Abschnitt mit n die Länge der Individuen $\vec{x}^{(p)}$, also die Anzahl der Komponenten gemeint. Das Individuum wird durch p angegeben, daher gilt $p \in \{1, \ldots, |P|\}$, wobei P die Population ist. Weiterhin ist $x_i^{(p)}$ ($i \in \{1, \ldots, n\}$) der Wert der entsprechenden Komponente.

Die folgenden Ausführungen orientieren sich an Jansen (2004), Bäck, Fogel und Michalewicz (1997).

2.2.3.1 Repräsentation

Wie bereits in den vorstehenden Kapiteln bemerkt stellen die unterschiedlichen Kodierungsmöglichkeiten einen wichtigen Unterschied zwischen den einzelnen Strömungen der evolutionären Algorithmen dar. Insbesondere aus den genetischen Algorithmen folgen verschiedene Varianten.

Eine Unterscheidung ist, welchen Wertebereich die einzelnen Komponenten $x_i^{(p)}$ haben. Als Definitionsmenge ist hier zum Beispiel $\{0,1\}$, $\mathbb{R}$, $\mathbb{N}$ oder $\mathbb{Z}$ möglich. So lassen sich bei $\{0,1\}$ Parameter von Problemen mit Binär- oder Greykodierung in den Individuen darstellen. Das Setzen von Nebenbedingungen und Beziehungen zwischen Komponenten lässt sich nutzen, um zum Beispiel das Traveling Salesman Problem durch Permutationen zu modellieren. Hierdurch können ungültige Lösungen verhindert werden. Durch Nutzen von $\mathbb{N}$ oder $\mathbb{Z}$ als Grundmenge sind möglicherweise Operatoren einfacher zu entwerfen, als bei $\{0,1\}$, was auch in Teil II zu sehen sein wird.

Wie bei den Evolutionsstrategien üblich, stellt eine Kodierung der Individuen im $\mathbb{R}^n$ einen kontinuierlichen Suchraum dar. So könnten die Individuen als Positionen in einem n-dimensionalen Raum betrachtet werden.

2.2.3.2 Initialisierung

Vor dem Start des evolutionären Algorithmus muss eine erste Population geschaffen werden. Klassischerweise werden die Individuen gleichverteilt zufällig über den Suchraum verteilt ohne problembezogenes Wissen einzubringen. Dies entspricht dem Konzept der maximalen Entropie, was die Wahrscheinlichkeit erhöht, Lösungen nahe eines Optimums bereits in der ersten Generation zu erhalten. Bei reellwertigen Kodierungen wird hier üblicherweise der Suchraum zur Initialisierung auf einen Bereich beschränkt.

Eine andere, in der Praxis sicher häufiger anzutreffende Möglichkeit ist, durch Einbringen von problembezogenem Wissen Individuen mit einer höheren durchschnittlichen Güte als bei Gleichverteilung zu erzeugen. Dies könnte zum Beispiel durch den Einsatz von Approximationsalgorithmen oder auch von Heuristiken realisiert werden.

2.2.3.3 Selektion zur Reproduktion

Die Auswahl der Elternindividuen zur Reproduktion, also zur Ausführung von Rekombination und Mutation, kann auf unterschiedliche Art erfolgen. Bei den Evolutionsstrategien ist eine gleichverteilt zufällige Auswahl aus der Population üblich, wobei die Fitness der Individuen nicht betrachtet wird. Im Gegensatz dazu stammt aus den genetischen Algorithmen der Ansatz der fitnessproportionalen Selektion, bei der von positiven Fitnesswerten ausgegangen wird. Hier wird ein Individuum $\vec{x}^{(p)}$ aus der Population P mit Wahrscheinlichkeit

$$Prob_S(\vec{x}^{(p)}) = \left(\frac{f(\vec{x}^{(p)})}{\sum_{\vec{x} \in P} f(\vec{x})} \right), \tag{2.1}$$

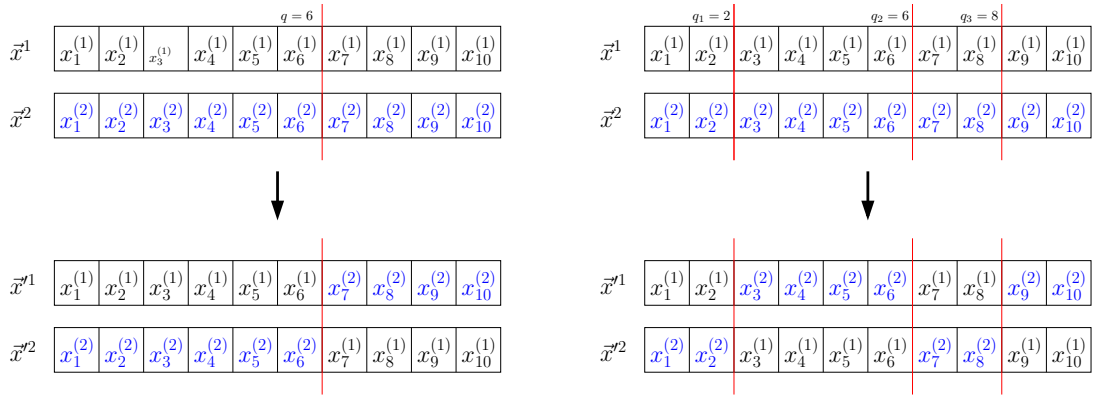
wobei f die Fitness angibt, gewählt.

Weitere Verfahren, welche auch bei der Selektion zur Reproduktion einsetzbar sind, werden in Kapitel 2.2.3.6 aufgeführt.

2.2.3.4 Rekombination

Die Kreuzung zweier oder mehr Elternindividuen zur Erzeugung von Nachkommen spielt je nach Problemstruktur eine wichtige Rolle.

Der klassische, durch die natürliche Evolution motivierte Ansatz ist die sogenannte *Ein-Punkt-Rekombination*, welche in Abbildung 2.3(a) dargestellt ist. Hierbei wird der Kreuzungs-Punkt q im allgemeinen gleichverteilt zufällig aus $\{1, \ldots, n\}$, möglicherweise auch mit anderer Verteilung, bestimmt und zwei neue Individuen erzeugt.



(a) Ein-Punkt-Rekombination

(b) 3-Punkt-Rekombination

Abbildung 2.3: Schematische Darstellung von Rekombination

Die Verallgemeinerung der Ein-Punkt-Rekombination ist die *k-Punkt-Rekombination*. Hierbei existiert nicht nur ein Schnittpunkt, sondern k viele. Die Kinderindividuen werden entsprechend alternierend aus den Eltern zusammengesetzt. Ein Beispiel für $k = 3$ findet sich in Abbildung 2.3(b).

Eine Variante dieser einfachen Rekombination ist die *uniforme Rekombination*, welche ebenfalls in diskreten wie auch in kontinuierlichen Suchräumen möglich ist. Hierbei wird je Position i mit einer Wahrscheinlichkeit von $\frac{1}{|P|}$ für jedes Elternindividuum $\vec{x}^{(p)}$ die Komponente zufällig aus den Eltern ausgewählt.

Insbesondere für den Raum $\mathbb{R}^n$ existieren weitere Varianten. Bei der recht häufig eingesetzten *intermediären Rekombination* wird für jede Komponente der Wert durch den Durchschnittswert der jeweiligen Komponenten der Eltern gegeben.

$$\begin{pmatrix} x'_1 \\ x'_2 \\ \vdots \\ x'_n \end{pmatrix} = \begin{pmatrix} (x_1^{(1)} + x_1^{(2)} + \ldots + x_1^{(v)})/v \\ (x_2^{(1)} + x_2^{(2)} + \ldots + x_2^{(v)})/v \\ \vdots \\ (x_n^{(1)} + x_n^{(2)} + \ldots + x_n^{(v)})/v \end{pmatrix}$$

Hierbei sei v die Anzahl der Eltern, da intermediäre und uniforme Rekombination nicht auf zwei Eltern je Kind beschränkt sind.

Die Rekombinationsvarianten für mit Permutationen kodierten Individuen müssen zur Erhaltung der Gültigkeit besonderen Bedürfnissen genügen. Hier sollen diese nicht weiter beschrieben werden, daher sei auf Bäck u. a. (1997) verwiesen.

Weiterhin lässt sich feststellen, dass das Einbringen von problemspezifischem Wissen, wie auch bei der Repräsentation, auch bei der Rekombination sehr sinnvoll sein kann.

2.2.3.5 Mutation

Die klassische Mutation, die Ein-Punkt-Mutation, ist bei Individuen aus $\{0,1\}^n$ das Ersetzen des Werts jeder Stelle mit dem jeweils anderen (also x_i durch $1 - x_i$) mit einer sogenannten Mutationswahrscheinlichkeit p_{mut}, welche üblicherweise sehr klein gewählt wird, da die Ähnlichkeit zum ursprünglichen Individuum hoch sein soll. Diese Mutation ist in Abbildung 2.4 dargestellt.

Diese Mutation kann ebenso bei Individuen aus $\mathbb{N}^n$ angewendet werden, wenn ein Maximalwert m gegeben wird. Dann ist die Mutation der Komponente x_i gegeben durch $m - x_i$.

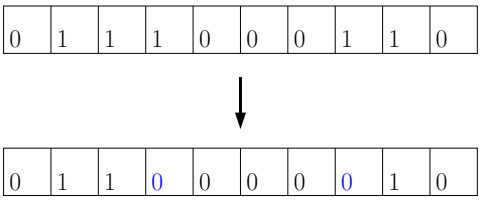


Abbildung 2.4: Schematische Darstellung von Ein-Punkt-Mutation

Bei Individuen aus $\mathbb{N}^n$ und $\mathbb{Z}^n$ kann auch mutiert werden, indem aus einem beschränkten Wertebereich das zu mutierende Bit gleichverteilt zufällig neu gesetzt wird. Eine andere Variante ist das Addieren oder Subtrahieren einer Zufallszahl entsprechend der Binomialverteilung (Feller, 1968) oder der Verteilung der maximalen Entropie (Rudolph, 1994). Auch die Normalverteilung gefolgt von Runden auf ganzzahlige Werte kann zum Einsatz kommen.

Eine vergleichbare Technik kommt klassischerweise bei Evolutionsstrategien zum Einsatz. Hier werden also die einzelnen Komponenten $x_i^{(p)}$ ($i \in \{1,\ldots,n\}$) eines Individuums p mutiert durch Addition einer normalverteilten Zufallszahl $N_{\mathbb{R}}(0,\sigma_i^{(p)})$ mit Erwartungswert 0:

$$x_i'^{(p)} = x_i'^{(p)} + N_{\mathbb{R}}(0,\sigma_i^{(p)})$$

Die Standardabweichungen σ_i (welche auch als Schrittweiten bezeichnet werden können) sind also abhängig von der Komponente und dem Individuum, weshalb sie als Strategiekomponente des Individuums bezeichnet werden. In den einfachen Varianten der Evolutionsstrategien gilt $\sigma_i^{(p)} = \sigma_j^{(p)}$ für alle $i,j \in \{1,\ldots,n\}$, was allerdings im allgemeinen ein ungünstigeres Explorationsverhalten verursacht.

Es existieren Verfahren, nach denen die Strategiekomponente erweitert und abhängig vom lokalen Suchverhalten verändert wird. Für eine Darstellung dieser Selbstadaption sei auf Hildebrand (2002, 2004a) verwiesen.

Mutationsoperatoren für Permutationen sind in Bäck u. a. (1997) zu finden.

2.2.3.6 Selektion

Wie bereits in Abschnitt 2.2.3.3 erwähnt ist die Selektion bei den klassischen genetischen Algorithmen fitnessproportional. Diese, entsprechend den Wahrscheinlichkeiten in Gleichung 2.1 durchgeführte Selektion, wird auch als *Rouletteradselektion* bezeichnet, was daher rührt, dass man sich vorstellt, jedem Individuum würde ein Bereich auf einem Rad zugewiesen werden, dessen Größe proportional zu seiner Fitness ist. Bei dem Drehen dieses Rouletterades werden nun Individuen mit höherer Fitness mit höherer Wahrscheinlichkeit ausgewählt. Allerdings kann bei stark unterschiedlichen Fitnesswerten eine Auswahl eines Individuums zu stark favorisiert werden, was oft nicht erwünscht ist, da hierbei schwächere Individuen keine Möglichkeit haben, in die nächste Generation zu kommen. Hierdurch würden möglicherweise wenige Individuen mehrfach für die Folgepopulation ausgewählt. Weiterhin werden bei sehr ähnlichen Fitnesswerten möglicherweise gar nicht die besseren Individuen gewählt.

Aus diesen Problemen heraus sind einige weitere Selektionmethoden entstanden. So werden bei der *Rangselektion* die Individuen in eine der Fitness entsprechenden Reihenfolge gebracht aus der eine Wahrscheinlichkeitsverteilung zur Selektion hergeleitet wird. Der Vorteil ist hierbei, dass die Dominanz besonders ‚starker' Individuen verringert wird, was die Diversität der Population fördert sowie dass bei sehr ähnlichen Individuen die Gefahr des Nichtüberlebens der besten Individuen nicht zu groß wird.

Bei der *Turnierselektion* wird aus einer Menge von k gleichverteilt zufällig ausgewählten Individuen das beste deterministisch übernommen. Umso größer hierbei also k ist, desto größer ist die Wahrscheinlichkeit,

dass sehr gute Individuen überleben und schwächere nicht in die nächste Generation übernommen werden. Ein kleines k erhöht die Diversität.

Bei den bisher beschriebenen Selektionmethoden wird jeweils ein Individuum selektiert, wobei dieses auch mehrfach in die Folgepopulation aufgenommen werden kann. Bei der aus den Evolutionsstrategien stammenden (+)- und (,)-Selektion wird ein anderer Ansatz verfolgt. Bei der (+)-Selektion werden aus den μ Individuen der letzten Generation und den λ Nachkommen wiederum μ Individuen ausgewählt. Dies geschieht deterministisch entsprechend der Fitness, es werden also die besten μ Individuen übernommen. Bei der (,)-Selektion werden aus den λ Nachkommen μ viele für die neue Generation ausgewählt. Hierbei wird also nur aus den Nachkommen, nicht aus der Vorgängerpopulation selektiert. Daher muss $\lambda \geq \mu$ gelten.

Üblicherweise werden diese Selektionsverfahren auch als $(\mu + \lambda)$ beziehungsweise (μ, λ)-Selektion bezeichnet (zusammengefasst kurz auch $(\mu \overset{+}{,} \lambda)$).

2.2.3.7 Abbruchkriterium

Dieses hier dargestellte Verfahren wird solange durchgeführt und wiederholt, bis ein gegebenes Abbruchkriterium erfüllt ist. Dies kann zum Beispiel das Erreichen eines bestimmten Fitnesswertes sein. Möglich ist auch, bei Unterschreiten einer Mindestverbesserung je Generation abzubrechen, da hieraus geschlossen werden kann, dass die Verbesserungen nur noch marginal sein dürften.

In jedem Fall sollte eine Maximalzahl von Generationen oder Fitnessauswertungen herangezogen werden, um die Laufzeit zu beschränken, damit der Algorithmus nicht aus Zeitgründen unplanmäßig abgebrochen werden muss.

2.2.4 Niching

Die Beschreibungen in diesem Kapitel folgen Bäck u. a. (1997, Kapitel C6.1).

Niching-Methoden ermöglichen den Einsatz evolutionärer Algorithmen auf Domänen, in denen die Erkennung von mehreren Lösungen gewünscht ist. Ebenso wird erreicht, dass diese im Verlauf der Evolution erhalten bleiben.

Hier werden zwei Ansätze, Fitness Sharing und Crowding kurz angesprochen.

Fitness Sharing

Fitness Sharing ist eine sogenannte *Fitness Scaling*-Technik. Dies bedeutet, dass nach Abschätzung der Fitness diese skaliert wird. Der Grundgedanke bei der Methode Fitness Sharing ist, dass sich ähnliche Individuen Fitness teilen, so dass die Anzahl der Individuen, welche einen Bereich der Fitnesslandschaft belegen durch diese Anzahl begrenzt wird. Dadurch folgt theoretisch die Anzahl von Individuen an einem hohen Fitnesswert der Höhe dieses Werts.

Die skalierte Fitness $f_s(i)$ eines Individuums i wird aus der ursprünglichen Fitness $f(i)$ bestimmt durch

$$f_s(i) = \frac{f(i)}{\sum_{j=1}^{\mu} sh(d(i,j))}$$

wobei μ die Anzahl aller Individuen der Population angibt. Weiterhin ist d eine Distanzfunktion und sh eine sogenannte Sharingfunktion, welche eine 1 bei Gleichheit der Individuen i und j und eine 0 bei Unterschreitung eines Grenzwerts der Ähnlichkeit zurückgibt. Beliebige Werte aus $[0;1]$ sind ebenfalls möglich.

Der Grenzwert wird gegeben durch σ_{share} wobei die Distanz zwischen zwei Individuen größer oder gleich σ_{share} angibt, dass sich deren Fitness nicht beeinflusst.

Entsprechend ist die *Sharing Funktion sh* gegeben durch

$$sh(d) = \begin{cases} 1 - (\frac{d}{\sigma_{share}})^{\alpha} & \text{wenn } d < \sigma_{share} \\ 0 & \text{sonst} \end{cases}$$

Hierbei ist α ein die Form der Sharing Funktion beeinflussender Parameter. Zu Hinweisen zur Belegung von α und σ_{share} empfiehlt Bäck u. a. (1997, Kapitel C6.1) die Lektüre von Deb (1989).

Crowding

Die Grundidee bei *Crowding* ist, während der Evolution die ähnlicheren Individuen miteinander zu vergleichen und das mit der höheren Fitness in die Folgegeneration zu übernehmen. Dadurch, dass unähnlichere Individuen von vornherein aussortiert werden, wird eine bestehende Diversität der Population erhalten.

Der Ablauf eines evolutionären Algorithmus mit Crowding ist in Algorithmus 2.2 dargestellt. Hierbei werden zunächst alle Individuen in $\frac{\mu}{2}$ Paare eingeteilt. Auf diesen wird Rekombination und Mutation angewandt und ein Vergleich der Fitness zwischen den ähnlicheren Nachkommen mit den Eltern findet statt wobei schließlich das Individuum mit der höheren Fitness die Nachfolgegeneration erreicht.

Eingabe: g Anzahl der Generationen, μ Populationsgröße
Ausgabe: P_g

Initialisiere Startpopulation P_0 in Array a von Länge μ
for $t = 1$ to g **do**
  Mische die Reihenfolge der Elemente in a
  for $i = 0$ to $\frac{\mu}{2} - 1$ **do**
    Elter $p_1 = a[2i + 1]$
    Elter $p_2 = a[2i + 2]$
    Kinder $\{c_1, c_2\} =$ Rekombination von p_1 und p_2
    Mutiere c_1 und c_2
    if $d(p_1, c_1) + d(p_2, c_2) \leq d(p_1, c_2) + d(p_2, c_1)$ **then**
      if $(f(c_1) > f(p_1))$ **then**
        $a[2i + 1] = c_1$
      end if
      if $(f(c_2) > f(p_2))$ **then**
        $a[2i + 1] = c_2$
      end if
    else
      if $(f(c_2) > f(p_1))$ **then**
        $a[2i + 1] = c_2$
      end if
      if $(f(c_1) > f(p_2))$ **then**
        $a[2i + 1] = c_1$
      end if
    end if
  end for
end for

Algorithmus 2.2: Evolutionärer Algorithmus mit Crowding

2.3 Neuronale Netze

Dieses Kapitel stellt die Grundlagen künstlicher neuronaler Netze vor. Hierbei wird kurz auf die biologische Motivation eingegangen woraufhin unterschiedliche Arten von Netzen erklärt werden. Ein besonderer Schwerpunkt wird auf Grund der Bedeutung für diese Arbeit auf sogenannte Feed-Forward-Netze gelegt.

2.3.1 Biologische Motivation und grundlegende Umsetzung

Die Modellierung künstlicher neuronaler Netze (Netze künstlicher Neuronen) ist an natürliche neuronale Netze (Netze natürlicher Neuronen) angelehnt. Zwar sind die künstlichen Netze stark vereinfacht, so dass nur grundlegende Parallelen erkennbar sind, dennoch sind die wichtigsten Elemente in beiden Neuronentypen wiederfindbar. Die hier gegebene Beschreibung orientiert sich an Anderson (1996), Borgelt u. a. (2003) und Beyer u. a. (2005).

Ein *Neuron* ist eine Nervenzelle, welche elektrische Energie sammelt und weiterleitet. Es besteht aus einem *Zellkörper* mit *Zellkern* und aus *Zellfortsätzen*. Der Zellkörper übernimmt einfache informationsverarbeitende Aufgaben. Zu den Zellfortsätzen zählen die *Dendriten*, welche für die Aufnahme von Energie und die Weiterleitung zum Zellkörper zuständig sind, sowie das *Axon*, welches Energie vom Zellkörper fort zu Muskelfasern oder anderen Neuronen leitet. Üblicherweise ist nur ein Axon vorhanden, welches in den *Synapsen* endet. Eine Darstellung eines solchen Neurons ist in Abbildung 2.5 zu finden.

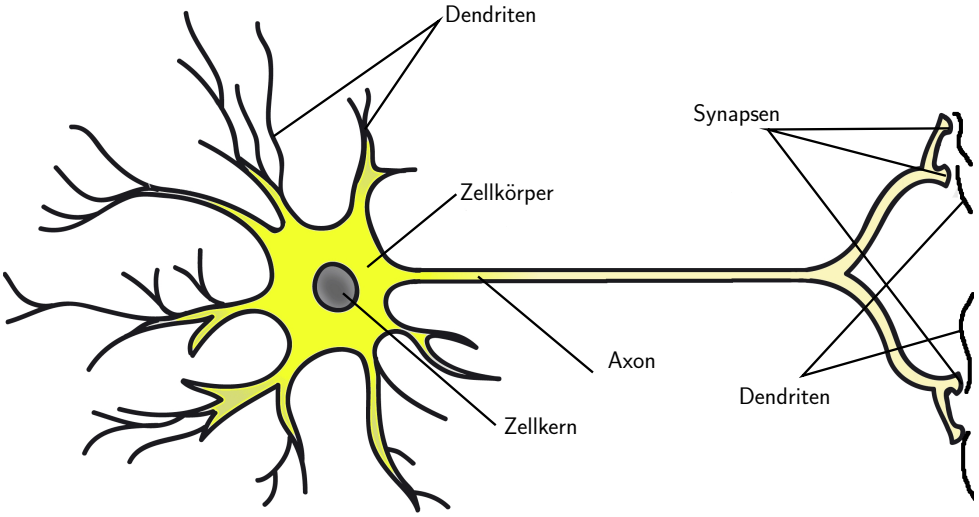


Abbildung 2.5: Schematische Darstellung eines natürlichen Neurons

Über die Dendriten werden chemische Stoffe, sogenannte *Neurotransmitter*, von den Synapsen anderer Neuronen empfangen. Diese Stoffe verursachen, je nach Typ der Synapse die diesen Stoff ausschüttet, eine Erhöhung (*exzitatorische Synapse*) oder Senkung (*inhibitorische Synapse*) des Potentials des Inneren des Neurons.

Normalerweise befindet sich ein Neuron im *Ruhezustand*, so dass die Spannung im Inneren etwa -80 Millivolt beträgt. Wird es nun hinreichend gereizt (der Schwellwert beträgt etwa -50 Millivolt), findet eine Depolarisierung statt, wodurch die innere Spannung im Verhältnis zum äußeren Potential kurzzeitig bis zu 30 Millivolt beträgt. Dieses *Aktionspotential* wird über das Axon weitergeleitet. Am Ende des Axons bewirkt dies wiederum die Ausschüttung von Neurotransmittern an den Synapsen.

Durch die Verschaltung vieler Neuronen wird eine parallele, schnelle und komplexe Informationsverarbeitung unter Wirkung elektrochemischer Prozesse ermöglicht. Wissen wird implizit durch die Vernetzungsstruktur gespeichert.

Aus den dargestellten Grundlagen ergibt sich eine Struktur für künstliche Neuronen und künstliche neuronale Netze (angelehnt an Hildebrand, 2004b):

Definition 12 *Ein* künstliches Neuron *(im weiteren* Neuron*) ist ein Tupel*

$$N = (\vec{x}, \vec{w}, \theta, f_v, f_a, f_o)$$

hierbei sind

- $\vec{x} = (x_1, \ldots, x_n)$ *Eingänge*
- $\vec{w} = (w_1, \ldots, w_n)$ *Gewichte*
- θ *Schwellwert*
- $f_v : \mathbb{R}^{2n+1} \rightarrow \mathbb{R}$ *Verknüpfung von Eingängen, Gewichten und Schwellwert*
- $f_a : \mathbb{R} \rightarrow \mathbb{R}$ *Aktivierungsfunktion*
- $f_o : \mathbb{R} \rightarrow \mathbb{R}$ *Ausgabefunktion*
- $x_i, w_i, \theta, z \in \mathbb{R}, i \in \{1, \ldots, n\}$.

Die Verknüpfung von Neuronen zu einem neuronalen Netz wird im folgenden gegeben:

Definition 13 *Ein* künstliches neuronales Netz *(im weiteren* neuronales Netz*) ist ein Tupel*

$$\mathfrak{N} = (\mathcal{N}, \mapsto, \mathcal{I}, \mathcal{O}, Alg_p, Alg_l)$$

mit

- $\mathcal{N} = \{N_1, \ldots, N_n\}$ *Menge der Neuronen*
- $\mapsto \subset \mathcal{N} \times \mathcal{N}$ *Verknüpfungsstruktur*
- $\mathcal{I} \subset \mathcal{N}$ *Menge der Eingangsneuronen*
- $\mathcal{O} \subset \mathcal{N}$ *Menge der Ausgabeneuronen*
- Alg_p *Propagierungsalgorithmus*
- Alg_l *Lernalgorithmus*

2.3.2 Perzeptron

Das *Perzeptron* ist ein einfaches Neuron, das 1958 von Frank Rosenblatt auf Basis des *McCulloch-Pitts-Neurons* (McCulloch und Pitts, 1943), welches als das erste auf natürlichen Neuronen basierende Modell gilt, entwickelt wurde (siehe Rojas, 1996; Rosenblatt, 1958).

Definition 14 *Das* Perzeptron *ist gegeben durch das Tupel*

$$N_{Perz} = (\vec{x}, \vec{w}, \theta, f_v, f_a, f_o)$$

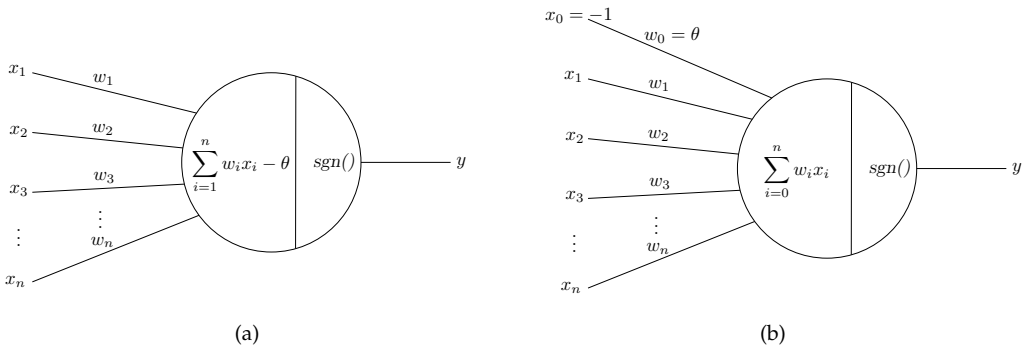
mit

- *Eingängen* $\vec{x} = (x_1, \ldots, x_n), x_i \in \mathbb{R}, i \in \{1, \ldots, n\}$
- *Gewichten* $\vec{w} = (w_1, \ldots, w_n), w_i \in \mathbb{R}, i \in \{1, \ldots, n\}$
- *Schwellwert* $\theta \in \mathbb{R}$
- *Verknüpfung* $f_v(\vec{x}, \vec{w}, \theta) = \sum_{i=1}^{n} w_i x_i - \theta$
- *Aktivierungsfunktion* $f_a = sgn(\cdot)$
- *Ausgabefunktion* $f_o = Identität$

wobei die Aktivierungsfunktion durch

$$sgn(x) = \begin{cases} -1 & \text{für } x < 0 \\ 1 & \text{für } x \geq 0 \end{cases}$$

gegeben ist. Ein solches Neuron ist in Abbildung 2.6(a) schematisch dargestellt.



(a) (b)

Abbildung 2.6: Perzeptron

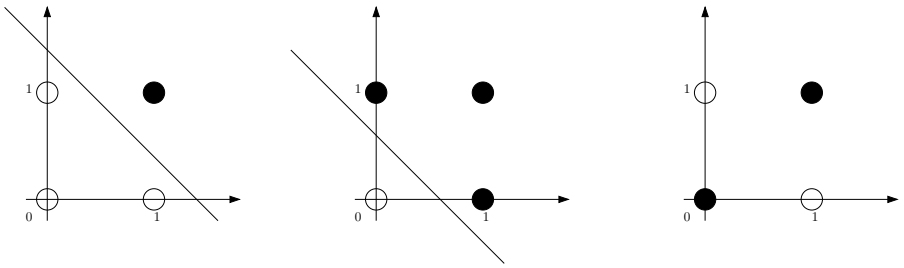
Hieraus ergibt sich auch, dass die Ausgabe y des Neurons berechnet wird durch:

$$y = sgn(\sum_{i=1}^{n} w_i x_i - \theta)$$

In Abbildung 2.6(b) ist eine äquivalente Variation zu sehen, welche implementatorische Vorteile bietet. Wir sprechen in diesem Fall von einem *impliziten Schwellwert*, wobei Eingänge, Gewichte und Verknüpfung folgendermaßen abgewandelt sind:

- Eingängen $\vec{x} = (x_0 = -1, x_1, \ldots, x_n), x_i \in \mathbb{R}, i \in \{1, \ldots, n\}$
- Gewichten $\vec{w} = (w_0 = \theta, w_1, \ldots, w_n), w_i \in \mathbb{R}, i \in \{1, \ldots, n\}$
- Verknüpfung $f_v(\vec{x}, \vec{w}) = \sum_{i=0}^{n} w_i x_i$

Definition 15 *Zwei Mengen von Punkten A und B in einem n-dimensionalen Raum heißen* linear separierbar, *wenn* $n + 1$ *reelle Zahlen* $w_1, \ldots, w_{n+1}$ *existieren, so dass für jeden Punkt* $(x_1, x_2, \ldots, x_n) \in A$ *gilt* $\sum_{i=1}^{n} w_i x_i > w_{n+1}$ *und für jeden Punkt* $(x_1, x_2, \ldots, x_n) \in B$ *gilt* $\sum_{i=1}^{n} w_i x_i < w_{n+1}$.



(a) OR (b) AND (c) XOR

Abbildung 2.7: OR und AND sind linear separierbar, jedoch nicht XOR

Das Perzeptron kann nur linear separierbare Eingaben klassifizieren. So sind die beiden Funktionen OR und AND klassifizierbar, jedoch nicht XOR. Geometrisch bedeutet dies, wie in Abbildung 2.7 zu sehen ist, dass die Klassifikation durch Trennung der Eingabeklassen mit einer Geraden möglich sein muss. Dies ist bei XOR (Abbildung 2.7(c)) nicht der Fall.

Die Gewichte und der Schwellwert des Perzeptrons geben eine solche Gerade an:

$$f(\vec{x}) = w_1 x_1 + w_2 x_2 + \ldots + w_n x_n - \theta$$

Das Auffinden einer solchen Trennlinie ist nun Aufgabe des Lernalgorithmus.

Perzeptron Lernalgorithmus

Im folgenden wird von einem impliziten Schwellwert ausgegangen. Gegeben ist eine Trainingsmenge $T = A \cup B$. Gesucht ist nun ein Vektor $\vec{w}$ der alle Beispiele aus A korrekt in die eine Klasse und alle Beispiele aus B korrekt in die andere Klasse einteilt.

Initialisiere Gewichtsvektor $\vec{w}$ zufällig;
while ($\exists$ Fehlklassifikation) **do**
  Wähle $\vec{x} \in A \cup B$;
  if ($\vec{w} \in A$ und $\vec{w} \cdot \vec{x} \leq 0$) **then**
    $\vec{w} = \vec{w} + \vec{x}$
  else if ($\vec{w} \in B$ und $\vec{w} \cdot \vec{x} \geq 0$) **then**
    $\vec{w} = \vec{w} - \vec{x}$
  end if
end while

Algorithmus 2.3: Perzeptron Lernalgorithmus (nach Rojas, 1996; Hildebrand, 2004b)

Der Algorithmus 2.3 korrigiert den Gewichtsvektor $\vec{w}$ immer genau dann, wenn dieser eine Eingabe nicht korrekt einordnet. Dieses Verfahren ist äquivalent zur Gewichtsanpassung durch

$$\vec{w}_{i+1} = \vec{w}_i + \eta \cdot (d_i - y_i) \cdot \vec{x}_i$$

wenn für die sogenannte *Lernrate* $\eta = 0{,}5$ gilt. Hierbei ist y_i die aktuelle Ausgabe des Perzeptrons und $d_i = 1$ für $\vec{x}_i \in A$ und $d_i = -1$ für $\vec{x}_i \in B$.

Für den Beweis der Konvergenz des Perzeptrons für linear separierbare Eingaben sei auf Rojas (1996) verwiesen.

2.3.3 Feedforward-Netze

Um nicht auf linear separierbare Eingaben beschränkt zu bleiben besteht die Möglichkeit, mehrere Neuronen in Schichten zu verknüpfen. Hierzu definieren wir uns eine etwas andere Version der Neuronen:

Definition 16 *Ein Multi-Layer-Feed-Forward-Neuron (*MLFF-Neuron*) ist gegeben durch ein Tupel*

$$N_{MLFF} = (\vec{x}, \vec{w}, \theta, f_v, f_a, f_o)$$

mit

- *Eingängen* $\vec{x} = (x_0 = -1, x_1, \ldots, x_n), x_i \in [-1,1] \subset \mathbb{R}, i \in \{1, \ldots, n\}$
- *Gewichten* $\vec{w} = (w_0 = \theta, w_1, \ldots, w_n), w_i \in \mathbb{R}, i \in \{1, \ldots, n\}$
- *Schwellwert* $\theta \in \mathbb{R}$
- *Verknüpfung* $f_v(\vec{x}, \vec{w}) = \sum_{i=0}^{n} w_i x_i$
- *Aktivierungsfunktion* $f_a = tanh(\cdot)$
- *Ausgabefunktion* $f_o = Identität$

Zu bemerken ist hier, dass der Schwellwert in den Gewichtsvektor integriert wird. Die Aktivierungsfunktion *Tangens hyperbolicus* $tanh(\cdot)$ ist gegeben durch

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

Der Graph ist in Abbildung 2.8(a) zu sehen. Die Berechnung der Ausgabe y des Neurons ist gegeben durch

$$y = tanh(\sum_{i=0}^{n} w_i x_i)$$

Dies ist die sogenannte bipolare Variante des MLFF-Neurons, welche für diese Arbeit die größere Rolle spielt. Eine weitere Variante ist das unipolare MLFF-Neuron, bei dem für die Eingaben $x_i \in [0,1] \subset \mathbb{R}$ gilt. Dazu passend wird als Aktivierungsfunktion die logistische Funktion genutzt:

$$lgc(x) = \frac{1}{1 + e^{-x}}$$

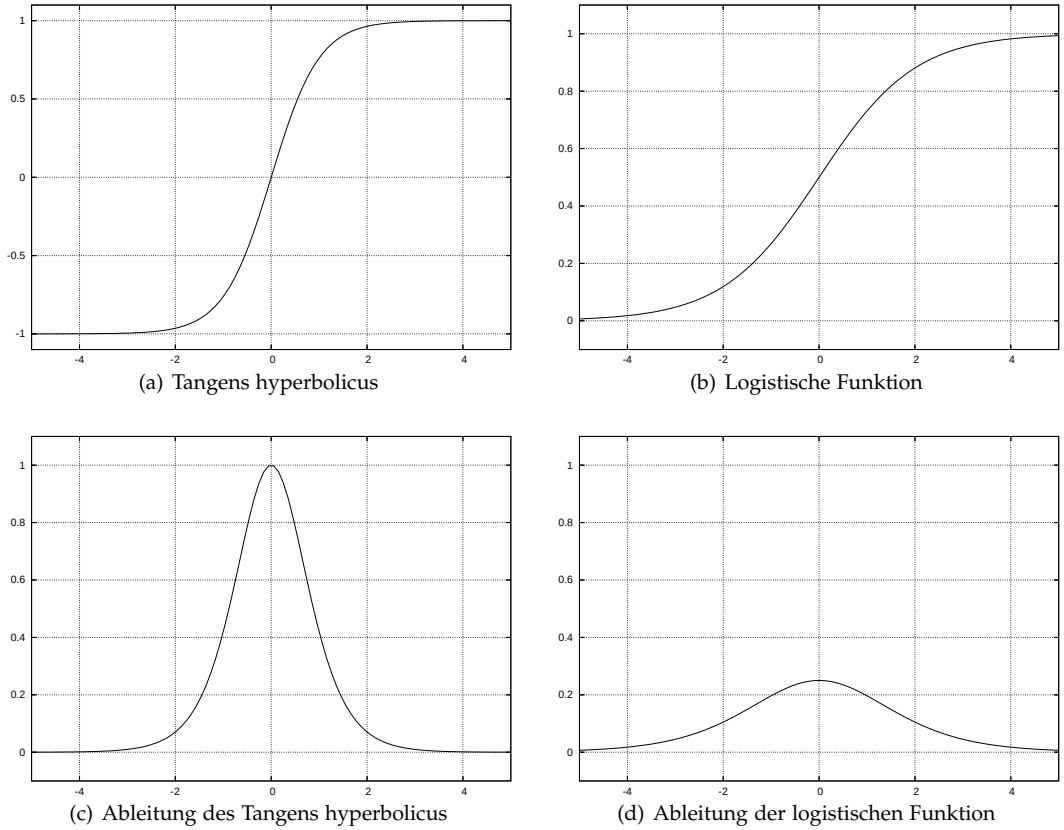
Sie ist in Abbildung 2.8(b) zu sehen. Die Berechnung bei dieser Variante folgt analog zum bipolaren Neuron.

Die für später vorgestellte Verfahren wichtigen ersten Ableitungen der Funktionen $lgc()$ und $tanh()$ sind gegeben durch

$$lgc'(x) \quad = \quad f(x) \cdot (1 - f(x))$$

und

$$tanh'(x) = 1 - f^2(x).$$



(a) Tangens hyperbolicus

(b) Logistische Funktion

(c) Ableitung des Tangens hyperbolicus

(d) Ableitung der logistischen Funktion

Abbildung 2.8: Aktivierungsfunktionen für Multi-Layer-Feed-Forward-Neuronen und ihre Ableitungen

Sie sind in Abbildung 2.8(c) und 2.8(d) zu sehen.

In Abbildung 2.9 ist ein Multi-Layer-Feed-Forward-Netz skizziert. Eine besondere Rolle spielen hier die Eingabeneuronen $N_{0,i}$ ($i \in \{1, \ldots, m_0\}$) da diese keine Berechnung durchführen sondern nur den einzugebenden Wert an das Netz weiterreichen. Die Ausgabe wird an den Ausgabeneuronen $N_{n,i}$ ($i \in \{1, \ldots, m_n\}$) abgenommen. Zu bemerken ist weiterhin die Einteilung in Ebenen und die ebenenweise Vernetzung (welche nicht vollständig sein muss).

Die Berechnung innerhalb des Netzes folgt dem Propagierungsalgorithmus 2.4.

2.3.3.1 Lernalgorithmus: Backpropagation

Der Fehler, den ein Multi-Layer-Feed-Forward-Netz bei Anlegen von Eingaben bezüglich einer Soll-Ausgabe macht, spielt eine wichtige Rolle. Aus der Fehlerfunktion wird im weiteren das Lernverfahren, welches auf einem Gradientenabstieg auf eben dieser basiert, hergeleitet.

Definition 17 *Sei $T = \{t_1, \ldots, t_n\}$ die Menge aller Trainingsbeispiele mit Eingaben $\vec{x}^{t_i} = (x_1^{t_i}, \ldots, x_m^{t_i})^T$ und Soll-Ausgaben $\vec{d}^{t_i} = (d_1^{t_i}, \ldots, d_o^{t_i})^T$. Weiterhin sei $\vec{y}^{t_i} = (y_1^{t_i}, \ldots, y_o^{t_i})^T$ die Ausgabe des Multi-Layer-Feed-Forward-Netzes $\mathfrak{N}$ bei Eingabe $\vec{x}^{t_i}$.*

Dann ist die Fehlerfunktion Total Summed Squared Error (TSSE) *gegeben durch:*

$$E_{TSSE}(\mathfrak{N}, T) = \sum_{i=1}^{n} \sum_{j=0}^{o} (d_j^{t_i} - y_j^{t_i})^2$$

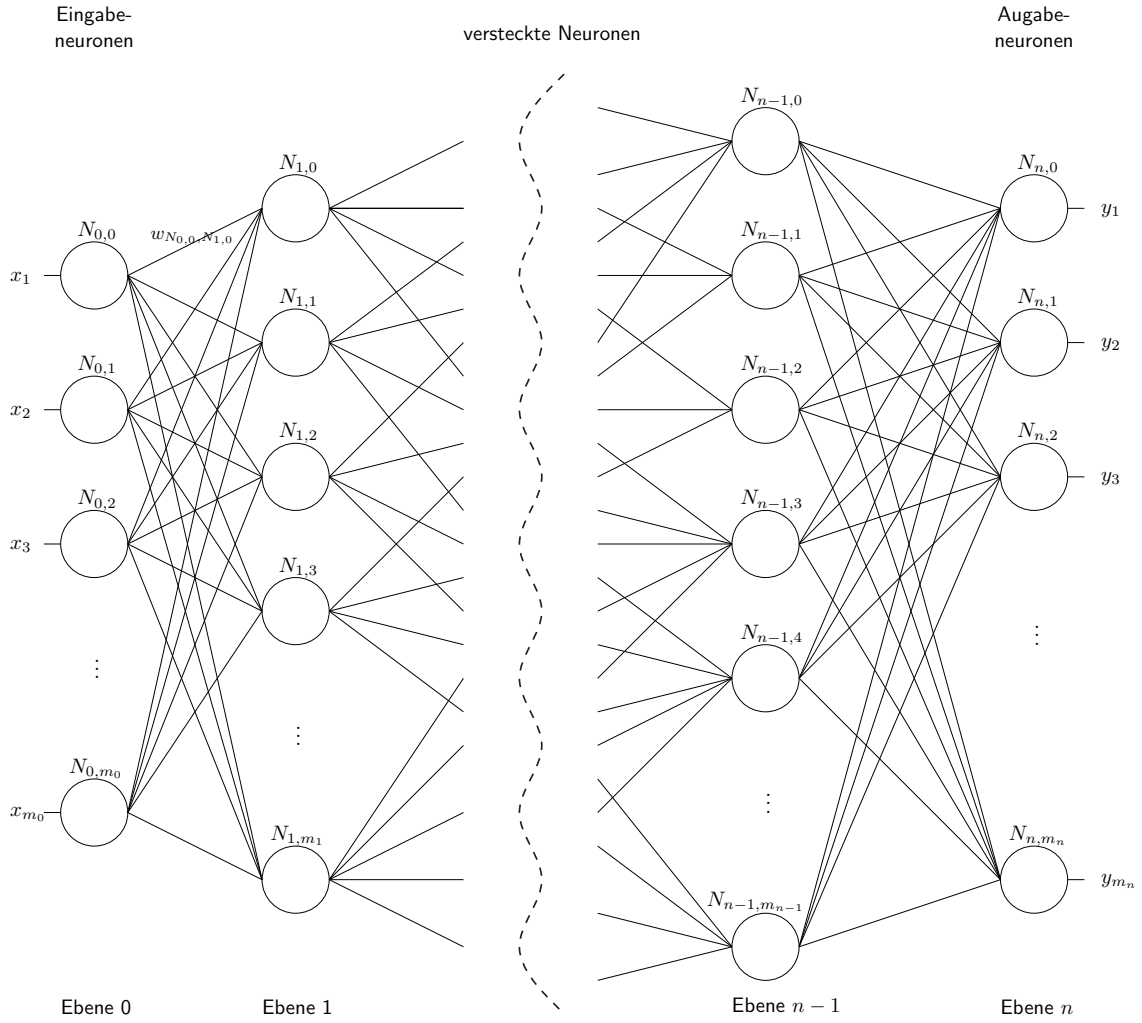


Abbildung 2.9: Skizzierung eines Multi-Layer-Feed-Forward-Netzes mit Bezeichnung der Neuronen, Ebenen und Ein- und Ausgaben. Exemplarisch ist an einer Verknüpfung ein Gewicht angezeichnet.

for all $N_{0,i}$ ($i \in \{1, \ldots, m_0\}$) **do**
  Stelle Eingabe am Ausgang zur Verfügung;
end for

for $i = 1$ to n **do**
  for $j = 0$ to m_n **do**
    Setze als Eingaben von $N_{i,j}$ die Ausgaben der
    Vorgängerneuronen im Netz;
    Berechne Ausgabe von Neuron $N_{i,j}$;
  end for
end for

Algorithmus 2.4: Propagierungsalgorithmus Alg_p im Multi-Layer-Feed-Forward-Netz

Der Gradientenabstieg ist ein Optimierungsverfahren, welches auf folgender Iteration beruht:

$$\vec{x}_{i+1} = \vec{x}_i - \eta \cdot \nabla f(\vec{x}_i)$$

Hierbei ist $\nabla f(\vec{x}_i)$ die Menge der partiellen Ableitungen von $f(\vec{x}_i)$. Der Parameter η gibt die Schrittweite an. Dieses Verfahren wird nun auf die Fehlerfunktion $TSSE$ angewendet. Daraus ergibt sich folgende Anpassung des Gewichts von Neuron a auf einer Schicht $i-1$ zu Neuron b auf Schicht i (nach Hildebrand, 2004b):

$$w_{a,b} = w_{a,b} + \eta \cdot \delta_b \cdot y_a \tag{2.2}$$

Für die sogenannten *lokalen Gradienten* δ gilt hierbei

$$\delta_b = \begin{cases} (d_b - y_b) \cdot f'(z_b) & \text{falls Neuron } b \text{ auf Ausgabeschicht} \\ \left(\sum_{c \in Succ(b)} (\delta_c \cdot w_{b,c}) \right) \cdot f'(z_b) & \text{sonst.} \end{cases} \tag{2.3}$$

Hierbei sind a, b, c Neuronen. Mit $Succ(x)$ ist die Menge aller Nachfolger auf Schicht $i+1$ von Neuron x auf Schicht i bezeichnet. Dies bedeutet, dass dies die Menge aller Neuronen $w \in Succ(x)$ ist, für die eine Verbindung $(x, w) \in \mapsto$ existiert. f' ist die erste Ableitung der Aktivierungsfunktion und y_x die Ausgabe des Neurons x. Weiterhin ist z_x der interne Zustand von Neuron x. Die Herleitung dieser Gewichtsanpassung ist in Hildebrand (2004b) und Zell (1994) zu finden.

Der Ablauf des Lernens mit Hilfe der Backpropagationregel ist in Algorithmus 2.5 skizziert. $Prec(x)$ bezeichnet hierbei die Menge aller Neuronen $w \in Prec(x)$ für die eine Verbindung $(w, x) \in \mapsto$ im neuronalen Netz existiert. Die Menge aller Neuronen ist $\mathcal{N}$. Die Menge aller Trainingsbeispiele ist T. Die Bezeichnung der Neuronen und Gewichte folgt Abbildung 2.9.

In Zeile 1–5 werden alle Gewichte zufällig initialisiert. Dies kann zum Beispiel normalverteilt in einem gegebenen Intervall geschehen. In Zeile 7 beginnt eine Schleife, wobei das Kriterium κ zum einen eine maximale Anzahl an Lernschritten realisiert, zum anderen aber auch einen Abbruch des Lernens bei einer hinreichend kleinen Verbesserung von Schleifendurchlauf zu Schleifendurchlauf sein kann. Jeder dieser Durchläufe heißt *Epoche*. In jeder Epoche geschieht folgendes: In Zeile 8–10 wird ein Trainingsbeispiel ausgewählt und die entsprechende Eingabe an die Eingabeneuronen des Netzes gesetzt. Weiterhin wird der Propagierungsalgorithmus 2.4 aufgerufen, so dass dann die Ausgabe an den Ausgangsneuronen vorliegt.

Von Zeile 11–16 werden dann für alle Ausgabeneuronen die lokalen Gradienten δ mit Gleichung 2.3 bestimmt und dann für alle Verbindungen im neuronalen Netz, welche zu jedem dieser Ausgabeneuronen führen, die Gewichte abhängig von den gerade bestimmten δs entsprechend 2.2 aktualisiert.

Von Zeile 17–24 geschieht das selbe für alle versteckten Neuronen. Hierbei werden die Schichten von der Schicht, welche direkt neben der Ausgabeschicht liegt in Richtung der Eingabeschicht durchlaufen. Für jedes Neuron auf der jeweils aktuellen Schicht werden wieder entsprechend den Gleichungen 2.2 und 2.3 die zu ihnen hinführenden Gewichte angepasst. Durch dieses schichtweise Vorgehen ist sichergestellt, dass die notwendigen lokalen Gradienten bei jeder Anpassung zur Verfügung stehen.

In Zeile 24 wird wieder zu Zeile 8 gesprungen und ein neues Trainingsbeispiel ausgewählt. Nachdem jedes Trainingsbeispiel als Grundlage für einen Durchlauf gedient hat beginnt die nächste Epoche.

Dies geschieht, bis Kriterium κ greift.

1: **for all** $(x \in \mathcal{N})$ **do**
2: **for all** $(w \in Prec(x))$ **do**
3: Setze $w_{w,x}$ zufällig;
4: **end for**
5: **end for**
6:
7: **while** (Kriterium κ nicht erfüllt) **do**
8: **for all** $t \in T$ **do**
9: Setze Eingabe x_t an Eingangsneuronen
10: Alg_p // Algorithmus 2.4
11: **for** $j = 0$ to m_n **do**
12: $\delta_{N_{n,j}} = (d_{N_{n,j}} - y_{N_{n,j}}) \cdot f'(z_{N_{n,j}})$
13: **for all** $X \in Prec(N_{n,j})$ **do**
14: $w_{X,N_{n,j}} = w_{X,N_{n,j}} + \eta \cdot \delta_{N_{n,j}} \cdot y_X$
15: **end for**
16: **end for**
17: **for** $i = n - 1$ downto 0 **do**
18: **for** $j = 0$ to m_i **do**
19: $\delta_{N_{i,j}} = \left(\sum_{Z \in Succ(N_{i,j})} (\delta_Z \cdot w_{N_{i,j},Z})\right) \cdot f'(z_{N_{i,j}})$
20: **for all** $X \in Prec(N_{i,j})$ **do**
21: $w_{X,N_{i,j}} = w_{X,N_{i,j}} + \eta \cdot \delta_{N_{i,j}} \cdot y_X$
22: **end for**
23: **end for**
24: **end for**
25: **end for**
26: **end while**

Algorithmus 2.5: Backpropagationalgorithmus zum Setzen der Gewichte eines Multi-Layer-Feed-Forward-Netzes

2.3.3.2 Lernalgorithmus: Resilient-Propagation

Der von Martin Riedmiller und Heinrich Braun entwickelte Lernalgorithmus Rprop (Riedmiller und Braun, 1992, 1993; Riedmiller, 1994) ist eine Variante des Backpropagation, welche im allgemeinen deutlich schneller konvergiert (Igel und Hüsken, 2003). Der Unterschied liegt in der Anpassung jedes einzelnen Gewichts, welche in zwei Schritten statt findet. Im ersten (Gleichung 2.4) wird der Betrag $\Delta_{ij}^{(t)}$ der Gewichtsänderung von Neuron i zu Neuron j bestimmt, im zweiten (Gleichung 2.5) die Änderung $\Delta w_{ij}^{(t)}$ mit Vorzeichen ($t \in \mathbb{N}$ gibt den Iterationsschritt an):

$$\Delta_{ij}^{(t)} = \begin{cases} \min\left(\eta^+ \cdot \Delta_{ij}^{(t-1)}; \Delta_{\max}\right) & \text{wenn } \frac{\partial E^{(t-1)}}{\partial w_{ij}^{(t-1)}} \cdot \frac{\partial E^{(t)}}{\partial w_{ij}^{(t)}} > 0 \\ \max\left(\eta^- \cdot \Delta_{ij}^{(t-1)}; \Delta_{\min}\right) & \text{wenn } \frac{\partial E^{(t-1)}}{\partial w_{ij}^{(t-1)}} \cdot \frac{\partial E^{(t)}}{\partial w_{ij}^{(t)}} < 0 \\ \Delta_{ij}^{(t-1)} & \text{sonst} \end{cases} \tag{2.4}$$

$$\Delta w_{ij}^{(t)} = \begin{cases} -\operatorname{sgn}\left(\frac{\partial E^{(t-1)}}{\partial w_{ij}^{(t-1)}}\right) \cdot \Delta_{ij}^{(t)} & \text{wenn } \frac{\partial E^{(t-1)}}{\partial w_{ij}^{(t-1)}} \cdot \frac{\partial E^{(t)}}{\partial w_{ij}^{(t)}} \geq 0 \\ -\Delta w_{ij}^{(t-1)} & \text{wenn } \frac{\partial E^{(t-1)}}{\partial w_{ij}^{(t-1)}} \cdot \frac{\partial E^{(t)}}{\partial w_{ij}^{(t)}} < 0 \ * \end{cases} \tag{2.5}$$

* In diesem Fall wird $\frac{\partial E^{(t-1)}}{\partial w_{ij}^{(t-1)}} = 0$ gesetzt.

Hierbei stellen $\Delta_{\max}$ und $\Delta_{\min}$ obere und untere Schranken für den Betrag der Gewichtsänderung dar. Die Parameter η^+ und η^- sind Faktoren, welche die Größe der Gewichtsänderung beeinflussen. Empfohlene Werte sind $\Delta_{\max} = 50$, $\Delta_{\min} = 0{,}000001$, $\eta^+ = 1{,}2$ und $\eta^- = 0{,}5$ sowie für alle Neuronen i und j zur Initialisierung $\Delta_{ij}^{(0)} = \frac{1}{2}$.

Die Gewichtsänderung wird schließlich durch

$$w_{ij}^{(t)} = w_{ij}^{(t-1)} + \Delta w_{ij}^{(t-1)}$$

durchgeführt. Bis auf die genannten Punkte entspricht der Algorithmus dem bereits vorgestellten Backpropagation.

2.3.4 Weitere Netze

Neben dem Perzeptron und den Feed-Forward-Netzen existieren weitere typische, häufig benutzte Formen der künstlichen neuronalen Netzen. Dieses Kapitel soll die wichtigsten Varianten kurz vorstellen. Diese Übersicht orientiert sich an Hildebrand (2004b).

McCulloch-Pitts Neuronen

Dieser Neuronentyp wurde im Jahr 1943 von Warren McCulloch und Walter Pitts (McCulloch und Pitts, 1943) als Modell einer natürlichen Nervenzelle entwickelt. Es existieren drei Typen:

Typ 1 unipolares Eingangsverhalten

Typ 2 bipolares Eingangsverhalten

Typ 3 in-/exhibitorisches Eingangsverhalten

Zu Typ 3 existieren inhibitorische und exhibitorische Eingänge. Bei Belegung eines inhibitorischen Eingangs gibt das Neuron einen dedizierten Wert aus, ansonsten verhält es sich wie Typ 2. Typ 1 und Typ 2 sind dem Perzeptron im Verhalten sehr ähnlich. Bei Typ 1 sind keine Gewichte an den Eingängen möglich, so dass das Neuron nur die Eingänge unter Überprüfung des Übersteigens eines Schwellwerts aufsummiert. Bei Typ 2 existieren auch Gewichte.

Adaline

Bernard Widrow und Marcian Hoff (1960) haben diese Variante des Perzeptrons entwickelt. Dieses Neuron verhält sich bei der Berechnung des Ausgangs wie das Perzeptron, in der Lernphase ist jedoch die Aktivierungsfunktion auf Identität gesetzt. Hierdurch besteht ein Vorteil hinsichtlich der Annäherung einer möglichst guten Trenngeraden: Auch wenn das Problem nicht linear separierbar ist wird die Anzahl der Fehlklassifikationen minimiert.

Rekurrente Netze
Bei rekurrenten Netzen werden im Gegensatz zu Feed-Forward-Netzen Rückkopplungen erlaubt, hierdurch wird die Schichtenstruktur nicht zwingend eingehalten.

Hopfield-Netze
Die nach ihm benannten Netze zum assoziativen Speichern stammen von John J. Hopfield (1982). Hierbei funktionieren alle Neuronen als Eingang wie auch als Ausgang. Weiterhin besteht eine Vollvernetzung.

Dem Netz werden sogenannte Fundamentalmuster präsentiert, welche es in den Verknüpfungen speichert. Bei verrauschter Informationspräsentation können diese Muster wiedererkannt werden.

Hopfield-Netze zählen zur Kategorie der rekurrenten Netze.

Adaptive-Resonance-Theory-Netze
Stephen Grossberg und Gail A. Carpenter (1992) haben diese Netzstruktur entwickelt. Sie ist ein Verfahren für das unüberwachte, also selbständige Kategorisieren von Eingaben aufgrund derer Ähnlichkeit.

Self-Organizing-Maps
Diese Netze gehen auf Teuvo Kohonen (2001) zurück. Sie realisieren wie auch die ART-Systeme unüberwachtes Klassifizieren, aber auch das Approximieren von Funktionen.

2.4 Entscheidungsbäume

Ein Entscheidungsbaum stellt eine Verknüpfung von Attributen mit Wertbelegungen nach bestimmten Zusammenhängen in Baumform dar. Hierbei wird jedes Attribut durch mindestens einen Knoten repräsentiert. Die Kanten zu den Kindern dieses Knotens stellen die Wertbelegung des ausgehenden Knotenattributs dar.

Ein solcher Baum ist von oben nach unten zu lesen. Liegt eine Belegung von den im Baum vorkommenden Attributen vor, so definiert diese einen Pfad im Baum bis hin zu einem Blatt. Dieses Blatt liefert so die Entscheidung, welche auch als Klassifikation der Belegung betrachtet werden kann. Ein Beispiel für einen solchen Baum ist in Abbildung 2.10 zu sehen.

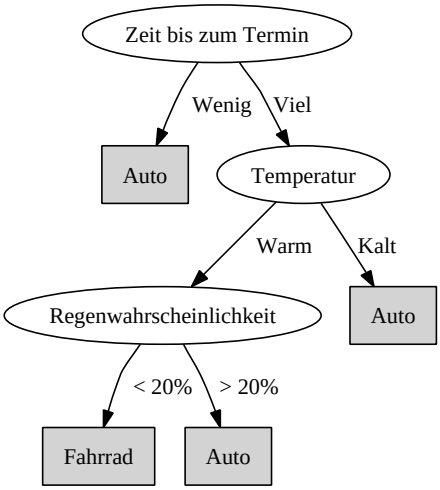


Abbildung 2.10: Baum zur Entscheidung ob mit dem Fahrrad oder dem Auto zu einem Termin gefahren wird.

Das Klassifikationsverfahren der Entscheidungsbäume wurde 1986 von John R. Quinlan eingeführt. Wie auch die neuronalen Netze basiert es auf einer Beispiel-Trainingsmenge, welche aus Eingaben und

Sollausgaben besteht. Vorteile der Entscheidungsbäume sind eine verhältnismäßig kurze Trainingszeit sowie eine verständliche Struktur im Ergebnis.

Im folgenden wird zunächst der grundlegende Algorithmus (Quinlan, 1986) vorgestellt. Im Anschluss werden für diese Arbeit notwendige Erweiterungen erläutert (Quinlan, 1992, 1993). Dieses Kapitel orientiert sich an Witten und Frank (2005a), Russell und Norvig (2003), Wrobel, Morik und Joachims (2000).

2.4.1 Entscheidungsbaumlernen

Dieses Verfahren entscheidet über den Punkt, an dem ein Attribut als Knoten in den Baum eingefügt wird, über ein Maß, welches von dem Bedarf an Information abhängt, um eine Fragestellung zu beantworten.

Definition 18 *Seien $v_1, \ldots, v_n$ mögliche Antworten auf eine Fragestellung und $P(v_1), \ldots, P(v_n)$ die Wahrscheinlichkeiten, diese Antwort zu geben ($n \in \mathbb{N}$). Dann ist der* Informationsgehalt *der aktuellen Antwort gegeben durch*

$$I(P(v_1), \ldots, P(v_n)) = -\sum_{i=1}^{n} \Big(P(v_i) \cdot \log_2 P(v_i) \Big)$$

Dieser Informationsgehalt ist in der Einheit *Bits* angegeben.

Definition 19 1 *Bit ist der Informationsgehalt, der die Antwort auf eine Ja/Nein-Frage beinhaltet.*

Entsprechend ist der Informationsgehalt des Wurfs einer Münze $I(\frac{1}{2}, \frac{1}{2}) = 1$.

Wenn eine Beispielmenge mit Eingaben und Ausgaben gegeben ist, dann unterteilt die Festlegung eines Attributs diese Beispielmenge. Sei die Beispielmenge in Tabelle 2.5 gegeben (zitiert aus Witten und Frank, 2005a). So unterteilt die Variable *Luftfeuchtigkeit* die Menge in die Teilmengen mit den Beispielen $\{1, 2, 3, 4, 8, 12, 14\}$ für hohe und $\{5, 6, 7, 9, 10, 11, 13\}$ für niedrige Luftfeuchtigkeit.

ID	Vorhersage	Temperatur	Luftfeuchtigkeit	windig	**Spielen**
1	sonnig	heiß	hoch	falsch	**Nein**
2	sonnig	heiß	hoch	wahr	**Nein**
3	bewölkt	heiß	hoch	falsch	**Ja**
4	regnerisch	mild	hoch	falsch	**Ja**
5	regnerisch	kühl	normal	falsch	**Ja**
6	regnerisch	kühl	normal	wahr	**Nein**
7	bewölkt	kühl	normal	wahr	**Ja**
8	sonnig	mild	hoch	falsch	**Nein**
9	sonnig	kühl	normal	falsch	**Ja**
10	regnerisch	mild	normal	falsch	**Ja**
11	sonnig	mild	normal	wahr	**Ja**
12	bewölkt	mild	hoch	wahr	**Ja**
13	bewölkt	heiß	normal	falsch	**Ja**
14	regnerisch	mild	hoch	wahr	**Nein**

Tabelle 2.5: Beispielmenge für die Belegung einiger das Wetter betreffenden Attribute und die davon abhängende Entscheidung, zum Spielen die Wohnung zu verlassen.

Definition 20 *Sei A ein Attribut, das eine Beispielmenge E in Teilmengen $E_1, \ldots, E_k$ zerlegt. Jede Menge E_i ($i \in \{1, \ldots, k\}$) enthält $v_i^1, \ldots, v_i^n$ viele verschiedene Beispiele bei n Teilmengen, welche durch die Unterscheidung*

anhand des Ergebniswerts der Beispiele unterteilt sind. Der Rest *an Informationsbedarf, um ein Beispiel zu klassifizieren, nachdem das Attribut A betrachtet wurde, ist dann gegeben durch*

$$Rest(A) = \sum_{i=1}^{k} \Big(\frac{|E_i|}{|E|} \cdot I(\frac{v_i^1}{|E_i|}, \ldots, \frac{v_i^n}{|E_i|}) \Big),$$

wobei zu bemerken ist, dass $E_i = \sum_{j=1}^{n} v_i^j$ *gilt.*

Hieraus lässt sich nun ableiten, welches Attribut den höchsten Informationsgewinn auf einer Beispielmenge erbringt.

Definition 21 *Der* Informationsgewinn *IG eines Attributs A ist gegeben durch*

$$IG(A) = I(\frac{v_1}{|E|}, \ldots, \frac{v_m}{|E|}) - Rest(A)$$

wobei $v_1, \ldots, v_m$ *die Anzahlen der Beispiele mit jeweils gleichem Ergebniswert angibt.*

Da aber nun der ursprüngliche Informationsbedarf $I(\frac{v_1}{|E|}, \ldots, \frac{v_m}{|E|})$ für alle Attribute identisch ist, genügt folgendes Qualitätsmaß.

Definition 22 *Die* Qualität *Q eines Attributs A ist gegeben durch*

$$Q(A) = -Rest(A)$$

Die Erstellung eines Entscheidungsbaums mit Hilfe der dargestellten Bewertungsmethoden ist in Algorithmus 2.6 dargestellt.

```
Entscheidungsbaumlernen(Beispielmenge E, Attributsmenge A){
    if alle Beispiele in E haben dieselbe Klassifikation then
      gebe Knoten mit dieser Klasse zurück;
    end if
    bestimme Attribut B ∈ A mit Q(B) = max_{A∈A}(Q(A));
    Erstelle Baum Baum mit Wurzelknoten B;
    Teile E in Teilmengen E_1, ..., E_k anhand von
                möglichen Belegungen w_1, ..., w_k von B;
    for all E_i (i ∈ {1, ..., k}) do
      Rekursiver Aufruf: Teilbaum = Entscheidungsbaumlernen(E_i, A\B);
      Erstelle Kante von B zu Teilbaum mit Beschriftung w_i;
    end for
    gebe Baum zurück;
}
```

Algorithmus 2.6: Induktive Erstellung eines Entscheidungsbaums (nach Wrobel u. a., 2000)

Beispiel

Der Informationsbedarf des in Tabelle 2.5 gegebenen Beispiels ist also

$$\begin{aligned} I(\frac{5}{14}, \frac{9}{14}) &= -\frac{5}{14}\log_2\frac{5}{14} - \frac{9}{14}\log_2\frac{9}{14} \\ &= 0{,}53 + 0{,}41 \\ &= 0{,}94 \end{aligned}$$

Die Wahl des Wurzelknotens für den Entscheidungsbaum findet wie folgt statt:

$$\begin{aligned}
\mathit{Rest}(\text{Temperatur}) &= \frac{|E_{\text{heiß}}|}{|E|} \cdot I\left(\frac{2}{|E_{\text{heiß}}|}, \frac{2}{|E_{\text{heiß}}|}\right) \\
&+ \frac{|E_{\text{mild}}|}{|E|} \cdot I\left(\frac{4}{|E_{\text{mild}}|}, \frac{2}{|E_{\text{mild}}|}\right) \\
&+ \frac{|E_{\text{kühl}}|}{|E|} \cdot I\left(\frac{3}{|E_{\text{kühl}}|}, \frac{1}{|E_{\text{kühl}}|}\right) \\
&= \frac{4}{14} \cdot I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{6}{14} \cdot I\left(\frac{4}{6}, \frac{2}{6}\right) + \frac{4}{14} \cdot I\left(\frac{3}{4}, \frac{1}{4}\right) \\
&= \frac{4}{14} \cdot (0{,}5 + 0{,}5) + \frac{6}{14} \cdot (0{,}39 + 0{,}53) + \frac{4}{14} \cdot (0{,}31 + 0{,}5) \\
&= 0{,}286 + 0{,}394 + 0{,}23 \\
&= 0{,}91
\end{aligned}$$

$$\begin{aligned}
\mathit{Rest}(\text{Vorhersage}) &= \frac{|E_{\text{sonnig}}|}{|E|} \cdot I\left(\frac{3}{|E_{\text{sonnig}}|}, \frac{2}{|E_{\text{sonnig}}|}\right) \\
&+ \frac{|E_{\text{bewölkt}}|}{|E|} \cdot I\left(\frac{4}{|E_{\text{bewölkt}}|}, \frac{0}{|E_{\text{bewölkt}}|}\right) \\
&+ \frac{|E_{\text{regnerisch}}|}{|E|} \cdot I\left(\frac{3}{|E_{\text{regnerisch}}|}, \frac{2}{|E_{\text{regnerisch}}|}\right) \\
&= \frac{5}{14} \cdot I\left(\frac{3}{5}, \frac{2}{5}\right) + \frac{4}{14} \cdot I\left(\frac{4}{4}, \frac{0}{4}\right) + \frac{5}{14} \cdot I\left(\frac{3}{5}, \frac{2}{5}\right) \\
&= 0{,}346 + 0 + 0{,}346 \\
&= 0{,}692
\end{aligned}$$

Analog wird bestimmt:

$$\begin{aligned}
\mathit{Rest}(\text{Luftfeuchtigkeit}) &= 0{,}788 \\
\mathit{Rest}(\text{windig}) &= 0{,}892
\end{aligned}$$

Daraus folgen die Qualitätswerte Q und der jeweilige Informationsgewinn IG

$$\begin{aligned}
Q(\text{Temperatur}) &= -0{,}91 & \qquad IG(\text{Temperatur}) &= 0{,}03 \\
Q(\text{Vorhersage}) &= -0{,}692 & IG(\text{Vorhersage}) &= 0{,}248 \\
Q(\text{Luftfeuchtigkeit}) &= -0{,}788 & IG(\text{Luftfeuchtigkeit}) &= 0{,}152 \\
Q(\text{windig}) &= -0{,}892 & IG(\text{windig}) &= 0{,}048
\end{aligned}$$

Aufgrund dieser Berechnung wird nun *Vorhersage* als erstes Attribut den Wurzelknoten im Entscheidungsbaum darstellen. Darunter entstehen drei Teilbäume an denen das Verfahren rekursiv erneut ansetzt. Daraus ergibt sich schließlich der Entscheidungsbaum in Abbildung 2.11. Hier ist ersichtlich, dass das Attribut *Temperatur* redundant ist.

2.4.2 Erweiterungen

Die im folgenden dargestellten Erweiterungen ermöglichen den Einsatz des vorgestellten Verfahrens in dieser Arbeit. Sie stellen keine grundlegenden Veränderungen dar sondern ergänzen das bereits

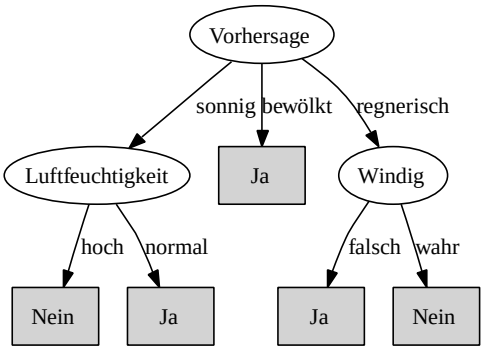


Abbildung 2.11: Entscheidungsbaum entsprechend der Daten in Tabelle 2.5

vorgestellte. Diese Beschreibung folgt Witten und Frank (2005a).

Numerische Attribute

Bisher sind nominelle Attribute angenommen worden. Tatsächlich finden sich bei real auftretenden Fragestellungen eher numerische Angaben. Diese lassen sich jedoch auf einfache Weise im vorgestellten Algorithmus einbinden.

Sei angelehnt an das Beispiel aus Tabelle 2.5 die Verteilung zwischen Temperatur und Entscheidung in Tabelle 2.6 gegeben, wobei mehrere Beispiele mit gleicher Temperaturangabe in einer Spalte zusammengefasst sind.

°C	17	18	19	20	21	22	23	24	26	27	28	29
Ergebnis	Ja	Nein	Ja	Ja	Ja	Nein	Nein Ja	Ja Ja	Nein	Ja	Ja	Nein

Tabelle 2.6: Numerische Temperaturverteilung entsprechend dem Beispiel in Tabelle 2.5

Entsprechend dieser Aufteilung existieren maximal 11 Punkte, an denen eine Aufteilung in zwei Klassen möglich ist, 8 Trennpunkte bestehen, wenn aufeinanderfolgende Elemente einer Klasse nicht getrennt werden dürfen. Der Informationgehalt für jede dieser Trennungen wird wie üblich entsprechend der Vorschrift in Definition 18 berechnet.

Zum Beispiel kann der Informationgehalt für den Test von Temperatur $< 22{,}5$ berechnet werden durch

$$I(\frac{6}{14}, \frac{8}{14}) = 0{,}52 + 0{,}46 = 0{,}98\,.$$

Beschneidung des Baums

Die Beschneidung des Baums kann helfen, das Übertrainieren zu verhindern, so dass die Verallgemeinerung bei der Klassifikation möglich ist. Die hier vorliegende Beschreibung folgt den Ausführungen von Witten und Frank (2005a), auf die zur weiteren Lektüre mit detaillierteren Schilderungen verwiesen sei. Es existieren zwei verschiedene Ansätze des Beschneidens. Das *Postpruning* (auch *Backward-Pruning*) bezeichnet das Verändern des Baums, nachdem dieser vollständig aufgebaut ist. Beim *Prepruning* (auch *Forward-Pruning*) wird während des Erstellungsprozesses versucht zu entscheiden, wann die Entwicklung eines Teilbaums abgebrochen werden kann. Möglicherweise kann jedoch durch Kombination von zwei Knoten die Generalisierungsfähigkeit verbessert werden. Solche Situationen sind mit nachträglichem Beschneiden eher zu erkennen, was dazu führt, dass im allgemeinen Postpruning eingesetzt wird.

Hierbei bestehen zwei Techniken: Das *Subtree-Raising* sowie das *Subtree-Replacement*. Subtree-Replacement bedeutet das Ersetzen eines Teilbaums durch ein Blatt. Das Subtree-Raising ist etwas komplexer. Hierbei kann ein Knoten durch seine Kinder und die daran befindlichen Teilbäume ersetzt werden.

Andere Kinder werden hierbei, wenn solche vorhanden sind, entfernt. Daher müssen die von diesen klassifizierten Beispiele in den ersetzenden Teilbaum eingeordnet werden.

Das eigentliche Pruning findet nun statt, indem jeder Knoten von den Blättern hin zur Wurzel betrachtet wird und darüber entschieden wird, ob dieser durch ein Blatt oder eines seiner Kinder ersetzt oder ob keines der beiden Verfahren angewendet wird. Um diese Entscheidung zu treffen werden die Blätter und internen Knoten potentiell entstehender Entscheidungsbäume mit einem Fehlermaß bewertet. Bei dem *Reduced-Error-Pruning* wird eine Testmenge benötigt. Wenn eine solche nicht gegeben werden kann besteht die Möglichkeit, einen Teil der Trainingsmenge nicht zum Trainieren zu benutzen, sondern als Testmenge. Dies ist allerdings problematisch, da hierdurch weniger Beispiele zur Erzeugung des Baums genutzt werden. Bei dem Verfahren C4.5 (Quinlan, 1993) wird ein anderer Ansatz verfolgt. Die Idee ist, die Menge der Beispiele, welche einen Knoten erreichen, zu betrachten. Angenommen wird, dass dieser Knoten durch die Klassifizierung repräsentiert wird, deren Beispiele an diesem Knoten am häufigsten vorkommen. Sei nun N die Anzahl der Beispiele an einem Knoten und E die der entsprechend falsch klassifizierten Beispiele. Dies bestimmt die beobachtete Fehlerrate $f = \frac{E}{N}$. Die Konfidenzgrenze z ergibt sich durch

$$P\left(\frac{f-q}{\sqrt{\frac{q\cdot(1-q)}{N}}} > z\right) = c\,,$$

wobei q die tatsächliche Fehlerrate darstellt und die *Konfidenz* c üblicherweise mit 0,25 belegt ist. Diese Grenze wird zu einer pessimistischen Abschätzung der oberen Konfidenzgrenze e genutzt:

$$e = \frac{f + \frac{z^2}{2N} + z\cdot\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}}$$

Um nun zu entscheiden, ob ein Teilbaum durch ein Blatt ersetzt wird, wird diese obere Konfidenzgrenze des Wurzelknotens des Teilbaums, sowie der entsprechend der Anzahl der Beispiele an den Kindern gewichtete Durchschnittswert der Kinder bestimmt. Ist der kombinierte Wert der Kinder größer als der des Teilbaumwurzelknotens wird dieser durch ein Blatt ersetzt. Auf vergleichbare Weise wird über das Beschneiden durch Subtree-Raising entschieden.

Alternative Bewertung der Qualität der Attribute

Die in Definition 22 gegebene Festlegung der Qualität eines Attributs kommt in dem Entscheidungsbaumlernverfahren ID3 (Quinlan, 1986) zum Einsatz. In der Erweiterung C4.5 (Quinlan, 1993) wird als alternative Qualitätszuweisung der Gewinnquotient genutzt:

Definition 23 *Der* Gewinnquotient *GQ ist gegeben durch*

$$GQ(A) = \frac{IG(A)}{I(A)}$$

Für weitere Erläuterungen sei auf Beierle und Kern-Isberner (2003) verwiesen.

Ausführungen zu anderen Erweiterungen des Entscheidungsbaumlernens sind in Russell und Norvig (2003) zu finden.

Kapitel 3

Musikalische Grundlagen

Dieses Kapitel stellt das grundlegend notwendige musikalische Wissen zum Verständnis von automatischer Komposition dar. Hierbei wird der Anspruch verfolgt, dass ohne jegliches Vorwissen die Anwendung der dargestellten Verfahren verstanden werden kann. Jedoch soll hier nicht musiktheoretisches Wissen umfassend und abgeschlossen vermittelt werden. Hierzu ist zum Beispiel Motte (1993, 2002, 2004) zu empfehlen.

3.1 Darstellung von Tonhöhen und Tonlängen in Notenschrift

3.1.1 Tonhöhen

Tonhöhen werden in ein fünfzeiliges System aus Linien mit Zwischenräumen eingetragen. Jede Linie und jeder Zwischenraum stellt eine unterschiedliche Tonhöhe dar. Tiefe Töne (also solche mit niedrigerer Frequenz) werden weiter unten in dem System eingetragen, höhere Töne weiter oben. Reichen die fünf Linien nicht für die darzustellende Tonhöhe, können Hilfslinien eingetragen werden. Die einzelnen Töne werden durch Noten dargestellt. In Abbildung 3.1 sind einige Noten mit Namensangabe in ein solches System eingetragen. Die Horizontale repräsentiert den zeitlichen Verlauf.

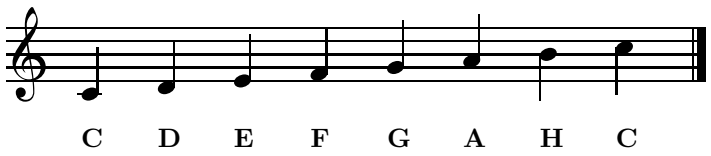


Abbildung 3.1: Notennamen

Welche Linie welche Tonhöhe darstellt wird durch den *Notenschlüssel* definiert. Der hier abgebildete Schlüssel heißt *Violin-* oder *G-Schlüssel* und definiert die Töne auf der zweiten Linie von unten als *G*. Ein anderer Schlüssel ist der *Bass-* oder *F-Schlüssel* (Abbildung 3.2), der die zweite Linie von oben für den Ton *F* definiert.

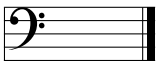


Abbildung 3.2: Bassschlüssel

Die in Abbildung 3.1 gezeigten Noten stellen den Umfang einer *Oktave* dar, den Umfang von 8 Tönen auf der Klaviatur, wie in Abbildung 3.3 zu sehen ist. Hier erkennt man auch, dass sich Oktaven stetig wiederholen.

Die Töne der schwarzen Tasten sind im Notensystem durch Erhöhen (mittels des Zeichens ♯) oder Erniedrigen (durch das Zeichen ♭) zu erreichen. Alle zwölf Töne einer Oktave sind in Abbildung 3.4

zu sehen. Sie werden je nachdem, in welchem Kontext sie stehen (abhängig von der Tonart) durch Anhängen der Silbe -es bei Erniedrigen oder der Silbe -is bei Erhöhen benannt. Ausnahmen dieser Regel sind das Zusammenziehen der Buchstaben beziehungsweise Verkürzen bei "A-es" zu "As", bei "E-es" zu "Es" und bei "H-es" zu "B".

Die sogenannten *Vorzeichen* ♯ und ♭ können am Anfang der Notenzeile stehen und gelten dann für diese gesamte Zeile oder aber direkt vor der zu erhöhenden oder erniedrigenden Note und gelten dann für den gesamten Takt, welcher durch einen senkrechten Strich beendet wird. Das sogenannte *Auflösungszeichen* ♮ hebt die Bedeutung des Vorzeichens bis Taktende (wenn Vorzeichen am Anfang der Zeile angegeben wurden) oder bis zum Auftreten des nächsten Vorzeichens auf. Zur Bedeutung von Takten kommen wir in Kapitel 3.1.2.

Wir werden häufig ein Klaviersystem nutzen, welches ein System mit einem Violin- und eines mit einem Bassschlüssel kombiniert (wobei auch die Kombination zweier anderer Systeme vorkommt, allerdings deutlich seltener). Ein solches ist in Abbildung 3.5 zu sehen.

3.1.2 Tonlängen

Um unterschiedliche Tonlängen und damit verschiedene Rhythmen möglich zu machen, wird in der Notendarstellung das Aussehen der einzelnen Noten verändert. Die Noten in den bisherigen Darstellungen waren sogenannte Viertelnoten. Dieser Name leitet sich von der Annahme eines Taktes mit vier Schlägen ab, bei dem jede solche Note einem Schlag Länge entspricht.

Andere Tonlängen sind in Tabelle 3.1 zu sehen.

Symbol	𝅝	𝅗𝅥	♩	♪	𝅘𝅥𝅯	𝅘𝅥𝅰	...
Länge	$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$	$\frac{1}{16}$	$\frac{1}{32}$	...

Tabelle 3.1: Notenlängen

Notenlängen, welche kürzer oder gleich der Länge $\frac{1}{8}$ sind, lassen sich in Gruppen darstellen, indem die Fähnchen mehrerer Noten verbunden werden. Dies ist unter anderem in Abbildung 3.7 zu sehen.

Ein Punkt hinter einer Note bedeutet die Verlängerung um die Hälfte ihres eigenen Werts. Einige Beispiele sind in Abbildung 3.2 zu sehen.

Symbol	𝅗𝅥.	♩.	♪.	𝅘𝅥𝅯.	...
Länge	$\frac{3}{4}$	$\frac{3}{8}$	$\frac{3}{16}$	$\frac{3}{32}$	...

Tabelle 3.2: Punktierte Notenlängen

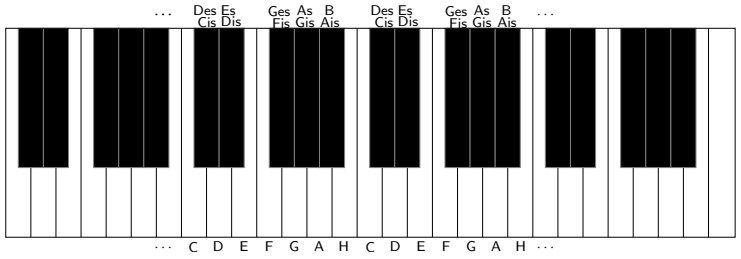


Abbildung 3.3: Klaviatur

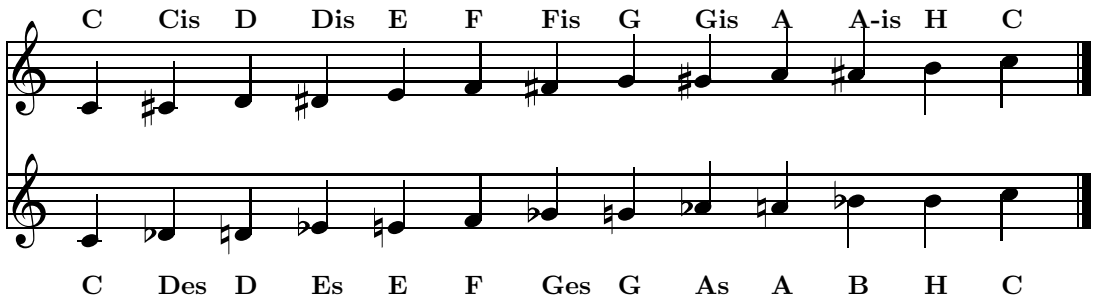


Abbildung 3.4: Alle zwölf Töne einer Oktave (die übereinander stehenden Noten stellen dieselbe Tonhöhe dar)

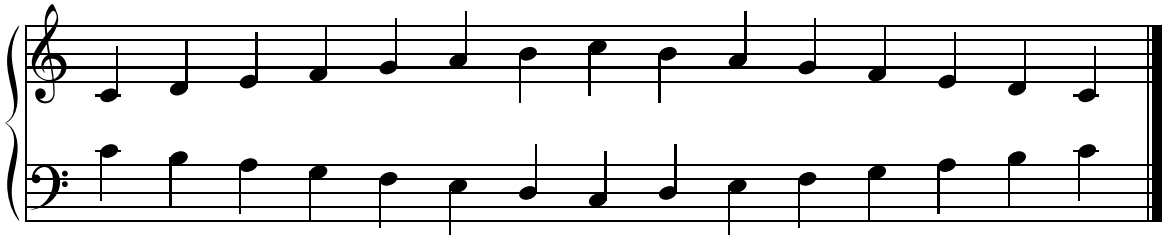


Abbildung 3.5: Ein Klaviersystem, welches zwei Systeme kombiniert.

Neben Noten gibt es Pausen, welche der selben Systematik folgen, wie aus Abbildung 3.6 zu entnehmen ist. Auch hier ist Punktierung möglich. Eine Ausnahme gilt jedoch: Die Länge der hier als $\frac{1}{1}$ dargestellten Pause ist nicht durch 4 Schläge definiert sondern durch das Ausfüllen eines gesamten Taktes. Bei einem $\frac{4}{4}$ Takt ist dies identisch, bei einem $\frac{3}{4}$ Takt jedoch nicht, hat dann diese Pause doch nur die Länge von 3 Schlägen.

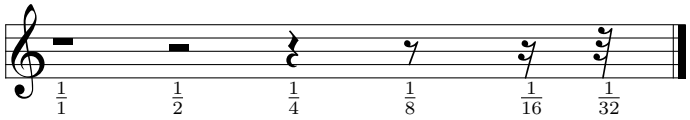


Abbildung 3.6: Pausen und ihre Längen

Wenn auch nicht direkt auf die Notenlängen bezogen, so ist doch gerade für diese der Begriff des *Taktes* wichtig.

Takte sind eine zeitliche Einteilung der Melodie. Sie geben kleine Abschnitte an, welche für Melodie und Rhythmus Sinneinheiten bilden. So definieren sie auch, welche Zeiten innerhalb eines Taktes stärker betont sind als andere.

Eine übliche Taktart ist der $\frac{4}{4}$-Takt, bei dem die Betonungen auf der ersten und der zweiten Zählzeit liegt. Mit Zählzeit ist gemeint, dass man die Viertel durchzählt. In diesem Takt haben, entsprechend der Namensgebung, 4 Noten der Länge $\frac{1}{4}$ Platz.

Der $\frac{3}{4}$-Takt wird auf der ersten Zählzeit betont. In ihm haben entsprechend 3 $\frac{1}{4}$-Noten Platz.

Das Ende von Takten wird im Notensystem durch einen senkrechten Strich angegeben. Die Taktart wird am Anfang des Systems angegeben. Beispiele sind in Abbildung 3.7 zu sehen.

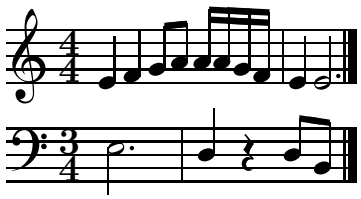


Abbildung 3.7: Ein System mit $\frac{4}{4}$- und eines mit $\frac{3}{4}$-Takt

3.2 Stammtonreihe und Tonleitern

Die Stammtonreihe besteht aus den Tönen *c*,*d*,*e*,*f*,*g*,*a*,*h*,*c* entsprechend der Abbildung 3.1. Diese liegen auf den weißen Tasten der Klaviatur (siehe Abbildung 3.3). Wir bezeichnen die so nah wie möglich beieinanderliegenden Tonschritte auf der Stammtonreihe als Halbtonschritte (H). Zwei Halbtonschritte stellen einen Ganztonschritt (G) dar. Die Stammtonreihe besteht also aus der Folge

G — G — H — G — G — G — H

Dies entspricht einer sogenannten Dur-Tonleiter. Diese Tonleiter kann unter Berücksichtigung der gegebenen Folge von Ganz- und Halbtonschritten von jedem Ton begonnen werden, der typische Klang einer Dur-Tonart bleibt erhalten. Moll-Tonleitern haben die Ganzton-/Halbtonschrittfolge

G — H — G — G — H — G — G

Weitere in dieser Arbeit genutzte Tonleitern werden wir in Teil II kennenlernen.

Der sogenannte *Quintenzirkel* gibt den Zusammenhang zwischen den Tonarten. So sind auf diesem Zirkel, wie in Abbildung 3.8 zu sehen ist, Tonarten unterschiedlich weit voneinander entfernt. Je größer die Entfernung, desto weniger Töne haben sie gemeinsam. Im Inneren des Kreises sind die Tonarten angezeichnet. Die Großbuchstaben bedeuten die Dur-, die Kleinbuchstaben die Molltonart. Außerhalb des Kreises ist abgebildet, welche Vorzeichen die jeweilige Tonart hat.

Dies bedeutet, dass man bei Spielen der Durtonreihe beginnend mit dem in Großbuchstaben geschriebenen Ton die Töne benutzt, welche durch die angegeben Vorzeichen vorgeschrieben sind. Spielt man dieselben Töne, jedoch beginnend mit dem Ton, welcher in Kleinbuchstaben angegeben ist, spielt man die Tonleiter der sogenannten parallelen Molltonart.

Die Herkunft des Begriffs Quintenzirkel wird in Kapitel 3.3 erläutert.

3.3 Intervalle

Intervalle bezeichnen die Abstände zwischen zwei Tönen. Das Intervall “Quinte”, haben wir bereits kennengelernt, wobei dieses bei dem Begriff “Quintenzirkel” den Abstand der Töne auf dem Kreis bezeichnet. So ist der Abstand zwischen *C* und *G* fünf Schritte auf der Stammtonreihe groß, ebenso wie der Abstand zwischen *G* und *D*, *D* und *A* und so weiter.

Die gebräuchlichsten Intervalle sind in Tabelle 3.3 mit je einem Beispiel vom Ton *C* aus dargestellt.

3.4 Dreiklänge und Akkorde

Mit Hilfe der durch Intervalle definierten Tonabstände lassen sich verschiedene Kombinationen von gleichzeitig erklingenden Tönen definieren. Der grundlegende Dreiklang ist der *Dur-Dreiklang* welcher durch einen Grundton sowie zwei (beziehungsweise eigentlich drei, da der Grundton als Prime bekannt

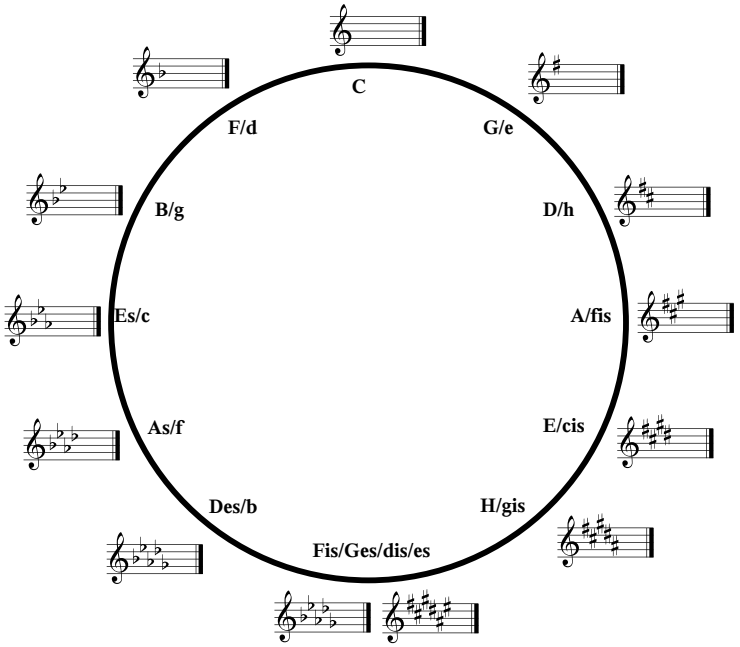


Abbildung 3.8: Der Quintenzirkel

Intervall	Schritte	Abstand	Beispiel
Prime	0	1	
Übermäßige Prime, Kl. Sekunde	1		
Große Sekunde	2	2	
Kleine Terz	3		
Große Terz	4	3	
Quarte	5	4	
Gr. Quarte, Kl. Quinte, Tritonus	6		
Quinte	7	5	
Kleine Sexte	8		
Große Sexte	9	6	
Kleine Septime	10		
Große Septime	11	7	
Oktave	12	8	

Tabelle 3.3: Die gebräuchlichsten Intervalle. *Schritte* bezeichnet Halbtonschritte, *Abstand* bezeichnet den Abstand der Töne auf der Stammtonreihe, wobei der Ton, bei die Zählung begonnen wird, mitgezählt wird.

ist,) Intervalle gegeben ist. So bildet sich der *C*-Dur Dreiklang durch den Grundton als Prime sowie das Intervall große Terz und Quinte. Es ergibt sich die Tonkombination "*C — E — G*". Der sogenannte *Moll-Dreiklang* ergibt sich aus der Prime, der kleinen Terz sowie der Quinte. Bei dem Grundton *C* ist dies die Tonfolge "*C — Es — G*".

Diese Dreiklänge sind sogenannte Akkorde. Akkorde können auch durch mehr als drei Töne zusammengesetzt werden. Ein gebräuchliches Beispiel hierzu ist der sogenannte *Septimeakkord* bei dem zu dem Durdreiklang oder dem Molldreiklang die kleine Septime des Grundtons hinzugefügt wird. Mit dem Grundton *C* als Beispiel ergibt sich für den Fall des Mollseptimeakkords die Tonfolge "*C — Es — G — B*".

Die in diesem genannten Abschnitt genannten Akkorde sind in Abbildung 3.9 in Notenschrift mit zugehöriger Kurzschreibweise notiert.

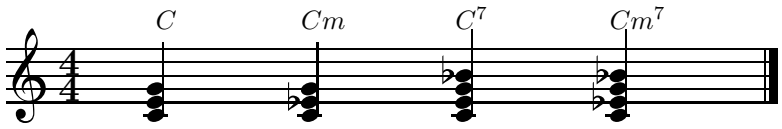


Abbildung 3.9: Die Akkorde C-Dur (*C*), C-Moll (*Cm*), C-Dur-Septime (C^7), C-Moll-Septime (Cm^7)

Weitere Akkorde werden wir in Teil II kennenlernen.

Zusammenhänge zwischen dem Wohlempfinden von Harmonien und Melodien werden in benötigtem Umfang im Teil II erläutert. Für tiefergehende Beschreibungen sei Jourdain (2001) und Roederer (2000) empfohlen.

Kapitel 4

MIDI-Technologie

4.1 Einführung

MIDI ist die Abkürzung für den Begriff „Musical Instrument Digital Interface“. Dieser Standard wurde 1983 von den Herstellern Sequential Circuits und Roland vorgestellt. Von diesem aus wurden im Laufe der Zeit einige proprietäre Weiterentwicklungen vorgenommen. Zu nennen sind hier als Beispiele *General MIDI* (GM) und *General Standard* (GS) von Roland und *Extended General Standard* (XG) von Yamaha. Im allgemeinen wird hier die ursprüngliche Form beschrieben, wenn auf gesonderte Standards eingegangen wird, wird dies ausdrücklich erwähnt.

Der MIDI-Standard (und der kurzlebige, direkte Vorgänger *Universal Synthesizer Interface*, welcher ebenfalls von Sequential Circuits entworfen war,) wurde eingeführt, um verschiedene MIDI-fähige Tonerzeuger miteinander über ein 5-poliges DIN-Kabel zu verbinden. Dies bietet die Möglichkeit, durch Betätigen einer Klaviatur mehrere Tonerzeuger gleichzeitig zur Klangerzeugung anzuregen. Dies wurde in den Anfangstagen der Synthesizer, in denen sich diese durch einen recht dünnen Klang auszeichneten, als wichtig empfunden.

Die große Verbreitung folgte schließlich mit dem Einsatz der MIDI-Technologie in Hardware-Sequenzern (eine Form digitaler Aufnahmegeräte für die durch die MIDI-Schnittstelle übertragenen Signale) und die Verbreitung in Home-Computern. Hierbei ist insbesondere der Atari ST zu nennen, der bereits bei seinem Erscheinen 1985 eine MIDI-Schnittstelle eingebaut hatte.

Weitergehende Schilderungen zur Geschichte der MIDI-Schnittstelle finden sich zum Beispiel in Zielinsky (2000)

4.2 Technologie

Die MIDI-Schnittstelle ist ursprünglich für den Transport von Nachrichten in serieller Form zwischen zwei mit Kabeln verbundenen Geräten spezifiziert worden. Hierbei wird in der einfachsten Form eine MIDI-Out-Buchse eines Synthesizers mit der MIDI-In-Buchse eines anderen Synthesizers verbunden und umgekehrt. Dann ist es möglich, beide Synthesizer durch Betätigen einer Klaviatur zu spielen. Ähnlich einfach ist das Verbinden eines Synthesizers mit einem Sequenzer. In dem Fall können die übermittelten Nachrichten aufgezeichnet werden und bei Abspielen dieser Nachrichten verhält sich das ursprünglich sendende und nun empfangende Gerät wie bei dem Sendevorgang.

Aus diesen Abläufen heraus wurde das *Standard-MIDI-File*-Format (smf) definiert. Dieses folgt den ursprünglichen Anforderungen, so dass auch in diesem Format die Herkunft der Kabelgebundenheit erkennbar ist.

So spielen die wichtigste Rolle bezüglich der Spezifikation die sogenannten *MIDI Messages*. Dies sind Nachrichten, welche sich stark an der Kabelherkunft orientieren, sind diese doch direkt aktionsgebunden. So gibt es Nachrichten, welche besagen, welche Taste man drückt und welchen Regler man auf welchen Wert stellt.

Jede dieser MIDI Messages setzt sich aus genau einem Statusbyte und einem oder mehreren Datenbytes zusammen. Jedes Statusbyte hat an seiner höchstwertigen Stelle den Wert 1. Jedes Datenbyte hat an

Befehl	Statusbyte	Datenbyte1	Datenbyte2
Note on	`1001 nnnn` Kanalnr. 0–15	`0nnn nnnn` Notennr. 0–127	`0nnn nnnn` Velocity 1–127
Note off	`1000 nnnn` Kanalnr. 0–15	`0nnn nnnn` Notennr. 0–127	`0nnn nnnn` Releasevelocity
Program Change	`1100 nnnn` Kanalnr. 0–15	`0nnn nnnn` Programmnr. 0–127	
Control Change	`1010 nnnn` Kanalnr. 0–15	`0nnn nnnn` Kontrollnr. 0–127	`0nnn nnnn` Wert 0–127

Tabelle 4.1: Übersicht über einige wichtige MIDI Messages

seiner höchstwertigen Stelle eine 0. Tabelle 4.1 gibt eine Übersicht über wichtige MIDI Messages. Hierbei steht der Buchstabe `n` als Platzhalter. Die Erklärung, welche Werte diese Bits annehmen können und ihre Bedeutung steht jeweils in der selben Spalte eine Zeile tiefer. Die *MIDI-Kanäle* stellen eine Kapselung der jeweiligen Befehle dar. So gelten Befehle (bis auf wenige Ausnahmen) grundsätzlich nur für den Kanal, auf dem sie gesendet wurden. Ein Ton, welcher mit "Note On" auf Kanal 1 gestartet wurde kann also nicht mit einem "Note Off" auf Kanal 2 beendet werden.

Der Befehl Control Change lässt die Veränderung von teilweise fest vorgegebenen, teilweise vom Hersteller eines Geräts selbst belegbaren Controllern zu. Fest vorgegeben sind hier zum Beispiel die Controllernummern für die Hauptlautstärke, ein Haltepedal oder auch einen sogenannten Breathcontroller, mit dem mit Atemluft das Verhalten des Tonerzeugers manipuliert werden kann. Als frei belegbar könnte man sich zum Beispiel die Steuerung der Parameter eines programmierbaren Synthesizers vorstellen.

Die Bedeutung der Programmnummern für den Wechsel der Instrumente entsprechend des Befehles Program Change finden sich in Tabelle 4.2. Ein Auszug der Zuordnung der die Notenhöhe bestimmenden Werte bei den Befehlen *Note On* und *Note Off* zu den entsprechenden Höhen ist in Tabelle 4.3 gegeben.

Für eine vollständige Auflistung aller möglichen Befehle verweise ich auf die MIDI Manufacturers Association (2001) und für einen umfassenden Überblick auf die MIDI Manufacturers Association (1995).

Besonderheiten des Standard-MIDI-File Formats

Zur Speicherung von MIDI-Messages treten einige Besonderheiten im Vergleich zur Übertragung per Kabel auf. So werden die Messages mit einem Zeitstempel versehen, der die beim Kabel naturgemäß vorgegebene zeitliche Ordnung auch bei der Speicherung möglich macht. Hierzu werden sie in sogenannte MIDI-Events gekapselt, welche die zusätzliche Information bereitstellen.

Die Bedeutung dieser Zeitstempel wird weiterhin durch die Auflösung sowie die Art der Zeitinformation bestimmt. Hierfür stehen zwei grundlegende Einteilungsarten zur Verfügung. Zum einen existiert *Puls per Quarternote* (PPQ), zum anderen die Norm der *Society of Motion Picture and Television Engineers* (SMPTE).

Bei PPQ wird die Auflösung in Puls je Viertelnote angegeben. Wenn also die Auflösung auf 1 gesetzt ist, ist eine Viertelnote die kleinstmögliche Zeiteinheit. Übliche Auflösung bei PPQ sind 384 oder höher.

Auf die SMPTE Zeitangabe wird hier nicht näher eingegangen, da dies ein Verfahren ist, welches aus der analogen Aufnahmetechnik stammt und für diese Arbeit keine Relevanz hat.

Weiterhin stellt das Standard-MIDI-File Format die Möglichkeit zur Verfügung, die Daten in unterschiedliche *Tracks* einzuteilen. Unter einem Track kann man sich eine logische Unterteilung der MIDI-Messages mit weiteren Zusatzinformationen vorstellen, wobei ein Track häufig mit einem MIDI-Kanal und damit auch üblicherweise mit einem Instrument assoziiert wird. Die Möglichkeit, Tracks zu verwenden, wird nur im sogenannten Standard-MIDI-File Format 1 unterstützt. Im Standard-MIDI-File

Nr.	Instrument	Nr	Instrument	Nr	Instrument
0	Piano 1	43	Contrabass	86	5th Saw Wave
1	Piano 2	44	Tremolo Strings	87	Bass & Lead
2	Piano 3	45	Pizzicato Strings	88	Fantasia
3	Honky-Tonk-Piano	46	Harp	89	Warm Pad
4	E-Piano 1	47	Timpani	90	Polysynth
5	E-Piano 2	48	Strings	91	Space Voice
6	Harpsichord	49	Slow Strings	92	Bowed Glass
7	Clavinet	50	Syn-Strings 1	93	Metal Pad
8	Celesta	51	Syn-Strings 2	94	Halo Pad
9	Glockenspiel	52	Choir Aahs	95	Sweep Pad
10	Music Box	53	Voice Oohs	96	Ice Rain
11	Vibraphone	54	Syn Vox	97	Soundtrack
12	Marimba	55	Orchestra Hit	98	Crystal
13	Xylophone	56	Trumpet	99	Atmosphere
14	Tubular Bell	57	Trombone	100	Brightness
15	Dulcimer	58	Tuba	101	Goblin
16	Organ 1	59	Muted Trumpet	102	Echo Drops
17	Organ 2	60	French Horn	103	Star Theme
18	Organ 3	61	Brass 1	104	Sitar
19	Church Organ	62	Synth Brass 1	105	Banjo
20	Reed Organ	63	Synth Brass 2	106	Shamisen
21	French Accordeon	64	Soprano Sax	107	Koto
22	Harmonica	65	Alto Sax	108	Kalimba
23	Tango Arccordeon	66	Tenor Sax	109	Bag Pipe
24	Nylon-String-Guitar	67	Baritone Sax	110	Fiddle
25	Steel-String-Guitar	68	Oboe	111	Shannai
26	Jazz Guitar	69	English Horn	112	Tinkle Bell
27	Clean Guitar	70	Bassoon	113	Agogo
28	Muted Guitar	71	Clarinet	114	Steel Drums
29	Overdrive Guitar	72	Piccolo	115	Woodblock
30	Distortion Guitar	73	Flute	116	Taiko
31	Guitar Harmonics	74	Recorder	117	Melodic Tom
32	Acoustic Bass	75	Pan Flute	118	Synth Drum
33	Fingered Bass	76	Bottle Blow	119	Reverse Cymbal
34	Picked Bass	77	Shakuhachi	120	Guitar Fret Noise
35	Fretless Bass	78	Whistle	121	Breath Noise
36	Slap Bass 1	79	Ocarina	122	Seashore
37	Slap Bass 2	80	Square Wave	123	Bird
38	Synth Bass 1	81	Saw Wave	124	Telephone 1
39	Synth Bass 2	82	Synth Calliope	125	Helicopter
40	Violin	83	Chiffer Lead	126	Applause
41	Viola	84	Charang	127	Gun Shot
42	Cello	85	Solo Vox		

Tabelle 4.2: Instrumentennummern entsprechend dem General MIDI Standard

Nr.	Höhe	Nr.	Höhe	Nr.	Höhe	Nr.	Höhe
48	*C*	60	*c*	72	*c′*	84	*c″*
49	*Cis/Des*	61	*cis/des*	73	*cis′/des′*	85	*cis″/des″*
50	*D*	62	*d*	74	*d′*	86	*d″*
51	*Dis/Es*	63	*dis/es*	75	*dis/es′*	87	*dis″/es″*
52	*E*	64	*e*	76	*e′*	88	*e″*
53	*F*	65	*f*	77	*f′*	89	*f″*
54	*Fis/Ges*	66	*fis/ges*	78	*fis′/ges′*	90	*fis″/ges″*
55	*G*	67	*g*	79	*g′*	91	*g″*
56	*Gis/As*	68	*gis/as*	80	*gis′/as′*	92	*gis″/as″*
57	*A*	69	*a*	81	*a′*	93	*a″*
58	*A-is/B*	70	*a-is/b*	82	*a-is′/b′*	94	*a-is″/b″*
59	*H*	71	*h*	83	*h′*	94	*h″*

Tabelle 4.3: Ausschnitt Tonhöhen entsprechend der MIDI-Spezifikation
(Mit kleinen Buchstaben wird die sogenannte mittlere Oktave bezeichnet (bei der die Frequenz des Tons *a* 440 Hz beträgt), die Anzahl der Striche gibt den Abstand nach oben in Oktaven zu der mittleren Oktave an. Großbuchstaben bezeichnen die Oktave unter der mittleren Oktave.)

Format 0 kann nur genau ein Track existieren.

Das Standard-MIDI-File Format wurde im Laufe der Zeit um einige Ergänzungen erweitert, wie zum Beispiel die Möglichkeit, Liedtexte, Tracknamen und anderes zu speichern. Solche Erweiterungen werden hier nicht besprochen. Für die vollständige Spezifikation sei wiederum auf die MIDI Manufacturers Association (2001) verwiesen.

Zur Erläuterung des dargestellten Formats findet sich in Tabelle 4.4 die Bytefolge der MIDI-Datei der Melodie aus Abbildung 4.1. Die Datei wurde durch das Sequenzersystem Rosegarden (McIntyre, 2005; Cannam u. a., 2005) erzeugt. Die Bytefolge ist durch eine Implementierung in Java[1] erzeugt worden. Die dargestellte Datei hat die PPQ-Auflösung 480 und hat zwei Tracks, obwohl nur einer Noten enthält. Der erste Track beinhaltet Initialisierungsdaten wie zum Beispiel Abspielgeschwindigkeit. Dargestellt wird nur der zweite Track mit den jeweiligen Zeitpunkten (*Ticks*) der Midi-Nachrichten.

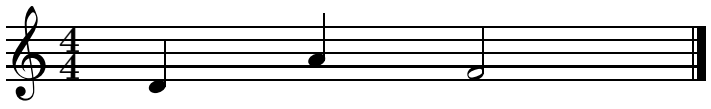


Abbildung 4.1: Beispiel zur Erläuterung des Standard-MIDI-File-Formats

4.3 Java

In dem vorstehenden Abschnitt wurde die MIDI-Technologie dargestellt. Hier soll nun die Realisierung dieser in der Standard-API von Sun (Sun Developer Network, 2005) eingeführt werden.

Die Datenstruktur, welche in Java zur Verfügung steht, lässt direkt die zugrundeliegende Technologie wiedererkennen. Sie ist graphisch in Abbildung 4.2 dargestellt wobei die Pfeile als „benutzt“ zu lesen sind. Die Klasse MidiMessage stellt mit seinen geerbten Formen MetaMessage, ShortMessage und SysexMessage die elementare Struktur dar, in der direkt die Bytes der Nachricht gespeichert werden. ShortMessages stellen hierbei den meistverwendeten Nachrichtentyp dar. SysexMessages enthalten meist herstellerspezifische Informationen und können üblicherweise recht lang sein. MetaMessages haben keinen Einfluss auf den Tonerzeuger sondern enthalten zum Beispiel Liedtexte.

[1] Hilfsprogramm *MidInfo*, siehe CD bzw. `http://www.roman-klinger.de`

	Byte		
Tick	Binär	Dezimal	Bedeutung
0	11111111	255	Reset
	00000011	3	Beginn des Tracks
	00000000	0	
0	11000000	192	Programmwechsel
	00000000	0	Instrument 0
0	10110000	176	Control Change Ch. 0
	00000111	7	Lautstärke des Kanals
	01100100	100	Lautstärke 100
0	10110000	176	Control Change Ch. 0
	00001010	10	Panorama
	01000000	64	mittig
0	10010000	144	Note On Ch. 0
	00111110	62	Tonhöhe 62 (D)
	01100100	100	Anschlagsdynamik 100
480	10000000	128	Note Off Ch. 0
	00111110	62	Tonhöhe 62 (D)
	01111111	127	Loslassdynamik 127
480	10010000	144	Note On Ch. 0
	01000101	69	Tonhöhe 69 (A)
	01100100	100	Anschlagsdynamik 100
960	10000000	128	Note Off Ch. 0
	01000101	69	Tonhöhe 69 (A)
	01111111	127	Loslassdynamik 127
960	10010000	144	Note On Ch. 0
	01000001	65	Tonhöhe 65 (F)
	01100100	100	Anschlagsdynamik 100
1920	10000000	128	Note Off Ch. 0
	01000001	65	Tonhöhe 65 (F)
	01111111	127	Loslassdynamik 127
1920	11111111	255	Reset
	00101111	47	Ende des Tracks
	00000000	0	

Tabelle 4.4: Bytefolge der Standard-MIDI-Datei des Melodiestücks aus Abbildung 4.1

Durch MidiEvent werden nun diese Nachrichten um eine Zeitinformation (den sogenannten *Tick*) ergänzt. Ein Track enthält mehrere solcher MidiEvents wobei schließlich eine Sequence mehrere Tracks zusammenfasst und um die Auflösungsinformation (*PPQ*) ergänzt wird.

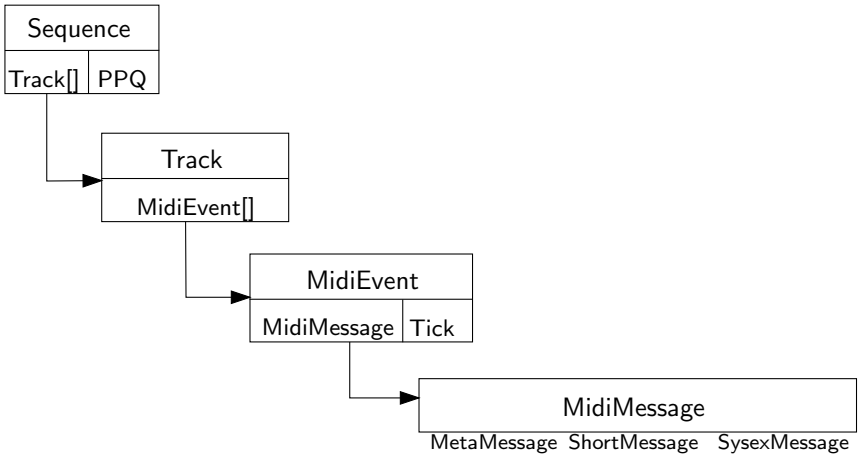


Abbildung 4.2: Datenstruktur zur MIDI-Verarbeitung in Java

Diese Sequences sind durch das Sequencer-Objekt von Java direkt abspielbar. Für eine genauere Beschreibung sei auf Sun Microsystems Inc. (2001) verwiesen.

4.4 jMusic

Die zur Musikverarbeitung angelegte Java-Bibliothek *jMusic* (Sorensen und Brown, 1998) stellt viele Funktionen im Zusammenhang mit MIDI zur Verfügung. So stehen zum Beispiel Editoren in Notendarstellung und Synthesizer zur Verfügung. Hier soll jedoch nur auf die von der Java-API stark abweichende Datenstruktur zur MIDI-Verarbeitung eingegangen werden. Ein diese darstellendes Diagramm ist in Abbildung 4.3 zu sehen.

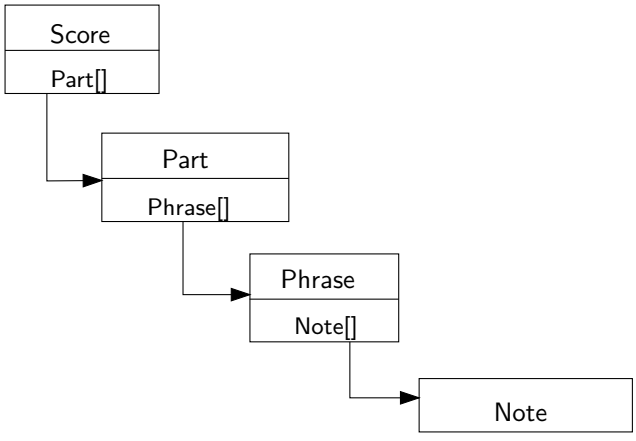


Abbildung 4.3: Datenstruktur zur MIDI-Verarbeitung in jMusic

Hierbei stellt die Klasse Note die elementaren Daten zur Verfügung, wobei nicht wie bei Java die MidiMessages gespeichert werden, sondern Attribute wie Tonhöhe und Tonlänge. Es findet keine

Speicherung des Zeitpunkts einer Note statt. Die Zeitpunkte ergeben sich aus der in Phrase angegebenen Reihenfolge von Noten mit unterschiedlichen Längen (Pausen werden als besondere Noten behandelt). Phrasen sind daher grundsätzlich monophon. Um mehrere Töne gleichzeitig darstellen zu können, aber auch um eine den Tracks bei der Java-Datenstruktur ähnliche Unterteilung zu erhalten, werden die Phrasen in Parts zusammengefasst. Die wiederum werden in Scores vereint, was eine Entsprechung in den Sequences aus der Java-Datenstruktur hat. Hier wird jedoch keine Auflösung mitgespeichert; diese wird automatisch von jMusic verwaltet. Scores enthalten Zusatzinformationen wie zum Beispiel das Tempo oder die Taktart, was bei der Java-API aufgrund dem eher technischen als musikalischen Schwerpunkt nicht der Fall ist.

Für eine genauere Betrachtung der Klassen sei auf die Sound-API von Java[2] (Sun Developer Network, 2005) und die API von jMusic[3] (Sorensen und Brown, 1998) verwiesen.

[2] http://java.sun.com/j2se/1.5.0/docs/api/javax/sound/midi/package-summary.html
[3] http://jmusic.ci.qut.edu.au/jmDocumentation/

Kapitel 5

Bisherige Arbeiten

Dieses Kapitel gibt einen Überblick über bereits bestehende Systeme aus dem Bereich der automatischen und auch interaktiven Komposition. Einen Schwerpunkt bilden hierbei solche, die die in dem Kapitel 2 vorgestellten Methoden nutzen, es werden aber auch Ansätze erklärt, welche andere Verfahren in den Mittelpunkt stellen. In diesem Fall werden die notwendigen Grundlagen kurz eingeführt.

Es werden nicht in jedem Fall alle Aspekte der jeweiligen Arbeiten beleuchtet. Vielmehr werden die besonders interessanten Punkte, welche die entwickelten Systeme von anderen abheben, herausgestellt. Dies soll zum einen einen Einblick in unterschiedliche Ansätze geben, zum anderen aber auch die Ideen herausstellen, welche für die vorliegende Arbeit richtungsweisend sind.

Diese Übersicht ist bezüglich verwendeter Methoden in der automatischen Komposition nicht vollständig, daher sei zur weiteren Übersicht hier auf Järveläinen (2000), Miranda (2001), Papadopoulos und Wiggins (1999), Santos, Arcay, Dorado, Romero und Rodriguez (2000) und Werner und Todd (1998) verwiesen.

5.1 Interaktive Melodieerzeugung durch genetisches Programmieren

Die Basis des von Jeffrey B. Putnam entworfenen Systems „Genetic Programming of Music“ (Putnam, 1994) bildet das genetische Programmieren, wie es in Abschnitt 2.2.2.4 beschrieben ist. Als Anspruch setzt er nicht die Notwendigkeit, musiktheoretische Erkenntnisse zu beweisen, sondern möchte experimentell feststellen, ob genetisches Programmieren in einer durch Menschen bestimmten Domäne erfolgreich sein kann. Als Ziel wird die Erzeugung eines Klanges, der im weitesten Sinne als ‘musikalisch’ bezeichnet werden kann, erwähnt.

Die Arbeit beschreibt zwei unterschiedliche Ansätze. Zum einen sollen direkt Klänge („Waveforms“) erzeugt werden (im .au-Format), des Weiteren werden Melodien evolviert (auf Basis von Sinusschwingungen unterschiedlicher Frequenz; die Ausgabe erfolgt ebenfalls im .au-Format).

Die direkte Erzeugung von Klängen wird als recht erfolglos bezeichnet. Es ist nicht gelungen, interessante Klänge in akzeptabler Zeit zu erzeugen. Als ein Grund wird hier die zeitaufwändige Fitnesszuweisung durch Menschen genannt.

Als besonderer Vorteil der Erzeugung von Melodien („Music“) wird der deutlich kleinere Suchraum bezeichnet, wodurch deutlich weniger Fitnessauswertungen notwendig sind.

Um hinreichend viele menschliche Fitnessauswertungen erreichen zu können ist ein Interface für Terminals entwickelt worden, welches, aufgrund der Tatsache, dass an öffentlichen Plätzen Musikhören als störend empfunden werden kann, bald auf ein Interface für das World Wide Web erweitert wurde.

Durch intensives Bewerben des Projekts konnten in einer Woche 100 Generationen mit 36 Individuen evaluiert werden. Das Interesse ebbte jedoch danach sehr zügig ab.

Insgesamt wird die Arbeit seitens des Autors nicht als Erfolg gewertet, jedoch sieht dieser die Möglichkeit, durch weitere Experimente, wie längere Laufzeiten des Systems, interessante Ergebnisse erzielen zu können.

5.2 Interaktive Erzeugung von Jazzmelodien

Das Programm GenJam von John A. Biles wurde 1994 entworfen (Biles, 1994) und seitdem aktiv weiterentwickelt und variiert.

Biles sieht als Vorbild für sein System einen Schüler, der die Jazz-Improvisation lernt und durch seine Zuhörer eine Rückmeldung über die Qualität seiner Darbietung erhält. Es erhält als Vorgabe einen rhythmischen Rahmen, eine Akkordfolge sowie eine automatisch (durch Fremdsoftware) erzeugte Begleitung aus Klavier, Bass und Schlagzeug im MIDI-Format.

Es werden nun zwei Populationen verwaltet: Zum einen die Taktpopulation, wobei jeder Takt durch eine Tonfolge gegeben ist, zum anderen die Phrasenpopulation, welche aus Anordnungen von Takten besteht.

Der Evaluationsprozeß findet während des ununterbrochenen Abspielens der Individuen statt, indem der Zuhörer durch wiederholtes Drücken von „g“ für ‘gut’ („good“) beziehungsweise „b“ für ‘schlecht’ („bad“) dem System seine Meinung mitteilt. Die Häufigkeit des Drückens beeinflußt die Höhe der Fitness wobei der Zeitpunkt festlegt, ob Takt oder Phrase beurteilt wurde.

Um den Suchraum einzuschränken werden nur bestimmte Skalen zu unterschiedlichen Akkorden zugelassen. Die Repräsentation basiert auf einer Abbildung möglicher Tonhöhen auf Zahlen, ähnlich der MIDI-Spezifikation.

Die implementierten Mutationsoperatoren für die Taktpopulation sind

- Reihenfolge der Töne umdrehen
- Rotieren[1]
- Tonhöhen invertieren (also Subtrahieren jedes Tones von dem höchsten vorkommenden Ton)
- Töne nach Tonhöhe absteigend oder aufsteigend sortieren
- Transponieren

Die Mutationsoperatoren für die Phrasenpopulation sind

- Reihenfolge der Takte umdrehen
- Rotieren
- Genetische Reparatur (Austauschen des Taktes mit schlechtester Fitness durch zufälligen Takt)
- Erzeugen einer neuen Phrase durch Nutzung der Takte mit den höchsten Fitnesswerten
- Ersetzen des in der Population am häufigsten auftretenden Taktes durch einen zufällig ausgewählten anderen Takt
- Erzeugen eines neuen Individuums durch die am seltensten in der Population vorhandenen Takte

Grundsätzlich wird die Arbeit seitens des Autors positiv bewertet, allerdings führt Biles hier auch den Begriff *Fitness Bottleneck* (‘Fitness Flaschenhals’) ein, mit dem er das durch die menschliche Bewertung auftretende Effizienzproblem bezeichnet. Mit diesem Problem beschäftigt sich der Ansatz, als Fitnessfunktion ein künstliches neuronales Netz zu nutzen (Biles u. a., 1996), was allerdings nicht als erfolgreich bezeichnet wird. Eine weitere Arbeit setzt sich mit dem Eliminieren des Fitnessproblems durch reines zufälliges Auswählen von bereits als gut befundenen Phrasen unter Verwendung von einem Kreuzungsoperator auseinander (Biles, 2001).

Einen guten Überblick über die Arbeiten zu GenJam geben auch Biles (2005) und Biles (2002).

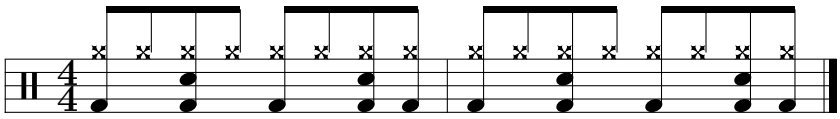
[1] Beispiel: Aus den Tönen `65 – 57 – 78 – 66 – 64` könnte durch Rotation `78 – 66 – 64 – 65 – 57` werden.

5.3 ART-Netze als Fitnessschätzer

In den Arbeiten von Anthony R. Burton (Burton und Vladimirova, 1997a,b; Burton, 1998) werden Schlagzeugstücke bestehend aus verschiedenen Schlaginstrumenten unter Nutzung eines genetischen Algorithmus als wichtiges Element evolviert. Das Bemerkenswerte ist hierbei die Fitnessbewertung, welche durch ein ART-1-Netz (siehe auch Abschnitt 2.3.4) erfolgt.

Hierzu werden dem Netz zunächst unterschiedliche Beispielstücke präsentiert, welche kategorisiert werden. Hierbei bilden sich Cluster die zum Beispiel entsprechend der Trainingsdaten „Rock“, „Funk“, „Fusion“, „Disco“ oder „Latin“ benannt werden. Zur Fitnessbewertung beim Lauf des genetischen Algorithmus wird schließlich die Ähnlichkeit zu dem Cluster, in das das zu bewertende Individuum eingeordnet wird, herangezogen. Wenn keine Einteilung in ein vorhandenes Cluster möglich ist, wird ein neues angelegt und die höchstmögliche Fitness zugewiesen.

Die Repräsentation, welche auch direkt als Eingabe für das ART-Netz fungiert, ist in einem Beispiel in Abbildung 5.1 dargestellt. Der hier verwendete Notenschlüssel, der in Abschnitt 3 nicht eingeführt wird, findet Verwendung zur Darstellung von Schlaginstrumenten. Die Noten auf Höhe des Tones g stellen hier eine Hi-Hat, die Noten auf Höhe des c eine Snare und die auf Höhe eines f eine Bass-Drum dar (jeweils unter Annahme eines Violin-Schlüssels).



1010101010101010 1010101010101010
0000100000001000 0000100000001000
1000100010001010 1000100010001010

Abbildung 5.1: Repräsentation von einem Schlagzeugstück bei Burton (1998)

Nach Beenden des Evolutionsprozesses finden sich ‘erfolgreiche’ Individuen in Zuordnung zu den entsprechend bereits bestandenen, stilistisch vorgelegten Clustern wieder. Diese stellen dann Variationen zu den beim Training etablierten Kategorien dar.

Der Autor betont die Vorteile dieses Ansatzes im Vergleich zu Multi-Layer-Feed-Forward-Netzen, der darin besteht, dass ein geringerer Trainingsaufwand betrieben werden muß. Ein Problem, das er erwähnt, ist die sehr hohe Anzahl von zu bildenden Kategorien durch die hohe Anzahl von Variationen, welche durch den genetischen Algorithmus erstellt werden. Durch dieses und weitere Probleme motiviert, werden schließlich Experimente mit einem ARTMAP-Netzwerk, einer Variante des ART-Netzes, durchgeführt.

5.4 Erzeugung von Jazzmelodien mit automatischer Fitnessbewertung

In dem von Geraint Wiggins und George Papadopoulos entwickelten System „A genetic Algorithm for the generation of Jazz Melodies“ (Wiggins und Papadopoulos, 1998) wird ein genetischer Algorithmus mit einer algorithmischen Zielfunktion eingesetzt. Es werden Akkorde vorgegeben, zu denen das System eine Jazz-Melodie entwickelt.

Die Initialisierung der Individuen für den genetischen Algorithmus erfolgt zufällig aus einer Menge von möglichen Notenlängen und einer gleichverteilt zufälligen Belegung dieser mit Tonhöhen. Mit einer Wahrscheinlichkeit von 12,5% wird ein Ton in eine Pause gewandelt.

Zur Mutation werden Operatoren genutzt, welche in drei Klassen eingeteilt werden:

1. Lokale Mutation (Operation auf Teilstück zufälliger Länge)
   - Transponieren[2] um zufällige Schrittweite
   - Permutation
   - Sortieren nach Tonhöhe
   - Ändern einiger Tonhöhen ohne Änderung des Rhythmus
   - Zufälliges Mischen der Tonlängen ohne Änderung der Tonhöhen
   - Verknüpfen von Pausen und identischen Tonhöhen
   - Ändern der Tonhöhe eines einzelnen Tones
2. Kopiermutation
   - Kopieren eines zufällig gewählten Teilstücks an eine andere Position unter eventueller Anwendung von lokaler Mutation (das Stück an der Zielposition wird überschrieben)
3. eingeschränkte Kopiermutation
   - Die selbe Prozedur wie die Kopiermutation, jedoch sind die Positionen, an denen das zu kopierende Teilstück beginnen kann, sowie die Länge dessen festgelegt.

Die Fitnessauswertung wird durch eine gewichtete Summe von Merkmalen vorgenommen. Die Gewichte sind, wenn möglich, jeweils in Klammern angegeben.

- Große Intervalle zerstören den Eindruck eines Melodieflusses (-20)
- Wiederholung von Patterns, nur auf den Tonhöhenverlauf bezogen (benutzerspezifiziert)
- Halten von Tönen über Akkordwechsel hinaus. Für jeden Akkordwechsel werden verschiedene Fälle betrachtet:
  - Note ist Teil beider Akkorde ($+10$)
  - Note ist Teil des ersten aber nicht des zweiten Akkords (-20)
  - Kein Halten eines Tones über beide Akkorde ($+5$)
  - Kein Ton, sondern eine Pause liegt über dem Akkord ($+5$)
- Betonte Schläge haben eine tragende Rolle im Verlauf. Es werden für den ersten Schlag und den dritten Schlag (im $\frac{4}{4}$-Takt) folgende Fälle unterschieden[3]:
  - Note ist ein Akkordton ($+10$, $+5$)
  - Auf dem Schlag liegt eine Pause ($+10$, $+5$)
  - Note ist kein Akkordton, aber einer, der in der entsprechenden Skala vorhanden ist (-10, $+5$)
  - Note ist weder im Akkord noch in der Skala (-20, -20)
- Lange Töne (ab wann ein Ton lang ist, wird vom Benutzer spezifiziert) sind ebenfalls besonders wichtig:
  - Note ist akkordeigen ($+10$)
  - Note ist nicht skaleneigen (-20)

[2] Ändern der Tonhöhe
[3] Hier gibt die erste Zahl in Klammer die Gewichtung für den ersten und die zweite die Gewichtung für den dritten Schlag an.

  - 'Note' ist eine Pause (-20)
  - Note ist konsonant über Akkordwechsel hinaus gehalten ($+20$)
  - Note ist dissonant über Akkordwechsel hinaus gehalten (-20)
- Der Benutzer kann eine gewünschte Kontur der Melodie angeben (also einen ungefähren Tonhöhenverlauf). Es wird die Ähnlichkeit zu dieser betrachtet.
- Ähnlich wie bei der Kontur kann eine gewünschte Geschwindigkeit (schnell, mittel, langsam) angegeben werden, wobei die Ähnlichkeit des Individuums hierzu bewertet wird.

Die Autoren stellen heraus, dass musikalisch ansprechende Individuen erst durch Implementierung der eingeschränkten Kopiermutation möglich werden. Sie bezeichnen die erzeugten Melodien als „viel versprechend" („encouraging"), bezogen auf das geringe kodierte Wissen.

5.5 Genetisches Programmieren mit automatischer Fitnessbewertung

Brad Johanson nutzt in seinem System „GP-Music" (Johanson und Poli, 1998) genetisches Programmieren, um Melodien zu erzeugen. Hervorzuheben ist hier, dass die Fitnessbewertung durch ein künstliches neuronales Netz, aber auch durch eine interaktive, menschliche Bewertung vorgenommen wird. Leider wird in der angegebenen Literatur nicht entgültig klar, wie genau das neuronale Netz genutzt wird.

Der Autor begründet die Verwendung des „automatischen Bewerters" mit der Vermeidung des bei interaktiven genetischen Algorithmen vorhandenen Flaschenhalsproblems. Allerdings muß er einräumen, dass die erzeugten Melodien bei automatischer Bewertung weniger interessant sind als bei interaktiver Bewertung.

Er regt daher eine kombinierte Bewertung durch beide Varianten an.

5.6 Merkmalsextraktion zur Nutzung in genetischen Algorithmen

Der Ansatz von Michael Towsey, Andrew Brown, Susan Wright und Joachim Diederich (Towsey u. a., 2000) ist dem aus Abschnitt 5.4 sehr ähnlich. Der besondere Schwerpunkt in dieser Arbeit ist jedoch die Extraktion von Merkmalen. So werden hier die Durchschnittswerte dieser Merkmale in Bezug auf bestimmte Musikstile bestimmt, um stilistische Besonderheiten von Stücken festzustellen.

Auf eine Angabe der von Towsey und Brown entwickelten Merkmale wird hier verzichtet. Stattdessen sind sie in der Beschreibung der Merkmalsextraktion in Kapitel 10.1 gekennzeichnet.

Die Autoren stellen heraus, dass ihre Absicht nicht ist, aus diesen Merkmalen eine Zielfunktion für genetische Algorithmen zu erzeugen. Sie möchten vielmehr einen Anteil zu der statistischen Analyse von Musik leisten, was wiederum in Zielfunktionen nutzbar gemacht werden kann.

5.7 Ein künstliches neuronales Netz als Melodiegenerator

In den Abschnitten 5.1, 5.2, 5.3, 5.5 und bezüglich zugrunde liegender Ideen auch in 5.6 geht es um die Erzeugung von Melodien mit evolutionären Algorithmen, wobei die komplizierteste Aufgabe hierbei ist, die 'Kreativität' durch eine Fitnessfunktion sinnvoll einzuschränken.

Einen anderen Weg gehen Chun-Chi J. Chen and Risto Miikkulainen (Chen und Miikkulainen, 2001). In ihrer Arbeit „Creating Melodies with Evolving Recurrent Neural Networks" nutzen sie einen evolutionären Algorithmus zur Optimierung eines künstlichen neuronalen Netzes, welches als Melodiegenerator fungiert. Die Verbesserung dieses rekurrenten Netzes soll sicherstellen, dass ansprechende Melodien

erzeugt werden. Weiterhin werden Nebenbedingungen genutzt, und zwar stellen diese rhythmische und tonale Abwechslung, eine hohe Anzahl an Tönen je Takt und Skalenzugehörigkeit der Töne sicher und realisieren weiterhin die Einhaltung bestimmter Vorgaben bezüglich des Melodieverlaufs. Die Struktur des Netzes ist in Abbildung 5.2 zu sehen.

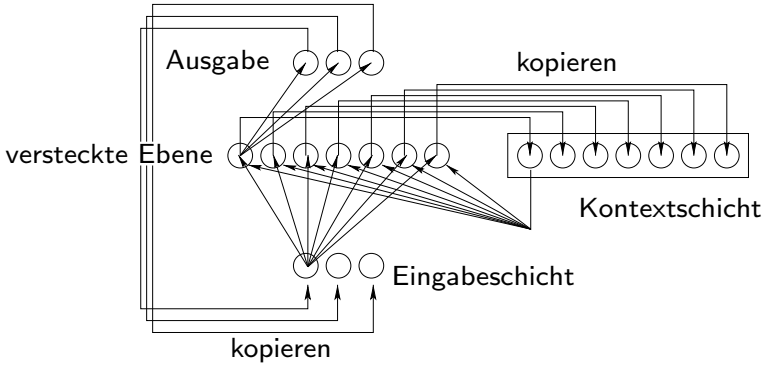


Abbildung 5.2: Die Netzstruktur aus Chen und Miikkulainen (2001)

Als Ziel nennen die Autoren die Erzeugung von Melodien im Stil des Komponisten Bela Bartok. Zur Vereinfachung werden einige Einschränkungen in Kauf genommen, wie eine starke Beschränkung der Anzahl unterschiedlicher Notenlängen und Tonhöhen. In diesen Grenzen bezeichnen die Autoren ihr System als erfolgreich.

5.8 Interaktive Komposition

In dem von Andrew Gartland-Jones entworfenen System MusicBlox (Gartland-Jones, 2003) liegt der Schwerpunkt in der interaktiven Entwicklung von Musik. In einem simulierten dreidimensionalen Raum kann der Benutzer Blöcke auf- und nebeneinander legen. Jeder dieser Blöcke bekommt, zufällig oder interaktiv, eine „Home-Music“ zugewiesen. Diese wird nun von Block zu Block gesandt. Wenn ein Block eine Melodie empfängt, verändert er seine eigene „Home-Music“ mit einem genetischen Algorithmus, wobei als Zielfunktion die Ähnlichkeit zu der empfangenen Melodie fungiert. Eine Abbildung der Bedienungsoberfläche seines Systems ist in Abbildung 5.3 zu sehen.

5.9 Zelluläre Automaten zur Generierung von Musik

Der Ansatz im Programm „CAMUS“ von Eduardo R. Miranda und Kenny McAlpine (Miranda, 2001, 1993) baut auf den Einsatz von zellulären Automaten. Der zelluläre Automat „Game of Life“ ist eine Matrix aus Feldern wobei jedes Feld den Zustand 1 oder den Zustand 0 einnehmen kann. Der Folgezustand hängt dabei nur von seinen Nachbarfeldern ab, und zwar nimmt eine Zelle den Zustand 1 genau dann an, wenn drei Nachbarn in Zustand 1 sind. Der Zustand 0 wird dann angenommen, wenn 4 oder mehr, 1 oder 0 Nachbarn in Zustand 1 sind. Im Fall von 2 Nachbarn in Zustand 1 ändert sich der Zustand der Zelle nicht. Der Prozeß der Änderung der Zustände der Zellen bei diesem und einem weiteren zellulären Automaten findet eine Abbildung auf eine Musikerzeugung. Hierbei werden die Zellen, die ihren Zustand von 0 auf 1 wechseln über ein Koordinatensystem und eine Menge von Vorlagen in ein Tontripel überführt.

Abbildung 5.3: Bedienungsoberfläche des Systems MusicBlox (Leider liegt das System selbst nicht vor so dass keine qualitativ höherwertige Abbildung zugänglich ist.)

5.10 Interaktive Evolution von Akkorden

Das System „Vox Populi“ von Artemis Moroni, Jônatas Manzolli, Fernando Von Zuben und Ricardo Gudwin (Moroni u. a., 2002; Miranda, 2001) nutzt genetische Algorithmen, um Akkorde zu evolvieren. Jeder Akkord hat hierbei vier Töne, wobei jeder durch einen 7-Bit-String repräsentiert wird, wodurch ein Akkord durch einen 28-Bit-String gegeben wird. Auf den Individuen, welche mit dieser Repräsentation kodiert sind, werden im Verlauf des genetischen Algorithmus die Operatoren Rekombination und Mutation entsprechend der Beschreibung in den Abschnitten 2.2.3.4 und 2.2.3.5 ausgeführt.

Die Fitness wird anhand der Kombination der folgenden Merkmale bestimmt:

- Das *Konsonanzkriterium* bestimmt aufgrund der Obertonreihe[4] eines Tones, ob ein anderer Ton zu ihm passt.
- Die *Melodische Fitness* wird bestimmt, in dem aus einem benutzergegebenen tonalen Zentrum und seiner Obertonreihe die Dissonanz zum zu bewertertenden Akkord bestimmt wird.
- Die *Harmonische Fitness* wird über eine Kombination von mehrfachen Anwendungen des Konsonanzkriteriums auf die Töne eines Akkords bestimmt.
- Das *Stimmenumfangskriterium* bestimmt, ob die Töne des Akkords sich über eine unscharfe Menge in die Stimmumfangsbezeichnungen für menschlichen Gesang, „Bass“, „Tenor“, „Alt“ und „Sopran“ einordnen lassen.

Das System kann über eine geeignete Bedienungsoberfläche (siehe 5.4) durch graphische Festlegung des tonalen Zentrums durch den Benutzer in Echtzeit genutzt werden.

[4]Die Obertonreihe eines Tones besteht aus allen Tönen, welche sich aus Frequenzverdopplungen des Ursprungstones ergeben. Wenn also der Ursprungston die Frequenz f hat, dann ergibt sich die Obertonreihe durch die Folge $f_n = (f_{n-1} * 2)$ mit $f_0 = f$. Hierbei ist $n \in \mathbb{N}$)

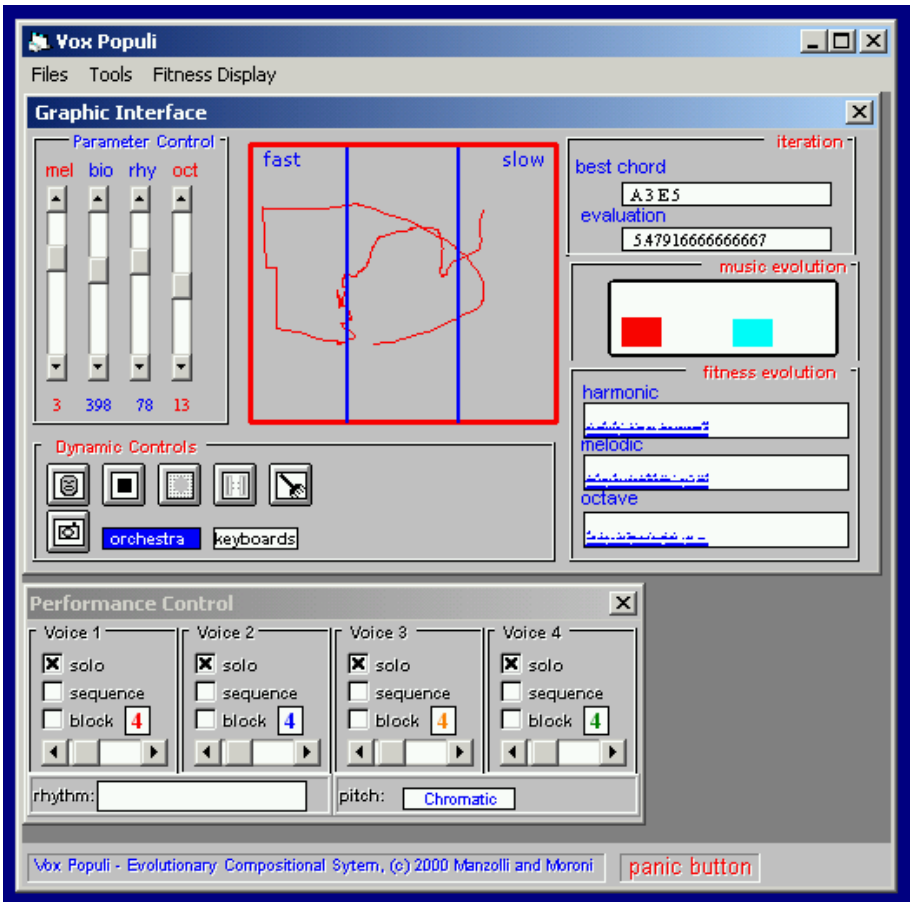


Abbildung 5.4: Bedienungsoberfläche des Systems Vox Populi

Teil II

Entwicklung und Implementierung des Systems *MusiComp*

Kapitel 6

Überblick und Systemarchitektur

In diesem Kapitel wird ein Überblick über das implementierte System gegeben wobei grundlegende Zusammenhänge sowie die Systemarchitektur im Vordergrund stehen.

Der zentrale Bestandteil des implementierten Systems basiert auf einem evolutionären Algorithmus mit konfigurierbaren Komponenten. So sind verschiedene Parameter des genetischen Algorithmus zu belegen, zu denen neben der Wahl der Initialisierungsverfahren, Angabe von Mutations- und Rekombinationsoperatoren sowie der Bewertungsfunktion der Individuen auch die Bestimmung von Populationgröße und Variante des eingesetzten Verfahrens zählt.

Die Struktur des Systems ist in Abbildung 6.1 gezeigt.

In der Graphik bezeichnet das Symbol  besondere Benutzerinteraktion wobei Parameterfestlegungen auch an anderen Stellen notwendig sind. Das Symbol  kennzeichnet die Verarbeitung von MIDI-Dateien.

Der evolutionäre Algorithmus ist unterteilt in *Initialisierung*, *Rekombination*, *Mutation*, *Bewertung* und *Selektion*. Hierbei greifen diese Teile jeweils auf parametrisierbare Komponenten zu. Einen entscheidenden globalen Parameter stellt die Angabe einer Akkordfolge mit damit verbundener Skalenzuweisung dar (siehe Abschnitt 7.1), welche von mehreren Seiten genutzt wird. Ebenso hat die Wahl, ob Crowding genutzt wird, Auswirkungen auf weitere Komponenten. So werden dann die unterstrichenen Optionen automatisch gewählt. Grundlegend ist auch die Festlegung der Populationsgröße.

Die detaillierten Beschreibungen beginnen in Kapitel 7 mit der grundlegenden Darstellung von Melodien und damit verbundenen Methoden. Weitergehend wird auf die Initialisierung eingegangen, welche unterteilt wird in Methoden zur Festlegung des Rhythmus und Bestimmung der auf dieser geführten Melodie. Hierzu kommen jeweils Markovketten unterschiedlicher Ordnung zum Einsatz. Zur Bestimmung des Rhythmus ist auch das Anlegen einer Gruppe von Mustern möglich, welche zufällig ausgewählt aneinandergereiht werden. Weitere von diesen abgeleitete Verfahren stehen zur Verfügung und werden in Kapitel 8 erläutert. Zur Rekombination können Ein-Punkt- und intermediäre Rekombination genutzt werden, wobei zweitgenanntes nicht zusammen mit Crowding zum Einsatz kommen kann. Die genauere Beschreibung findet sich in Abschnitt 9. Zur Mutation werden einige Operatoren genutzt, welche sich unterteilen lassen in Methoden, welche die Tonhöhe verändern (Punktmutation, Abschnittstransposition, Abschnittsinversion), einen Bereich strukturell verändern (Sortieren, Spiegeln, Rotieren) und in erster Linie den Rhythmus manipulieren (Verschieben, Teilen, Verbinden von Tönen) (siehe Kapitel 9.2).

Verschiedene Bewertungsmethoden und deren Entwurf und Implementierung werden in Kapitel 10 diskutiert. Hier wird auf die interaktive Bewertung der Individuen eingegangen, sowie auf den Versuch, diese mit künstlichen neuronalen Netzen und Entscheidungsbäumen nachzubilden. Auch die Kombination der unterschiedlichen Verfahren ist möglich und in bestimmten Anwendungsfällen nützlich. Des weiteren werden hier auch die eingesetzten Selektionsverfahren beschrieben.

Zum Abschluss werden in Kapitel 11 Erfahrungen mit verschiedenen Parameterbelegungen und den zur Verfügung stehenden Verfahren diskutiert.

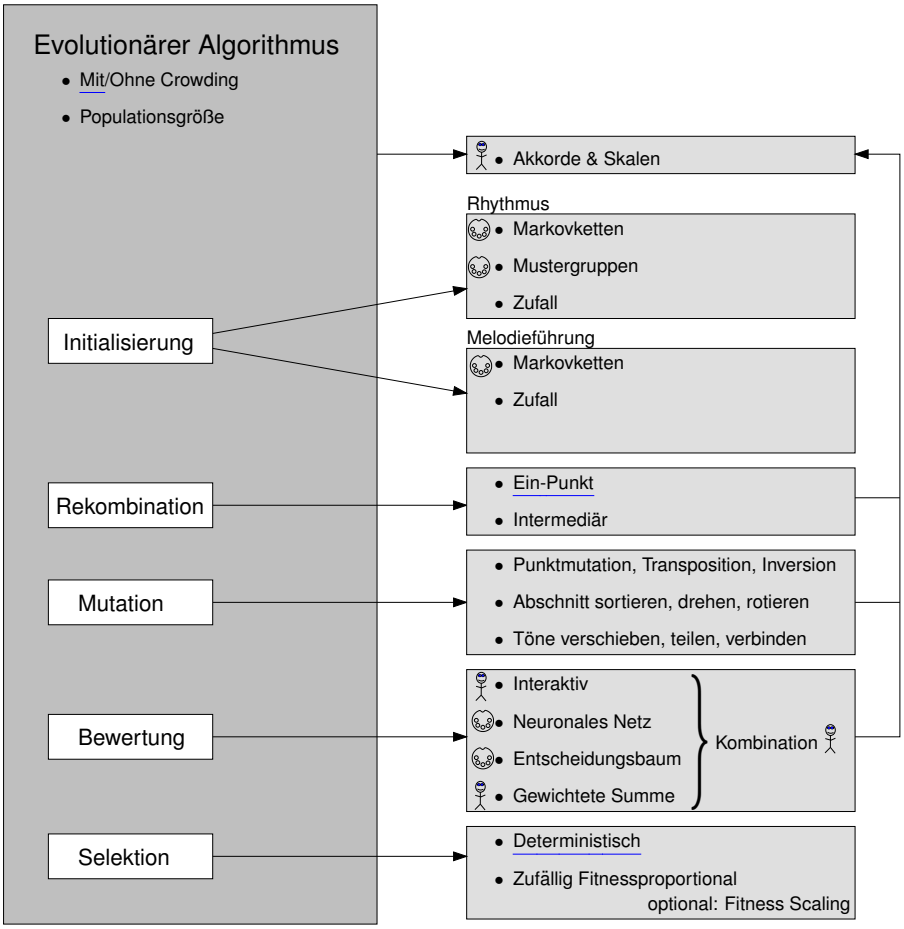


Abbildung 6.1: Überblick über die Architektur der Implementierung. Das Symbol bezeichnet eine entscheidende Benutzeraktion, das Zeichen steht für die Verarbeitung von MIDI-Dateien an entsprechender Stelle. Unterstrichen sind die Optionen, welche bei der Wahl von Crowding automatisch genutzt werden. Die Pfeile sind zu lesen als „benutzt“.

Kapitel 7

Repräsentation

7.1 Akkorde und Skalen

In Kapitel 3 insbesondere in Abschnitt 3.2 wurden die Begriffe Stammtonreihe und Tonleiter eingeführt. Solche Tonleitern, auch als Skalen bezeichnet, definieren Tonarten. Zu jeder Tonart zugehörig sind Akkorde, welche aus Tönen der entsprechenden Skala bestehen. Die Stammtonreihe beginnend mit dem Ton *C* korrespondiert so mit dem *C*-Dur-Akkord.

Zum automatischen Entwurf einer Melodie werden in der hier beschriebenen Implementierung Skalen und damit implizit Akkorde auf bestimmten Grundtönen vom Benutzer angegeben. Hierbei werden, wie in Tabelle 7.1 dargestellt ist, Akkorde wie auch Skalen durch eine Folge von Schritten auf der chromatischen, also aus allen zwölf Tönen einer Oktave bestehenden, Tonleiter angegeben.

7.2 Repräsentation eines Individuums

Ein Individuum in der Evolution repräsentiert eine Melodie auf einer angegebenen Akkordfolge. Hierbei ist die Länge der Melodie $t \in \mathbb{N}\backslash\{0\}$ in Takten und die Anzahl der Viertelschläge je Takt $v \in \mathbb{N}\backslash\{0\}$ vorzugeben. Des Weiteren ist die *Auflösung* r anzugeben, also die kürzeste Dauer eines Ereignisses, also eines Tons oder einer Pause. Diese wird in Anteilen einer Viertelnote spezifiziert, welche die Dauer 1 zugewiesen bekommt. Eine Auflösung von $r = 0{,}5$ definiert so die kürzeste mögliche Dauer eines Ereignisses als $\frac{1}{8}$. Daraus ergibt sich die Anzahl möglicher Positionen von Ereignissen $n \in \mathbb{N}\backslash\{0\}$ innerhalb einer Melodie eines Individuums durch

$$n = \frac{t \cdot v}{r}$$

Ein Akkord wird bestimmt durch ein Wertepaar (T, S) wobei $T \in \{c, d, e, f, g, a, h\}$ den Grundton angibt und $S \in \{1, \ldots, 47\}$ die Kennzahl entsprechend Tabelle 7.1. Eine Akkordfolge eines Individuums ist daher ein n-Tupel $a \in (T,S)^n$.

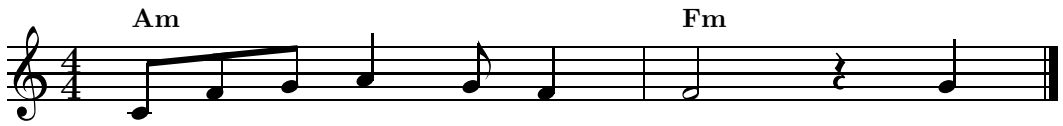
Die Melodie selbst wird schließlich repräsentiert durch ein Tupel $m \in \{-2, -1, 0, \ldots, 127\}^n$. Die Werte 0 bis 127 stellen den Beginn eines Tones mit der mit dem Wert nach Tabelle 4.3 korrespondierenden Tonhöhe dar. Der Wert -1 bedeutet, den im Tupel vorhergehenden Wert zu halten (den Ton also nicht neu anzuschlagen) und eine -2 stellt den Beginn einer Pause dar. Ein Individuum I ist also als Paar $I = (m, a)$ zu begreifen. Ein Beispiel zur Darstellung von Melodien mit gegebenen Akkorden findet sich in Abbildung 7.1.

7.3 Speicherung von Individuen

Die Melodien werden in MIDI-Dateien gespeichert. Damit aus einer gespeicherten MIDI-Datei jedoch wieder ein Individuum des evolutionären Algorithmus rekonstruiert werden kann bedarf es einiger Zusatzinformationen. Dazu gehören neben der Fitness des Individuums auch die über die Positionen hin geltenden Akkorde und Skalen. Die Auflösung des Individuums wird nicht gespeichert, da davon ausgegangen wird, dass diese nicht verändert wird.

Kennzahl	Bezeichnung	Akkordkürzel	Akkord	Skala
1	Dur	*d*	0 4 7	0 2 4 5 7 9 11
2	Dur ohne Quarte	d	0 4 7	0 2 4 7 9 11
3	Mixolydisch	+7	0 4 7 10	0 2 4 5 7 9 10
4	Mixolydisch ohne Quarte	+7	0 4 7 10	0 2 4 7 9 10
5	Moll (natürlich)	*m*	0 3 7	0 2 3 5 7 8 10
6	Moll Septime (natürlich)	*m7*	0 3 7 10	0 2 3 5 7 8 10
7	Moll (natürlich) ohne Sexte	m	0 3 7	0 2 3 5 7 10
8	Moll Septime (natürlich) ohne Sexte	m7	0 3 7 10	0 2 3 5 7 10
9	Moll (harmonisch)	m	0 3 7	0 2 3 5 7 8 11
10	Moll (harmonisch) ohne Sexte	m	0 3 7	0 2 3 5 7 11
11	Moll (melodisch)	m	0 3 7	0 2 3 5 7 8 9 10 11
12	Lokrisch	*m7b5*	0 3 6 10	0 1 3 5 6 8 10
13	Lokrisch ohne Sekunde	m7b5	0 3 6 10	0 3 5 6 8 10
14	Vermindert	*dim*	0 3 6	0 2 3 5 6 8 9 11
15	Lydisch erweitert	+ oder *#5*	0 4 8	0 2 4 6 8 9 11
16	Ganztöne	*7+* oder *7#5*	0 4 8 10	0 2 4 6 8 10
17	Lydisch dominant	*7#11*	0 4 7 10 19	0 2 4 6 7 9 10
18		*7#9*	0 4 7 10 16	0 3 4 7 9 10
19		7#9	0 4 7 10 16	0 1 3 4 6 8 10
20	Vermindert Variante A	*7b9*	0 4 7 10 14	0 1 4 5 7 8 10
21	Vermindert Variante A ohne Sexte	7b9	0 4 7 10 14	0 1 4 5 7 10
22	Vermindert Variante B	7b9	0 4 7 10 14	0 1 3 4 6 7 9 10
23		*mMaj7*	0 3 7 11	0 2 3 5 7 9 11
24		*m6*	0 3 7 9	0 2 3 5 7 9
25	Phrygisch	*m7b9*	0 3 7 14	0 1 3 5 7 9 10
26	Lydisch	*Maj7#11*	0 4 7 11 19	0 2 4 6 7 9 11
27	Aeolisch	Grundton	0	0 2 3 5 7 8 10
28	Dorisch	Grundton	0	0 2 3 5 7 9 10
29	Blues	Grundton	0	0 2 3 4 5 7 9 10 11
30	Türkisch	Grundton	0	0 1 3 5 7 10 11
31	Indisch	Grundton	0	0 1 1 4 5 8 10
32	Pentatonisch	Grundton	0	0 2 4 7 9
33	Chromatisch	Grundton	0	0 1 2 3 4 5 6 7 8 9 10 11
34	Dur	d	0 4 7	0 4 7
35	Dur Septime	+7	0 4 7 10	0 4 7 10
36	Moll	m	0 3 7	0 3 7
37	Moll Septime	m7	0 3 7 10	0 3 7 10
38	Moll Septime, verminderte Quinte	m7b5	0 3 6 10	0 3 6 10
39	Vermindert	dim	0 3 6	0 3 6
40	Erweitert	+ oder #5	0 4 8	0 4 8
41	Septime erweitert	7+ oder 7#5	0 4 8 10	0 4 8 10
42	Septime mit Duodezime	7#11	0 4 7 10 19	0 4 7 10 19
43	Septime, verminderte None	7b9	0 4 7 10 14	0 4 7 10 14
44	Moll, große Septime	mMaj7	0 3 7 11	0 3 7 11
45	Moll-Sext	m6	0 3 7 9	0 3 7 9
46	Moll Septime, verminderte None	m7b9	0 3 7 14	0 3 7 14
47	Dur Septime mit Duodezime	7#11	0 4 7 10 19	0 4 7 10 19

Tabelle 7.1: Implementierte Skalen und dazugehörige Akkorde (1–24 stammen von John Biles (Biles, 1994, 2005) oder sind direkt von den von ihm genannten Skalen abgeleitet, 25–33 sind dem Quelltext der jMusic-Bibliothek entnommen (Sorensen und Brown, 1998), zu unbenannten Tabellenzeilen konnte kein eindeutiger Name ausfindig gemacht werden)



$$
\begin{aligned}
m &= (60,-1{,}65,-1{,}67,-1{,}69,-1,-1,-1{,}67,-1,65,-1,-1,-1,\\
&\quad 65,-1,-1,-1,-1,-1,-1,-1,-2,-1,-1,-1,67,-1,-1,-1,)\\
a &= \Big((A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),\\
&\quad (A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),(A{,}5),\\
&\quad (F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5),\\
&\quad (F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5),(F{,}5)\Big)
\end{aligned}
$$

Abbildung 7.1: Repräsentation eines Individuums mit $r = 0{,}25$, $t = 2$ und $v = 4$

Diese Zusatzinformationen werden in einer XML[1]-Datei abgelegt (zur Spezifikation von XML sei auf Bray u. a. (2004) verwiesen). Diese XML-Datei folgt der durch die in Abbildung 7.2 angegebene DTD[2]. In Abbildung 7.3 ist ein Beispiel für eine solche XML-Datei, die eine Midi-Datei zu einem Individuum erweitert, gegeben.

```
<!ELEMENT addinfo (chords , fitness , length)>
<!ELEMENT chords (chord)*>
<!ELEMENT chord (pitch , scale)>
<!ATTLIST chord pos CDATA #IMPLIED>
<!ELEMENT pitch (#PCDATA)>
<!ELEMENT scale (#PCDATA)>
<!ELEMENT length (#PCDATA)>
<!ELEMENT fitness (#PCDATA)>
```

Abbildung 7.2: Spezifikation der XML-Datei zur Speicherung eines Individuums als DTD

7.4 Bestimmung der Ähnlichkeit

Die Ähnlichkeit von Individuen lässt sich in Phänotyp und Genotyp bestimmen, wobei sich hier die Berechnung der genotypischen Ähnlichkeit anbietet.

Definition 24 *Die Positionen der Melodie eines Individuums I_l seien mit m_i^l mit $i \in \{1,\ldots,n\}$ bezeichnet. Dann ist die* Ähnlichkeit *$sim(I_1, I_2)$ zweier Individuen I_1 und I_2 gegeben durch*

$$sim(I_1,I_2) = 1 - \frac{\sum_{i=1}^{n} dist(m_i^1, m_i^2)}{h(I_1, I_2)}$$

wobei

$$dist(m_i^1, m_i^2) = \begin{cases} 0 & \text{wenn } m_i^1 \neq -2 \text{ und } m_i^2 \neq -2 \\ \min(|m_i^1 - m_i^2|, \Delta_{max}) & \text{sonst} \end{cases}$$

und

$$h(I_1, I_2) = \sum_{i=1}^{n} a(x,y) \text{ mit } a(x,y) = \begin{cases} 0 & \text{für } x = y = -2 \\ \Delta_{max} & \text{sonst} \end{cases}$$

ist.

[1] Extensible Markup Language
[2] Document Type Definition

```
<?xml version="1.0" encoding="UTF-8"?>
<addinfo>
  <chords>
    <chord pos="0">
      <pitch>A</pitch>
      <scale>1</scale>
    </chord>
    <chord pos="16">
      <pitch>E</pitch>
      <scale>5</scale>
    </chord>
    <chord pos="32">
      <pitch>F</pitch>
      <scale>1</scale>
    </chord>
    <chord pos="48">
      <pitch>A</pitch>
      <scale>5</scale>
    </chord>
  </chords>
  <fitness>0.3</fitness>
 <length>64</length>
</addinfo>
```

Abbildung 7.3: Beispiel einer XML-Datei zur Speicherung eines Individuums

Der Parameter Δ_{max} gibt die maximal zur Ähnlichkeitsfindung betrachtete Distanz zweier Töne an und ist in dieser Arbeit mit $\Delta_{max} = 4$ belegt. Zu bemerken ist, dass $sim(I_1, I_2) = 1$ bei höchstmöglicher Ähnlichkeit von I_1 und I_2 gilt sowie $sim(I_1, I_2) = 0$ bei maximaler Unähnlichkeit.

Diese Funktion betrachtet einige Aspekte der Ähnlichkeit von Melodien nicht und kann nicht allgemein als Maßstab gesehen werden. Die Vorteile (und Nachteile) des Einsatzes dieser Funktion werden in Kapitel 11 und 12 geschildert. Des weiteren sei hier auf Grachten, Arcos und de Mántaras (2004) verwiesen, in deren Arbeit eine eher im allgemeinen gültige Ähnlichkeitsfunktion angegeben ist, welche jedoch aus Effizienzgründen hier nicht zum Einsatz kommt.

Kapitel 8

Initialisierung

Der dem implementierten System zugrundeliegende evolutionäre Algorithmus benötigt eine initiale Population, von der die Optimierung ausgeht.

Diese erste Erstellung von Individuen erfolgt in zwei Schritten. Zunächst wird der Rhythmus der Melodie erzeugt. Hierbei können Markovketten unterschiedlicher Ordnung zum Einsatz kommen wie auch Mustergruppen aus denen unterschiedliche Rhythmusstücke zufällig aneinandergehängt werden. Des Weiteren besteht die Möglichkeit, aus einem vorgegebenen Rahmen gleichverteilt zufällig Notenlängen aneinanderzuhängen. Das Ergebnis dieses Schritts ist ein Individuum mit Tönen auf nur einer Höhe, welche als Grundlage für den zweiten Schritt dienen.

In diesem zweiten Schritt wird auf diesem festgelegten Rhythmus die Melodieführung erzeugt. Hierzu stehen ebenfalls Markovketten in zwei verschiedenen Varianten zur Verfügung. Eine gesonderte Form ist vorgefertigt, nämlich die Erzeugung durch einen Random Walk. Des weiteren ist eine zufällige Erzeugung implementiert.

Es sind zur Initialisierung der Population mehrere Verfahren gleichzeitig wählbar. Hierzu sind jeweils Prioritäten zuweisbar, entsprechend denen dann zum einen für die Erzeugung des Rhythmus, zum anderen für die Erzeugung der Melodie eine Wahrscheinlichkeitsverteilung erzeugt wird, mit deren Hilfe dann für jedes Individuum zufällig ein Verfahren für den Rhythmus und eines für die Melodie gewählt wird.

8.1 Modelle zur Erstellung des Rhythmus

8.1.1 Markovketten

Markovketten zur Erstellung des Rhythmus betrachten die Folge von Notenlängen. Respektiert werden hierbei Töne und Pausen mit Längen zwischen der Länge eines Takts und der minimalen Tonlänge, gegeben durch die Auflösung des Individuums.

Des weiteren stehen Markovketten mit Ordnung zwischen 1 und 5 zur Verfügung. Diese werden aus Beispielen geschätzt, welche als MIDI-Dateien zur Verfügung gestellt werden (hierbei kommt die in Abschnitt 2.1.4.1 vorgestellte Methode zum Einsatz). Die Melodieführung wird ignoriert. Mögliche Ungenauigkeiten innerhalb der MIDI-Dateien werden vor der Verarbeitung durch Quantisierung ausgeräumt.

In Abbildung 8.1 ist die Umrechnung von zwei Melodien in Beispiele zur Schätzung einer Markovkette dargestellt. Hierbei stellen die Zahlen die Notenlänge wie bereits in Abschnitt 7.2 durch Vielfache der Länge 1, welche eine Viertelnote repräsentiert, dar. Pausen (was hier nicht zu sehen ist) werden durch negative Zahlen dargestellt, wobei deren Betrag entsprechend die Länge angibt.

Aus diesen Beispielen kann nun, wie in Abschnitt 2.1.4.1 erläutert, eine Markovkette erzeugt werden. Graphisch ist die hier entstehende Markovkette erster Ordnung in Abbildung 8.2 dargestellt. Wie man hier sieht, wird der Bereich vor der Melodie als längstmögliche Pause betrachtet. Führende Pausen bei den Beispielen zur Erstellung der Markovkette werden ignoriert, um durch möglicherweise notwendiges wiederholtes Initialisieren der Markovkette bei der Erstellung eines Rhythmus den Anteil der Pausen nicht zu erhöhen. Auf diese Technik wird weiter unten noch genauer eingegangen.

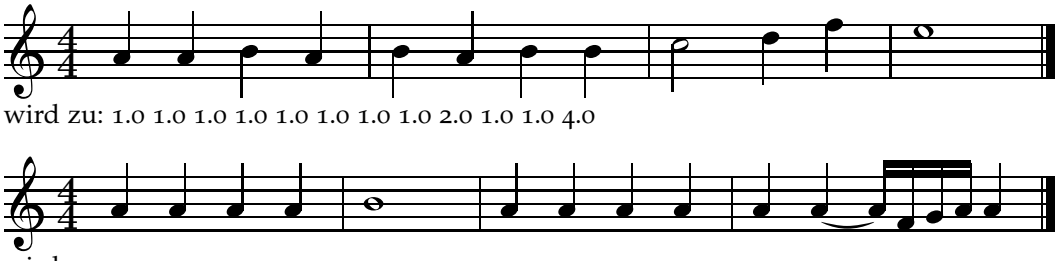


Abbildung 8.1: Beispiel für die Erzeugung der Eingabe zur Schätzung einer Markovkette

Zu bemerken ist, dass, entsprechend der Beispiele aus denen die Markovkette erstellt wurde, die Viertelnoten in dem Beispiel eine zentrale Rolle spielen, die mit hoher Wahrscheinlichkeit von 0,789 auf eine Note der selben Länge und mit Sicherheit auf halbe und ganze Noten folgen. Die Markovketten der Ordnungen 1 und 3 sind in Tabelle 8.1 angegeben. Hierbei sind alle möglichen Folgen von Notenlängen genannt, bei denen die Wahrscheinlichkeiten für deren Auftreten größer 0 sind. Auf die Angabe der charakteristischen Matrix wird aus Platzgründen verzichtet.

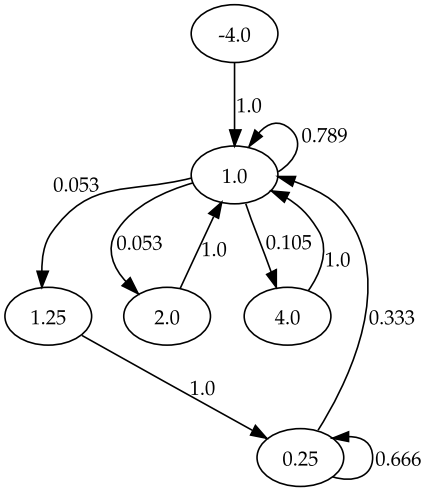


Abbildung 8.2: Markovkette erster Ordnung zur Erzeugung des Rhythmus entsprechend der Beispiele in Abbildung 8.1

Die zur Erzeugung von Rhythmen aus auf die hier beschriebene Weise erstellten Markovketten benötigt eine initiale Wahrscheinlichkeitsverteilung aus der sich die erste Notenlänge ergibt. Hierbei wird die Verteilung genutzt, welche von vorhergehenden längstmöglichen Pausen ausgeht. Diese entspricht auch dem Beginn der Rhythmen der Beispiele, da der Beginn dieser als auf längstmögliche Töne folgend interpretiert wird. Die initiale Verteilung ist in den Beispielen in Tabelle 8.1 jeweils nur in der ersten Spalte zu sehen, da es sicher ist, dass die Melodie mit einer Viertelnote beginnt (vergleiche auch Abbildung 8.1).

Diese initiale Wahrscheinlichkeitsverteilung kommt auch dann zum Tragen, wenn ein Zustand erreicht ist, bei dem es keinen Folgezustand geben kann, wie dies bei Erreichen einer ganzen Note in der in Abbildung 8.3 gezeigten Markovkette der Fall wäre. Hier wird deutlich, warum es sinnvoll erscheint, die führenden Pausen bei den Dateien, aus denen die Markovketten erstellt werden, zu ignorieren.

x_{t-1}	-4	0.25	0.25	1	1	1	1	1.25	2	4
x_t	1	0.25	1	1	1.25	2	4	0.25	1	1
P	1	0.667	0.333	0.7895	0.053	0.053	0.105	1	1	1

x_{t-3}	-4	-4	-4	0.25	1	1	1	1	1	1	1	1	1	1	1.25	2	4
x_{t-2}	-4	-4	1	0.25	1	1	1	1	1	1	1	1.25	2	4	0.25	1	1
x_{t-1}	-4	1	1	0.25	1	1	1	1	1.25	2	4	0.25	1	1	0.25	1	1
x_t	1	1	1	1	1	1.25	2	4	0.25	1	1	0.25	1	1	0.25	4	1
P	1	1	1	1	0.72	0.09	0.09	0.09	1	1	1	1	1	1	1	1	1

Tabelle 8.1: Markovkette 1.(oben) und 3. (unten) Ordnung zur Erzeugung von Rhythmus geschätzt aus den Beispielen in Abbildung 8.1. P gibt die Wahrscheinlichkeit des Auftretens des Zustands des in der mit x_t beschrifteten Zeile unter der Bedingung, dass die darüber stehenden vorgehenden Zuständen gelten, an.

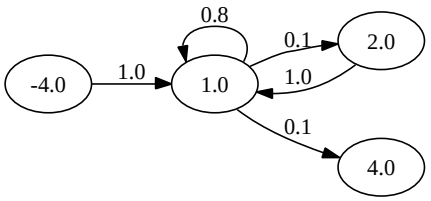


Abbildung 8.3: Markovkette erster Ordnung zur Erzeugung des Rhythmus entsprechend des ersten Beispiels in Abbildung 8.1

8.1.2 Mustergruppen

Definition 25 *Ein* Muster *ist ein Tupel* $p \in \{-2, -1{,}64\}^u$ *wobei* $u \leq n$ *gilt* ($u \in \mathbb{N}^{\geq 1}$, *n ist die Länge eines Individuums entsprechend Abschnitt 7.2).*

Die Bedeutung der Werte des Tupels entsprechen denen in Abschnitt 7. Die Tonhöhe 64 ist willkürlich festgelegt und repräsentiert das Anschlagen eines Tons. Der Melodieverlauf wird auch hier erst in einem weiteren (in Abschnitt 8.2 beschriebenen) Schritt erzeugt.

Definition 26 *Eine* Mustergruppe *ist eine Menge* $P = \{p_1, \ldots, p_k\}$ *von Mustern mit* $k \in \mathbb{N}^{\geq 1}$.

Eine Mustergruppe wird aus monophonen Beispiel-MIDI-Dateien erzeugt, von denen die ersten u Ereignisse Muster definieren. Ein Beispiel für die Erzeugung einer Mustergruppe aus aus MIDI-Dateien extrahierten Melodien ist in Tabelle 8.2 gegeben. An diesem erkennt man, dass grundsätzlich Muster der vorgegebenen Länge erzeugt werden: Ist die MIDI-Datei zu lang, wird der Bereich hinter den ersten u Ereignissen ignoriert (p_4). Ist die MIDI-Datei zu kurz, wird der letzte Ton hinreichend lang gehalten (p_2).

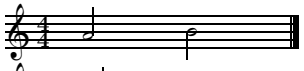
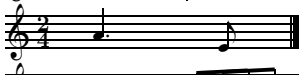
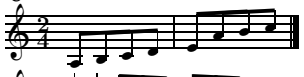
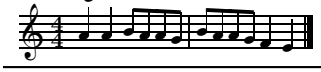
	$p_1 = (64, -1, -1, -1, -1, -1, -1, -1, 64, -1, -1, -1, -1, -1, -1, -1)$
	$p_2 = (64, -1, -1, -1, -1, -1, 64, -1, -1, -1, -1, -1, -1, -1, -1, -1)$
	$p_3 = (64, -1, 64, -1, 64, -1, 64, -1, 64, -1, 64, -1, 64, -1, 64, -1)$
	$p_4 = (64, -1, -1, -1, 64, -1, -1, -1, 64, -1, 64, -1, 64, -1, 64, -1)$

Tabelle 8.2: Erzeugung von Mustern der Länge 16 aus MIDI-Dateien (bei einer Auflösung von 0,25)

Erzeugen eines Rhythmus zur Initialisierung geschieht durch zufälliges Auswählen eines Musters und Übernahme in das zu erzeugende Individuum hinter das vorherige Muster bis jede Position des Individuums belegt ist. Dieses Verfahren ist in Algorithmus 8.1 dargestellt.

```
Mustergruppeninitialisierung(
        Individuum ind, Mustergruppe P) {
    pos = 0
    u = Länge der Muster in P
    n = Anzahl der Ereignisse in ind
    while pos ≤ n do
      Wähle zufällig Muster p ∈ P
      for i = 0 to u − 1 do
        m^ind_(pos+i) = m^p_i
      end for
    end while
}
```

Algorithmus 8.1: Initialisierung des Rhythmus durch Mustergruppen (m_i^{ind} gibt die i-te Position der Melodie des Individuums an, m_i^p die i-te Position des Musters)

Vorgefertigte Mustergruppen

Es stehen Mustergruppen mit je einem Muster zur direkten Auswahl in der Implementierung bereit, ohne dass diese durch MIDI-Dateien angelegt werden müssten. Diese erzeugen die auch aus der Verslehre bekannten Metren Jambus, Daktylus, Anapäst und Trochäus. Die Muster sind in Tabelle 8.3 für den $\frac{4}{4}$-Takt bei einer Auflösung von $r = 0{,}25$ angegeben. Für andere Taktarten weichen sie leicht ab.

Metrum	Muster
Trochäus	$p_t = (64, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 64, -1, -1, -1)$
Jambus	$p_j = (64, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 64, -1, -1, -1)$
Daktylus	$p_d = (64, -1, -1, -1, -1, -1, -1, -1, 64, -1, -1, -1, 64, -1, -1, -1)$
Anapäst	$p_a = (64, -1, -1, -1, -1, -1, -1, -1, 64, -1, -1, -1, 64, -1, -1, -1)$

Tabelle 8.3: Vorgefertigte Mustergruppen entsprechend der Grundmetren

Wie in der Tabelle zu sehen ist, sind die Muster bei Trochäus und Jambus sowie bei Daktylus und Anapäst jeweils identisch. Der Unterschied entsteht in einer Nachbearbeitung des entstehenden Individuums, die bei individuell angelegten Mustergruppen nicht stattfindet. Und zwar wird bei Jambus und Anapäst der erste Ton durch eine Pause ersetzt. Weiterhin wird bei Trochäus und Jambus der letzte Ton entfernt und stattdessen der davorliegende gehalten sowie bei Anapäst die letzten beiden Töne durch Halten des davorliegenden Tons ersetzt. Durch dieses Verfahren entsteht der Auftakt bei Jambus und Anapäst.

8.1.3 Zufällige Festlegung

Nach Angabe von minimaler und maximaler Ereignislänge sowie der Wahrscheinlichkeit, dass ein Ereignis eine Pause ist, kann der Rhythmus in diesen Grenzen gleichverteilt zufällig angelegt werden. Die Angabe findet auch hier in Werten, die Vielfache einer Viertelnote sind, welche mit dem Wert 1 korrespondiert, statt. Möglich sind Vielfache der durch die Auflösung des Individuums bestimmten kürzestmöglichen Ereignis.

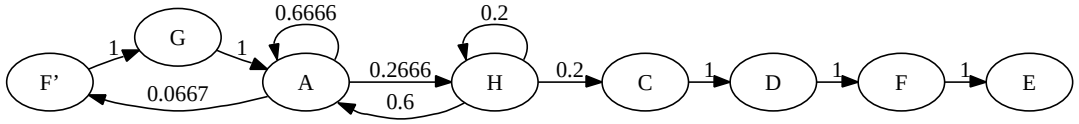
8.2 Modelle zur Erstellung der Melodieführung

8.2.1 Relative und absolute Markovketten

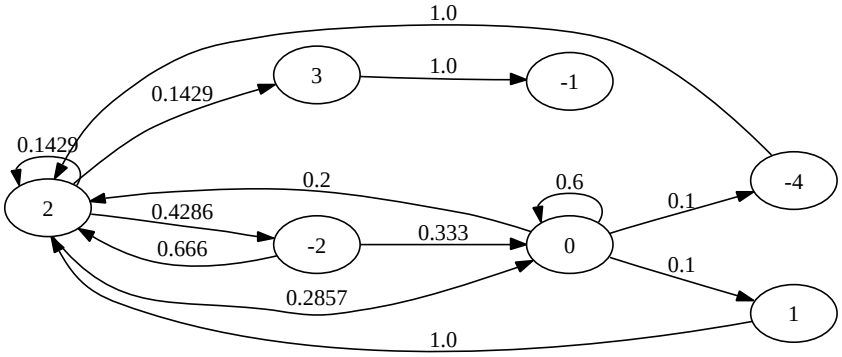
Markovketten zur Melodieerzeugung sind in zwei Varianten implementiert. Zum einen die relative Variante, welche die Folge von Tonhöhenänderungen betrachtet. Bei dieser spielen die eigentlichen Tonhöhen keine Rolle sondern nur die Schritte von einem Ton zum nächsten. Um die Komplexität einzuschränken werden nur Schritte bis zu einer Größe von 12 Halbtönen betrachtet. Zum anderen wird in der sogenannten absoluten Variante die Folge von tatsächlichen Tönen betrachtet. Relative Markovketten sind bis zu einer Ordnung von 5 verfügbar, absolute Markovketten können bis zu einer Ordnung von 3 genutzt werden.

Die Erstellung der Markovketten geschieht auf Basis von anzugebenden MIDI-Dateien, in denen die Tonhöhen betrachtet werden. Der Rhythmus wird ignoriert. Zum Einsatz kommt das in Abschnitt 2.1.4.1 vorgestellte Verfahren.

In Abbildung 8.4 finden sich die graphischen Darstellungen einer absoluten Markovkette erster Ordnung und einer relativen Markovkette erster Ordnung, welche jeweils aus den Melodien in Abbildung 8.1 erzeugt sind.



(a) Absolute Markovkette



(b) Relative Markovkette

Abbildung 8.4: Markovketten zur Melodieerzeugung, geschätzt aus den Beispielen in Abbildung 8.1

Da keine naheliegende initiale Wahrscheinlichkeitsverteilung zur Erzeugung der Melodieführung vorhanden ist, wird bei absoluten Markovketten neben dieser auch die Menge aller Töne x gespeichert, für die ein Ton y existiert, so dass $\mathcal{P}(x,y) > 0$ gilt (entsprechend der Notation in Abschnitt 2.1.1). Analog wird bei den relativen Markovketten verfahren, so dass eine Menge von Intervallen zur Verfügung steht. Aus diesen Mengen wird gleichverteilt zufällig der Beginn der Markovkette ausgewählt und im folgenden entsprechend Abschnitt 2.1.4.2 vorgegangen.

Die eigentliche Festlegung des Melodieverlaufs findet in der Art statt, dass jedes Ereignis in dem zu initialisierenden Individuum entsprechend der Markovkette belegt wird, welches weder eine Pause noch das Halten eines Tons angibt. Bei absoluten Markovketten gibt diese die festzulegende Tonhöhe direkt

an, bei relativen Markovketten ergibt sie sich aus der Addition des Intervalls auf die vorhergehende Tonhöhe. Daher muss die erste Tonhöhe vorgegeben werden, was durch gleichverteilt zufällige Auswahl aus den akkordeigenen Tönen an der Stelle des ersten Ereignis im Individuum geschieht. Hierdurch wird auf Basis des durch den vorhergehenden Schritt vorgegebenen Rhythmus eine Melodieführung festgelegt.

Wird ein Ton beziehungsweise ein Intervall erzeugt, bei dem kein Folgeton beziehungsweise kein Folgeschritt möglich ist (wie in Abbildung 8.4(a) bei Ton E und in Abbildung 8.4(b) bei Intervall -1) wird aus der vorgespeicherten Menge die Markovkette neu initialisiert.

Nach Abschluss der vollständigen Initialisierung der Melodieführung wird jeder Ton, der nicht zur an dieser Stelle vorgegebenen Skala gehört, auf die skalenzugehörige Tonhöhe mit minimalem Abstand gesetzt.

Verdeutlicht wird das beschriebene Initialisierungsverfahren in Algorithmus 8.2 und 8.3.

```
AbsoluteMarkovketteInit(
        Individuum ind, absolute Markovkette M) {
    Queue lastValues mit Kapazität Ordnung(M);
    lastValues = null;
    for i = 0 to Länge(ind) do
      if m_i ≥ 0 then
        m_i = nextTone(M,lastValues);
        pop(lastValues); lastValues ← m_i ;
      end if
    end for
    Korrigiere Skalenzugehörigkeit aller Töne in ind;
}

nextTone(
      abs.Markovkette M, Queue lastValues) {
    if lastValues = null then
      return Initialwert zu M;
    else
      if es ex. Folgewert zu lastValues aus M then
        return Folgewert zu lastValues aus M; // s. 2.1.4.2
      else
        return null;
      end if
    end if
}
```

Algorithmus 8.2: Initialisierung der Melodie mit absoluten Markovketten (m_i gibt den Wert an Position i im Individuum an)

8.2.2 Random Walk

Die Initialisierung mit Hilfe eines Random Walks arbeitet auf die selbe Weise wie die Initialisierung mit relativen Markovketten. Die Besonderheit ist, dass unabhängig von dem vorhergehenden Schritt die Wahrscheinlichkeit für den Folgeschritt zwischen den beiden Möglichkeiten -1 und $+1$ gleichverteilt ist.

Hier sind nun allerdings nicht Schritte auf der chromatischen Tonleiter gemeint, wie dies in Abschnitt 8.2.1 der Fall war. Stattdessen werden die Schritte direkt auf der jeweils an der zu initialisierenden Stelle

```
RelativeMarkovketteInit(
        Individuum ind, relative Markovkette M) {
    Queue lastValues mit Kapazität Ordnung(M);
    int lastTone = null;
    lastValues = null;
    for i = 0 to Länge(ind) do
      if m_i ≥ 0 then
        if lastTone = null then
          lastTone = Zufallston aus a_i
        end if
        m_i = nextTone(M,lastValues,lastTone);
        if m_i = null then
          m_i = Zufallston aus a_i
        end if
        pop(lastValues); lastValues ← m_i ;
        lastTone = m_i ;
      end if
    end for
    Korrigiere Skalenzugehörigkeit aller Töne in ind;
}

nextTone(
      rel.Markovkette M, Queue lastValues, int lastTone) {
    if lastValues = null then
      return (Initialwert zu M) +lastTone;
    else
      if es ex. Folgewert zu lastValues aus M then
        return (Folgewert zu lastValues aus M) +lastTone;
        // s. 2.1.4.2
      else
        return null;
      end if
    end if
}
```

Algorithmus 8.3: Initialisierung der Melodie mit relativen Markovketten (m_i gibt den Wert der Melodie und a_i den Akkord an Stelle i im Individuum an)

im Individuum geltenden Skala gemacht. Hierdurch wird die nach der Initialisierung folgende Korrektur der Töne auf Skalenzugehörigkeit unnötig.

8.2.3 Zufällige Festlegung

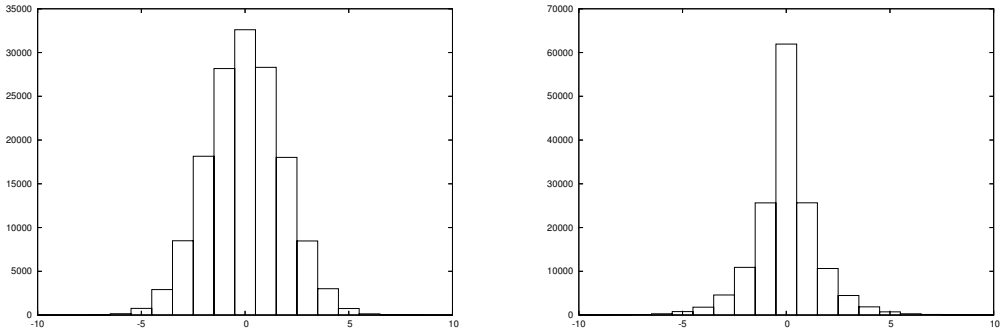
Dieses Verfahren betrachtet jeden möglichen Ton m_i der Melodie des Individuums unabhängig von den vorhergehenden Tönen. Hierzu wird der Grundton des Akkords an der selben Stelle (hier mit ${}_0a_i$ bezeichnet) genutzt. Auf diesen wird ein zufälliger Wert addiert, der nach Angabe der Standardabweichung aus der entsprechenden Normalverteilung durch Runden auf einen ganzzahligen Wert folgt.

Die Absicht, den ganzzahligen Zufallswert auf diese Weise zu bestimmen ist, bei einer verhältnismäßig kleinen Streuung keine zu hohe Dominanz des Zufallswerts 0 zu erhalten. Eine solche Verteilung ist in Abbildung 8.5(a) zu sehen. Die bilateral geometrische Verteilung in 8.5(b) wird nicht genutzt, stellt aber bei einer ähnlichen Streuung dar, dass der Wert 0 deutlich häufiger im Verhältnis zu den anderen Werten auftritt als es bei der gerundeten Normalverteilung der Fall ist.

Nach Abschluss der Festlegung der Melodieführung wird auch hier eine Korrektur der Skalenzugehörigkeit der Töne durchgeführt. Die Melodie des Individuums *ind* ergibt sich also durch Algorithmus 8.4.

```
NormalverteiltMelodieInit(
    Individuum ind,Standardabweichung σ){
    for i = 0 to Länge(ind) do
      if m_i ≥ 0 then
        m_i = {}_0a_i + [N_IR(0,σ)]
      end if
    end for
    Korrigiere Skalenzugehörigkeit aller Töne in ind;
}
```

Algorithmus 8.4: Festlegung der Melodieführung durch gerundete normalverteilte Zufallszahlen



(a) Normalverteilung nach Runden auf ganzzahlige Werte (die Standardabweichung beträgt 1,8)

(b) Bilateral geometrische Verteilung (siehe Rudolph, 1994)

Abbildung 8.5: Wahrscheinlichkeitsverteilungen zur Initialisierung der Melodieführung

8.3 Vergleich und Bewertung der Initialisierungsverfahren

Die in den vorstehenden Kapiteln beschriebenen Initialisierungsverfahren werden nun anhand von Beispielen erläutert. Hierbei sollen auch die Grenzen der Verfahren deutlich werden und damit eine

Hilfestellung bestehen, günstige Parameterbelegungen finden zu können.

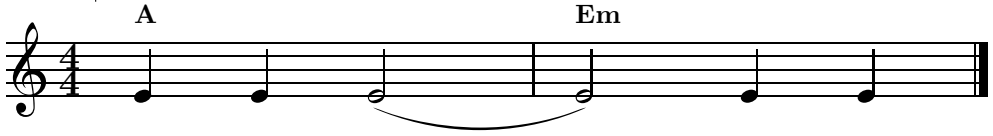
Zunächst sei der Ablauf einer Initialisierung eines Individuums mit zwei Takten ausführlich beschrieben. In den Tabellen ist jeweils in der ersten Zeile die Melodie, in der zweiten der Grundton des Akkords und in der dritten die Kennung der geltenden Skala-Akkord-Kombination angegeben.

1. Nach Anlegen eines leeren Individuums beinhaltet es zunächst nur den Befehl „Ton Halten“ in der Melodie. Die Akkorde müssen übergeben werden:

Takt 1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Takt 2	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1	−1
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

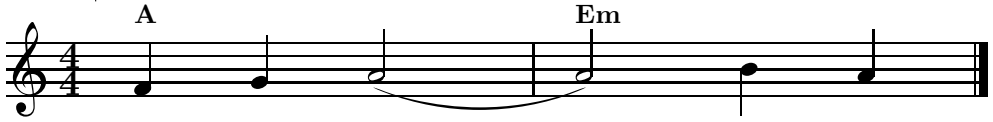
2. Nun wird zunächst der Rhythmus festgelegt. Hierzu wird an den Stellen der Melodie des Individuums, an denen ein Ton angeschlagen werden soll, der Ton *e* (mit MIDI-Tonhöhe 64), und an denen, an denen eine Pause beginnen soll eine −2 eingetragen. In diesem Beispiel wird die Markovkette aus Abbildung 8.2 genutzt.

Takt 1	64	−1	−1	−1	64	−1	−1	−1	64	−1	−1	−1	−1	−1	−1	−1
	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Takt 2	−1	−1	−1	−1	−1	−1	−1	64	−1	−1	−1	−1	64	−1	−1	−1
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5



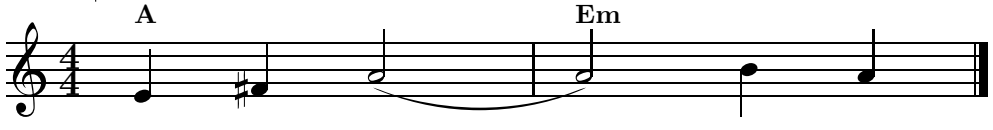
3. Nun folgt die Festlegung der Melodieführung auf den bereits bestehenden Punkten, an denen ein Ton angeschlagen wird. Hier wird die Markovkette aus Abbildung 8.4(a) genutzt.

Takt 1	65	−1	−1	−1	67	−1	−1	−1	69	−1	−1	−1	−1	−1	−1	−1
	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Takt 2	−1	−1	−1	−1	−1	−1	−1	71	−1	−1	−1	−1	69	−1	−1	−1
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5



4. Zu guter Letzt werden die Töne, welche nicht zur Skala gehören, auf die nächstliegende, skalenzugehörige Tonhöhe verschoben.

Takt 1	64	−1	−1	−1	66	−1	−1	−1	69	−1	−1	−1	−1	−1	−1	−1
	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A	A
	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Takt 2	−1	−1	−1	−1	−1	−1	−1	71	−1	−1	−1	−1	69	−1	−1	−1
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5



Im folgenden werden die Vor- und Nachteile der einzelnen Initialisierungsverfahren diskutiert. Auf die Darstellung in Tabellen wird auf Grund der besseren Übersichtlichkeit der Notendarstellung verzichtet.

8.3.1 Initialisierung des Rhythmus

8.3.1.1 Markovketten

Als Eingabe für die Erzeugung der Markovketten sei zunächst die Melodie in Abbildung 8.6 angenommen. In Abbildung 8.7 sind Beispiele für Rhythmen gezeigt, welche aus diesen erstellt sind. Hierbei soll deutlich werden, welchen Einfluss die Ordnung der Markovkette auf die erzeugten Rhythmen hat. So finden sich bei den Beispielen der ersten Ordnung (Abbildung 8.7(a)) drei deutlich unterschiedliche Individuen. Das erste zeigt ein Problem, welches grundlegend bei der Erzeugung von Rhythmen mittels Markovketten ist: Die Struktur des Taktes mit entsprechend betonten Schlägen findet keine besondere Beachtung. Es ist zu vermuten, dass dieses Problem mit steigender Ordnung und sinkender Komplexität der Markovkette abnimmt.

Die Beispiele zweiter Ordnung sind hinsichtlich der Ähnlichkeit zueinander nicht signifikant anders strukturiert als die der ersten Ordnung, jedoch fällt der identische Beginn der Individuen auf. Diese immer gleiche Passage wird hier mit steigender Ordnung länger.

Insgesamt ist zu erwarten, dass sich bei höherer Ordnung identische Passagen in den erzeugten Individuen häufen und diese länger sind als bei niedrigerer Ordnung. Daraus folgt aber auch eine steigende Wahrscheinlichkeit einer häufigeren Neuinitialisierung der Markovkette während der Erzeugung des Rhythmus, was zu weniger Ähnlichkeit mit den Beispielen zur Erstellung der Markovkette führt.

Bei diesen Beispielen ist diese Vermutung in der vierten Ordnung durch zwei identische Individuen sowie ein weiteres, ausgesprochen ähnliches, bestätigt.

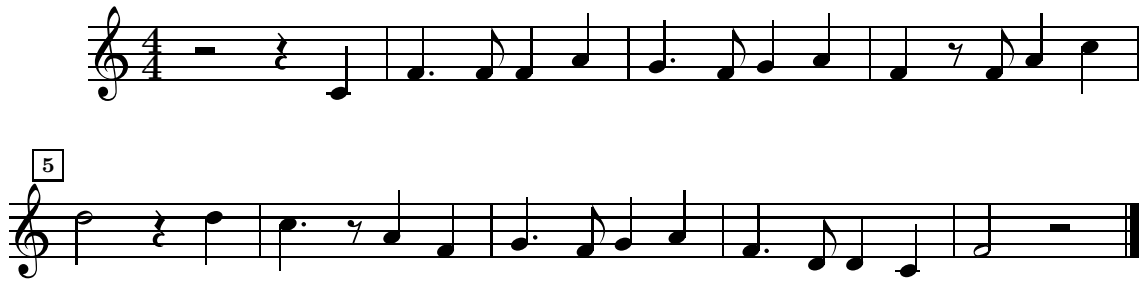


Abbildung 8.6: Melodie zur Erzeugung einer Markovkette (Auld Lang Syne (Schottisches Volkslied))

8.3.1.2 Mustergruppen

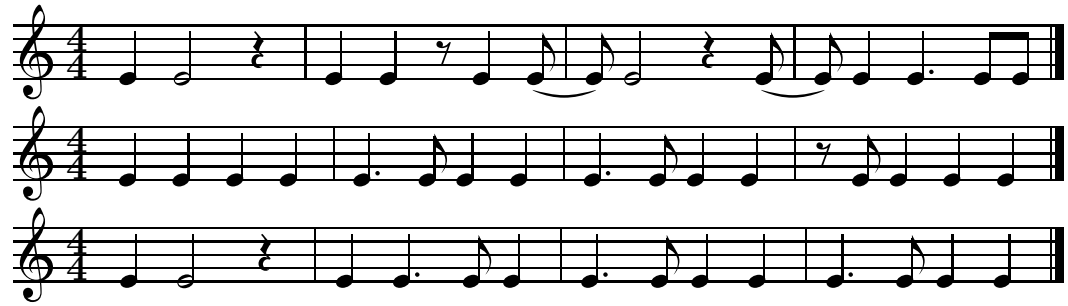
Das Problem, dass die Grenzen der Takte und die entsprechenden Betonungen nicht betrachtet werden, wie dies bei den Markovketten zur Erzeugung des Rhythmus der Fall ist, tritt bei der Verwendung von Mustergruppen nicht auf, wenn die Muster auf die Anzahl der Schläge eines Takts abgestimmt sind.

Für die Beispiele in Abbildung 8.8 ist die in Tabelle 8.2 gezeigte Gruppe genutzt worden. Durch die Länge jedes Musters von genau einem Takt werden die Taktgrenzen respektiert und auf den wichtigen betonten Schlägen beginnen Töne.

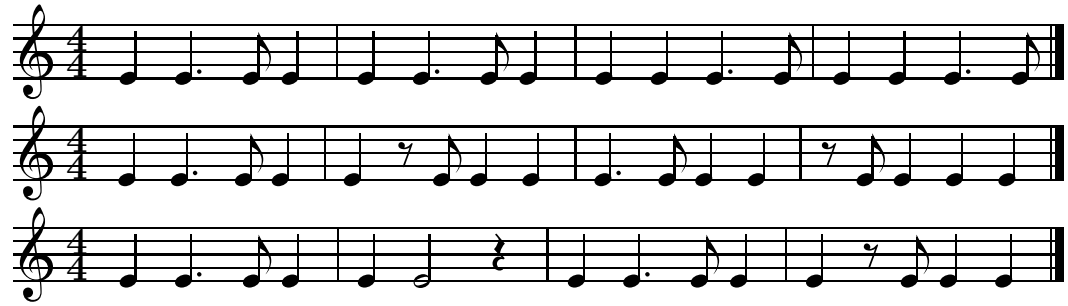
Festzustellen ist, dass auf diese Weise recht komplexe Rhythmen nach einer entsprechenden Vorgabe erzeugbar sind. Allerdings sind die generierten Strukturen sehr leicht vorhersehbar.

8.3.1.3 Zufällige Festlegung

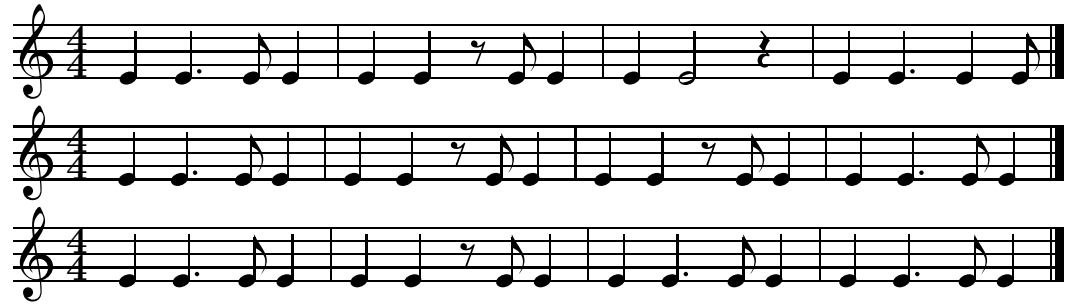
Den größten Freiheitsgrad bei der Festlegung des Rhythmus bietet die zufällige Festlegung. Bei kürzester zugelassener Ereignisdauer von 0,25 und maximaler Ereignisdauer von 4 mit einer Pausenwahrschein-



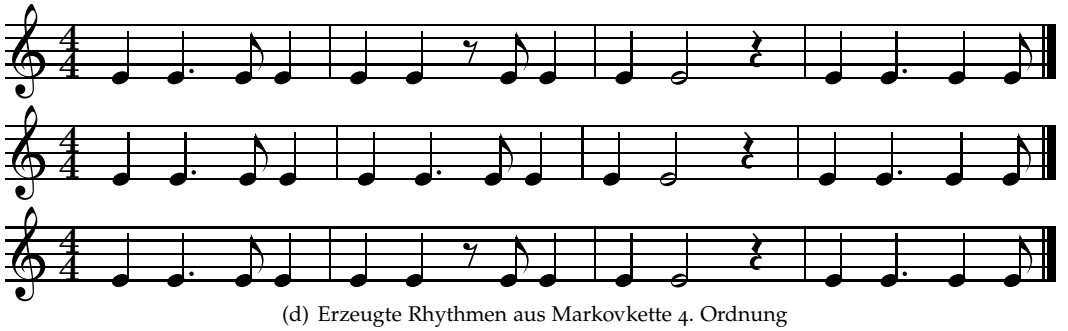
(a) Erzeugte Rhythmen aus Markovkette 1. Ordnung



(b) Erzeugte Rhythmen aus Markovkette 2. Ordnung



(c) Erzeugte Rhythmen aus Markovkette 3. Ordnung



(d) Erzeugte Rhythmen aus Markovkette 4. Ordnung

Abbildung 8.7: Beispiele für Rhythmen, welche aus Markovketten, die aus der Melodie in Abbildung 8.6 erzeugt wurden, erzeugt wurden.

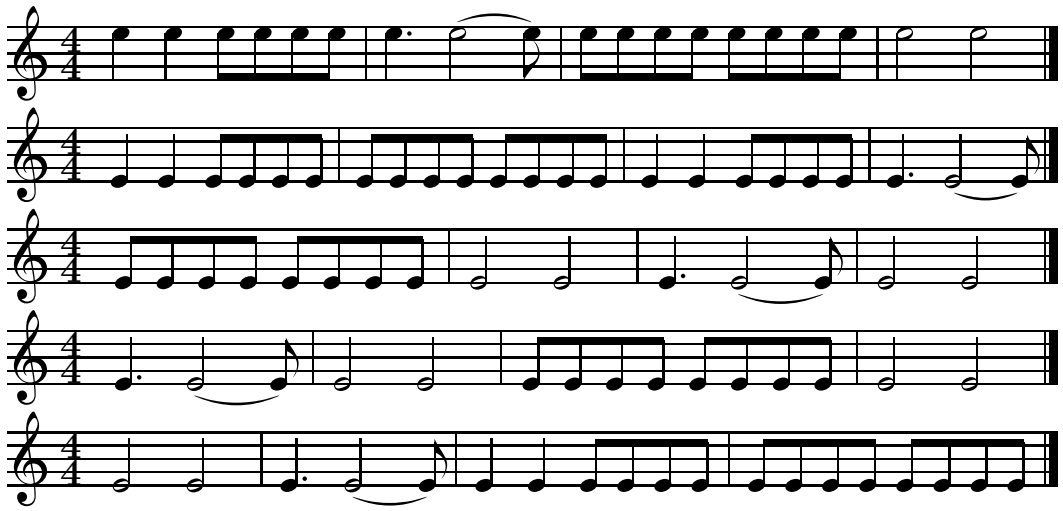


Abbildung 8.8: Beispiele für aus der Mustergruppe in Tabelle 8.2 erstellte Rhythmen

lichkeit von $\frac{1}{5}$ sind die in Abbildung 8.9 gezeigten Rhythmen entstanden.

Hierbei sind keinerlei Strukturen und Regelmäßigkeiten, insbesondere keine Beachtung der Betonungen zu beobachten. Daher scheint dieses Verfahren zunächst nicht sinnvoll einsetzbar zu sein. Insbesondere zur unkomplizierten Erzeugung von einfachen Rhythmen, welche nur aus Tönen einer Tonlänge bestehen und nur durch wenige Pausen unterbrochen sind, ist dieses Verfahren allerdings nützlich. Hierzu ist dann die minimale und die maximale Dauer eines Ereignisses auf den selben Wert zu setzen.

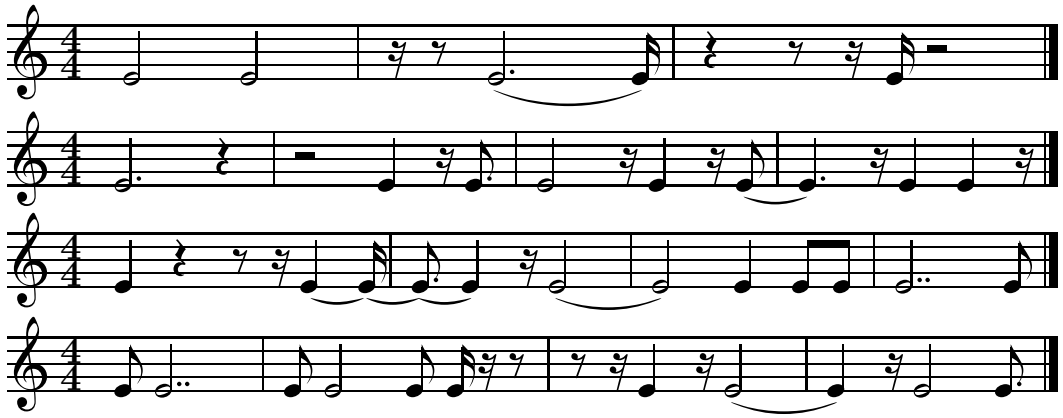


Abbildung 8.9: Beispiele für zufällig erzeugte Rhythmen mit minimaler Ereignisdauer von 0,25, maximaler Dauer von 4 und Pausenwahrscheinlichkeit von $\frac{1}{5}$

8.3.2 Erstellung der Melodieführung

In den folgenden Abschnitten werden Beispiele zu der Erstellung der Melodieführung gegeben. Da diese unabhängig von der Festlegung des Rhythmus ist, werden grundsätzlich $\frac{1}{8}$-Noten genutzt.

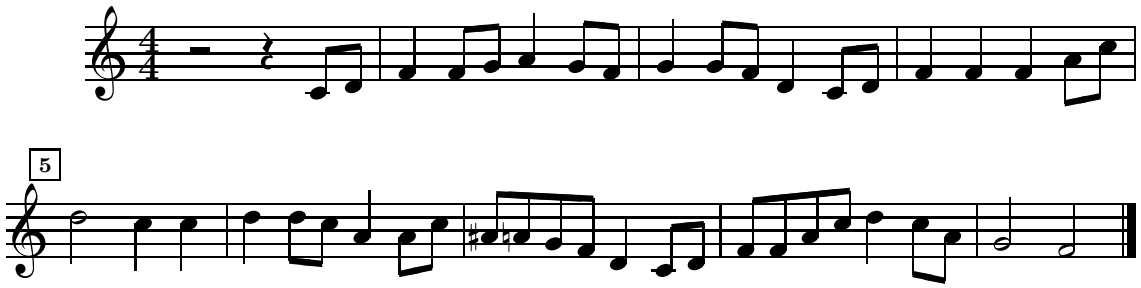


Abbildung 8.10: Melodie zur Erzeugung einer Markovkette (Loch Lomond (Schottisches Volkslied))

8.3.2.1 Relative und absolute Markovketten

Den hier beschriebenen Beispielen liegen Markovketten zugrunde, welche aus der Melodie in Abbildung 8.10 erstellt wurden. So sind in Abbildung 8.11 Melodieführungen zu sehen, welche aus relativen Markovketten unterschiedlicher Ordnung stammen. Hier ist es äußerst schwierig, den Ursprung der Markovkette in den erzeugten Melodien wiederzuerkennen. Einige Merkmale sind dennoch bezeichnet. So dominieren bei der Melodie des Stücks „Loch Lomond“ Sekund- und Terzschritte. Die Melodie bewegt sich (bis auf eine Ausnahme) nur in dem Tonumfang einer Oktave. Dieser Tonumfang wird bei Markovketten höherer Ordnung eher eingehalten, als bei Markovketten niedriger Ordnung, da sich die Melodie von „Loch Lomond“ als Wellenbewegung bezeichnen lässt, die von Markovketten höherer Ordnung besser abgebildet werden kann.

In Abbildung 8.12 sind Melodien dargestellt, wie sie durch eine absolute Markovkette entstanden sind. Hier wird selbstverständlich der Tonumfang der die Markovkette erzeugenden Melodie eingehalten, da (bis auf die Skalenkorrektur) nur Töne genutzt werden, die auch in dieser vorhanden sind.

Hier finden sich nun auch Teile von „Loch Lomond“ wieder, wobei die zusammenhängenden Stücke in Abbildung 8.12(b) länger sind als bei Abbildung 8.12(a).

8.3.2.2 Random Walk

Ein einfaches und doch teilweise überraschend überzeugende Ergebnisse lieferndes Verfahren ist, die Melodie mit Hilfe eines Random Walks erzeugen zu lassen. Beispiele hierzu sind in Abbildung 8.13 zu sehen.

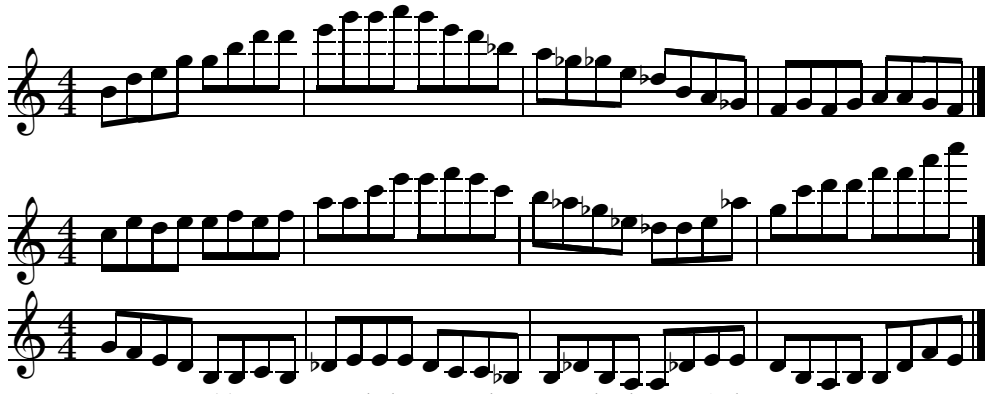
8.3.2.3 Zufällige Festlegung

Bei Erzeugung der Melodie durch die zufällige Festlegung wird der Grundton des jeweiligen Akkords umspielt. Dies erzeugt im Allgemeinen wenig eingängige Melodien, wie in den Beispielen in Abbildung 8.14 zu sehen ist. Zur Fehlersuche sowie der Analyse der Rhythmuserzeugung ist dieses Verfahren jedoch unabdingbar, da bei einer Standardabweichung von 0 und nur einem angegebenen Akkord nur ein Ton genutzt wird. Auf diese Weise sind die Beispiele der Rhythmuserzeugung entstanden.

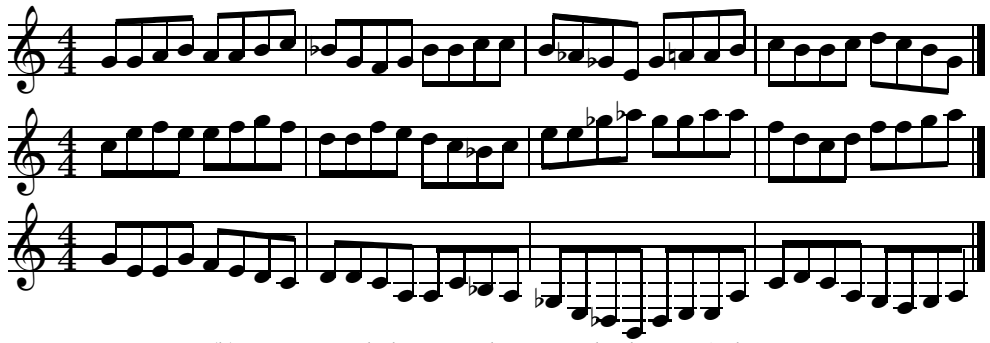
Eine Relevanz hat dieses Initialisierungsverfahren auch im Kontext des evolutionären Algorithmus, wie später zu sehen sein wird.

8.3.3 Zusammenfassung

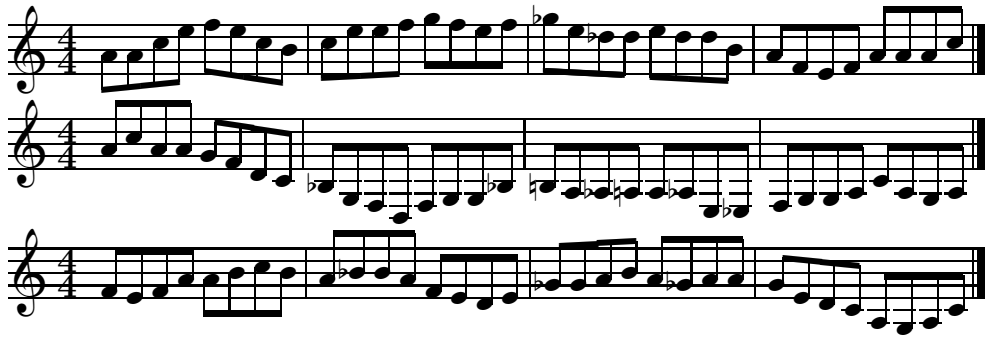
Es ist festzustellen, dass es einen Zielkonflikt zwischen der Erstellung von innovativen sowie strukturell sinnvollen Rhythmen und Melodien gibt. Eine Gegenüberstellung von innovativen und konservativen Verfahren findet sich in Tabelle 8.4.



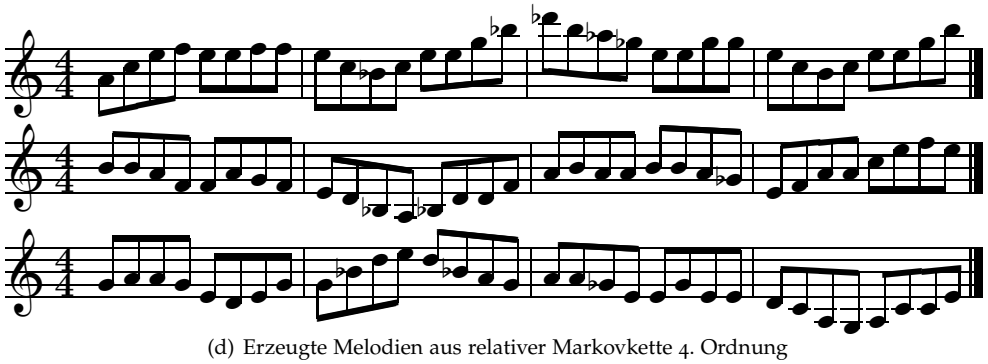
(a) Erzeugte Melodien aus relativer Markovkette 1. Ordnung



(b) Erzeugte Melodien aus relativer Markovkette 2. Ordnung

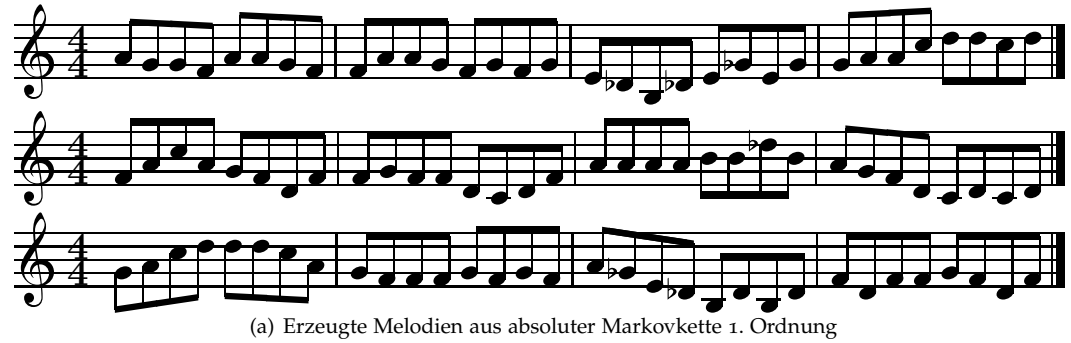


(c) Erzeugte Melodien aus relativer Markovkette 3. Ordnung

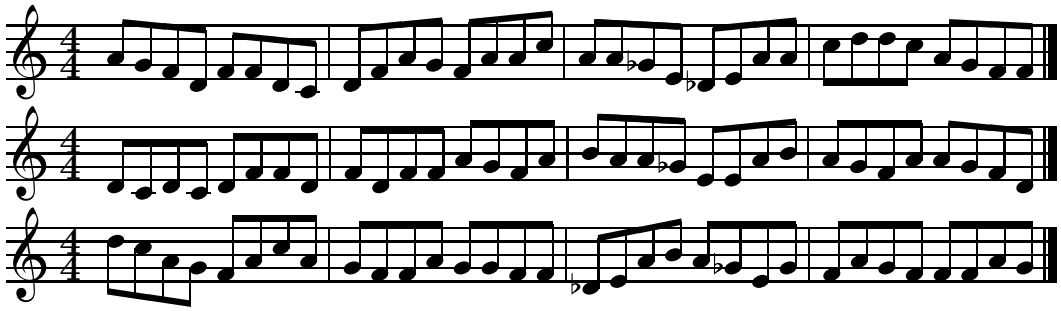


(d) Erzeugte Melodien aus relativer Markovkette 4. Ordnung

Abbildung 8.11: Beispiele für Melodien auf der Akkordfolge *Am, Dm, E, Am*, welche aus relativen Markovketten erzeugt wurden, die aus der Melodie in Abbildung 8.10 erzeugt wurden



(a) Erzeugte Melodien aus absoluter Markovkette 1. Ordnung



(b) Erzeugte Melodien aus absoluter Markovkette 2. Ordnung

Abbildung 8.12: Beispiele für Melodien auf der Akkordfolge *Am, Dm, E, Am*, welche aus absoluten Markovketten erzeugt wurden, die aus der Melodie in Abbildung 8.10 erzeugt wurden

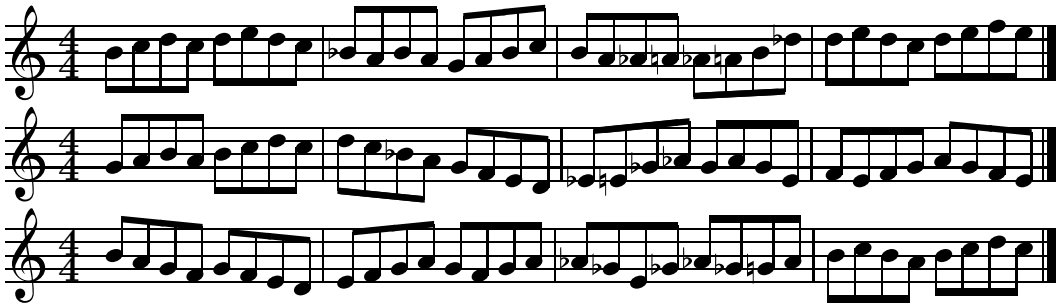


Abbildung 8.13: Melodieführungen auf der Akkordfolge *Am, Dm, E, Am*, die durch einen Random Walk erzeugt sind

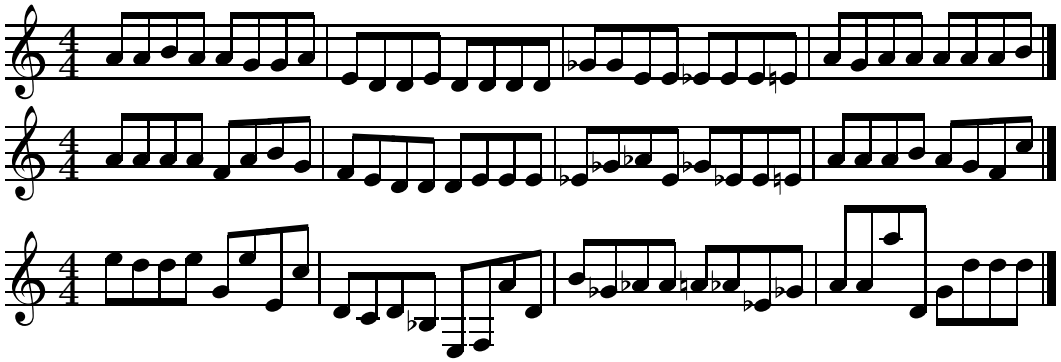


Abbildung 8.14: Melodieführungen auf der Akkordfolge *Am, Dm, E, Am*, die durch zufällige Festlegung erzeugt sind (von oben nach unten mit einer Standardabweichung von 0.5, 1 und 3)

Komplexe Markovketten sind durch eine hohe Anzahl von möglichen Zuständen sowie vielen Zustandsübergängen gekennzeichnet. Umso mehr Melodien mit unterschiedlichen Strukturen zur Erzeugung genutzt werden, desto komplexer wird die Markovkette. Eine einfache Markovkette entsteht zum Beispiel durch Nutzung von nur einer einfachen Melodie (wie zum Beispiel einem Kinderlied).

Je höher die Ordnung ist, desto genauer werden die Zusammenhänge innerhalb der die Kette erzeugenden Melodie modelliert und entsprechend abgebildet. Ist die Ordnung allerdings hoch und bestehen wenige Zustände besteht das Risiko, häufig neu initialisieren zu müssen, was kontraproduktiv ist.

Durch viele, kurze Muster in einer Mustergruppe steigt die Anzahl der Kombinationsmöglichkeiten in den erzeugten Melodien, durch wenige, lange Muster sinkt sie.

Es ist also darauf zu achten, die Initialisierung so zu wählen, dass ein gewisses Innovationsmaß vorhanden ist, allerdings sollten die entstehenden Melodien nicht zu stark von unserer Gewöhnung abweichen. Kombinationen unterschiedlicher Verfahren sind möglich und sinnvoll, wie im weiteren zu sehen sein wird.

	Innovativ	Konservativ
Markovketten	Komplex niedrige Ordnung	Einfach hohe Ordnung
Mustergruppen	viele Muster kurze Muster	wenige Muster lange Muster
Zufällige Festlegung	große Spanne zwischen max. und min. Dauer bzw. hohe Standardabw.	identischer Wert für max. und min. Dauer bzw. niedrige Standardabw.

Tabelle 8.4: Gegenüberstellung von innovativer und konservativer Initialisierung

Kapitel 9

Operatoren

Nachdem initiale Individuen für die Verwendung im evolutionären Algorithmus erstellt sind, kann auf diesen das Optimierverfahren gestartet werden. Elementar sind hierbei die Rekombinations- und Mutationsoperatoren, welche in Anlehnung an die Beschreibungen in Abschnitt 2.2.3.4 und 2.2.3.5 entworfen sind. Eine Besonderheit stellt allerdings dar, dass sie mit domänenspezifischem Wissen arbeiten, also musikalisch bedeutsam sind.

9.1 Rekombination

Die Durchführung der Rekombination erzeugt potentielle Nachfolgeindividuen. Zur Verfügung stehen hier zwei Verfahren, die *Ein-Punkt-Rekombination* und die *intermediäre Rekombination*. Je nach Parametereinstellung verhalten sich diese unterschiedlich:

Ist der evolutionäre Algorithmus ohne Crowding gewählt, erzeugen beide Verfahren ein Nachkommen, wobei vom Benutzer bestimmbar ist, ob jeweils aus beiden zufällig eines genutzt wird oder nur eines zum Einsatz kommen kann. Aus der aktuellen Population werden gleichverteilt zufällig so oft Eltern gewählt und jeweils ein Nachkommen generiert, bis die gewünschte Anzahl Kinder erreicht ist.

Ist die Verwendung von Crowding gewählt, steht nur die Ein-Punkt-Rekombination zur Verfügung. Diese erzeugt in diesem Fall je Elternpaar zwei Nachfolgeindividuen.

Grundsätzliche Annahmen sind die selbe Länge aller potentiellen Elternindividuen, wodurch auch die Kinder dieselbe Länge haben sowie die Basierung auf derselben Akkordfolge. Diese Bedingungen sind aufgrund der möglichen Parameterfestlegungen bei der Initialisierung sichergestellt.

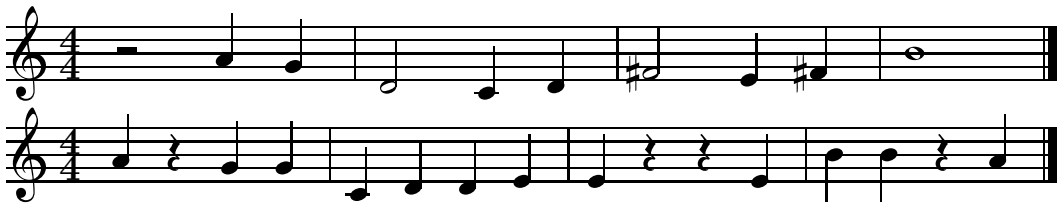


Abbildung 9.1: Eltern der Beispiele für Rekombination in Abbildung 9.2 und 9.3

Ein-Punkt-Rekombination

Die Ein-Punkt-Rekombination folgt genau dem in Abbildung 2.3(a) gezeigten Schema. Der Schnittpunkt zwischen den beiden Individuen wird gleichverteilt gewählt. Abbildung 9.2 zeigt ein entsprechendes Beispiel.

Intermediäre Rekombination

Intermediäre Rekombination kombiniert die beiden Elternteile durch Bestimmung des Mittelwerts der Tonhöhen an den entsprechenden Positionen des Melodie-Tupels, wie das in Abschnitt 2.2.3.4 beschrieben ist. Da es allerdings keinen Sinn macht, den Mittelwert zwischen einer Pause (mit dem repräsentierenden

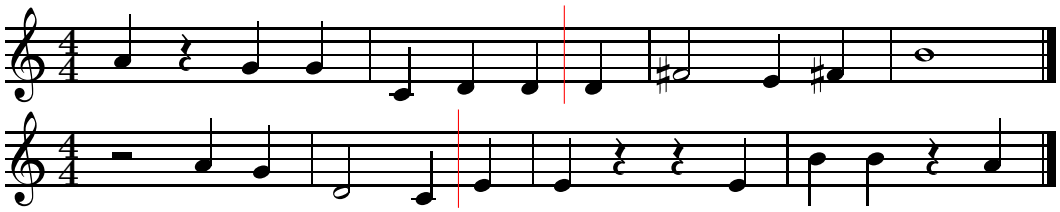


Abbildung 9.2: Beispiel für die Anwendung von Ein-Punkt-Rekombination (Der Schnittpunkt liegt zwischen dem 3. und 4. Schlag des zweiten Takts.)

Wert -2) und einer Tonhöhe an der entsprechenden Position im Nachfolgeindividuum zu bilden, gelten einige Besonderheiten.

Zunächst wird zufällig der Elter gewählt, von dem der Rhythmus identisch auf das Kind übertragen wird. Im folgenden wird der Durchschnitt von jedem der Töne dieses Elters mit dem zu diesem Zeitpunkt klingenden oder angeschlagenen Ton des anderen Elters gebildet. Hat der andere Elter zu diesem Zeitpunkt eine Pause wird die Tonhöhe des rhythmusgebenden Elters übernommen. Den Abschluss bildet eine Korrektur der Skalenzugehörigkeit aller Töne.

Ein Beispiel ist in Abbildung 9.3 zu sehen.

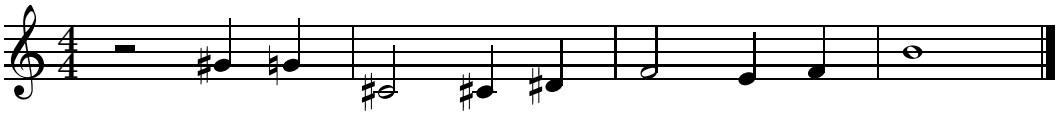


Abbildung 9.3: Beispiel für die Anwendung von intermediärer Rekombination

9.2 Mutation

Bevor die Selektion auf den durch die Rekombination erzeugten Individuen und deren Eltern stattfindet werden die Kinder mutiert. In diesem Abschnitt werden die zur Verfügung stehenden Mutationsoperatoren vorgestellt, aus denen gleichverteilt zufällig das jeweils eingesetzte Verfahren ausgewählt wird.

Sie sind in Anlehnung an die Methoden von Biles (1994, siehe Abschnitt 5.2) entworfen und lassen sich unterteilen in *Veränderung der Tonhöhe* (Ein-Punkt-Mutation, Abschnittstransposition, Abschnittsinversion), *strukturelle Veränderung* eines Abschnitts (Sortieren, Spiegeln, Rotieren) und *Veränderung des Rhythmus* (Verschieben, Teilen, Verbinden von Tönen), wobei sich diese Bereiche teilweise etwas überschneiden, da zum Beispiel die Rotation Einfluss auf den Verlauf der Tonhöhe und den Rhythmus hat.

In Abbildung 9.4 ist ein kurzes Melodiestück zu sehen, an dem die jeweiligen Operatoren veranschaulicht werden. Hierbei wird grundsätzlich nur der zweite Takt mutiert, der erste entspricht dem Original, so dass ein Vergleich leicht möglich ist.

Nach jeder Mutation wird im übrigen eine Korrektur der Skalenzugehörigkeit der Tonhöhen durchgeführt.

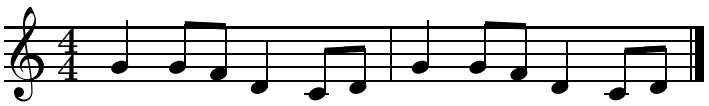


Abbildung 9.4: Melodie zur Veranschaulichung der verschiedenen Mutationsoperatoren

Einige der Mutationsoperatoren verändern nur einen Bereich des Individuums. Die Festlegung dieser Abschnittsgrenzen findet statt, indem die erste Grenze aus allen möglichen Positionen gleichverteilt zufällig ausgewählt wird. Die zweite Grenze wird als Abstand zur ersten zufällig entsprechend der bilateralen geometrischen Verteilung

$$G = \left\lfloor \frac{\log(1-U)}{\log(1-p)} \right\rfloor$$

bestimmt. G ist hierbei die zu bestimmende Zufallsvariable, U eine gleichverteilte Zufallszahl. Weiterhin wird p bestimmt durch

$$p = 1 - \frac{S}{(1+S^2)^{\frac{1}{2}} + 1}\,.$$

Für genauere Erläuterung sei hier auf Rudolph (1994) verwiesen.

Der Parameter S, der die Streuung der Verteilung beeinflusst, wird in Abhängigkeit von der Länge des zu mutierenden Individuums gesetzt:

$$S = \frac{n}{q}$$

Durch den Parameter q hat der Benutzer Einfluss auf die Länge des zu mutierenden Bereichs, allerdings immer in Abhängigkeit von der Länge des Individuums. Empfohlen wird $5 \leq q < 8$ (je kleiner q, desto länger der Mutationsabschnitt). Eine Verteilung für die Länge des Individuums in Abbildung 9.4 bei einer Auflösung von 0,25 (woraus $n = 32$ folgt) mit $q = 7$ findet sich in Abbildung 9.5.

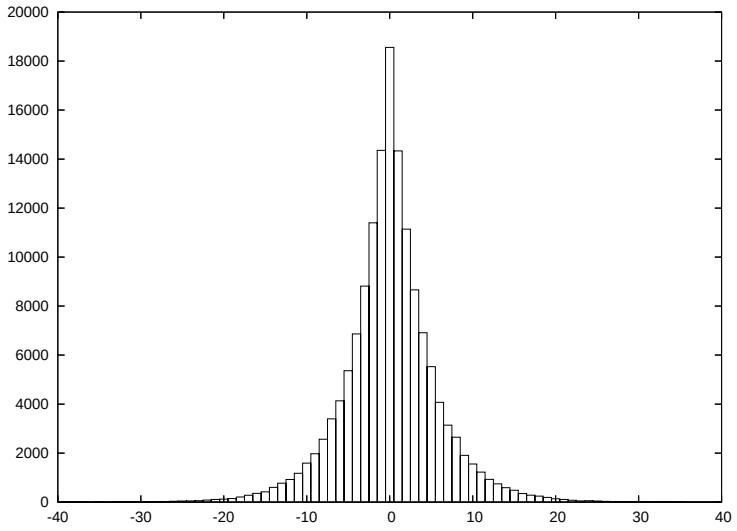


Abbildung 9.5: Bilateral geometrische Verteilung zur Abschnittsbestimmung

9.2.1 Veränderung der Tonhöhe

Ein-Punkt-Mutation

Die Ein-Punkt-Mutation bildet das in 2.2.3.5 beschriebene Verfahren nach. So wird jede Position des Individuums, an der ein Ton angeschlagen wird, mit einer vom Benutzer spezifizierten Wahrscheinlichkeit mutiert, dass heißt, es wird eine normalverteilte Zufallszahl erzeugt und auf einen ganzzahligen Wert gerundet. Dieser beschreibt je nach Vorzeichen und Betrag Richtung und Schrittzahl auf der an dieser Position geltenden Skala und damit die neue Tonhöhe im Verhältnis zur ursprünglichen.

Auf diese Weise ist die Mutation in Abbildung 9.6 entstanden.

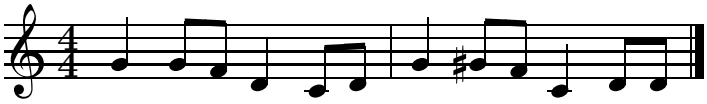


Abbildung 9.6: Beispiel für Punktmutation

Abschnittstransposition

Bei diesem Mutationsverfahren werden alle Töne, die innerhalb eines Abschnitts liegen, um den gleichen Wert transponiert. Die Schrittweite wird durch eine normalverteilte Zufallszahl bestimmt. Die Melodie in Abbildung 9.7 ist in dem Bereich des gesamten zweiten Takts um 3 Schritte nach oben, also den Wert $+3$ transponiert.

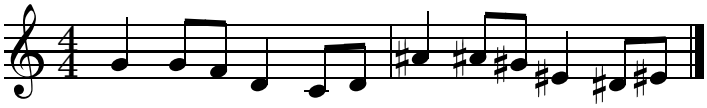


Abbildung 9.7: Beispiel für Abschnittstransposition

Abschnittsinversion

Die Tonhöhen aller im Abschnitt angeschlagenen Töne werden invertiert. Dazu wird der höchste Ton $t_{\max}$ und der niedrigste Ton $t_{\min}$ bestimmt und die Tonhöhe jedes angeschlagenen Tons t_i im Abschnitt verändert durch $t_i = (t_{\max} - t_i) + t_{\min}$.

Der zweite Takt in Abbildung 9.8 ist entsprechend verändert worden.

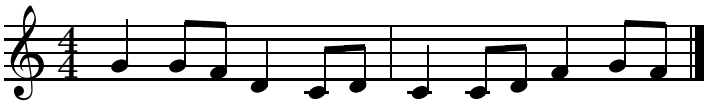


Abbildung 9.8: Beispiel für Abschnittsinversion

9.2.2 Strukturelle Veränderung

Abschnittssortierung nach Tonhöhe

Bei diesem Mutationsverfahren werden die Tonhöhen in einem Bereich sortiert, was absteigend oder aufsteigend geschehen kann. Der Rhythmus wird nicht verändert. In Abbildung 9.9(a) ist eine aufsteigende, in Abbildung 9.9(b) eine absteigende Sortierung des jeweils zweiten Takts zu sehen.

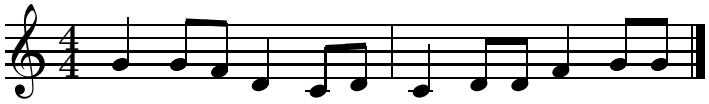
Dieses Verfahren arbeitet zwar auf einem Abschnitt, bestimmt diesen jedoch nicht durch oben beschriebenes Verfahren sondern setzt die erste Grenze gleichverteilt zufällig auf den möglichen Positionen des Individuums, die zweite jedoch nicht vor oder nach dem an der ersten Grenze geltenden Akkord. Hierdurch wird nur innerhalb einer Skala sortiert.

Abschnittsweises Spiegeln der Reihenfolge der Tonhöhen

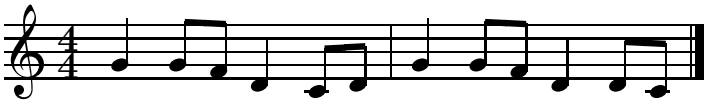
Ähnlich wie bei der Mutation durch Sortieren werden hierbei nur die Tonhöhen verändert. Die Reihenfolge dieser wird im gewählten Abschnitt umgedreht. Ein Beispiel ist in Abbildung 9.10 zu sehen.

Abschnittsweises Spiegeln der Reihenfolge der Tonlängen

Im Gegensatz zum vorherigen Verfahren werden hierbei die Tonlängen gespiegelt und die Tonhöhen nicht verändert. Ein Beispiel zeigt Abbildung 9.11.



(a) Aufsteigende Sortierung



(b) Absteigende Sortierung

Abbildung 9.9: Beispiel für Mutation durch Sortierung von Tonhöhen

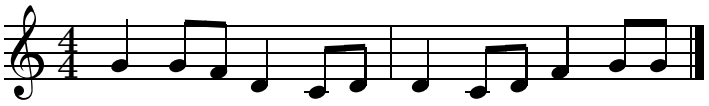


Abbildung 9.10: Beispiel für Mutation durch Spiegeln der Tonhöhen

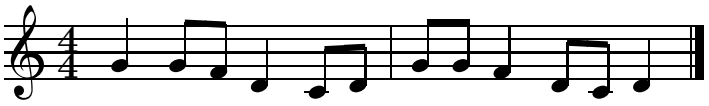


Abbildung 9.11: Beispiel für Mutation durch Spiegeln der Tonlängen

Abschnittsweise Rotation der Noten

Dieses Verfahren erhält im Gegensatz zu den bisherigen Mutationsmethoden in diesem Abschnitt die Kombination zwischen Tonhöhe und -länge. In dem entsprechenden Abschnitt wird jeder Ton zyklisch um einen gleichverteilt zufälligen Wert zwischen 0 und der Anzahl der Töne in dem Bereich nach rechts verschoben. Ein Beispiel, bei dem um den Wert 3 rotiert wurde, ist in Abbildung 9.12 zu sehen.

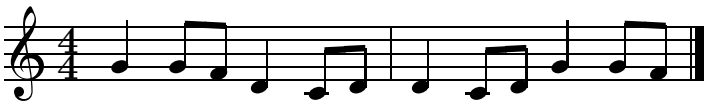


Abbildung 9.12: Beispiel für Mutation durch Rotation

9.2.3 Veränderung des Rhythmus

Verschieben des Beginns einer Note

Dieser Operator verschiebt den Beginn eines Ereignisses nach vorne, so dass der vorherige Ton (beziehungsweise die Pause) kürzer, und das verschobene Ereignis länger wird, oder nach hinten, so dass das verschobene Ereignis kürzer und der vorherige Ton (beziehungsweise die Pause) länger wird.

Ähnlich wie bei der Ein-Punkt-Mutation wird dieses Verfahren mit einer anzugebenden Wahrscheinlichkeit auf jeden Ton angewendet. Die Anzahl der zu verschiebenden Positionen wird normalverteilt zufällig mit einer Standardabweichung von 2 bestimmt. Grenzen setzen hierbei die davor beziehungsweise dahinterliegenden Startpositionen von Ereignissen.

Ein Beispiel, bei dem der zweite Ton um zwei Positionen nach vorne verschoben und der fünfte Ton um eine Position nach hinten verschoben wurde (jeweils in Takt 2, bei einer angenommenen Auflösung von 0,25) ist in Abbildung 9.13 zu sehen.

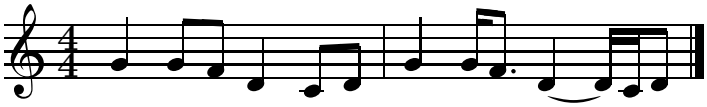


Abbildung 9.13: Beispiel für Mutation durch Verschieben von Tönen

Ersetzung einer Note durch kürzere Noten

Diese Methode ersetzt jeden Ton mit einer anzugebenden Mutationswahrscheinlichkeit durch zwei Töne mit jeweils der halben Länge des Ursprungstons. Der erste der beiden neuen Töne erhält die selbe Tonhöhe wie der Ursprungston. Der zweite wird auf der Basis dieser Tonhöhe durch Addition einer auf einen ganzzahligen Wert gerundeten normalverteilten Zufallszahl erhöht oder erniedrigt. Die Noten werden in der Reihenfolge vom Anfang hin zum Ende betrachtet.

Ein Beispiel, bei dem der zweite und dritte Ton des zweiten Takts durch jeweils zwei neue Töne substituiert ist, ist in Abbildung 9.14 zu sehen.

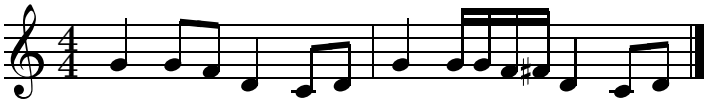


Abbildung 9.14: Beispiel für Mutation durch Ersetzen von Tönen

Verbinden von Noten

Diese Mutation kombiniert jeden Ton mit einer anzugebenden Wahrscheinlichkeit mit seinem Nachfolger. Hierbei werden die Noten in der Reihenfolge vom Anfang hin zum Ende betrachtet, wodurch nicht mehr als zwei hintereinanderliegende Töne zusammengefasst werden. Die Tonhöhe der neuen, längeren Note ergibt sich aus der ersten der beiden kombinierten.

Dieses Mutationsverfahren kommt nur zum Einsatz, wenn mehr als vier Noten in der Melodie vorhanden sind. Diese Einschränkung ist zur späteren Bewertung des Individuums notwendig. Ein Beispiel, in dem der erste und fünfte Ton des zweiten Takts mit seinem Nachfolger kombiniert worden ist, zeigt Abbildung 9.15.

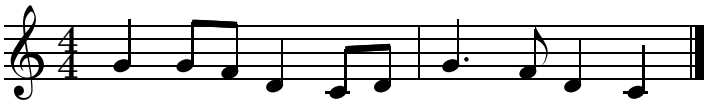


Abbildung 9.15: Beispiel für Mutation durch Verbinden von Tönen

9.3 Vergleich und Bewertung der Operatoren

Die Mutationsoperatoren sind mit der Absicht entworfen, jedes musikalisch sinnvolle Individuum im Suchraum erzeugen zu können. Diese Formulierung birgt jedoch Risiken, da die musikalische Sinnhaftigkeit nicht trivial erkennbar ist. Daher ist die einzige strenge Einschränkung, nur Töne der jeweils geltenden Skala zu erlauben. Eine umso größere regulierende Rolle kommt den Bewertungs- und Selektionsverfahren zu.

Elementare Funktionen haben die Ein-Punkt-Mutation zur Veränderung der Tonhöhe sowie die drei Operationen Verschieben, Verbinden und Ersetzen von Tönen, da damit bereits jeder Punkt im Suchraum erreichbar ist. Die weiteren transponierenden, sortierenden, rotierenden und spiegelnden Verfahren stellen Abkürzungen auf der Suche nach den dadurch erreichten Individuen dar.

Wie auch bei Initialisierungsverfahren besteht ein Zielkonflikt zwischen einer das Individuum stark verändernden, also mit einer bemerkenswerten Innovation verbundenen Mutation und einer eher konservativen Mutation. Die melodieführungsverändernden Verfahren erscheinen subjektiv weniger dramatische Eingriffe vorzunehmen als die nur rhythmusverändernden Verfahren. In Abbildung 9.16 sind einige Mutationen gezeigt, wie sie bei der Evolution mit nur einem Individuum und nicht genutzten rhythmusverändernden Verfahren entstanden sind. Es sind insgesamt 15 Mutationversuche zufällig ausgeführt worden, allerdings sind einige zu dem Ursprungsindividuum identische Melodien entstanden, welche nicht abgebildet sind.

Im Vergleich dazu sind Mutationen durch die rhythmusverändernden Verfahren durchgeführt worden, die in Abbildung 9.17 zu sehen sind. Umfangreichere Tests bestätigen die These, das Verschieben und Verbinden von Tönen zu wenig gefälligen Rhythmen führen kann, aus Platzgründen muss leider auf die ausführliche Darstellung verzichtet werden.

Genauer ist der Umgang mit diesem Problem in Kapitel 11 erklärt. Dort finden sich auch Erläuterungen zu den Rekombinationsoperatoren, da diese im Kontext der Evolution betrachtet werden müssen.

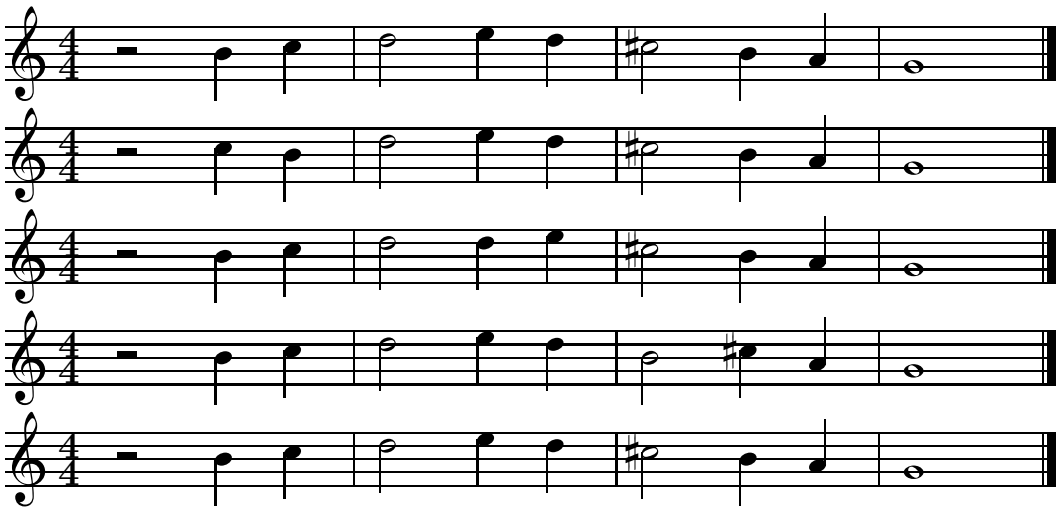


Abbildung 9.16: Mutationen eines Individuums (des obersten) mit einer Punktmutationswahrscheinlichkeit von 0,3 und einem Abschnittsparameter $q = 3$

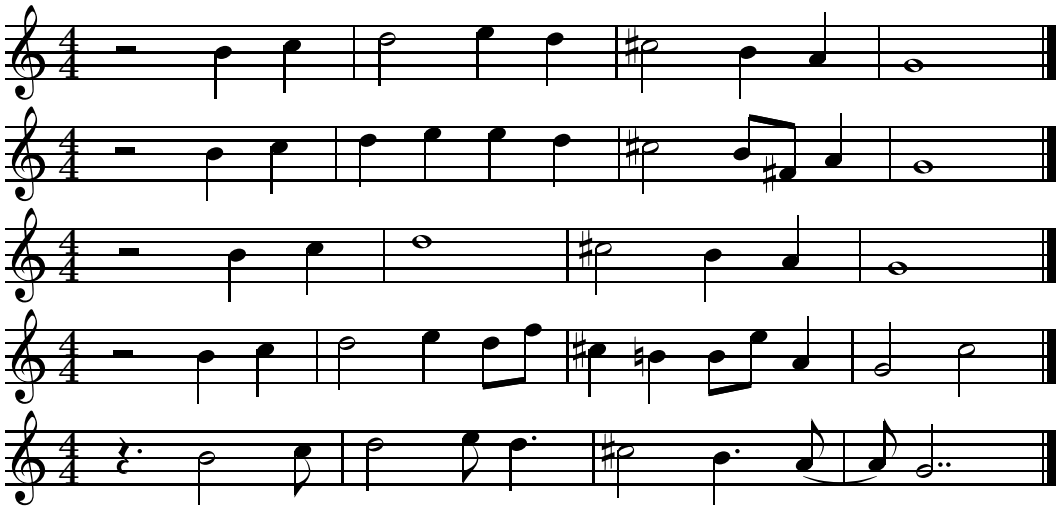


Abbildung 9.17: Mutationen eines Individuums (des obersten) mit Mutationswahrscheinlichkeiten von 0,3 für die rhythmusverändernden Verfahren

Kapitel 10

Bewertung und Selektion

Durch die Anwendung von Initialisierung sowie Rekombination und Mutation sind Individuen entstanden, aus denen nach einem gewissen Maß die günstigsten, also solche mit einer hohen Fitness, ausgewählt werden. Dieses Kapitel setzt sich mit dieser Bewertung und Selektion auseinander.

Neben der Möglichkeit, die Individuen interaktiv zu bewerten, stellt eine Merkmalsextraktion die Grundlage für die automatische Bewertung dar. Nachdem diese erläutert ist, wird auf verschiedene Verfahren eingegangen, aus diesen einen Fitnesswert zu berechnen. Im Anschluss findet sich ein Vergleich und eine Beurteilung der Bewertungsverfahren.

Zum Abschluss der Bildung der Folgepopulation von Individuen findet die Selektion statt, welche am Ende dieses Kapitels besprochen wird.

10.1 Merkmalsextraktion

Dieser Abschnitt beschreibt die implementierten Merkmale, welche aus den Individuen extrahiert werden, wobei jedes einen Wert im Intervall $[0;1]$ angibt. Weitestgehend wird hier den Vorschlägen von Towsey u. a. (2000) und von Wiggins und Papadopoulos (1998) entsprochen, welche in Abschnitt 5.6 und 5.4 eingeführt sind. Hier findet sich nun ein Überblick mit einem Schwerpunkt auf die implementationsspezifischen Besonderheiten und Erweiterungen. Einige Merkmale sind in der aktuellen Implementierung ohne Funktion (es wird jeweils darauf hingewiesen), was sich allerdings durch geringe zukünftige Änderungen am System ändern kann und die Nennung rechtfertigt.

In Klammern ist die Bezeichnung in der Implementierung angegeben. Bemerkungen zu der möglichen Wahrnehmung eines Hörers bezüglich bestimmter Werte des Merkmals sind subjektiv. Merkmale, welche von Towsey u. a. (2000) übernommen sind, sind mit „[1]“ markiert. Solche, die an Wiggins und Papadopoulos (1998) angelehnt sind, tragen eine „[2]“.

10.1.1 Tonhöhenmerkmale

Tonhöhenvielfalt (Pitch Variety)[1]

Dieses Merkmal beschreibt das Verhältnis der Anzahl unterschiedlicher Tonhöhen zu der Anzahl aller Töne im Individuum.

$$\text{feature}_1(I) = \frac{\text{Anzahl unterschiedliche Tonhöhen}}{\text{Anzahl Töne}}$$

Tonhöhenumfang (Pitch Range)[1]

Dies beschreibt das Verhältnis des Tonumfangs des Individuums im Verhältnis zum angenommenen maximalen Tonumfang von 24.

$$\text{feature}_2(I) = \max\left(1, \frac{\text{höchste Tonhöhe} - \text{niedrigste Tonhöhe}}{24}\right)$$

10.1.2 Tonale Merkmale

Schlüsselzentrierung (Key Centering)

Die Schlüsselzentrierung betrachtet das Verhältnis der Anzahl aller Töne, welche auf der Prime oder der Quinte liegen, im Verhältnis zu der Anzahl aller Töne. Dies entspricht nicht dem Vorschlag von Towsey u. a. (2000), der Dominante und Tonika, also im allgemeinen Quarte und Quinte statt Prime und Quinte als zentral annimmt. Hier ist also eine andere Interpretation dieses Merkmals implementiert. Als Prime wird der erste Schritt des Akkords betrachtet, als Quinte der dritte Schritt.

Es ist zu erwarten, dass Melodien mit einer geringen Schlüsselzentrierung eher nicht harmonisch wirken.

$$\text{feature}_3(I) = \frac{\text{Anzahl Töne die Prime oder Quinte des Akkords sind}}{\text{Anzahl Töne}}$$

Schlüsselzentrierung bezüglich Zeiteinheiten (Key Centering (Quanta))

Hierbei wird nicht die Anzahl der Töne betrachtet, wie im vorhergehenden Merkmal, sondern die Anzahl der Zeiteinheiten, welche hier als die Anzahl der Position des Individuums angesehen werden können, in denen entsprechende Töne der Prime oder Quinte des aktuell geltenden Akkords klingen.

$$\text{feature}_4(I) = \frac{\text{Anzahl Zeiteinheiten m. Tönen d. Prime oder Quinte d. Akkords sind}}{\text{Anzahl aller Zeiteinheiten}}$$

Skalenfremde Töne (Non Scale Notes)

Dieses Merkmal stellt das Verhältnis der Anzahl aller nicht skalenzugehöriger Töne zu der Anzahl aller Töne dar. Zu häufig auftretende skalenfremde Töne wirken stark dissonant.

Dieses Merkmal ist in der aktuellen Implementierung wirkungslos, da nur skaleneigene Töne zugelassen sind.

$$\text{feature}_5(I) = \frac{\text{Anzahl skalenfremder Töne}}{\text{Anzahl Töne}}$$

Skalenfremde Töne bezüglich Zeiteinheiten (Non Scale Notes (Quanta))[1]

Dieser Merkmal betrachtet im Gegensatz zum vorherigen die Anzahl der Zeiteinheiten mit entsprechenden Tönen.

$$\text{feature}_6(I) = \frac{\text{Anzahl Zeiteinheiten, in denen ein Ton klingt, der skalenfremd ist}}{\text{Anzahl aller Zeiteinheiten}}$$

Dissonante Intervalle (Dissonant Intervals)[1]

Hier wird das Verhältnis zwischen der Summe der Dissonanzwerte aller vorkommenden Intervalle entsprechend Tabelle 10.1 zur Anzahl aller Intervalle betrachtet.

Zu wenig dissonante Intervalle könnten eine Melodie langweilig klingen lassen, zu viele erzeugen möglicherweise eine unangenehme Reibung.

Schritte auf der Skala	Dissonanzwert
$0, 1, 2, 3, 4, 5, 7, 8, 9, 12$	0
10	0,5
$6, 11, \geq 13$	1

Tabelle 10.1: Dissonanzwerte von Intervallen zur Merkmalsextraktion

$$\text{feature}_7(I) = \frac{\text{Summe aller Dissonanzwerte}}{\text{Anzahl Intervalle} = \text{Anzahl Noten} - 1}$$

Harmonie (Harmonicity)

Mit diesem Merkmal wird die Anzahl aller akkordeigenen Töne der Melodie der Anzahl aller Töne gegebenübergestellt.

$$\text{feature}_8(I) = \frac{\text{Anzahl akkordeigene Töne}}{\text{Anzahl Töne}}$$

Lange Töne in Akkord (Long Notes in Chord)[2]

Als lange Töne werden hier solche bezeichnet, die mehr als oder genau zwei Viertelschläge andauern. Unabhängig von ihrer Position gibt dieses Kriterium das Verhältnis der Anzahl der langen Töne, die akkordeigen sind, zu der Anzahl aller langen Töne an.

$$\text{feature}_9(I) = \frac{\text{Anzahl langer akkordeigener Töne}}{\text{Anzahl langer Töne}}$$

Lange Töne nicht in Skala (Long Notes not in Scale)[2]

Dieses Kriterium entspricht dem vorhergehenden, nur wird nicht die Anzahl der akkordeigenen, sondern der skalenfremden Töne betrachtet. Dies hat in der aktuellen Implementierung keine Funktion, da nur skaleneigene Töne zugelassen sind.

$$\text{feature}_{10}(I) = \frac{\text{Anzahl langer skalenfremder Töne}}{\text{Anzahl langer Töne}}$$

10.1.3 Konturmerkmale

Konturrichtung (Contour Direction)[1]

Hier wird die Summe aller steigenden Intervalle im Verhältnis zur Summe aller Intervalle betrachtet. Dies ergibt die Tendenz der Melodie, also ob sie fällt oder steigt.

$$\text{feature}_{11}(I) = \frac{\text{Summe aller steigenden Intervalle}}{\text{Summe aller Intervalle}}$$

Konturstabilität (Contour Stability)[1]

Dieses Merkmal beschreibt die Stabilität der Melodie und hängt so von der Anzahl aufeinanderfolgender Intervalle selber Richtung ab.

$$\text{feature}_{12}(I) = \frac{\text{Anzahl aufeinanderfolgender Intervalle mit selber Richtung}}{\text{Anzahl Intervalle} - 1}$$

Schrittweise Bewegung (Movement by Step)[1]

Als Schritt wird eine große oder kleine Sekunde bezeichnet und entsprechend bezeichnet dieses Merkmal das Verhältnis solcher (auch diatonisch genannter) Schritte.

$$\text{feature}_{13}(I) = \frac{\text{Anzahl diatonischer Schritte}}{\text{Anzahl Intervalle}}$$

Sprungrückkehr (Leap Return)[1]

Als großer Sprung wird ein Intervall zwischen zwei aufeinanderfolgenden Tönen, welches größer als eine kleine Sexte ist, bezeichnet. Von einer Rückkehr wird gesprochen, wenn der Ton folgend auf das

Intervall zwischen den beiden vorhergehenden Tönen liegt. Hierbei darf er gleich dem ersten, aber nicht gleich dem zweiten Ton sein. Daher ergibt sich ein Rücksprungintervall, welches mindestens einen Halbton groß, aber kleiner als das vorhergehende große Intervall sein muss.

Das Verhältnis großer Sprünge ohne Rückkehr zu allen großen Sprüngen wird so charakterisiert. Ein zu hoher Wert könnte zu einer sehr nervösen, wenig eingängigen Melodie führen.

$$\text{feature}_{14}(I) = \frac{\text{Anzahl großer Sprünge, denen keine Rückkehr folgt}}{\text{Anzahl großer Sprünge}}$$

Höhepunktstärke (Climax Strength)[1]

Wie stark ein Höhepunkt der Melodie wahrgenommen wird, bestimmt dieses Merkmal, welches die Häufigkeit des Vorkommens des höchsten Tons beschreibt.

$$\text{feature}_{15}(I) = \frac{1}{\text{Anzahl der Vorkommen der höchsten Note}}$$

Große Intervalle (Large Intervals)[2]

Dieses Kriterium berechnet die relative Häufigkeit von großen (mehr als 8 chromatische Schritte umfassenden) Intervallen, wobei im Gegensatz zu dem Kriterium „Sprungrückkehr“ der Verlauf der Melodie nach dem großen Intervall nicht betrachtet wird.

$$\text{feature}_{16}(I) = \frac{\text{Anzahl großer Intervalle}}{\text{Anzahl aller Intervalle}}$$

10.1.4 Rhythmische Merkmale

Notendichte (Note Density)[1]

Dieses Merkmal beschreibt das Verhältnis der Anzahl aller Töne zu der Anzahl aller möglichen Töne. Dies könnte als eine Beschreibung der Geschwindigkeit der Melodie wahrgenommen werden.

$$\text{feature}_{17}(I) = \frac{\text{Anzahl Noten}}{\text{Anzahl Zeiteinheiten}}$$

Pausendichte (Rest Density)[1]

Die Dichte der Pausen stellt nicht das selbe Merkmal wie die Notendichte dar, sondern repräsentiert die verhältnismäßige Zahl der Zeiteinheiten, an denen kein Ton zu hören ist.

$$\text{feature}_{18}(I) = \frac{\text{Anzahl stille Zeiteinheiten}}{\text{Anzahl Zeiteinheiten}}$$

Tonlängenvielfalt (Rhythmic Variety)[1]

Die Anzahl der unterschiedlichen Tonlängen in dem Individuum könnte als Konstanz und Ruhe des Rhythmus wahrgenommen werden. Sie steht im Verhältnis zu der angenommenen maximalen Anzahl unterschiedlicher Tonlängen von 16.

$$\text{feature}_{19}(I) = \max\left(1; \frac{\text{Anzahl verschiedener verwendeter Notenlängen}}{16}\right)$$

Tonlängenumfang (Rhythmic Range)[1]

Dieses Merkmal bestimmt die Spanne zwischen der geringsten und der größten Notenlänge. Es gilt die Annahme, dass es sinnvoll ist, von einer höchsten Anzahl von 16 unterschiedlichen Tonlängen auszugehen.

$$\text{feature}_{20}(I) = \max\left(1; \frac{\text{Maximale Notenlänge} - \text{Minimale Notenlänge}}{16}\right)$$

Synkopisierung (Syncopation)

Als synkopisierte Noten werden solche bezeichnet, welche nicht auf einem Vielfachen ihrer eigenen Länge im Takt beginnen. Dieses Kritierium könnte die Stetigkeit des Rhythmus bewerten, da hier eingeht, ob auf betonten Schlägen Töne beginnen.

$$\text{feature}_{21}(I) = \frac{\text{Anzahl synkopisierter Noten}}{\text{Anzahl Noten}}$$

Synkopisierung in Bezug auf Viertel (Syncopation w. Resp. to Quarters)[1]

Mit Synkopisierung werden hier Noten bezeichnet, die eine Länge von mindestens einer Viertel haben, aber nicht auf einem Viertelschlag beginnen. Dies ist also eine Spezialisierung des vorhergehenden Kriteriums.

$$\text{feature}_{22}(I) = \frac{\text{Anzahl synkopisierter}^{3} \text{ Noten}}{\text{Anzahl Noten}}$$

10.1.5 Mustermerkmale

Wiederholte Tonhöhe aufeinanderfolgender Töne (Repeated Pitch)[1]

Dieses Kriterium gibt das Verhältnis der Anzahl wiederholter Tonhöhen zu der Gesamtzahl zweier aufeinanderfolgender Töne an.

$$\text{feature}_{23}(I) = \frac{\text{Anzahl zweier aufeinanderfolgender Töne mit gleicher Tonhöhe}}{\text{Anzahl Intervalle}}$$

Wiederholte Tonlänge aufeinanderfolgender Töne (Repeated Rhythmic Value)[1]

Dieses Merkmal bestimmt die Häufigkeit zweier aufeinanderfolgender Töne mit selbem Rhythmuswert im Verhältnis zur Gesamtzahl zweier aufeinanderfolgender Töne.

$$\text{feature}_{24}(I) = \frac{\text{Anzahl zweier aufeinanderfolgender Töne mit gleicher Tonlänge}}{\text{Anzahl Intervalle}}$$

Aufeinanderfolgende Wiederholung einer Tonhöhenfolge von drei Tönen (Repeated Pitch Patterns of Three Notes)[1]

Dieses Merkmal gibt die Häufigkeit des direkten Aufeinanderfolgends von drei Tönen gleicher Tonhöhe im Verhältnis zu dem entsprechend höchstmöglichen Wert an.

$$\text{feature}_{25}(I) = \frac{\text{Anzahl wiederholter Sequenzen von drei Tonhöhen}}{\text{Anzahl Noten} - 4}$$

Aufeinanderfolgende Wiederholung einer Tonhöhenfolge von vier Tönen (Repeated Pitch Patterns of Four Notes)[1]

Dieses Merkmal entspricht dem vorhergehenden mit dem Unterschied, dass hier eine Folge von vier Tonhöhen betrachtet wird.

$$\text{feature}_{26}(I) = \frac{\text{Anzahl wiederholter Sequenzen von vier Tonhöhen}}{\text{Anzahl Noten} - 5}$$

[3] Zu beachten: Hier gilt ein anderes Verständnis von Synkopisierung als im vorhergehenden Merkmal.

Aufeinanderfolgende Wiederholung einer Tonlängenfolge von drei Tönen (Repeated Rhythm Patterns of Three Notes)[1]
Dieses Kriterium entspricht den vorhergehenden Merkmalen, jedoch wird hier nicht die Tonhöhe sondern die Tonlänge betrachtet.

$$\text{feature}_{27}(I) = \frac{\text{Anzahl wiederholter Sequenzen von drei Tonlängen}}{\text{Anzahl Noten} - 4}$$

Aufeinanderfolgende Wiederholung einer Tonlängenfolge von vier Tönen (Repeated Rhythm Patterns of Four Notes)[1]
Dieses Merkmal entspricht dem vorhergehenden, jedoch werden folgen von vier Tonlängen betrachtet.

$$\text{feature}_{28}(I) = \frac{\text{Anzahl wiederholter Sequenzen von vier Tonlängen}}{\text{Anzahl Noten} - 5}$$

Aufeinanderfolgende Wiederholung einer Tonlängenfolge von drei Tönen (Globally Repeated Rhythm Patterns of Three Notes)
Dieses Merkmal betrachtet entgegen des vorletzten Kriteriums zwar ebenfalls die Wiederholung von Tonlängenfolgen, bewertet diese aber unabhängig davon, ob die Wiederholung der Folge von drei Tönen direkt stattfindet oder andere Töne dazwischen liegen.

$$\text{feature}_{29}(I) = \frac{\text{Anzahl wiederholter Sequenzen von drei Tonlängen}}{\text{Anzahl Noten} - 4}$$

Aufeinanderfolgende Wiederholung einer Tonlängenfolge von vier Tönen (Globally Repeated Rhythm Patterns of Four Notes)
Dieses Kriterium entspricht dem vorhergehenden, wobei allerdings nicht Folgen von drei sondern von vier Tonlängen betrachtet werden.

$$\text{feature}_{30}(I) = \frac{\text{Anzahl wiederholter Sequenzen von vier Tonlängen}}{\text{Anzahl Noten} - 5}$$

10.1.6 Merkmale bei Akkordwechsel

Über Akkordwechsel passend gehaltene Töne (Holding Notes Fitting to Chord Change)[2]
Dieses Merkmal betrachtet alle Töne, welche über einen Akkordwechsel hinaus gehalten werden und in beiden Akkorden enthalten sind. Diese Häufigkeit wird im Verhältnis zu allen Akkordwechseln gesetzt.

$$\text{feature}_{31}(I) = \frac{\text{Anzahl über Akkordwechsel passend gehaltener Töne}}{\text{Anzahl Akkordwechsel}}$$

Über Akkordwechsel unpassend gehaltene Töne (Holding Notes not Fitting to Chord Change)[2]
Ähnlich wie in dem vorhergehenden Beispiel arbeitet dieses Merkmal, betrachtet nun aber nicht die Töne, die zu beiden Akkorden passen, sondern solche, die im ersten Akkord aber nicht im zweiten enthalten sind.

$$\text{feature}_{32}(I) = \frac{\text{Anzahl über Akkordwechsel unpassend gehaltener Töne}}{\text{Anzahl Akkordwechsel}}$$

Tonhöhe wechselt mit Akkordwechsel (Note Pitch Changing with Chord Change)[2]
Hier werden nun die Häufigkeiten der Akkordwechsel mit Wechsel der Tonhöhe im Verhältnis gesehen.

$$\text{feature}_{33}(I) = \frac{\text{Anzahl Akkordwechsel mit Tonhöhenwechsel}}{\text{Anzahl Akkordwechsel}}$$

Pause über Akkordwechsel gehalten (Rests Holding while Chord Changes)[2]

Dieses Merkmal betrachtet die relative Häufigkeit von Pausen, die über einen Akkordwechsel hinaus gehalten werden im Verhältnis zu allen Akkordwechseln.

$$\text{feature}_{34}(I) = \frac{\text{Anzahl Akkordwechsel mit gehaltener Pause}}{\text{Anzahl Akkordwechsel}}$$

10.1.7 Betonungsmerkmale

Betonung liegt in Akkord (Downbeat Notes in Chord)[2]

Dieses Merkmale betrachtet die Anzahl der Töne, die auf einem betonten Schlag erster Ordnung liegen und akkordeigen sind im Verhältnis zu allen betonten Schlägen erster Ordnung. Unter einer solchen Betonung ist der erste Schlag eines Takts zu verstehen.

Die Positionen im Individuum, die diesem Schlag entsprechen sind zu bestimmen, indem jedes ganzzahlige Vielfache der Anzahl der Positionen eines Takts n_t betrachtet wird. Dabei ist $n_t = \frac{v}{r}$, wobei v die Anzahl der Schläge und r die Auflösung angibt.

$$\text{feature}_{35}(I) = \frac{\text{Anzahl akkordeigener Töne auf Betonungen erster Ordnung}}{\text{Anzahl Betonungen erster Ordnung}}$$

Pause auf Betonung (Rests on Downbeats)[2]

Hier wird die Anzahl aller Betonungen betrachtet, auf denen eine Pause liegt. Es macht keinen Unterschied, ob diese Pause an der Betonung beginnt oder zu dieser Position gehalten ist.

$$\text{feature}_{36}(I) = \frac{\text{Anzahl Betonungen mit Pause}}{\text{Anzahl Betonungen erster Ordnung}}$$

Betonung liegt in Skala (Downbeat Notes in Scale)[2]

Dieses Kriterium betrachtet die Anzahl aller betonten Töne, die in der geltenden Skala liegen. In der vorliegenden Implementierung hat dieses Merkmal keine Wirkung, da nur skaleneigene Töne zugelassen sind.

$$\text{feature}_{37}(I) = \frac{\text{Anzahl skaleneigener Töne auf Betonungen erster Ordnung}}{\text{Anzahl Betonungen erster Ordnung}}$$

Betonung liegt nicht in Skala/Akkord (Downbeat Notes not in Scale/Chord)[2]

Dieses Merkmal bestimmt die relative Anzahl von Tönen auf Betonungen, die weder im geltenden Akkord noch in der Skala liegen. Da in der vorliegenden Implementierung nur skaleneigene Töne zugelassen sind hat dieses Merkmale keine Wirkung.

$$\text{feature}_{38}(I) = \frac{\text{Anzahl Betonungen mit skalen- und akkordfremdem Ton}}{\text{Anzahl Betonungen erster Ordnung}}$$

Betonung zweiter Ordnung liegt in Akkord (Halfdownbeat Notes in Chord)[2]

Dieses Kriterium entspricht dem Merkmal „Betonung liegt in Akkord“ mit dem Unterschied, dass hier die Betonungen zweiter Ordnung betrachtet werden. Die Positionen dieser Betonungen zweiter Ordnung werden bestimmt durch ganzzahlige Vielfache von

$$n_h = \left\lfloor \frac{v}{2r} + 1 \right\rfloor .$$

$$\text{feature}_{39}(I) = \frac{\text{Anzahl akkordeigener Töne auf Betonungen zweiter Ordnung}}{\text{Anzahl Betonungen zweiter Ordnung}}$$

Pause auf Betonung zweiter Ordnung (Rests on Halfdownbeats)[2]
Dieses Merkmal entspricht dem Merkmal „Pause auf Betonung“, allerdings werden hier Betonungen zweiter Ordnung betrachtet.

$$\text{feature}_{40}(I) = \frac{\text{Anzahl Betonungen zweiter Ordnung mit Pause}}{\text{Anzahl Betonungen zweiter Ordnung}}$$

Betonung zweiter Ordnung liegt in Skala (Halfdownbeat Notes in Scale)[2]
Diese Merkmale entspricht „Betonung liegt in Skala“, allerdings werden hier Betonungen zweiter Ordnung betrachtet.

$$\text{feature}_{41}(I) = \frac{\text{Anzahl skaleneigener Töne auf Betonungen zweiter Ordnung}}{\text{Anzahl Betonungen zweiter Ordnung}}$$

Betonung zweiter Ordnung liegt nicht in Skala/Akkord (Halfdownbeat Notes not in Scale/Chord)[2]
Dieses Merkmal entspricht „Betonung liegt nicht in Skala/Akkord“, allerdings werden hier Betonungen zweiter Ordnung betrachtet.

$$\text{feature}_{42}(I) = \frac{\text{Anzahl Betonungen zweiter Ordnung mit skalen- und akkordfremdem Ton}}{\text{Anzahl Betonungen zweiter Ordnung}}$$

10.2 Bewertung der Individuen

10.2.1 Interaktive Bewertung

Die Schwierigkeit, eine Rechenvorschrift oder eine Klassifikationsregel zur Erstellung der Fitness eines Individuums zu entwerfen stellt sich bei diesem Verfahren nicht. Die Idee ist, die subjektive, unscharfe Komponente des Geschmacks eines Menschen auszunutzen und diesen zur Bewertung heranzuziehen. Hierzu wird jeweils das zu bewertende Individuum vorgespielt, wobei zur Unterstützung die zugrundeliegenden Akkorde hörbar sein können, wenn das gewünscht wird. Weiterhin wird dieses Individuum in Notenschrift dargestellt. Der Benutzer soll nun, nachdem er sich eine Meinung gebildet hat, wobei er das Abspielen der Melodie beliebig oft anstoßen, aber auch vorzeitig abbrechen kann, eine Bewertung abgeben. Dies findet durch einen Schieberegler statt, auf dem Beträge zwischen 0 und 10 in Schritten von 0,1 eingestellt werden können, was eine hinreichend genaue Abstufung möglich macht.

Diese Bewertungsfunktion hat noch zwei weitere, besondere Funktionen. Zum einen besteht die Möglichkeit, einzuschätzende Melodien gezielt zu verändern. Dadurch entsteht eine Möglichkeit der interaktiven Komposition. Des weiteren können bei jeder Evolution alle bewerteten Individuen automatisiert gespeichert werden. Dies macht jedoch bei dieser Funktion besonders Sinn, da hierdurch recht einfach eine Trainingsmenge für neuronale Netze beziehungsweise die Grundlage zur Erstellung von Entscheidungsbäumen geschaffen werden kann. Auf diese Verfahren wird im folgenden eingegangen.

10.2.2 Neuronale Netze

Da die Prozedur des interaktiven Bewertens recht mühselig sein kann, da möglicherweise recht wenig gefällige Melodien angehört werden müssen, ist es naheliegend, automatische Klassifikationssysteme zu nutzen. Hierzu werden unter anderem vollvernetzte Feed-Forward-Netze eingesetzt (siehe Abschnitt 2.3). Da diese auf den (auf den Wertebereich $[-1;1]$ skalierten) extrahierten Merkmalen als Eingaben arbeiten,

[1] Dieses Merkmal ist ohne Veränderung von (Towsey u. a., 2000) übernommen.
[2] Dieses Merkmal folgt (Wiggins und Papadopoulos, 1998).

ist zunächst anzugeben, welche dieser Merkmale genutzt werden sollen. Dies soll die Möglichkeit schaffen, als unwichtig oder kontraproduktiv eingeschätzte Merkmale ausschalten zu können sowie die Wirkung einzelner Merkmale isoliert betrachten zu können.

Neben der Auswahl der als Eingabe genutzten Merkmale, aus der die Anzahl der Eingabeneuronen folgt, ist die Anzahl der Neuronen je versteckter Schicht anzugeben. Es existiert grundsätzlich genau ein Ausgabeneuron, welches nicht explizit anzugeben ist. Dessen Ausgabewerte, welche entsprechend der für alle Neuronen (bis auf die Eingabeneuronen) verwendeten Aktivierungsfunktion *tanh* (siehe Definition 16) im Intervall $[-1;1]$ liegen, werden zum Vergleich und für die Zuweisung an die Individuen auf das Intervall $[0;10]$ skaliert.

Als Lernverfahren stehen in einer eigenen Implementierung Backpropagation und unter Nutzung der Bibliothek Joone (Marrone, 2005) Backpropagation und Resilientpropagation zur Verfügung. Die Anpassung der Gewichte des Netzes mittels Resilientpropagation ist deutlich effizienter, so dass empfohlen ist, dieses Lernverfahren einzusetzen.

Zum Test der neuronalen Netze ist eine Beispielmenge von Individuen erstellt worden. Diese teilen sich auf in bekannte Musikstücke, aus denen die Melodie extrahiert ist und die mit der geltenden Akkordfolge sowie einer Bewertung versehen sind (45 Melodien) sowie automatisch erzeugter Individuen (136 Melodien). Diese sind durch automatische Speicherung aller während einer Evolution durch den Benutzer bewerteten Individuen entstanden. Weiterhin sind einige Melodien von Hand erstellt worden, die subjektiv besonders wenig ästhetisch erscheinen (25 Melodien).

Um zu evaluieren, welche Netzstruktur sinnvoll ist, wurden Netze mit einer oder mehreren versteckten Schichten betrachtet, bei denen die Anzahl der Neuronen variiert wurde. Der Test fand grundsätzlich mit allen Merkmalen als Eingabe des Netzes statt. Zu jeder Netzstruktur wurden 10 Lernversuche mit jeweils 1000 Wiederholungen durchgeführt und der durchschnittliche TSSE als Güte betrachtet. Diese Durchschnittswerte sind in Abbildung 10.1 gezeigt.

Wie man hier sieht, ist eine Erhöhung der Anzahl der versteckten Neuronen bei einer versteckten Schicht über 35 nicht sinnvoll, da der dort geltende TSSE von 0,215 nicht signifikant weiter verbessert wird (bei 100 Neuronen beträgt er 0,0213). Auch durch andere Netzstrukturen konnten keine weitergehenden Erfolge erzielt werden.

Insgesamt scheint der Fehler schwierig weiter zu unterschreiten zu sein. Auch Versuche mit 100000 Iterationen brachten nur Verbesserungen im Bereich von 0,0003.

Der Versuch ist auch mit einer als besonders informativ (entsprechend Definition 18) angenommenen Teilmenge von Merkmalen (siehe Tabelle 10.2) sowie der komplementären Menge als Eingabe des neuronalen Netzes durchgeführt worden. Erwartungsgemäß ist der Fehler bei den informativeren Merkmalen geringer, jedoch nicht in signifikantem Maß. Dies lässt darauf schließen, dass einige der genutzten Merkmale redundant sind, was naheliegend ist, da zwischen einigen nur kleine Unterschiede bestehen. Der TSSE nach 1000 Lernschritten beider Merkmalsmengen ist in Abbildung 10.2 gezeigt.

10.2.3 Entscheidungsbäume

So flexibel künstliche neuronale Netze doch sind, ist eine Interpretation der Gewichte hinsichtlich einer Bedeutung der Art und Weise der Kombination der Merkmale äußerst schwierig. Dieser Nachteil besteht bei Entscheidungsbäumen nicht, da diese eine Menge von Regeln darstellen, welche intuitiv verständlich ist. Ein weiterer Vorteil von Entscheidungsbäumen gegenüber künstlichen neuronalen Netzen ist die im Verhältnis sehr kurze Trainings- beziehungsweise Erstellungszeit.

Die verwendete Implementierung zur Erzeugung von Entscheidungsbäumen entstammt der Bibliothek Weka (Witten und Frank, 2005b), welche eine Weiterentwicklung der in Abschnitt 2.4 vorgestellten Verfahren darstellt. Besonderheiten sind die Verfügbarkeit verschiedener parametrisierbarer[1] Beschneidungstechniken, aber auch die sogenannte *10-Fold-Cross-Validation*, welche die Generalisierungsfähigkeit verbessern soll. Bei dieser werden mit $\frac{9}{10}$ der Beispiele der Baum erstellt und $\frac{1}{10}$ die Klassifikation

[1] Alle hier durchgeführten Tests nutzen die empfohlenen Standardeinstellungen der Weka-Bibliothek.

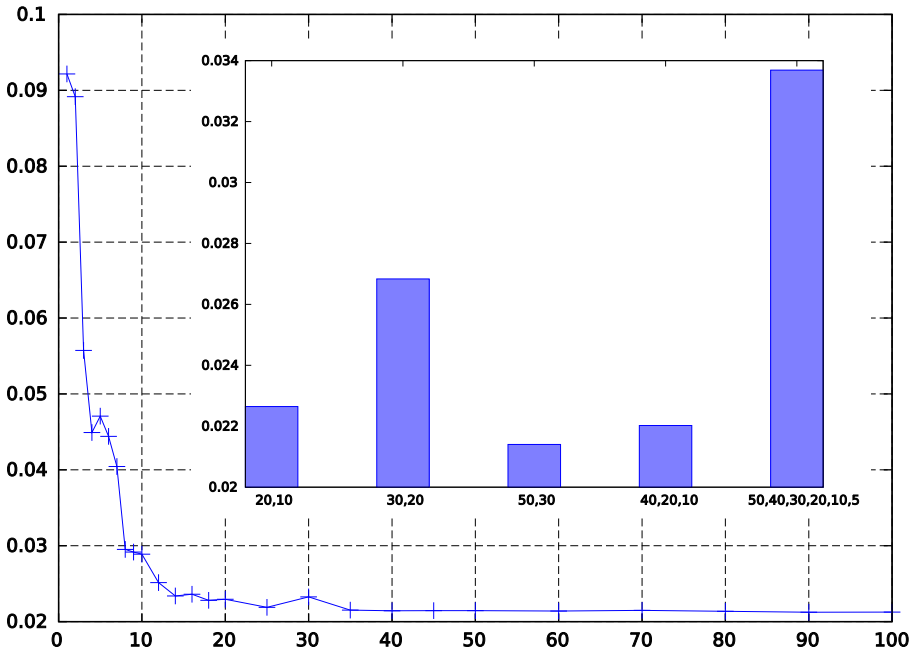


Abbildung 10.1: TSSE nach 1000 Iterationen von Resilient-Propagation auf Neuronalen Netzen mit unterschiedlich vielen versteckten Neuronen.

Große Graphik: Veränderung des erreichten TSSE bei Erhöhen der Anzahl versteckter Neuronen bei einer versteckten Schicht (Die horizontale Achse beschreibt die Anzahl der Neuronen, die vertikale Achse den erreichten TSSE).

Kleine Graphik: TSSE bei Netzstrukturen mit unterschiedlich vielen versteckten Schichten (Die kommaseparierten Zahlen geben die Anzahl der Neuronen auf versteckten Schichten an: 50,30 deutet also auf zwei versteckte Schichten mit 50 und 30 Neuronen hin.)

kontrolliert. Die Einteilung der Beispiele wird 10 mal verändert, so dass alle Beispiele zur Erstellung sowie zur Validierung herangezogen werden. Details, welche hier aus Platzgründen ausgelassen sind, finden sich in Witten und Frank (2005a).

Als Attribute werden die in Abschnitt 10.1 beschriebenen Merkmale genutzt. Anhand der Belegung dieser Merkmalsmenge ist über die Fitness des korrespondierenden Individuums zu entscheiden. Da die Fitness kontinuierlich ist, muss sie diskretisiert werden. Hierzu wird vom Benutzer die Anzahl der Klassen angegeben, welche als gleich groß angenommen sind. Erstellt wird der Entscheidungsbaum anhand von Beispielindividuen, aus denen die Merkmale berechnet werden und die Fitness angegeben ist. Tests sind mit derselben Menge durchgeführt, die in Abschnitt 10.2.2 erwähnt ist. Die Auflistung in Tabelle 10.2 basiert auf einem Entscheidungsbaum aus diesen Merkmalen mit 11 Fitnessklassen.

Die Komplexität des entstehenden Entscheidungsbaums (bei angenommener festgelegter Beispielmenge) wird neben der Wahl, ob beschnitten werden soll, in hohem Maße von der gewählten Zahl der Klassen bestimmt. In Abbildung 10.3 ist der Anstieg der Anzahl der Knoten mit dem Anstieg der Anzahl der Klassen, auf die die Fitness der Individuen diskretisiert wird zum einen ohne, zum anderen mit Beschneidung (wobei die Standardeinstellungen von Weka genutzt werden), dargestellt. Zwar beeinflussen diese Unterschiede die Effizienz der Bewertung während der Evolution nicht merklich. Allerdings hängt der Überblick zum Verständnis des Baums und damit den Zusammenhängen zwischen den Merkmalen von ihnen ab.

Exemplarisch ist in Abbildung 10.4 ein Baum mit 5 Fitnessklassen dargestellt, der auf der bereits erwähnten Beispielmenge basiert und so ein auch im Allgemeinen recht gutes Verhältnis der Merkmale zueinander darstellt. In den Blättern ist jeweils die zugeordnete Fitness angetragen. Der erste Wert in

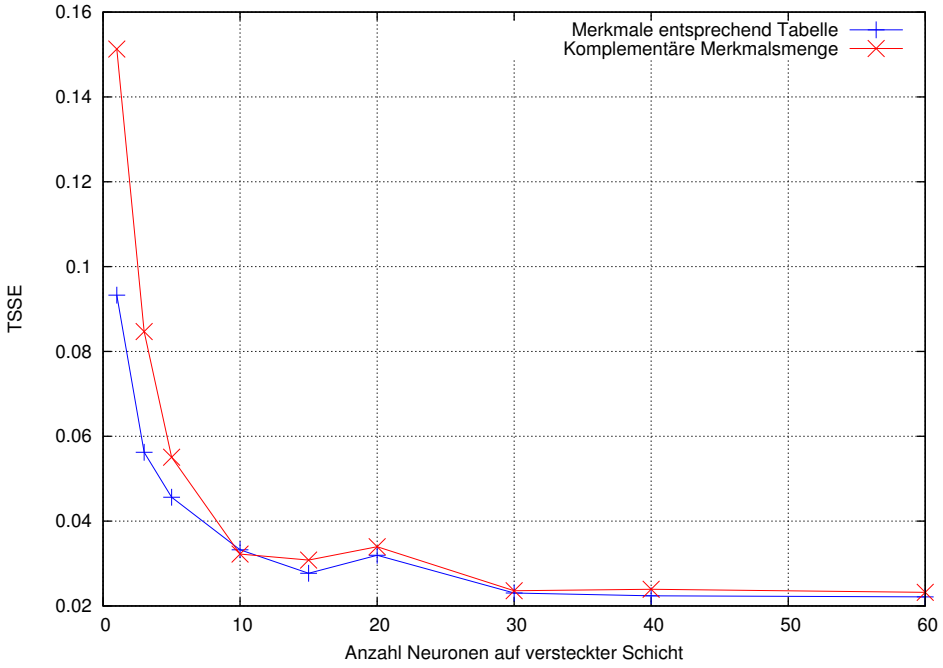


Abbildung 10.2: TSSE nach 1000 Lernschritten auf der Merkmalsmenge aus Tabelle 10.2 sowie aus der komplementären Merkmalsmenge. Die horizontale Achse stellt die Anzahl der Neuronen auf der versteckten Schicht dar.

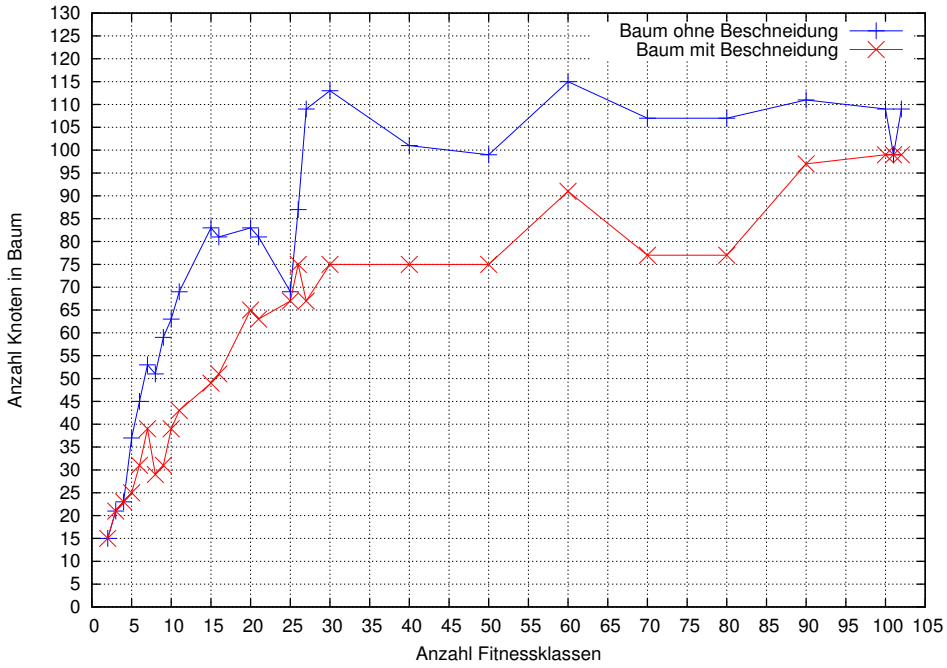


Abbildung 10.3: Anzahl der Knoten im erzeugten Entscheidungsbaum in Abhängigkeit von der Anzahl der Klassen, auf die die Fitness der Individuen diskretisiert werden

Ordnung entspr. Informationsgehalt	Merkmalsname
1	Repeated Rhythm Patterns of Four Notes
2	Note Pitch Changing with Chord Change
2	Repeated Rhythmic Value
3	Downbeat Notes in Chord
3	Syncopation w. Resp. to Quarters
3	Harmonicity
4	Rests on Downbeats
4	Halfdownbeat Notes in Chord
4	Contour Direction
4	Contour Stability
5	Repeated Pitch Patterns of Four Notes
5	Rests Holding while Chord Changes
6	Globally Repeated Rhythm Patterns of Three Notes
6	Holding Notes not Fitting to Chord Change
6	Repeated Pitch
7	Leap Return
7	Rhythmic Range
7	Non Scale Notes (Quanta)
8	Pitch Range
8	Large Intervals
9	Holding Notes Fitting to Chord Change
10	Dissonant Intervals
11	Rests on Halfdownbeats
12	Downbeat Notes in Scale
13	Pitch Variety

Tabelle 10.2: Ordnung einer Teilmenge der Merkmale nach Informativität auf einer Beispielmenge (Auf die Erstellung der Liste wird in Abschnitt 10.2.3 eingegangen.)

Klammern beschreibt die Anzahl der Beispielindividuen, welche dieses Blatt bei einer Klassifikation erreichen. Der zweite Wert in Klammern (wenn er fehlt ist er als 0 angenommen) bezeichnet die Anzahl der fehlklassifizierten Beispiele. Dies kann auch bei ausgeschalteter Beschneidung durch die 10-Fold-Cross-Validation vorkommen. An dem Beispielbaum kann man die Relevanz und Bedeutung der unterschiedlichen Merkmale erkennen. So ist als am wichtigsten eingestuft, dass wenigstens in geringem Maße Wiederholungen von Folgen gleicher Tonlängen vorhanden sind („Repeated Rhythm Patterns of Four Notes“). Im rechten Teilbaum finden sich 37 Individuen mit einer Fitness von mindestens 7,5, im linken nur 3 solche. Gut erkennbar ist auch die Unterscheidung anhand des Attributs, „Note Pitch Changing with Chord Change“. Hier ist ein recht hoher Wert nötig, um dem Individuum eine hohe Fitness zuzuweisen, was möglicherweise zunächst nicht trivial zu interpretieren ist. Tatsächlich sind die meisten Beispielindividuen 4 Takte lang mit taktweisen Akkordwechseln. Wenn also ein Ton nicht mit dem Akkord wechselt deutet dies neben der harmonischen Komponente auch auf eine Betonungsverschiebung hin. Dies dürfte der Grund für die hohe Relevanz dieses Merkmals sein, insbesondere, da es nur zum Tragen kommt, wenn eine hohe rhythmische Spannweite („Rhythmic Range“) vorhanden ist.

Auf diese Weise lassen sich alle Merkmale mit Interpretationen versehen. Solche eher musiktheoretischen Analysen sollen hier jedoch nicht weiter ausgeführt werden.

Verändern des Verhaltens einer bestehenden Zielfunktion ist nicht vorgesehen. Daher besteht die

Möglichkeit der Integration verschiedener Zielfunktionen in einer einzelnen, wie im folgenden zu sehen ist.

10.2.4 Gewichtete Summen und kombinierte Zielfunktionen

Es besteht die Möglichkeit, eine gewichtete Summe der zur Verfügung stehenden Merkmale zu erstellen und als Zielfunktion zu nutzen. Dies kann in mehreren Fällen sinnvoll sein. Zum einen besteht so die Möglichkeit, einzelne Merkmale und ihre Auswirkung in der Evolution zu beobachten und so einen subjektiven Eindruck zu bekommen, welche Eigenschaften einer Melodie hierdurch unterstützt werden. Es sei empfohlen diesen Versuch mit leicht verständlichen Merkmalen wie „Repeated Pitch“ selbst durchzuführen.

Zum anderen kann durch iteratives Erweitern einer gewichteten Summe um Merkmale eine Zielfunktion erstellt werden, welche eine bestimmte Absicht des Benutzers verfolgt. Solch ein gezieltes Vorgehen wäre durch Nutzung von Entscheidungsbäumen und neuronalen Netzen eher schwierig, da die Auswahl der Beispielindividuen sehr sorgfältig getroffen werden müsste.

Ein weiterer Anwendungszweck kommt bei der Erstellung von kombinierten Fitnessfunktionen zum Tragen. Mit diesen ist es möglich, eine gewichtete Summe von beliebigen Zielfunktionen, also neuronalen Netzen, Entscheidungsbäumen und gewichteten Summen zu definieren. So kann bei Beobachten eines unerwünschten Verhaltens einer sonst recht guten Zielfunktion ein bestimmtes Merkmal verstärkt oder geschwächt werden, indem eine gewichtete Summe mit dieser kombiniert wird.

Bei beiden Zielfunktionen, also den kombinierten Fitnessfunktionen wie auch den gewichteten Summen sind also die Gewichte w_i einer Funktion $f_g(I) = \sum_{i=1}^{n} v_i(I) \cdot w_i$ anzugeben (I ist ein Individuum, $v_1, \ldots, v_n$ sind die zu kombinierenden Zielfunktionen beziehungsweise die zu kombinierenden Merkmale). Die Werte von f_g können auf Wunsch des Benutzers auf das Intervall $[0;10] \subset \mathbb{R}$ skaliert werden. Hierzu werden die maximal und minimal erreichbaren Fitnesswerte $f_{\max}$ und $f_{\min}$

$$f_{\max} = \sum_{w_i > 0} w_i \cdot v_{\max} \qquad f_{\min} = \sum_{w_i < 0} |w_i| \cdot v_{\max}$$

bestimmt, wobei $v_{\max}$ den Maximalwert aller v_i angibt. Dieser wird bei der Erstellung kombinierter Zielfunktionen als $v_{\max} = 10$ und bei der Gewichtung von Merkmalen als $v_{\max} = 1$ angenommen. Die Bewertung des skalierten Fitnesswerts f_c wird aus f_g berechnet durch

$$f_c = \frac{f_g + f_{\min}}{f_{\max} + f_{\min}}.$$

10.2.5 Vergleich der verschiedenen Bewertungsfunktionen

Im folgenden soll ein Eindruck von den Besonderheiten der unterschiedlichen Zielfunktionen vermittelt werden. Hierbei wird nicht auf den Kontext der Evolution eingegangen, da hierzu durchgeführte Untersuchungen in Kapitel 11 dargestellt sind. Zur Erläuterung werden einige Beispielindividuen genutzt, welche in Abbildung 10.5 und 10.6 zusammen mit ihrer jeweiligen Charakterisierung gezeigt sind.

Zum Vergleich kommen die folgenden Zielfunktionen zum Einsatz:

Interaktiv Diese von mir vorgenommene Bewertung der Individuen stellt eine Referenz dar. Die Durchführung der Bewertung ist aufgrund des Umfangs dieser Arbeit auf eine Person beschränkt.

Gewichtete Summen

- Gewichtete Summe angelehnt an Wiggins und Papadopoulos (1998)
  Die in Abschnitt 5.4 genannten Merkmale sind zur Grundlage einiger implementierter Merkmale (siehe Abschnitt 10.1) genutzt. Diese gewichtete Gleichung ist an die Vorschläge der

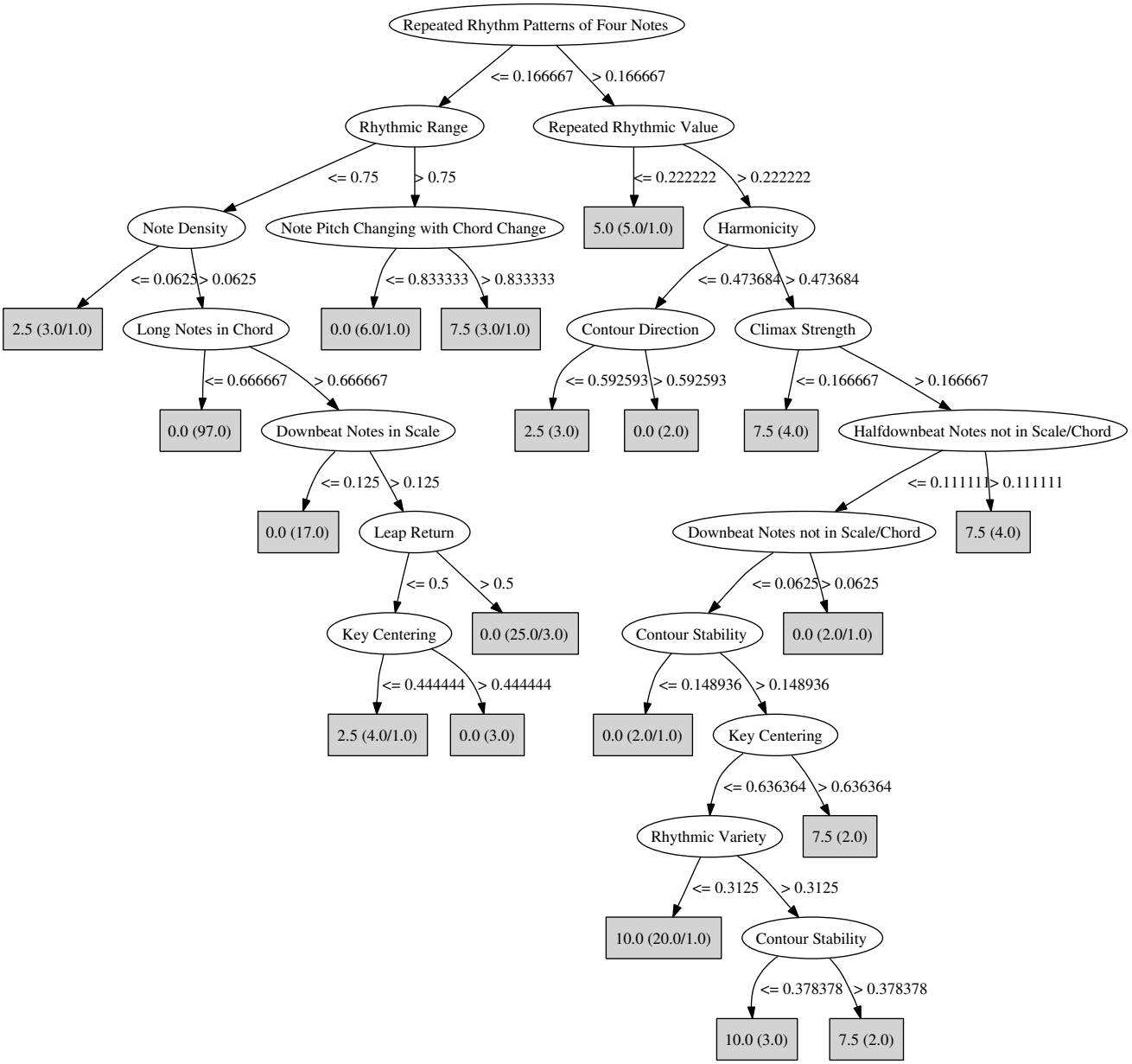


Abbildung 10.4: Entscheidungsbaum mit 5 Fitnessklassen basierend auf den in Abschnitt 10.2.2 vorgestellten Beispieldaten

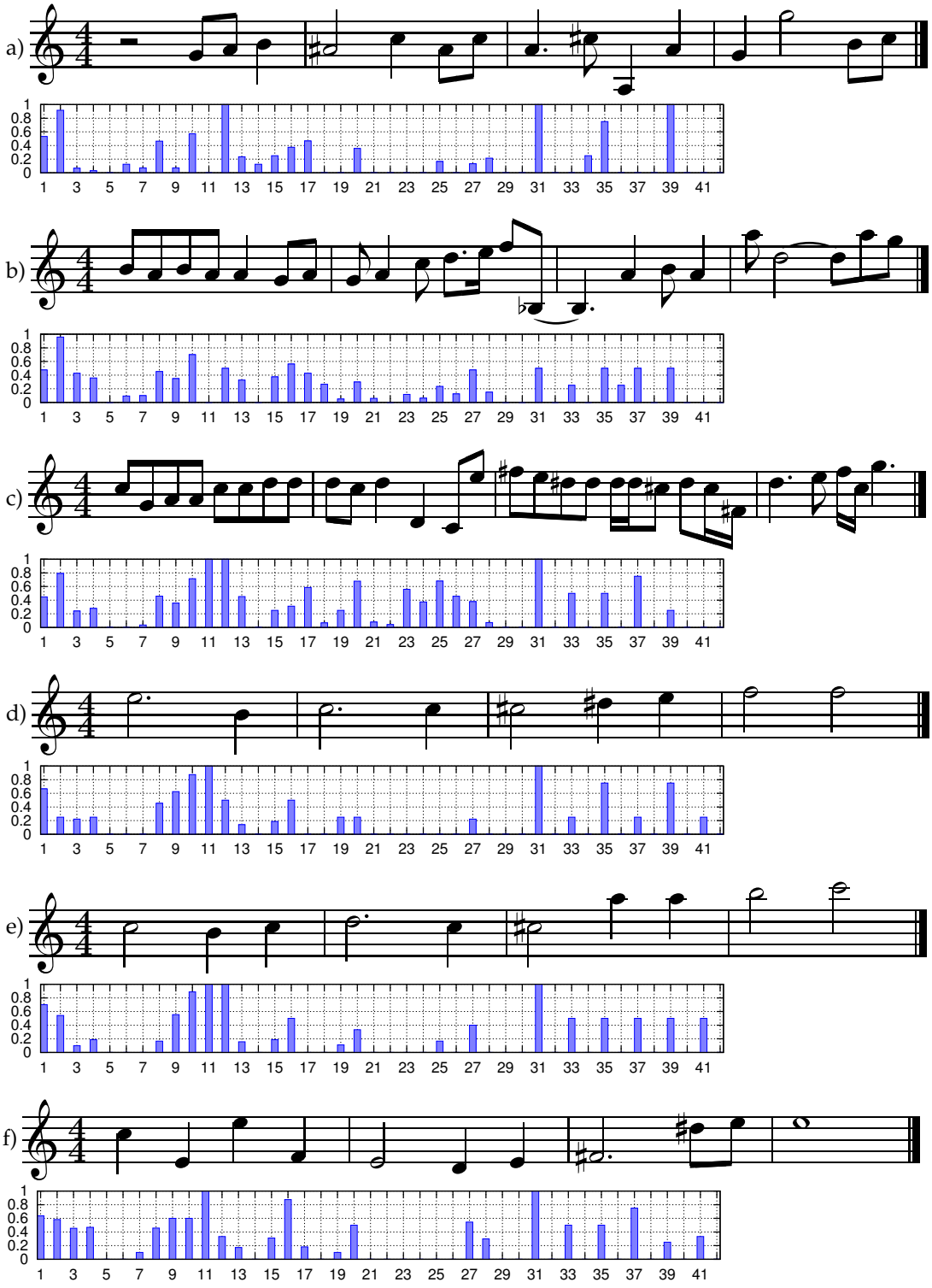


Abbildung 10.5: Beispielindividuen zum Vergleich verschiedener Bewertungsfunktionen mit Charakterisierung (Es gelten jeweils taktweise die Akkord *Am – Dm – E – Am*. Die Kennzahlen der Charakterisierung entsprechen der Zuordnung in Tabelle 10.3.)

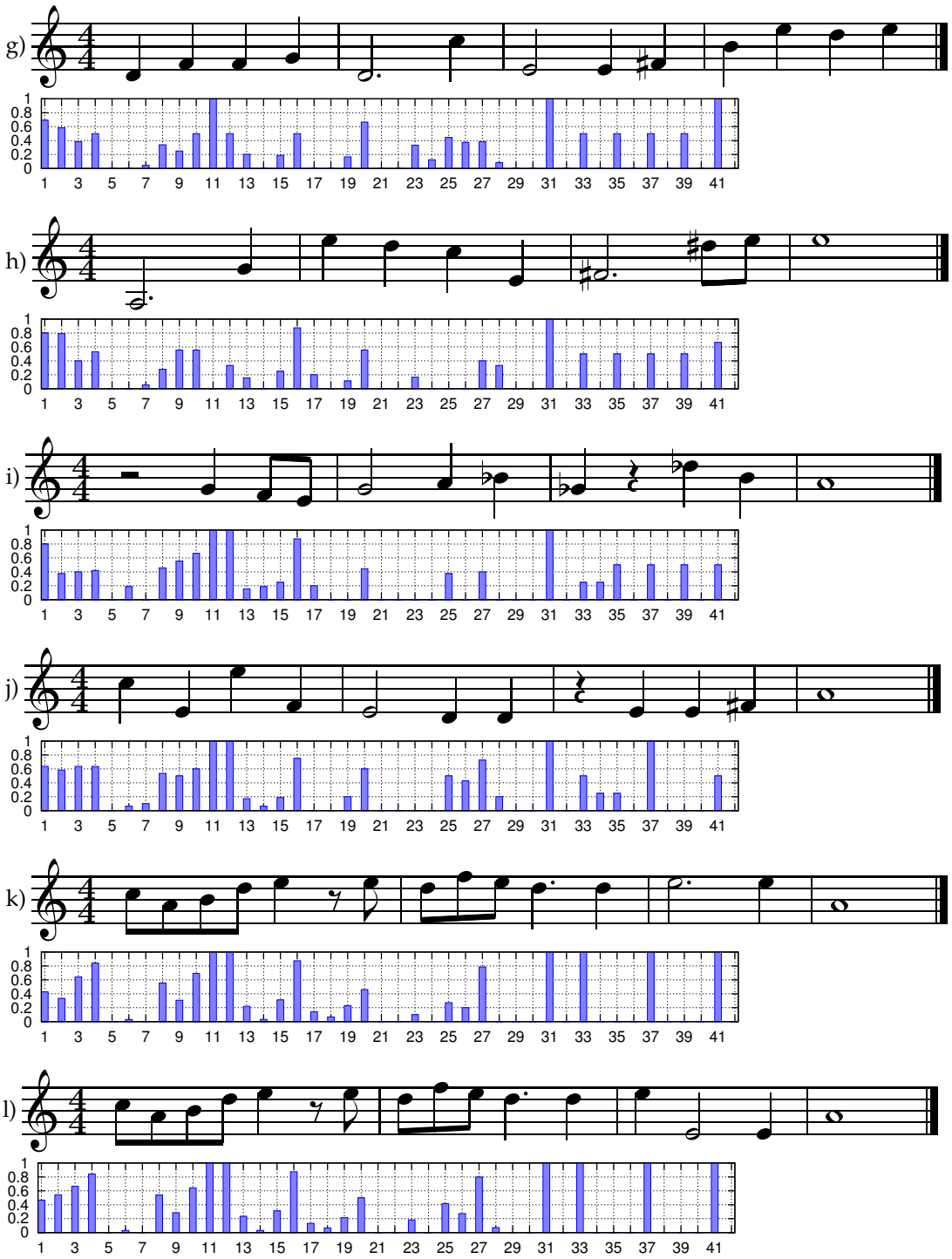


Abbildung 10.6: Beispielindividuen zum Vergleich verschiedener Bewertungsfunktionen mit Charakterisierung(Fortsetzung) (Es gelten jeweils taktweise die Akkord *Am – Dm – E – Am*. Die Kennzahlen der Charakterisierung entsprechen der Zuordnung in Tabelle 10.3.)

Nutzung dieser Merkmale angelehnt. Eine exakte Umsetzung ist aufgrund implementatorischer Unterschiede nicht möglich.

- Intuitiv erstellte gewichtete Summe
  Diese Berechnungsregel basiert auf der nach Wiggins erstellten, ist jedoch um eigene Ideen angereichert.

Entscheidungsbäume Die Bäume basieren auf der in Abschnitt 10.2.2 erwähnten Beispielmenge.

- 5 Fitnessklassen, unbeschnitten
  Dieser Baum entspricht dem in Abbildung 10.4 gezeigten. Knotenzahl: 37, Blattzahl: 19
- 5 Fitnessklassen, beschnitten
  Knotenzahl: 25, Blattzahl: 13
- 11 Fitnessklassen, unbeschnitten
  Knotenzahl: 69, Blattzahl: 35
- 11 Fitnessklassen, beschnitten
  Knotenzahl: 43, Blattzahl: 22

Neuronale Netze Die Netze sind in 1000 Iterationen mit Resilient-Propagation mit der in Abschnitt 10.2.2 erwähnten Beispielmenge trainiert.

- 6 Neuronen auf einer versteckten Schicht
  TSSE: 0,04016
- 35 Neuronen auf einer versteckten Schicht
  TSSE: 0,02136

Die Koeffizienten der Merkmale der gewichteten Summen sind in Tabelle 10.3 gezeigt. Hier ist auch die Zuordnung der Kennzahlen der Merkmale zu finden, wie sie in den Charakterisierungen der Beispielindividuen in Abbildung 10.5 und 10.6 genutzt ist.

Der Vergleich der unterschiedlichen Bewertungen der Beispielindividuen ist in Abbildung 10.7 zu sehen. In Diagramm 10.7(a) sind verschiedene Entscheidungsbäume dem interaktiven Verfahren gegenüber gestellt. Die Bewertungen des beschnittenen Baums mit 11 Fitnessklassen entspricht exakt der des unbeschnittenen Baums, weshalb zwischen diesen keine Unterscheidung gemacht wird. Die beiden Varianten des Baums mit 5 Klassen sind als ähnlich gut einzustufen. Allerdings scheinen alle Entscheidungsbäume mittelmäßige Bewertungen im Intervall $[2;5]$ (f, g, h, i) nicht allzu gut abzubilden. Dieser Effekt findet sich auch bei deutlich mehr Fitnessklassen wieder. Insgesamt erscheint die Bewertung mit 5 Klassen jedoch schlüssig: Individuen, welche deutlich als schlecht (a, b, c, d, e) oder gut (j, k, l) eingestuft sind, werden näherungsweise korrekt klassifiziert. Auch ist hier das Verhältnis zwischen den Individuen bis auf einen Fall gut wiedergegeben.

Die Bewertung der neuronalen Netze deckt sich nur schlecht mit der interaktiven Bewertung. Bei 6 Neuronen werden zwar die subjektiv guten Individuen (j, k, l) sowie die als sehr schlecht eingeschätzten Melodien (a, b, c, d) relativ gut klassifiziert, allerdings bestehen auch große Abweichungen (e, h, i). Mit 35 Neuronen sind die sehr guten Individuen (k, l) falsch klassifiziert. Dies lässt wegen des recht geringen Trainingsfehlers wohl auf eine schlechte Generalisierung des Netzes schließen. Da die Beispielmenge im Verhältnis zu der Zahl der Eingangsneuronen doch eher klein ist erscheint dies schlüssig. Aufgrund dieser Vermutung ist ein Netz mit 20 versteckten Neuronen und einer kleineren Zahl Eingangsneuronen erstellt worden, welches auf den Merkmalen in Tabelle 10.2 basiert. Allerdings sind die Ergebnisse hier ebenfalls nicht besser, wie in Abbildung 10.8 dargestellt ist. Hier ist auch die Kombination des unbeschnittenen Entscheidungsbaums mit 5 Klassen und des neuronalen Netzes mit 6 versteckten Neuronen zu sehen, welche eine ähnlich gute Bewertung liefert wie der Entscheidungsbaum, jedoch mit einem etwas geringeren Problem der schlecht klassifizierten mittleren Fitnesswerte. Der Ansatz der Kombination von verschiedenen Zielfunktionen sollte daher weiter verfolgt werden.

Wie in Abbildung 10.7(c) zu sehen ist, fallen die Differenzierungen zwischen den Individuen bei den gewichteten Summen geringer aus. Dennoch bilden sie trotz ihrer vergleichsweise einfachen Struktur die Verhältnisse zwischen den Individuen entsprechend der interaktiven Bewertung recht gut nach, wobei allerdings keine Verbesserung bei der intuitiv erweiterten Gleichung im Gegensatz zum Vorschlag von Wiggins zu beobachten ist.

Die interaktive Bewertung ist vor der automatischen Klassifikation durchgeführt worden. Auch in dieser können durchaus Fehleinschätzungen vorgenommen worden sein, da sie nur durch eine Person durchgeführt wurde. So fällt bei einigen Bewertungen bei einem subjektiven Vergleich auf, dass die automatische Klassifikation entgegen der ersten Annahme durchaus schlüssig erscheint.

Insgesamt erscheinen die Entscheidungsbäume die Klassifikationsaufgabe am besten zu erfüllen. Experimente mit einer Kombination mit anderen Methoden sind sinnvoll. Neuronale Netze funktionieren im Ansatz, bedürfen aber näheren Untersuchungen mit mehr Trainingsbeispielen. Die von Hand erstellten gewichteten Summen erzeugen näher beieinanderliegende Fitnesswerte, bilden jedoch das Verhältnis der Individuen zueinander gut ab. Allerdings ist das Erzeugen dieser Gleichung je nach Ziel mit einem erheblichen Aufwand verbunden, da die Rolle der unterschiedlichen Merkmale erkannt werden muss. Dieses Problem besteht bei den neuronalen Netzen und Entscheidungsbäumen nicht. Daher ist ein wichtiges ausstehendes Experiment, diese Klassifikatoren durch eine größere, möglicherweise einem Musikstil zugehörige Beispielmenge zu erzeugen. Zur genaueren Einschätzung sollte auch eine zuverlässige Daten liefernde Untersuchung der interaktiven Klassifikation mit hinreichend vielen Versuchspersonen durchgeführt werden.

10.3 Selektion

Nach der Bewertung der Individuen folgt der letzte Schritt des evolutionären Algorithmus. Durch die Selektion wird entschieden, welche Individuen in die Folgepopulation übernommen werden. Diese bildet nun das Ergebnis der Evolution, wenn die vorgegebene Zahl von Generationen erreicht ist oder der Benutzer einen Abbruch veranlasst hat (ein Abbruch ist nur zu diesem Punkt möglich). Im anderen Fall werden auf den selektierten Individuen wieder die Schritte Rekombination, Mutation und Bewertung durchgeführt. Zur Selektion stehen zwei Verfahren zur Verfügung.

Fitnessproportional

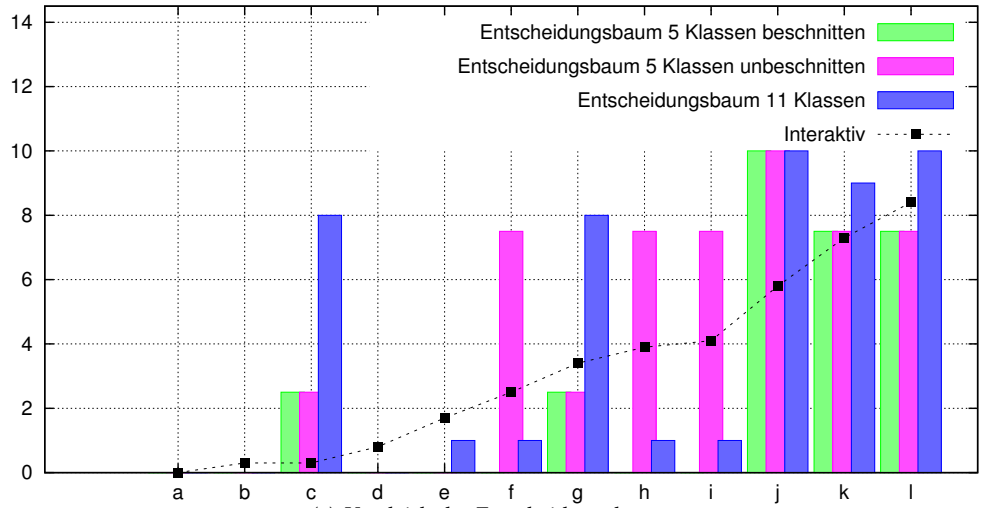
Diese Selektion entspricht dem in den Abschnitten 2.2.3.3 und 2.2.3.6 vorgestellten Verfahren. Hierbei wird also aus der Verteilung der Fitnesswerte auf die Individuen der aktuellen Population, wie auch der durch Rekombination und Mutation entstandenen Kindern, eine Wahrscheinlichkeitsverteilung bestimmt, entsprechend der dann zufällig die Folgeindividuen bestimmt werden. Ein Problem hierbei ist, dass Individuen mit einer hohen Fitness diese Verteilung dominieren können, was dazu führt, dass solche übermäßig häufig für die Folgepopulation gewählt werden.

Deterministisch

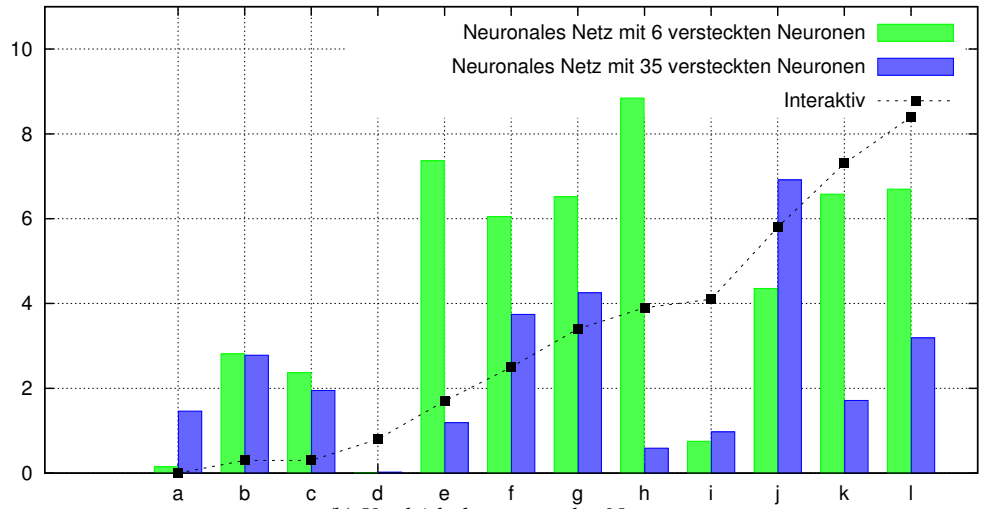
Die aus den Evolutionsstrategien stammende deterministische Selektion ist in Abschnitt 2.2.3.6 beschrieben. Hierbei werden aus der aktuellen Population und den erzeugten Kindern die Individuen mit der höchsten Fitness übernommen. Es handelt sich daher um die +-Selektion. Existieren mehr Individuen mit diesem höchsten Wert als selektiert werden sollen, wird aus der Menge der besten Melodien gleichverteilt zufällig ausgewählt. Jedes Individuum kann im Gegensatz zur fitnessproportionalen Selektion nur einmal in die Folgepopulation übernommen werden. Bei der Wahl von Crowding steht nur die deterministische Selektion zur Verfügung, wobei sie in der besonderen Form entsprechend der Beschreibung in Abschnitt 2.2.4 durchgeführt wird.

Kennung	Merkmal	Gewicht nach Wiggins	Gewicht intuitiv erstellt
1	Pitch Variety	0	-30
2	Pitch Range	0	0
3	Key Centering	0	0
4	Key Centering (Quanta)	0	10
5	Non Scale Notes	0	0
6	Non Scale Notes (Quanta)	0	0
7	Dissonant Intervals	0	0
8	Contour Direction	0	0
9	Contour Stability	0	0
10	Movement by Step	0	10
11	Leap Return	0	0
12	Climax Strength	0	10
13	Note Density	0	20
14	Rest Density	0	-10
15	Rhythmic Variety	0	0
16	Rhythmic Range	0	0
17	Syncopation	0	-20
18	Syncopation w. Resp. to Quarters	0	-30
19	Repeated Pitch	0	-15
20	Repeated Rhythmic Value	0	0
21	Repeated Pitch Patterns of Three Notes	0	5
22	Repeated Pitch Patterns of Four Notes	0	5
23	Globally Repeated Rhythm Patterns of Three Notes	0	0
24	Globally Repeated Rhythm Patterns of Four Notes	0	0
25	Repeated Rhythm Patterns of Three Notes	0	0
26	Repeated Rhythm Patterns of Four Notes	0	0
27	Harmonicity	0	10
28	Large Intervals	-20	-20
29	Holding Notes Fitting to Chord Change	10	10
30	Holding Notes not Fitting to Chord Change	-20	-20
31	Note Pitch Changing with Chord Change	5	5
32	Rests Holding while Chord Changes	5	5
33	Downbeat Notes in Chord	10	30
34	Rests on Downbeats	5	10
35	Downbeat Notes in Scale	-10	-10
36	Downbeat Notes not in Scale/Chord	-20	-20
37	Halfdownbeat Notes in Chord	5	5
38	Rests on Halfdownbeats	5	5
39	Halfdownbeat Notes in Scale	5	5
40	Halfdownbeat Notes not in Scale/Chord	-20	-20
41	Long Notes in Chord	10	10
42	Long Notes not in Scale	-20	-20

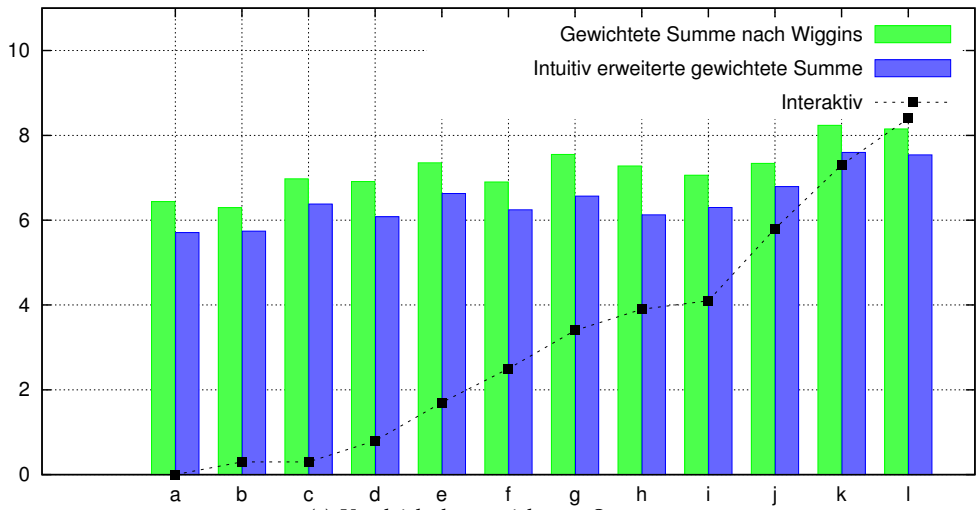
Tabelle 10.3: Beispiele für gewichtete Summen



(a) Vergleich der Entscheidungsbäume



(b) Vergleich der neuronalen Netze



(c) Vergleich der gewichteten Summen

Abbildung 10.7: Bewertungen der Individuen in Abbildung 10.5 und 10.6

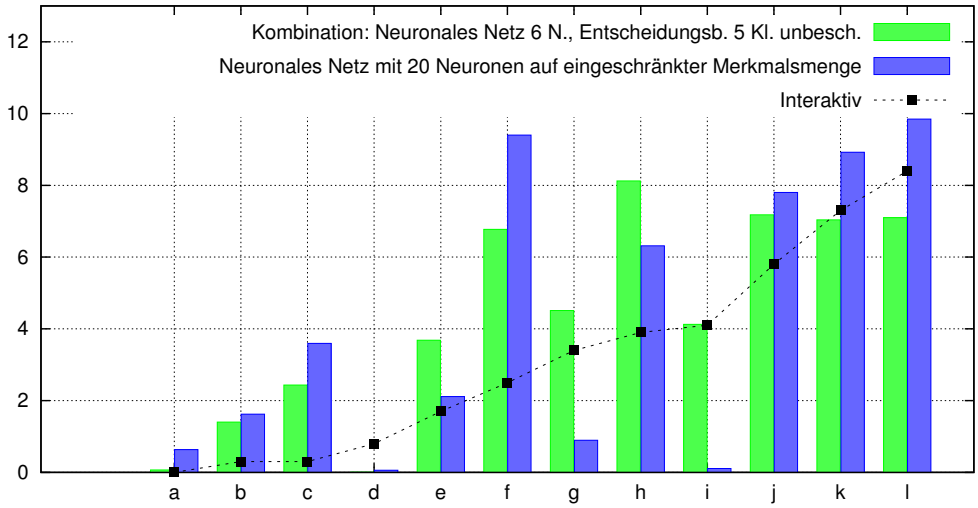


Abbildung 10.8: Weitere Bewertungen der Individuen in Abbildung 10.5 und 10.6

Kapitel 11

Erfahrungen mit verschiedenen Parameterbelegungen

In den bisherigen Kapiteln sind die Komponenten des evolutionären Algorithmus vorgestellt worden. Hierbei wurde auch auf die Vor- und Nachteile der einzelnen Verfahren eingegangen. In diesem Kapitel werden nun Probleme und Erfahrungen behandelt, welche systemübergreifend zu sehen, also nicht auf eine Komponente beschränkt sind, sondern gerade durch das Zusammenspiel der einzelnen Teile auftreten und auffallen.

Hierzu ist die Sichtweise hilfreich, dass die Initialisierung sowie die Veränderung der Individuen durch Rekombination und Mutation als kreative Komponente betrachtet wird. Die Bewertung dagegen ist notwendig, um diese Variationen in musikalisch sinnvoller Form zu halten. Daraus ergibt sich die Frage, ob die initialen Individuen bereits eine gefällige Struktur haben müssen, oder ob es möglich ist, diese durch den Evolutionsprozess herbeizuführen. Ebenso soll die Funktion der unterschiedlichen Rekombinations- und Mutationsoperatoren beleuchtet werden.

Des weiteren erscheint es sinnvoll, dass eine Generation mehrere, nicht vollkommen identische Individuen enthält, eine gewisse Diversität der Population also vorhanden ist. Dies widerstrebt jedoch dem üblichen Ansatz eines evolutionären Algorithmus, *eine* optimale Lösung zu finden.

Die geschilderten Erfahrungen basieren auf der Arbeit mit dem System während der Entwicklung und sind aus Gründen des Umfangs dieser Arbeit nicht durch statistische Untersuchungen belegbar. Sie bieten jedoch einen Anhaltspunkt für die Einstellungen des Systems sowie weitere Analysen.

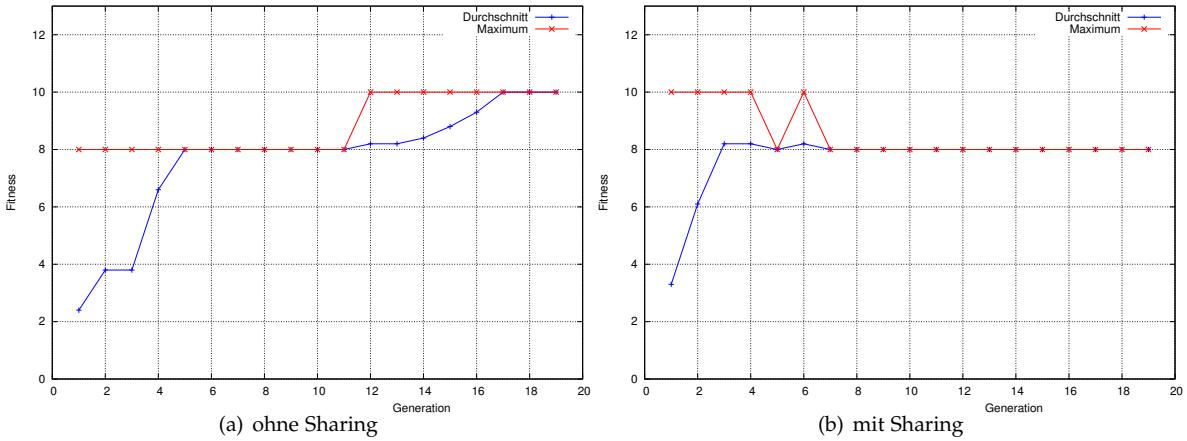
11.1 Diversität der Population

Um die initiale Verschiedenheit der Individuen zu erhalten, ist mit den in Abschnitt 2.2.4 beschriebenen Verfahren experimentiert worden. Als Ähnlichkeitsfunktion kommt die in Abschnitt 7.4 beschriebene zum Einsatz.

Die Nutzung von Fitness Sharing hat sich als nicht sinnvoll erwiesen. Es ist keine Erhöhung der Diversität der Population im Gegensatz zur Evolution ohne diese Technik zu beobachten. Vorteilhafter sieht die Nutzung von Crowding aus. Durch diese Methode halten sich durch die Gestaltung der Ähnlichkeitsfunktion insbesondere bezogen auf den Rhythmus bestimmte Merkmale in der Evolution über mehrere Generationen hinweg. Hierdurch bekommt die Initialisierung eine wichtigere Rolle, da sich die dort gebildeten Motive über viele Iterationen von Rekombination und Mutation hinweg halten.

Die Entwicklung der Fitness bei der Evolution ohne Crowding ist in Abbildung 11.1 zu sehen. Im Verhältnis zu der Entwicklung in einer Evolution mit Crowding, wie in Abbildung 11.6(a) zu sehen ist, fällt das frühe Erreichen von hohen Fitnesswerten auf. Bemerkenswert ist auch, dass bei Nutzung von Fitnesssharing die hier dargestellte, nicht skalierte Fitness abnehmen kann, da entsprechend einer skalierten Fitness selektiert wird.

Zur Verdeutlichung sind Evolutionen durchgeführt worden, wobei als Bewertungsfunktion ein auf der in Abschnitt 10.2.2 erwähnten Beispielmenge erstellter beschnittener Entscheidungsbaum mit 11 Fitnessklassen genutzt wird. Es sind alle Mutationsfunktionen aktiviert, die Mutationswahrscheinlichkeiten



(a) ohne Sharing (b) mit Sharing

Abbildung 11.1: Entwicklung der Fitness ohne Crowding

betragen 0,01. Intermediäre Rekombination wird vermieden, Ein-Punkt-Rekombination erzeugt 10 Nachfolgeindividuen je Population. Das Selektionsverfahren ist deterministisch. Die initialen Individuen sind in Abbildung 11.2 zu sehen. In Abbildung 11.3 sind die entstehenden Individuen nach 20 Generationen ohne Fitnesssharing gezeigt, in Abbildung 11.4 ist das Ergebnis mit Fitnesssharing zu sehen. Abbildung 11.13 zeigt evolvierte Individuen unter Nutzung von Crowding. Dort ist deutlich die wesentlich höhere Diversität sowie die Kombination von Merkmalen mehrerer initialer Individuen zu erkennen.

Es lässt sich festhalten, dass Crowding die Qualität der Evolution mit automatischen Klassifikationsverfahren verbessert. Bei interaktiver Bewertung kann es sinnvoll sein, auf Crowding zu verzichten, da dieses den kreativen Prozess verlangsamt und damit die Notwendigkeit schafft, mehr Individuen zu bewerten.

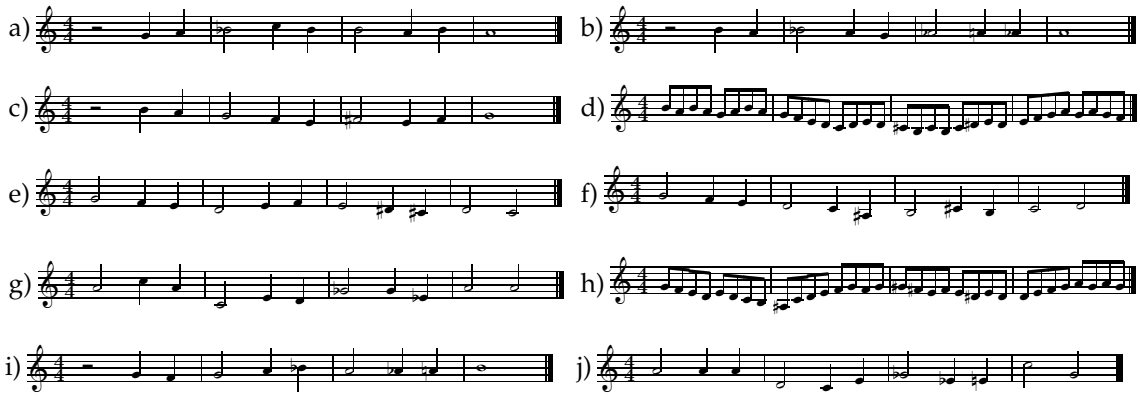


Abbildung 11.2: Initiale Individuen für Evolutionsbeispiele

11.2 Mutations- und Rekombinationsoperatoren im Kontext

Die beiden zur Verfügung stehenden Rekombinationsoperatoren sind in Abschnitt 9.1 erklärt. Die Bedeutung der intermediären Rekombination ist auf die Veränderung des Melodieverlaufs beschränkt, da der Rhythmus eines Individuums übernommen wird. Damit unterstützt dieser Operator, dass im Verlauf der Evolution die Individuen nahezu denselben Rhythmus erhalten. Eine weitere Wirkung ist,

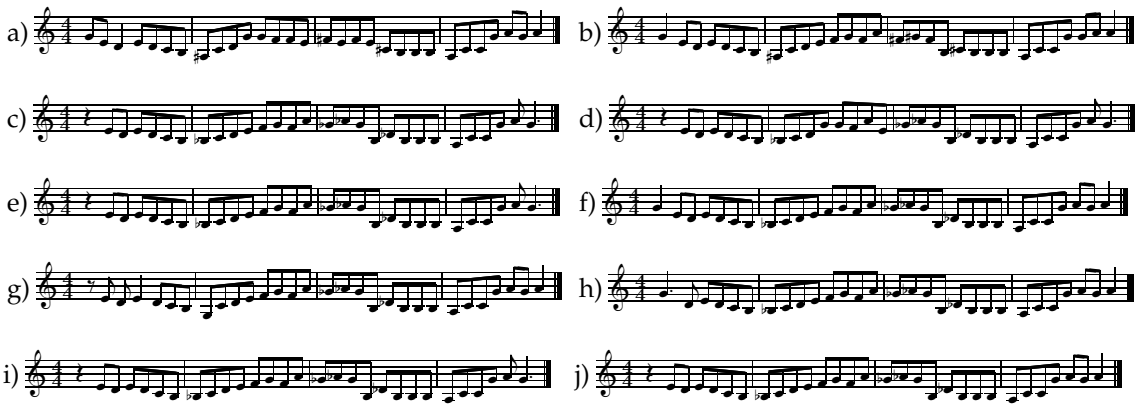


Abbildung 11.3: Ergebnis einer Evolution ohne Fitnesssharing unter Nutzung eines beschnittenen Entscheidungsbaums mit 11 Fitnessklassen nach 20 Generationen

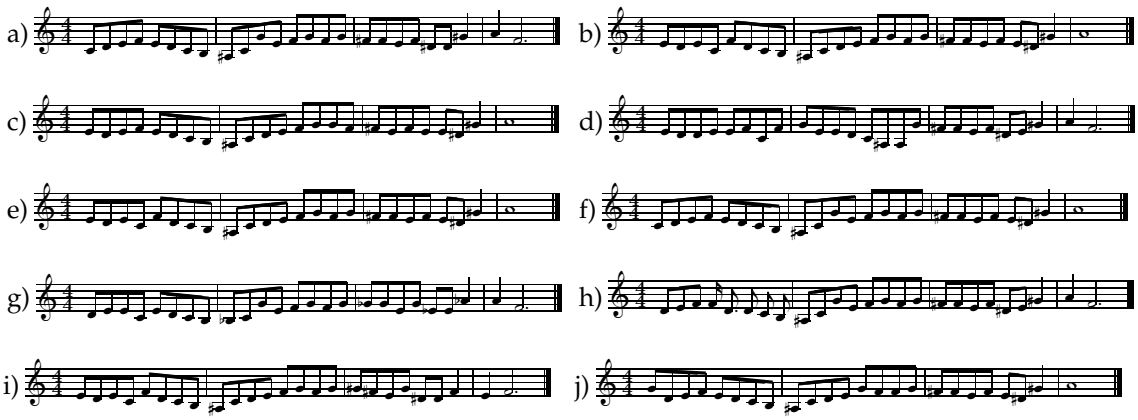


Abbildung 11.4: Ergebnis einer Evolution mit Fitnesssharing unter Nutzung eines beschnittenen Entscheidungsbaums mit 11 Fitnessklassen nach 20 Generationen

dass die Melodien in wenigen Generationen nahezu nur noch Tonrepetitionen darstellen. Dieser Operator sollte daher nicht zum Einsatz kommen.

Der Ein-Punkt-Rekombinationsoperator unterstützt dagegen neben einer Veränderung der Melodie auch die Variation des Rhythmus. Häufig ist zu erkennen, dass aus zwei rhythmisch ansprechenden Individuen auch gefällige Nachkommen generiert werden.

Bei den Mutationsoperatoren ist zur Beurteilung in Rhythmus und Melodie zu unterscheiden.

Die elementaren Operatoren zur Rhythmusänderung sind das Verschieben, das Ersetzen einer Note durch zwei kürzere (im folgenden auch „Auftrennen“ genannt) sowie das Verbinden von Noten. Das Verschieben eines Tones kann den Eindruck des Rhythmus sehr stark verändern. Wenn zum Beispiel eine $\frac{1}{4}$ Note um $\frac{1}{16}$ verschoben wird, hat dies häufig eine eher negativ wahrgenommene Wirkung, da hierdurch leicht Betonungen und rhythmische Gefüge zerstört werden. Diese Probleme werden jedoch durch die Bewertungsfunktionen im allgemeinen gut erkannt, so dass solche Individuen häufig nicht überleben und Verschieben von Tönen eher innerhalb eines Takts das Erreichen der nächsten Generation möglich macht.

Verbinden und Auftrennen von Tönen hat gegensätzliche Wirkung. Das Auftrennen sorgt so zum Beispiel nach einer Initialisierung mit nur $\frac{1}{4}$-Noten für rhythmische Variationen und zum Teil interessante Motive. Fehlt jedoch das Verbinden, können, insbesondere ohne Crowding, recht schnell viele Töne

dieselbe, kürzestmögliche Tonlänge erhalten. Verbinden von Tönen ohne Auftrennen stellt nicht so große Probleme dar, werden die daraus entstehenden rhythmischen Unansehnlichkeiten recht zuverlässig aussortiert. Eine Kombination beider Verfahren mit gleicher Mutationswahrscheinlichkeit ist jedoch angeraten.

Das elementare Verfahren zur Änderung des Melodieverlaufs stellt die Punktmutation dar. Die durch diese vollzogenen Veränderungen einer Melodie sind weniger stark auffallend, als die Veränderung des Rhythmus. Problematische Veränderungen, wie zum Beispiel das Erhöhen einer betonten Note von einem akkordeigenen Ton zu einem nicht akkordeigenen Ton wird von den Zielfunktionen zuverlässig erkannt und verhindert.

Da die bisher beschriebenen Methoden eher kleine Änderungen am System vornehmen, ist das Vorhandensein der Mutationsoperatoren höherer Ordnung nützlich. Deren Wirkung tritt umso stärker in den Vordergrund, umso größer die zu verändernden Abschnitte ausfallen. Sie haben eine gewisse regulierende Wirkung: Sortieren von Tonhöhen kann chaotische Melodien, welche von mehreren großen Sprüngen geprägt sind, in Ordnung bringen, Rotieren und Drehen korrigiert auffallende Schnittpunkte, welche durch Rekombination zustande gekommen sein können.

Es ist naheliegend, grundsätzlich alle Mutatoren anzuschalten. Alleine bei der Evolution ohne Crowding, insbesondere bei interaktiver Bewertung ist die Wahl der rhythmusverändernden Verfahren wohlüberlegt anzuwenden.

Grundsätzlich sollten bei interaktiver Bewertung eher hohe Mutationsraten und große Mutationsbereiche genutzt werden, damit der Verlauf der Evolution nicht zu ermüdend ist. Hohe Mutationsraten verursachen bei dem Verschieben von Tönen allerdings wiederum wenig gefällige Melodien. Hier gilt es, ein Mittelmaß zu finden.

Bei automatischer Bewertung darf man etwas vorsichtiger mit Mutationsraten sein, stellt es doch bei zu geringer Kreativität kein Problem dar, einige weitere Generationen zu erstellen.

11.3 Initialisierungsverfahren im Kontext

Je nach Wahl der Mutationsoperatoren sowie der Festlegung, ob Crowding genutzt wird, spielen die initialen Individuen eine mehr oder weniger große Rolle. So ist eine zufällige Festlegung des Rhythmus mit minimaler Ereignislänge von 0,5 und maximaler Länge von 2 ein Garant für eine Evolution, welche keine gefälligen Melodien erzeugt. Ohne Crowding werden die wenig sinnvollen Strukturen zwar in gewissem Rahmen eliminiert, allerdings benötigt dies viele Generationen, so dass das Ergebnis der Evolution identische Individuen sind. Bei Crowding tritt das Problem noch stärker in den Vordergrund, da die rhythmische Struktur der initialen Individuen erhalten bleibt. Dadurch gelingt die Eliminierung sinnloser rhythmischer Strukturen nicht.

In jedem Fall sollte daher auf einen gewissen Anteil rhythmisch gefälliger Individuen in der Initialisierung Wert gelegt werden. Hierzu eignen sich Markovketten und Mustergruppen sehr gut. Es sollten allerdings unterschiedliche Rhythmen vorhanden sein, um durch Rekombination die Möglichkeit auf innovative Strukturen zu erhalten. Insgesamt erscheint die Chance auf gefällige Rhythmen höher, wenn von ruhigen Rhythmen, also solchen mit wenigen unterschiedlichen, langen Tönen, ausgegangen wird. Daher kann es sinnvoll sein, grundsätzlich das Verfahren zur zufälligen Erzeugung auf minimale und maximale Länge von $\frac{1}{4}$-Noten einzustellen und mit anderen Verfahren zur Initialisierung der Population zu kombinieren.

Die Initialisierung der Melodie erscheint weniger bedeutungsvoll für den Verlauf der Evolution, da sie das Erreichen eines hohen Fitnesswerts nicht merklich beeinflusst. Allerdings hängt von ihr in bedeutendem Maße der Charakter der Individuen ab. Besonders deutlich wird dies bei der Verwendung von absoluten Markovketten, da hier der verwendete Tonumfang vorgegeben wird. Dieser Einfluss findet sich auch nach vielen Generationen in den Melodien. Entsprechend übernimmt die Initialisierung mit einem Random Walk die Funktion der Erzeugung von Melodien mit wenigen Sprüngen demgegenüber

die normalverteilt zufällige Festlegung gestellt werden kann. Entsprechend der Wahl dieser Methoden kann die Charakteristik der entstehenden Melodien gut beeinflusst werden.

Es ist festzustellen, dass sich als initiale eher einfach strukturierte Individuen eignen. Der Ansatz, mit Hilfe von Markovketten den Beispielmelodien ähnliche Verläufe und Rhythmen zu erzeugen, wird durch die Evolution verhindert. Allerdings bieten diese eine Methode, eine aus diversen Individuen bestehende Startpopulation zu erhalten.

11.4 Bewertung im Kontext

Im Voraus sei bemerkt, dass es sich bei diesem evolutionären Algorithmus zwar um ein häufig konvergierendes Verfahren handelt. Allerdings steht am Ende von diesem nicht zwingend eine gefällige Melodie, auch wenn sie eine hohe Fitness erhalten hat. Dies gilt für alle bisher genutzten Bewertungsfunktionen. Vielmehr ist die Evolution als zielgerichteter, kontrollierter, kreativer Prozess zu verstehen, der allerdings auch Richtungen einschlagen kann, die weniger gefällig sind.

Die Bewertungsfunktionen tragen hierbei neben dem Crowding mit der entsprechenden Ähnlichkeitsfunktion die Hauptrolle der Kontrolle der Kreativität. Zur Veranschaulichung sind in den Abbildungen 11.7 bis 11.18 die Ergebnisse unterschiedlicher Evolutionen auf Basis der initialen Individuen in Abbildung 11.2 gezeigt. Sie sind jeweils mit Crowding und allen Mutationsoperatoren durchgeführt, wobei die Mutationswahrscheinlichkeiten den Wert 0,01 haben. In diesen Abbildungen sind auch die erreichten Fitnesswerte gezeigt. Um einen Anhaltspunkt zur subjektiv wahrgenommenen Güte zur Verfügung zu stellen, ist zusätzlich zu allen Individuen eine interaktive Bewertung von mir vorgenommen worden.

Zunächst ist auffallend, dass bei allen Evolutionen in ähnlicher Weise die rhythmischen Strukturen aus den initialen Individuen erhalten bleiben. Des weiteren entspricht kein Verfahren tatsächlich der subjektiven Bewertung (selbstverständlich bis auf die interaktive Evolution). Dies ist jedoch nicht verwunderlich, da der Geschmack eines Menschen zeitlich nicht konstant bleibt und sich auch seit der Bewertung der Beispielindividuen, aus denen die Entscheidungsbäume erstellt und mit denen die neuronalen Netze trainiert sind, geändert haben kann.

Allerdings sind durchaus Melodien zu sehen, welche schwierig als akzeptabel gelten können, allerdings eine hohe Fitness erreicht haben wie zum Beispiel 11.10j), 11.11j), 11.12e) und 11.14b). Nicht jeder Evolutionslauf erzeugt also mehrere wohl klingende Individuen, besonders unschöne Ergebnisse sind die in 11.14. Hier erscheint die Harmoniezugehörigkeit der Töne schlicht nicht betrachtet zu sein. Als Gegenteil hierzu kann eine Evolution gesehen werden, welche nur das Merkmal „Harmonicity“ und dieses skaliert als Fitnesswert betrachtet (siehe 11.18). Hier sind trotz der extremen Einfachheit recht angenehme Melodien erzeugt. Allerdings sind diese bezüglich des Melodieverlaufs eher langweilig. Des weiteren ist der Rhythmus hier nicht durch die Fitnessfunktion kontrolliert. Angenehm und sicherlich spannender sind die Melodien in 11.16. Insgesamt scheint die Fitnessfunktion mit der eingeschränkten Merkmalsmenge möglicherweise aufgrund einer besseren Generalisierungsfähigkeit günstiger in der Evolution zu wirken als die anderen künstlichen neuronalen Netze.

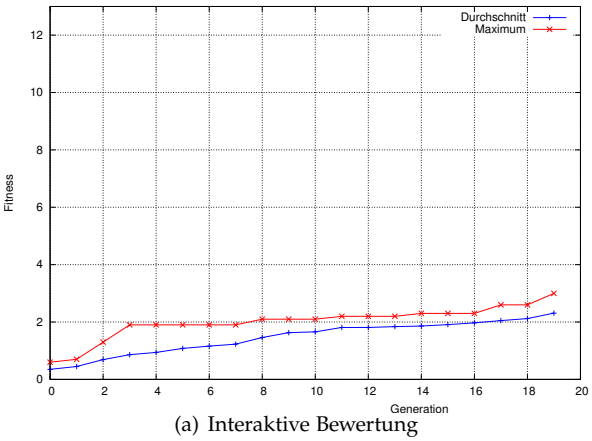
Eine Sonderrolle nehmen die in der interaktiven Evolution entstehenden Individuen ein. Hier ist aufgrund der Tatsache, dass der bewertende Benutzer die jeweils erzeugten Melodien mit den vorherigen vergleicht, die Fitness im Verhältnis zu den subjektiven Bewertungen der Individuen der anderen Verfahren sehr gering.

Die Entwicklung der Fitness über die Generationen hinweg ist in Abbildung 11.5 und 11.6 für den Durchschnitt sowie das beste Individuum der jeweiligen Population gezeigt. Alle dargestellten Verläufe bis auf die interaktive Bewertung machen deutlich, dass sehr zügig wenigstens ein Individuum der Population einen Grenzwert erreicht. Dies ist allerdings häufig kein günstiges Abbruchkriterium für die Evolution, da zu diesem Zeitpunkt noch keine als kreativ zu bezeichnende Kombination von Teilstücken der initialen Individuen durch Rekombination stattgefunden hat. Ein weiteres Argument, die Evolution länger durchzuführen ist, dass eine hohe Fitness nicht unbedingt auf eine dem persönlichen

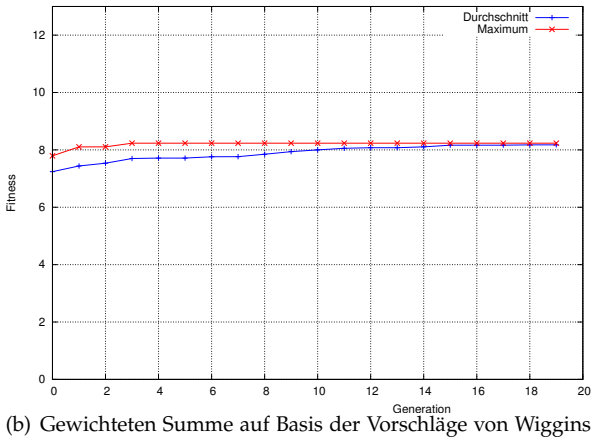
Geschmack entsprechende Melodie schließen lässt und mehreren Iterationen mehrere Individuen mit hoher Fitness zur Verfügung stehen, welche zum Teil auch Variationen voneinander sein können. So steigt die Wahrscheinlichkeit, dass ein dem Benutzer gefälliges Musikstück generiert wird. Dies ist bei den Evolutionen in Abbildung 11.1 nicht gültig, ist doch das Ansteigen der durchschnittlichen Fitness oft als Annäherung an das beste Individuum zu sehen.

Neben diesen Hinweisen zur Interaktion des Benutzers ergeben sich aus den Darstellungen der Fitnessentwicklung Beschreibungen zum Verhalten der Zielfunktionen. So ist bei der interaktiven Bewertung eine stetige Steigerung der Fitness ohne Erreichen eines Grenzwerts innerhalb der dargestellten Generationen erkennbar (Abbildung 11.5(a)). Dies ist verständlich, da diese Zielfunktion sicherlich als am differenzierendsten bezeichnet werden kann. Die Entwicklungen bei Bewertung mit Entscheidungsbäumen lässt die Diskretisierung der Fitness erkennen (Abbildungen 11.5(d),11.5(e),11.5(f),11.6(a)). So sind hier Sprünge von einem Fitnesswert zum nächsten erkennbar. Dies ist auch bei dem neuronalen Netz mit eingeschränkter Merkmalsmenge (Abbildung 11.6(d)) der Fall. Eine differenziertere Bewertung und damit stetigere Steigerung dieser ist bei den neuronalen Netzen mit vollständiger Merkmalsmenge (Abbildungen 11.6(b) und 11.6(c)) zu beobachten.

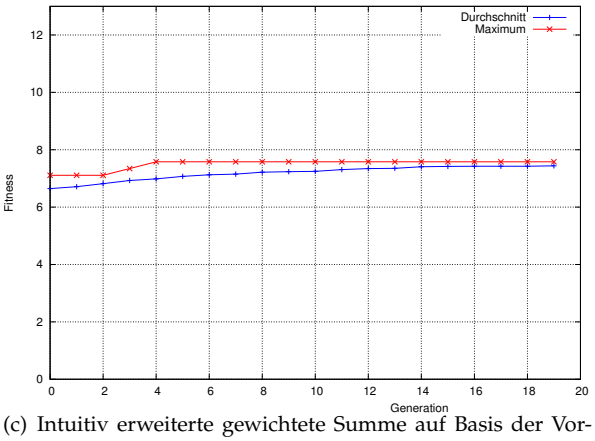
Es ist zu erwarten, dass die Güte der Bewertungsmethoden im Verlauf der Evolution von dem Geschmack des Benutzers sowie von der Beispielmenge zur Erstellung von Baum und Netz abhängt. In den hier dargestellten Fällen agiert der beschnittene Entscheidungsbaum mit 11 Fitnessklassen sowie das neuronale Netz mit eingeschränkter Merkmalsmenge recht gut. Dies ist möglicherweise jedoch bei anderen Initialisierungsverfahren nicht so und muss auf diese neu abgestimmt werden.



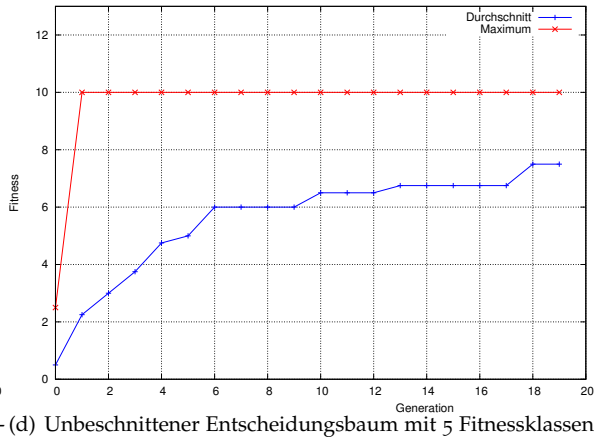
(a) Interaktive Bewertung



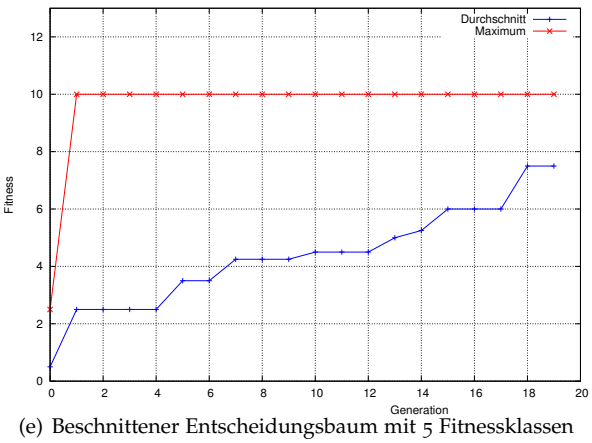
(b) Gewichteten Summe auf Basis der Vorschläge von Wiggins und Papadopoulos (1998)



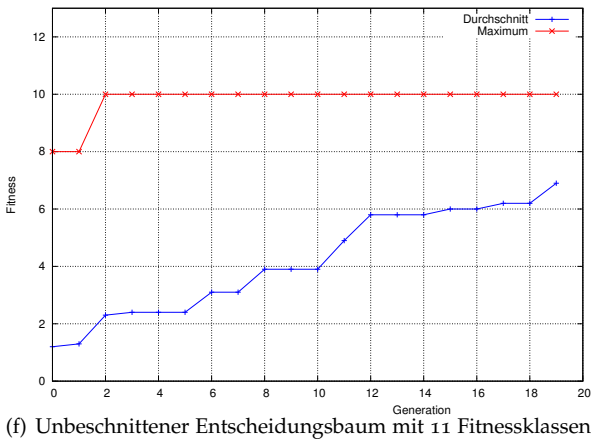
(c) Intuitiv erweiterte gewichtete Summe auf Basis der Vorschläge von Wiggins und Papadopoulos (1998)



(d) Unbeschnittener Entscheidungsbaum mit 5 Fitnessklassen

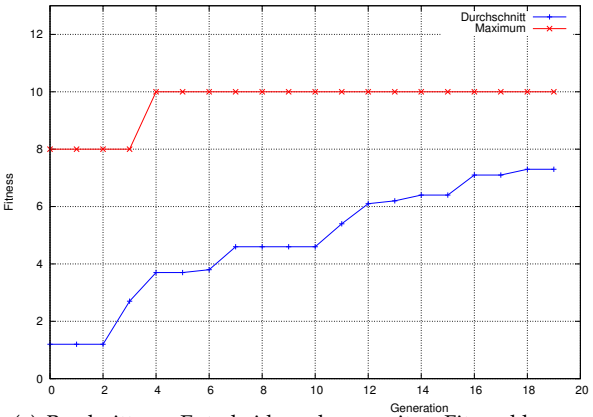


(e) Beschnittener Entscheidungsbaum mit 5 Fitnessklassen

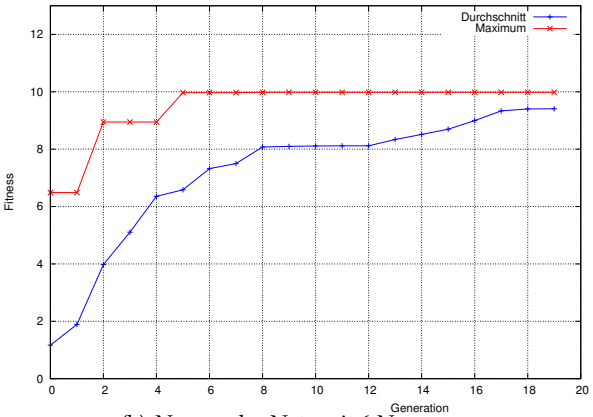


(f) Unbeschnittener Entscheidungsbaum mit 11 Fitnessklassen

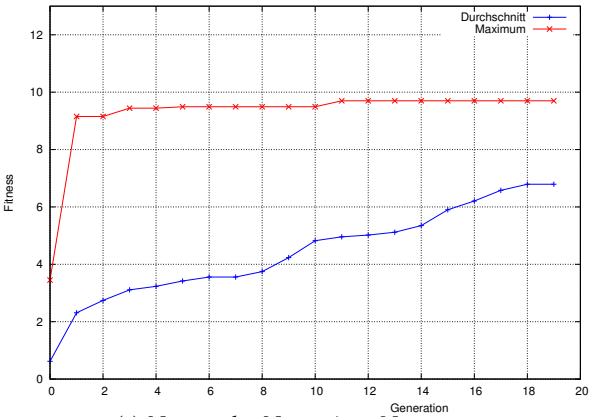
Abbildung 11.5: Entwicklungen der Fitness bei Evolutionen mit Crowding mit unterschiedlichen Bewertungsfunktionen



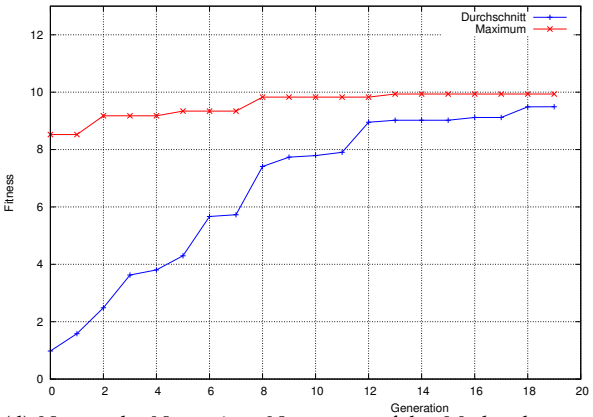
(a) Beschnittener Entscheidungsbaum mit 11 Fitnessklassen



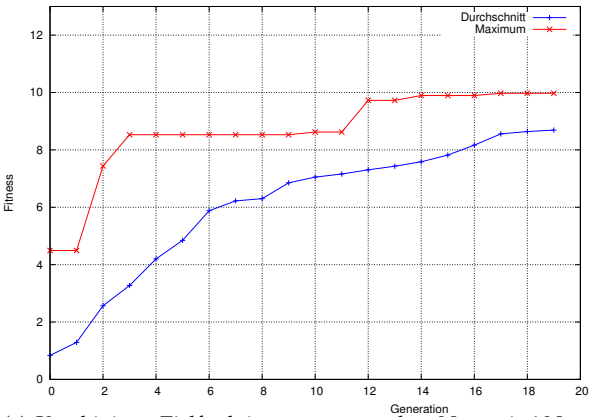
(b) Neuronales Netz mit 6 Neuronen



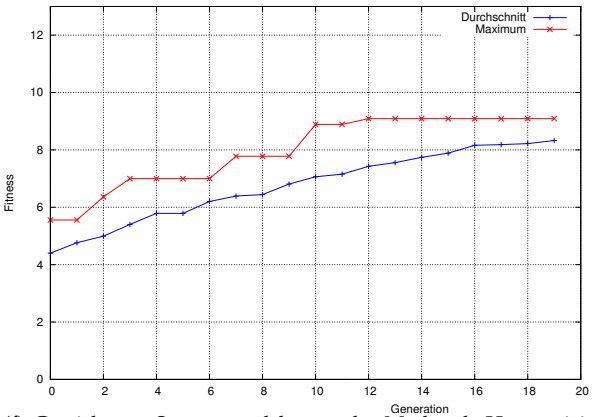
(c) Neuronales Netz mit 35 Neuronen



(d) Neuronales Netz mit 20 Neuronen auf den Merkmalen aus Tabelle 10.2



(e) Kombinierte Zielfunktion aus neuronalem Netz mit 6 Neuronen und unbeschnittenem Entscheidungsbaum mit 5 Fitnessklassen



(f) Gewichteten Summe, welche nur das Merkmal „Harmonicity“ betrachtet

Abbildung 11.6: Entwicklungen der Fitness bei Evolutionen mit Crowding mit unterschiedlichen Bewertungsfunktionen (Fortsetzung)

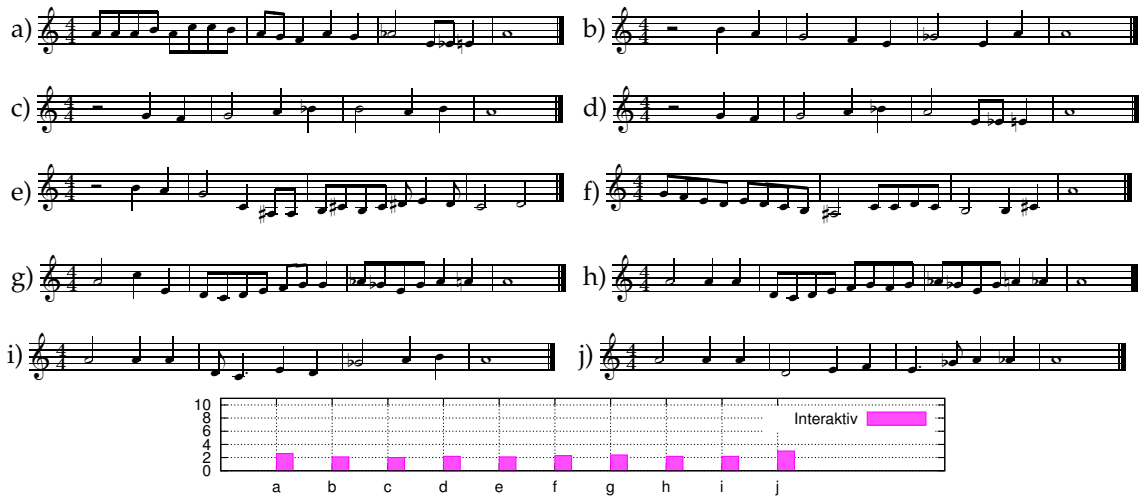


Abbildung 11.7: Ergebnis einer Evolution mit interaktiver Bewertung nach 20 Generationen

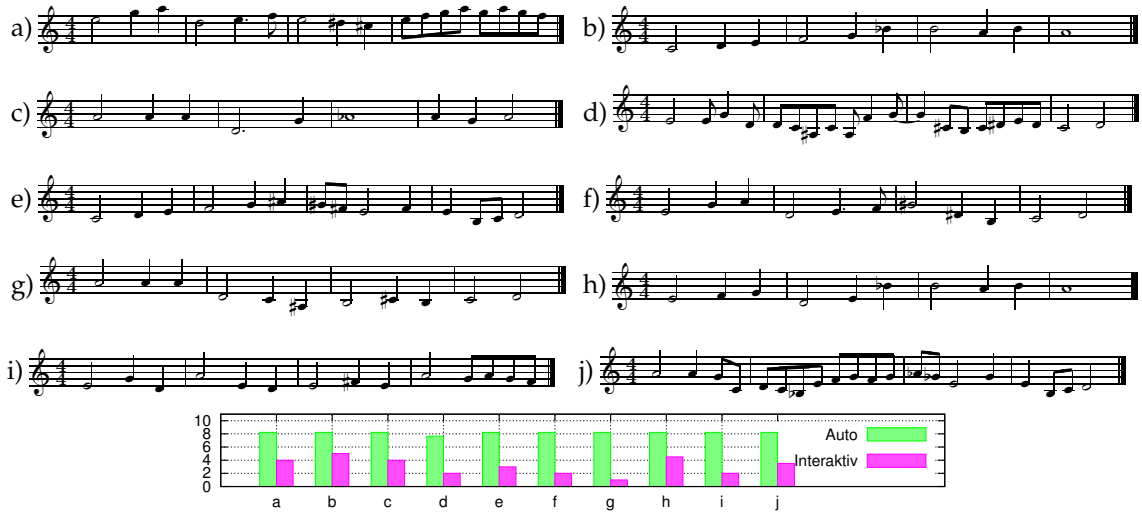


Abbildung 11.8: Ergebnis einer Evolution der gewichteten Summe auf Basis der Vorschläge von Wiggins und Papadopoulos (1998) als Zielfunktion nach 20 Generationen

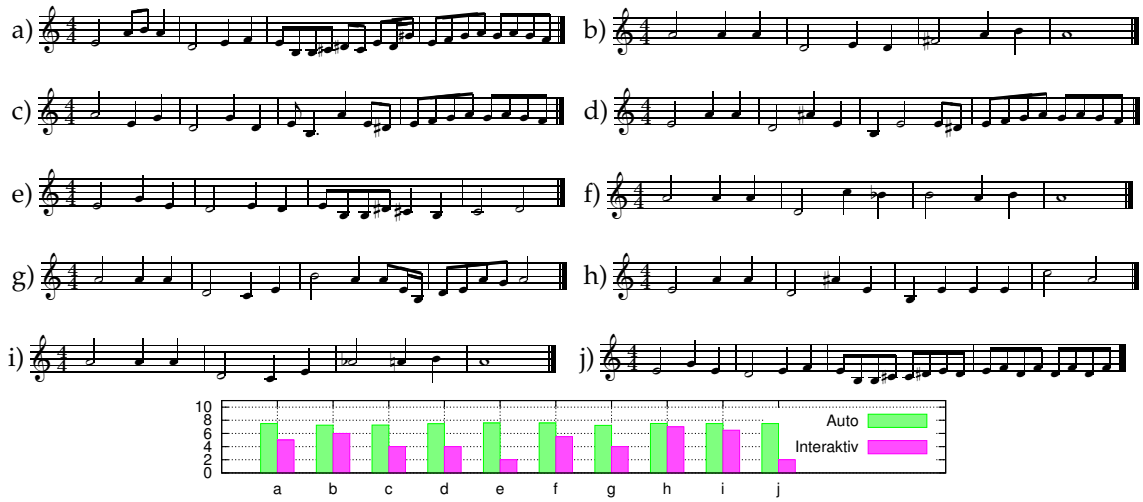


Abbildung 11.9: Ergebnis einer Evolution der intuitiv erweiterten gewichteten Summe auf Basis der Vorschläge von Wiggins und Papadopoulos (1998) als Zielfunktion nach 20 Generationen

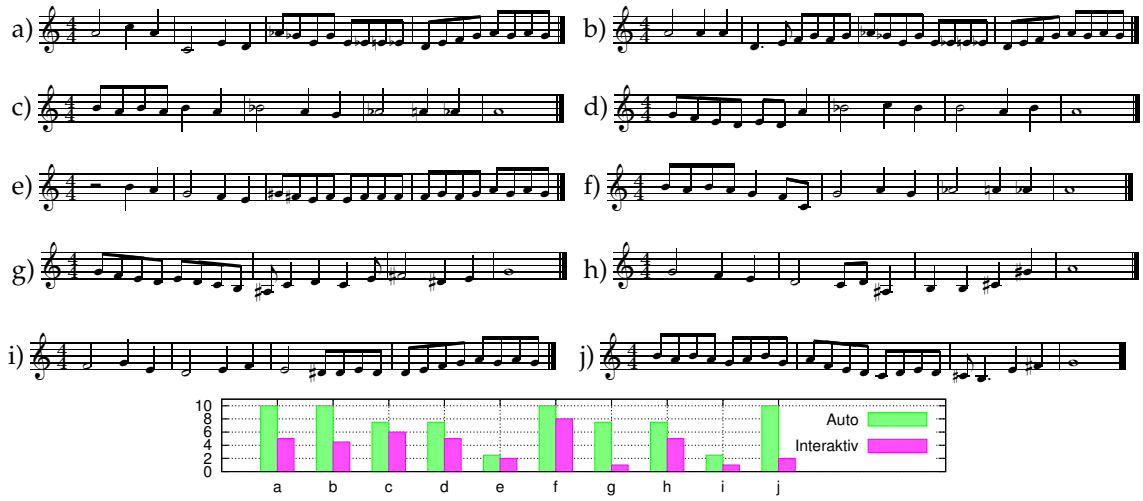


Abbildung 11.10: Ergebnis einer Evolution mit unbeschnittenem Entscheidungsbaum mit 5 Fitnessklassen nach 20 Generationen

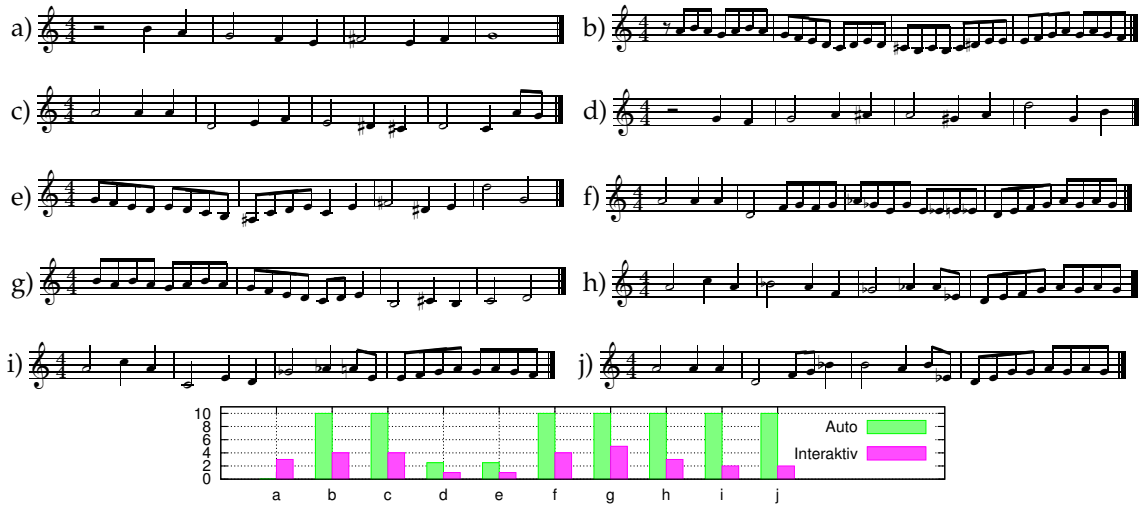


Abbildung 11.11: Ergebnis einer Evolution mit beschnittenem Entscheidungsbaum mit 5 Fitnessklassen nach 20 Generationen

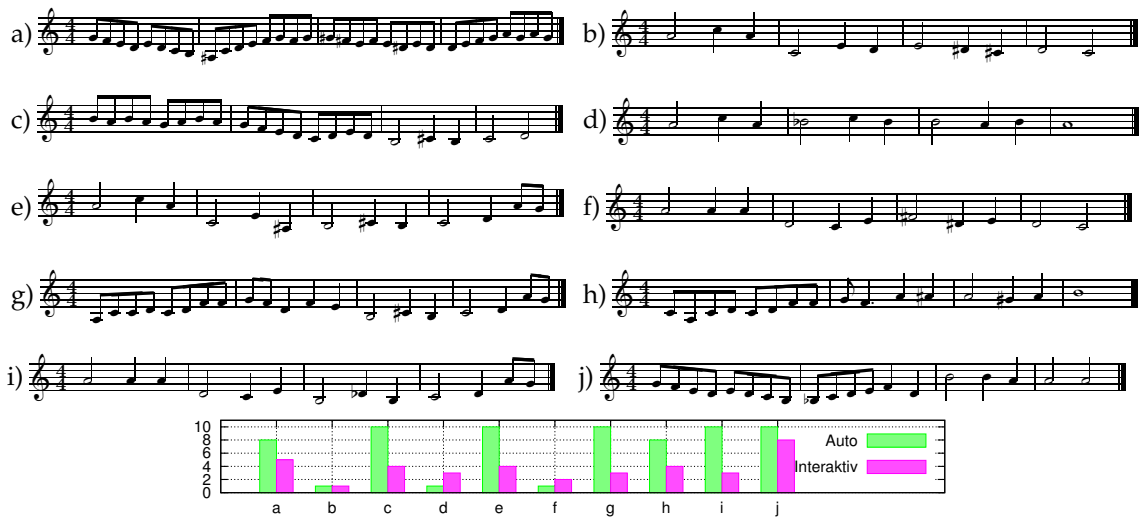


Abbildung 11.12: Ergebnis einer Evolution mit unbeschnittenem Entscheidungsbaum mit 11 Fitnessklassen nach 20 Generationen

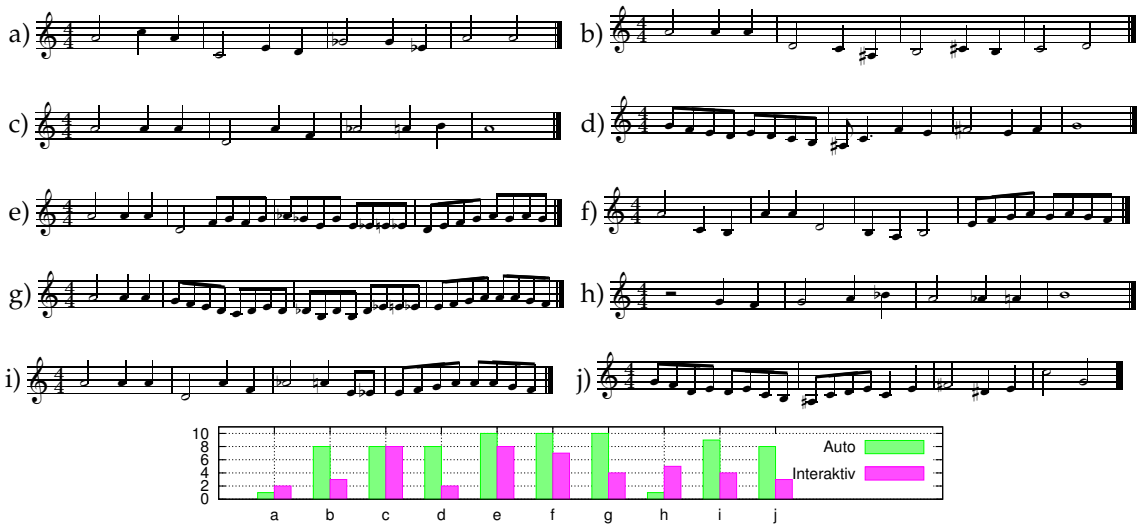


Abbildung 11.13: Ergebnis einer Evolution mit beschnittenem Entscheidungsbaum mit 11 Fitnessklassen nach 20 Generationen

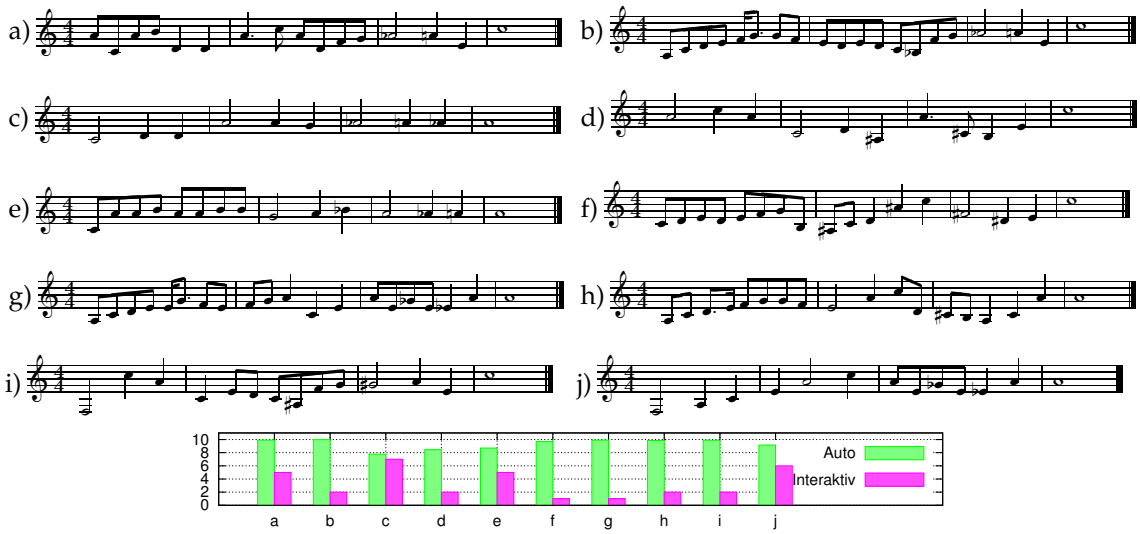


Abbildung 11.14: Ergebnis einer Evolution mit neuronalem Netz mit 6 Neuronen nach 20 Generationen

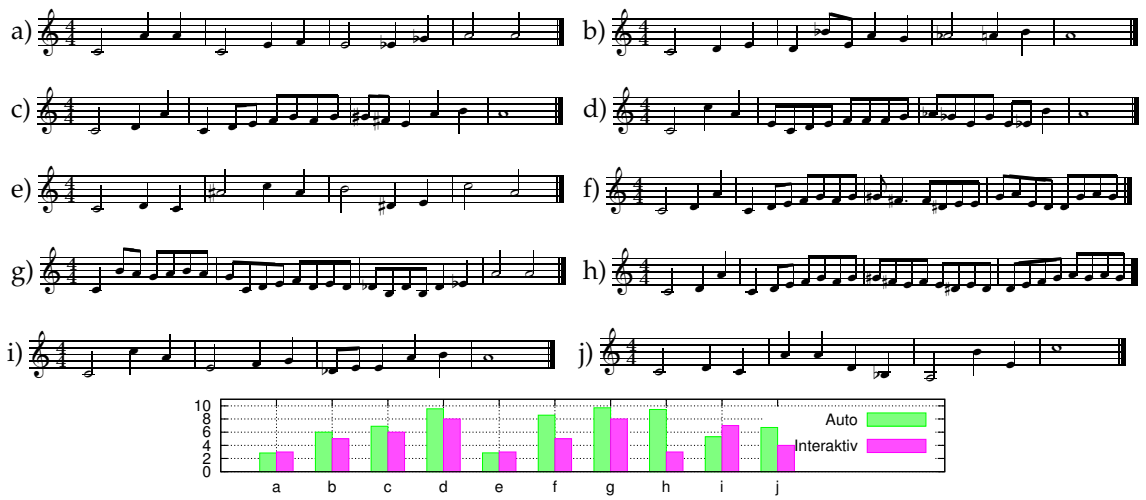


Abbildung 11.15: Ergebnis einer Evolution mit neuronalem Netz mit 35 Neuronen nach 20 Generationen

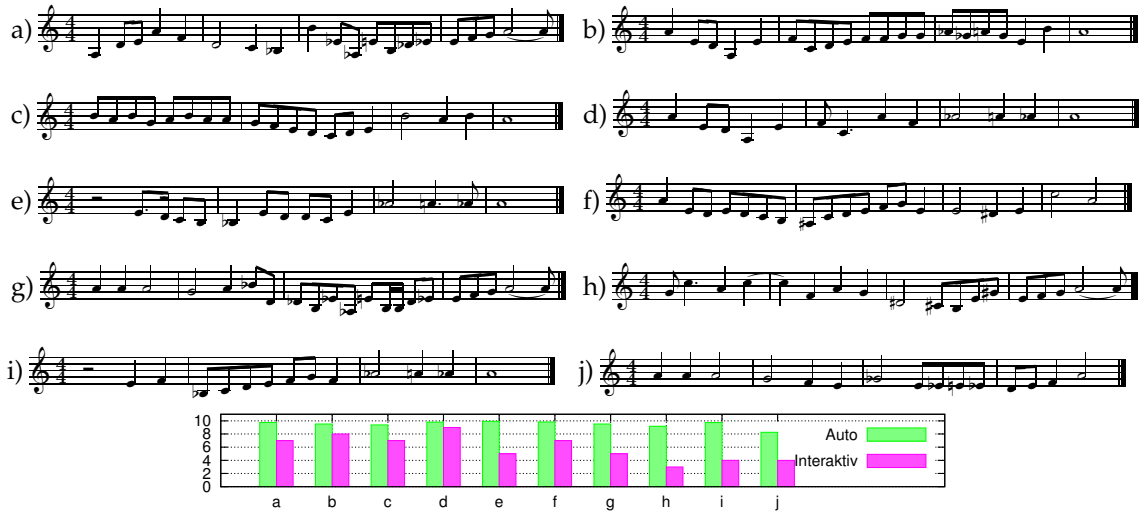


Abbildung 11.16: Ergebnis einer Evolution mit neuronalem Netz mit 20 Neuronen auf den Merkmalen aus Tabelle 10.2 nach 20 Generationen

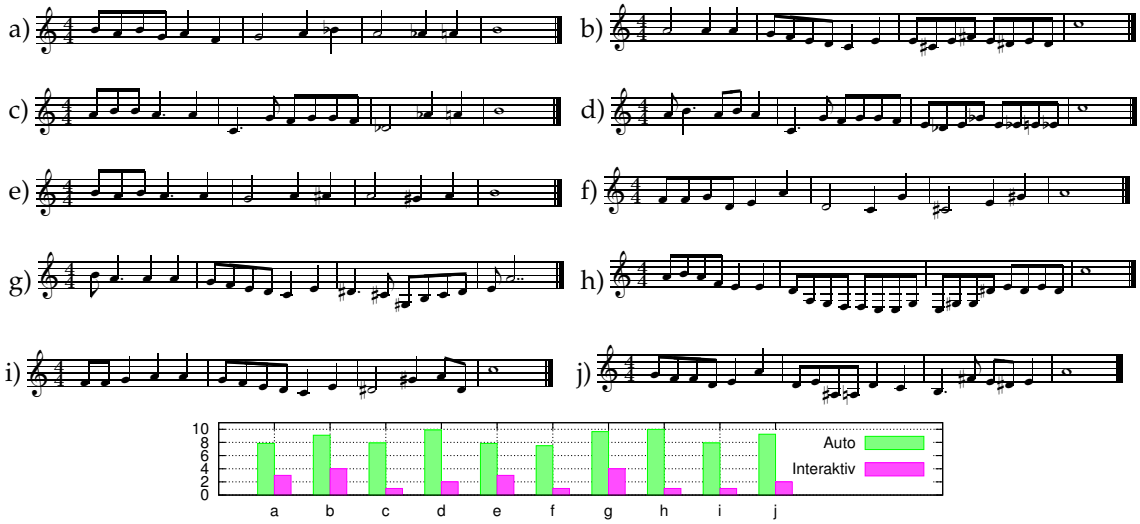


Abbildung 11.17: Ergebnis einer Evolution mit kombinierter Zielfunktion aus neuronalem Netz mit 6 Neuronen und unbeschnittenem Entscheidungsbaum mit 5 Fitnessklassen nach 20 Generationen

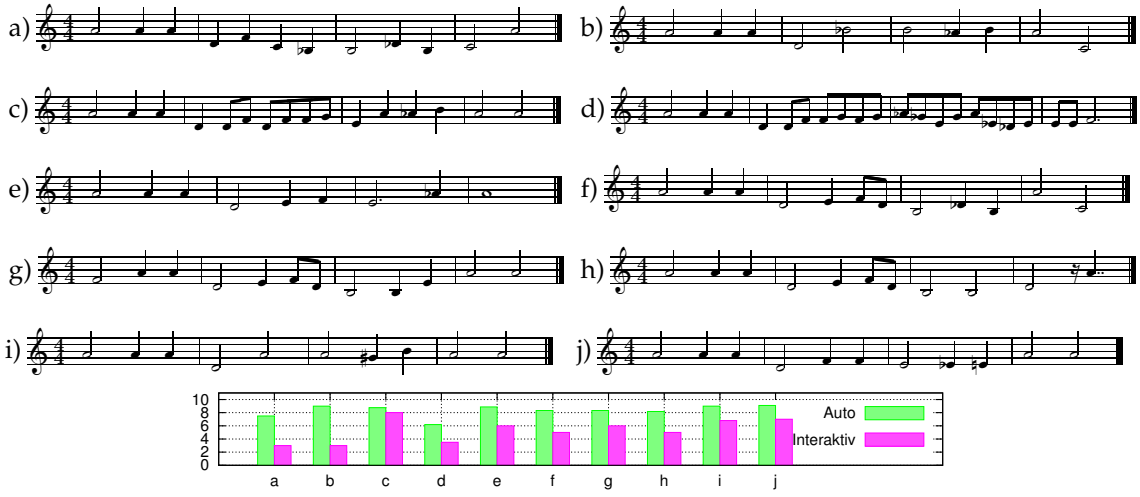


Abbildung 11.18: Ergebnis einer Evolution mit einer gewichteten Summe als Zielfunktion, welche nur das Merkmal „Harmonicity“ betrachtet, nach 20 Generationen

Kapitel 12

Zusammenfassung und Ausblick

In dieser Arbeit wird ein System zur automatischen Komposition von Melodien vorgestellt. Zugrundeliegend ist das Paradigma eines evolutionären Algorithmus, dessen einzelne Komponenten beleuchtet werden. So bestehen verschiedene Möglichkeiten, die initialen Individuen zu erzeugen. Diese sind die Verwendung von Markovketten, Random Walks, Mustergruppen und anderen randomisierten Verfahren. Es sind unterschiedliche Rekombinations- und Mutationsoperatoren entwickelt und implementiert. Die Güte der während des evolutionären Prozess entstehenden Melodien wird durch künstliche neuronale Netze, Entscheidungsbäume, interaktive Bewertung und gewichtete Summen sowie Kombinationen dieser beurteilt. Dies sowie der Einsatz von Crowding mit einer besonderen Ähnlichkeitsfunktion lässt die Evolution kontrolliert ablaufen, was erst die Möglichkeit schafft, gefällige Melodien zu erhalten.

Neben der Beschreibung der Grundlagen der einzelnen Komponenten findet eine ausführliche Erläuterung des Einsatzes im Bereich der Musikverarbeitung und Melodieerzeugung statt. Des weiteren wird anhand von Beispielen die Eignung des Einsatzes erläutert. Im Zuge der Entwicklung der beschriebenen Verfahren sind diese in einer Experimentierumgebung implementiert worden. Es sei dem Leser nahegelegt, selbst die Wechselwirkungen zwischen den unterschiedlich parametrisierbaren Komponenten des evolutionären Algorithmus kennenzulernen. Zwar sind in dieser Arbeit viele Beispiele automatisch erzeugter Melodien abgebildet, es liegt jedoch in der Natur von Musik, dass sie erst bei der akustischen Wahrnehmung wirklich zur Geltung kommt.

Durch entsprechende Parameterkonfigurationen der einzelnen Bestandteile des Systems ist es möglich, ästhetische Melodien zu erzeugen. Hierbei ist zwischen einer erhöhten Wahrscheinlichkeit zum Erhalt kreativer Individuen und der Sicherheit, dass wohlklingende Melodien entstehen, abzuwägen. Je geringer die Einschränkungen durch die Zielfunktionen sind, desto eher entstehen überraschende Ergebnisse, allerdings müssen dann auch mehr weniger wohlklingende Melodien in Kauf genommen werden.

Aus den bereits durchgeführten Experimenten sind einige offene Fragestellungen entstanden, welche im folgenden erläutert werden sollen. Die Erstellung der initialen Individuen muss nicht nur als Zweck zur Nutzung in einem evolutionären Algorithmus zu sehen sein. Vielmehr stellt bereits der Einsatz der Markovketten, aber auch der der Mustergruppen die Möglichkeit dar, aus vorgegebenen Musikstücken neue Melodien zusammenzustellen, zu komponieren. Da hier jedoch ein Schwerpunkt auf den evolutionären Algorithmus gelegt ist, wurde dies nicht weiter verfolgt. So ist die Verwendung der Markovketten getrennt für Rhythmus und Melodieverlauf entworfen. Dies funktioniert hervorragend, um das Ziel zu erfüllen, musikalisch sinnvolle initiale Individuen zu erhalten, stellt doch Rhythmus wie auch Melodieführung jeweils die Struktur der zur Erstellung der Markovkette eingegebenen Musikstücke dar. Allerdings ist zu beobachten, dass im Allgemeinen die Beziehung zwischen diesen beiden Komponenten vernachlässigt ist. Tatsächlich ist dies ein Problem, da zum Beispiel auf betonten, auf langen, auf Noten, auf denen eine Richtungsänderung der Melodieführung stattfindet, meistens harmonieeigene Töne klingen sollten. Dies wird in der vorliegenden Implementierung zwar durch den evolutionären Algorithmus erreicht, würde es jedoch bereits bei der Erstellung der initialen Individuen durch eine Markovkette geschehen, könnte man davon ausgehen, dass der Wiedererkennungswert der zur Erstellung genutzten Musikstücke nicht, wie dies im Moment der Fall ist, verloren geht. Hierzu müsste eine Repräsentation gefunden werden, bei der jeder Zustand innerhalb der Markovkette nicht eine Tonlänge *oder* ein Intervall beziehungsweise eine Tonhöhe darstellt. Vielmehr müssten diese Parameter eines Klangs

kombiniert betrachtet werden können. Ebenso findet zur Zeit keine Betrachtung der Positionen der Töne innerhalb eines Takts oder dem gesamten Musikstück statt. Insbesondere hinsichtlich Betonungen sollten diese Punkte ebenfalls in den Markovketten kodiert werden, da sie entscheidend für die Wahrnehmung von Melodien sind.

In ähnlicher Weise problematisch können die Mutationsfunktionen wirken, da die Abschnitte, auf denen einige der Operatoren agieren, diese unabhängig von Betonungen setzen. Es wäre sinnvoll, diese Bereiche mit Bezug auf Takte bestimmten zu können. Dies regt auch die Implementierung von Kopiermutationen an, die Teilstücke auf andere kopieren (möglicherweise auch getrennt in Rhythmus und Melodie), so dass Motivwiederholungen wahrscheinlicher sind.

Des Weiteren wurde bei dem Entwurf des Systems Wert auf möglichst geringe Einschränkungen gelegt. Selbstverständlich musste die maximale Tonlänge bei der Erstellung des Rhythmus nach oben beschränkt werden. Die absoluten Markovketten zur Erstellung der Melodie bilden jedoch den gesamten Tonraum, der von MIDI zur Verfügung gestellt ist, ab. Bei beiden Fällen, Rhythmus wie auch Melodie, ist der Speicherbedarf der Markovketten in der aktuellen Implementierung jedoch recht hoch (bei einer absoluten Markovkette dritter Ordnung werden etwa 268 Megabyte reserviert). Hier sollte optimiert werden, damit die Auswirkung von Markovketten höherer Ordnung getestet werden kann.

Eine zentrale Rolle nimmt der Einsatz des Verfahrens Crowding ein, konnte doch erst damit eine Diversität in der Population erreicht werden, die es möglich macht, am Ende der Evolution deutlich unterschiedliche Melodien erhalten zu können. Des Weiteren trägt sie aber mit der Ähnlichkeitsfunktion, welche einen Schwerpunkt auf die Ähnlichkeit des Rhythmus setzt, dazu bei, dass der initiale Rhythmus über viele Generationen hinweg erkennbar bleibt. Dies ist sinnvoll, weil es ausgesprochen schwierig zu sein scheint, die Gefälligkeit eines Rhythmus durch eine Bewertungsfunktion zu erhalten. Hier sollten weitere Experimente mit anderen Ähnlichkeitsfunktionen folgen, um zu ergründen, ob ähnliche Vorteile auch in Bezug auf die Melodieführung nutzbar sein könnten. Ebenso sollten weitere Versuche mit unterschiedlichen Ähnlichkeitsfunktionen im Zusammenhang mit Fitnesssharing durchgeführt werden.

Bei Vergleich der Nutzung eines neuronalen Netzes mit Teilmengen der Merkmale, welche aus den Individuen extrahiert werden, ist aufgefallen, dass möglicherweise Redundanz vorliegt. Dies ist prinzipiell nicht problematisch. Allerdings nimmt dadurch die Menge der sinnvollen Merkmale ab, so dass nicht eine so genaue Charakterisierung des Individuums möglich ist, wie es mit der selben Anzahl nicht redundanter Merkmale möglich wäre. Daher sollten weitere Merkmale zur Verfügung gestellt werden, wobei auch solche implementiert sein sollten, welche scheinbar nur minimal von den vorhandenen abweichen, da solch kleine Unterschiede möglicherweise signifikante Wirkung haben könnten. Mit einer großen Menge Merkmale sollte eine erneute Untersuchung des Informationsgehalts der einzelnen stattfinden. So könnte eine günstigere, kleinere Menge gefunden werden, welche die Möglichkeit von besseren und auch eher generalisierenden Klassifikationsverfahren ergibt.

Ähnliche Vorteile kann die Nutzung einer größeren Beispielmenge bringen, welche zum Training der neuronalen Netze sowie der Erstellung von Entscheidungsbäumen eingesetzt werden kann. Zwar ist ein Modul implementiert, mit dem die Bewertung und Anreicherung mit Akkordinformationen von MIDI-Dateien einfach von statten geht. Allerdings sind hierzu Midi-Dateien mit exakten Start- und Endpositionen von Tönen nötig. Üblicherweise sind solche aber nicht für genaue Notenschrift optimiert sondern zur akustischen Wiedergabe. Daher besteht nur die Möglichkeit, solche MIDI-Dateien aufwändig nachzubearbeiten oder selbst Melodien aufzunehmen. Auch die implementierte automatische Speicherung während einer interaktiven Evolution kann dies nicht überflüssig machen, da hierbei zunächst eher minderwertige Melodien entstehen. Beispielindividuen mit sehr hoher Fitness sollten auf anderem Wege bereitgestellt werden. Eine solche größere Beispielmenge hat auch den weiteren Vorteil, dass die Relevanz der unterschiedlichen Merkmale, wie oben erwähnt, besser erkannt werden könnte.

Die in dieser Arbeit implementierten Bewertungsfunktionen sind bisher eher subjektiv beurteilt worden, indem ein Vergleich mit einer von mir durchgeführten interaktiven Klassifikation stattgefunden hat. Um zuverlässige Daten über die Güte der automatischen Klassifikationmethoden zu haben, sollte eine diese liefernde Untersuchung durchgeführt werden. Hierzu wäre eine größere Zahl Versuchspersonen

nach der Einschätzung von vielen Melodien zu fragen. Eine solche Durchführung könnte auch mit der Erstellung von Beispielindividuen kombiniert werden.

Weiterhin sollte die Güte der entstehenden Melodien in Abhängigkeit von dem Geschmack der Person, der die Beispielmenge zur Erstellung des Klassifikationsverfahrens bewertet hat, bestimmt werden. Über diese Zusammenhänge konnten bisher nur Vermutungen geäußert werden.

Zu guter Letzt wäre der Entwurf weiterer Bewertungsfunktionen sinnvoll. Hierbei sollten verschiedene Verfahren des Data-Minings zu ihrer Eignung evaluiert werden. Des Weiteren ist eine Kombination von automatischer und interaktiver Klassifikation denkbar, bei der der Benutzer nur in fraglichen Fällen zu Rate gezogen würde. Dies würde die Evolution beschleunigen und damit auch Verbesserungen der Klassifikationsverfahren während der Erstellung der Melodien möglich machen.

Um diese Ziele erfüllen zu können, besteht die Absicht, das vorliegende System stark zu modularisieren, so dass leicht weitere Komponenten angebunden werden können. Über den Stand dieser Entwicklung wird auf `http://www.roman-klinger.de` zu lesen sein. Dort findet sich auch eine detaillierte Benutzeranleitung zum System sowie folgende Dokumentation der Weiterentwicklung.

Anhang A

Übersicht über die Bedienungsoberfläche

Im folgenden ist ein Überblick über die Bedienungsoberfläche des implementierten Systems mit kurzen Erläuterungen gegeben. Eine genauere Anleitung wird auf `http://www.roman-klinger.de` zu finden sein. Hier wird nur auf die wichtigsten Komponenten eingegangen.

In Abbildung A.1 ist der Hauptarbeitsbereich der Anwendung zu sehen, welcher in drei Bereiche unterteilt ist. Links ist das eigentliche Hauptfenster zu sehen, in dem sich die Menüleiste sowie Dropdown-Boxen für die Auswahl der angelegten genetischen Algorithmen[1] (üblicherweise dürfte nur einer zu einem Zeitpunkt benötigt sein), die Evaluierungsfunktionen und die Markovketten und Mustergruppen befindet. Des weiteren kann hier die Evolution mit einer angegebenen Generationszahl gestartet werden, wobei die gewählte Evaluierungsmethode genutzt wird. Die Entwicklung der Fitness kann in dem in Abbildung A.2 gezeigten Fenster verfolgt werden. Dominiert wird das Hauptfenster durch die Auflistung der aktuellen Population. Jede Zeile in der Liste stellt ein Individuum dar, wobei die erste Zahl die Fitness, die zweite die Länge (welche bei allen Individuen innerhalb eines genetischen Algorithmus identisch sein muss) und die weiteren die zugrundeliegende Datenstruktur angeben. Wird hier ein Individuum angewählt wird im unteren Fenster dieses in Notenschrift gezeigt. Im rechten Bereich des Hauptfensters wird die aktuelle Fitness auf dem linken Schieberegler eingestellt. Hier ist es möglich, Veränderungen vorzunehmen und diese mit Druck auf den Knopf *Set* dem Individuum zuzuweisen, was auch bei interaktiver Evolution genutzt wird. Der rechte Schieberegler zeigt die Fitnessbewertung der in der Dropdown-Box eingestellten Evaluierungsfunktionen an. Rechts befindet sich die Auflistung der Merkmale des selektierten Individuums.

Während der Evolution wie auch außerhalb dieser ist es möglich, die Melodien gezielt zu verändern. Hierzu kann mit der mittleren Maustaste auf einzelne Töne geklickt werden, was verschiedene Optionen wie zum Beispiel die Veränderung der Länge zur Verfügung stellt. Des weiteren können die Noten mit der linken Maustaste in ihrer Tonhöhe verändert werden. Die Änderungen sind erst nach Druck auf den Knopf *Accept Changes* hörbar, was diese auch direkt dem Individuum zuweist.

Die Menüzeile beinhaltet die Menüs

- *Info*, in dem sich ein Informationsdialog sowie Lizenzinformationen anzeigen lassen,
- *File*, mit dem die gesamte Arbeitsumgebung aber auch genetische Algorithmen, Evaluierungsmethoden und einzelne Individuen geladen und gespeichert werden können,
- *GA*, in dem Markovketten, Mustergruppen und evolutionäre Algorithmen definiert, gelöscht und geändert werden können,
- *Evaluation*, das das Anlegen, Löschen und Verändern von neuronalen Netzen, Entscheidungsbäumen, gewichteten Summen und kombinierten Funktionen möglich macht,
- *Tools*, in dem unter anderem auf die Erweiterung von Midi-Dateien zu nutzbaren Individuen zugegriffen wird,

[1] Der Begriff „genetische Algorithmen“ wird synonym zu „evolutionäre Algorithmen“ genutzt.

- *Properties*, in dem die automatische Speicherung von Individuen an und ausgeschaltet werden kann sowie
- *Windows*, das unterschiedliche Fenster sichtbar schalten kann und weitere Optionen zur Verfügung stellt.

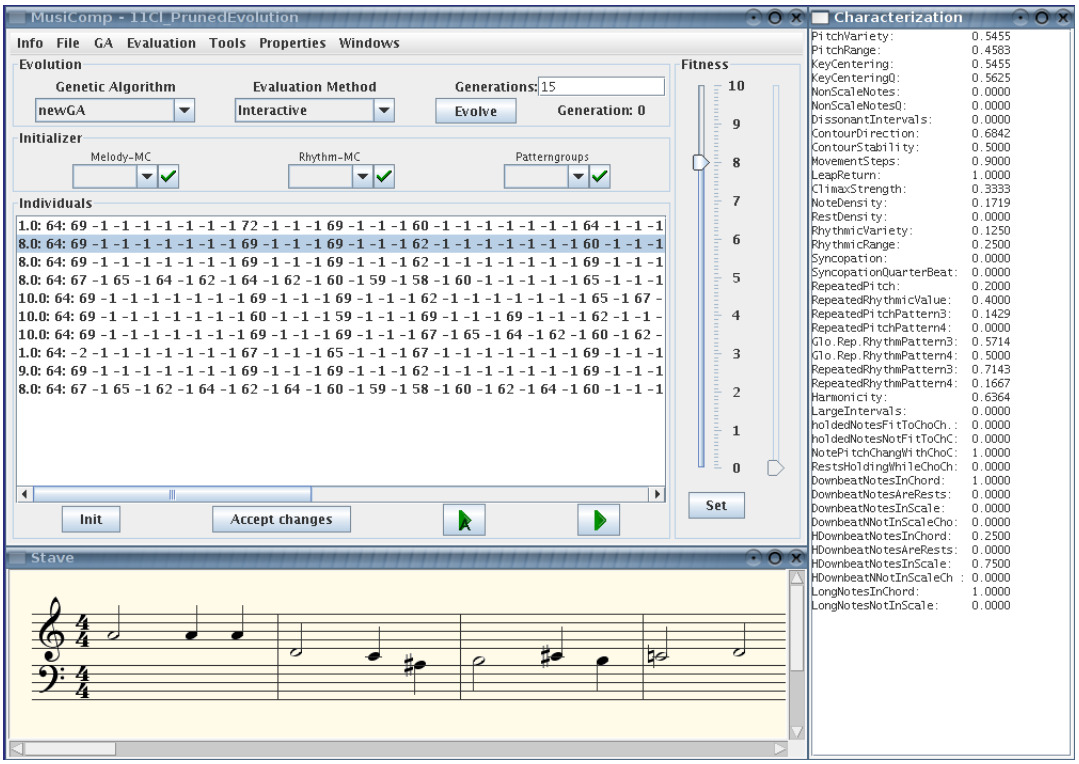


Abbildung A.1: Das Hauptfenster des Systems MusiComp

Die Optionen für das Anlegen eines evolutionären Algorithmus sind in den Abbildungen A.3, A.4 und A.5 gezeigt. Im Reiter *General* wird die Populationgröße und die Anzahl der Nachkommen, der Zwischenpopulation eingestellt. Letzteres ist allerdings nur möglich, wenn Crowding nicht eingeschaltet ist. Im Reiter *Initialisierung* werden die zur Initialisierung notwendigen Verfahren gewählt und einige weitere Parameter festgelegt. Die Zahlen hinter den Methodennamen deuten auf eine Priorität hin, mit der dieses Verfahren genutzt wird. So lassen sich unterschiedliche Verfahren für unterschiedlich viele Individuen in der initialen Population nutzen, da entsprechend dieser Prioritätsverteilung je Melodie zufällig das zu nutzende Verfahren bestimmt wird. In den Reitern *Rekombination* und *Mutation* werden die zur Verfügung stehenden Methoden gewählt, aus denen dann je Rekombination und Mutation ein Verfahren gleichverteilt zufällig ausgewählt wird. Der Reiter *Selektion* lässt das zu verwendende Selektionverfahren bestimmen sowie ob Fitness Sharing eingesetzt werden soll. Die Akkorde, die den Individuen zugrunde gelegt sind werden im Reiter *Chords* festgelegt. Hierzu stehen je Takt zwei Spalten zur Verfügung. In der jeweils ersten wird der Akkordgrundton angegeben, in der jeweils zweiten die Akkord-Skalen Kombination. Dies ist über Kürzel möglich, aber auch durch ein durch einen Druck auf die rechte Maustaste hervorgerufenes Pop-Up-Menü. Die Zahlen und Buchstaben in einfachen Anführungsstrichen (welche nicht überall vorhanden sind) können als Kürzel direkt eingetragen werden. Die Anzahl der Zeilen ist so voreingestellt, dass je Viertelschlag ein Akkord eingegeben werden kann. Durch Druck auf *Enlarge Table* wird die Anzahl jeweils verdoppelt.

Zum Anlegen der neuronalen Netze steht der in Abbildung A.6 gezeigte Dialog zur Verfügung. Die Anzahl der Eingangsneuronen folgt direkt aus der Wahl der Merkmale. Die in Tabelle 10.2 dargestellten

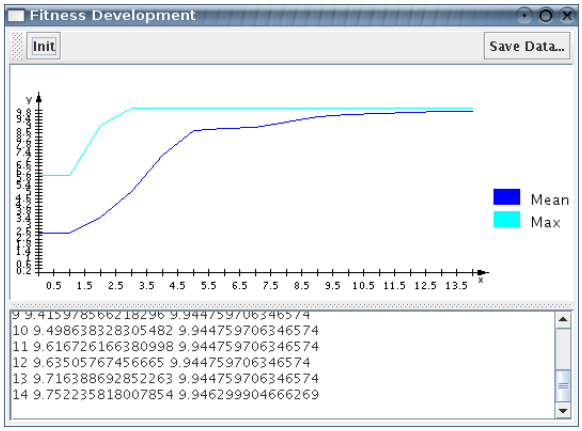


Abbildung A.2: Die Anzeige der Entwicklung der Fitness in einer Evolution

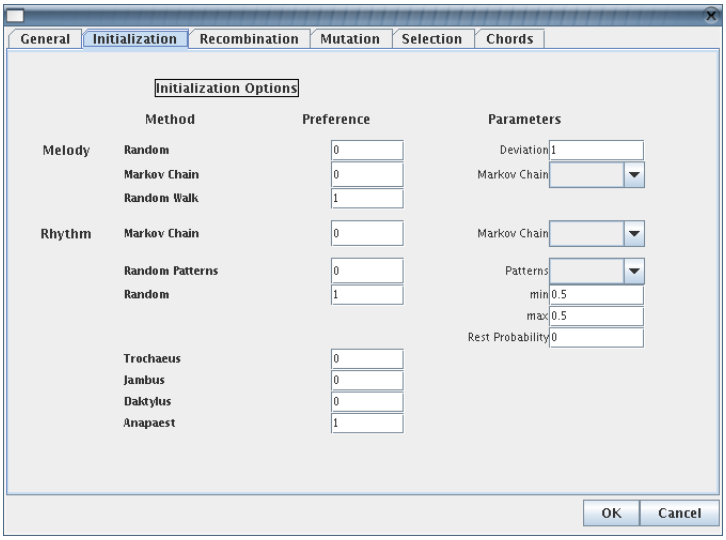


Abbildung A.3: Einstellen der Parameter für den evolutionären Algorithmus (1)

können durch den Knopf *Recommended* gewählt werden. Das Textfeld oben rechts gibt die Struktur der versteckten Neuronen an. Es wird eine kommaseparierte Liste von ganzen Zahlen erwartet, wobei jede Zahl eine versteckte Schicht mit der ihrem Wert entsprechenden Anzahl von Neuronen darstellt. Es existiert grundsätzlich ein Ausgangsneuron. Des weiteren kann gewählt werden, ob die eigene Implementierung oder die Bibliothek Joone genutzt werden soll.

Das Training des Netzes findet über die in Abbildung A.7 dargestellten Dialoge statt. Zunächst ist die Trainingsmenge über den Knopf *Load Individuals* zu laden. Über *Learn* kann das gewählte Verfahren gestartet werden, wobei die in *Steps* eingetragene Anzahl Iterationen durchgeführt wird. Bei Bedarf kann mit *Break* der Lernprozess unterbrochen werden. Wenn das Fenster zur Anzeige des Fehlers geöffnet ist (was mit dem Häkchen an *Show Error Window* beeinflusst wird), wird während des Trainings der TSSE aufgezeichnet. Mit Druck auf *Save Data* kann diese Entwicklung für die Verarbeitung in dem Programm *GnuPlot*[2] gespeichert werden.

Das Anlegen eines Entscheidungsbaums ist mit dem in Abbildung A.8 gezeigten Dialog möglich. Im oberen Bereich des Fensters findet die Wahl der Parameter statt, im unteren die Selektion der dem Baum zugrunde liegenden Beispiele. Zur Bedeutung der Parameter sei auf Kapitel 2.4 verwiesen. Im

[2]siehe `http://www.gnuplot.info`

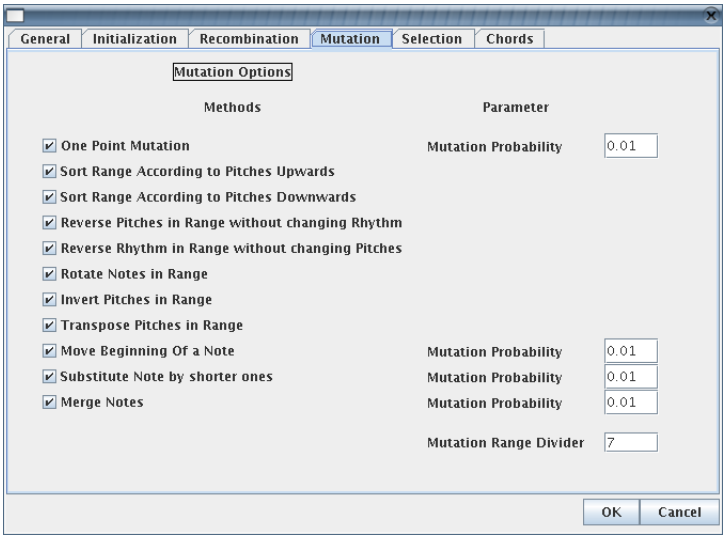


Abbildung A.4: Einstellen der Parameter für den evolutionären Algorithmus (2)

Allgemeinen sind die voreingestellten Parameter sinnvoll vorbelegt.

Nach Erstellen des Baums ist das in Abbildung A.9 gezeigte Fenster zu sehen. Hier sind einige Informationen über den entstandenen Baum angezeigt sowie der Baum selbst dargestellt. Mit der linken Maustaste kann man den sichtbaren Ausschnitt verschieben, mit der rechten kann er auf Festergröße skaliert oder neu gezeichnet werden. Mit einem Druck auf *Save as .dot* kann der Baum als Eingabedatei für das Programmpaket *GraphViz*[3] gespeichert werden.

[3]siehe http://www.graphviz.org

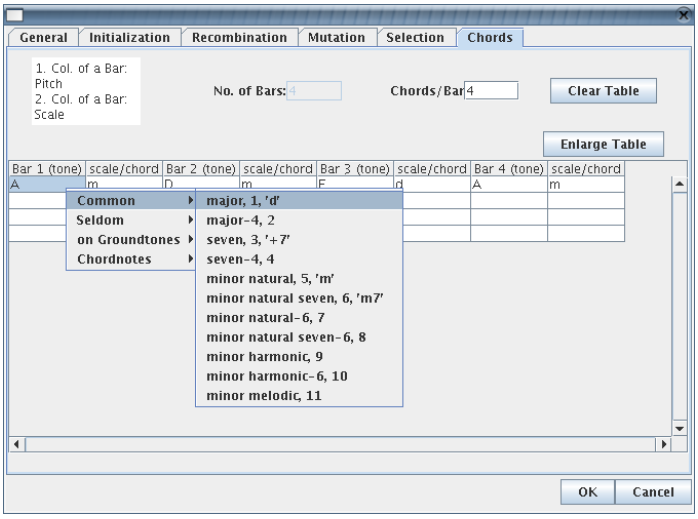


Abbildung A.5: Einstellen der Parameter für den evolutionären Algorithmus (3)

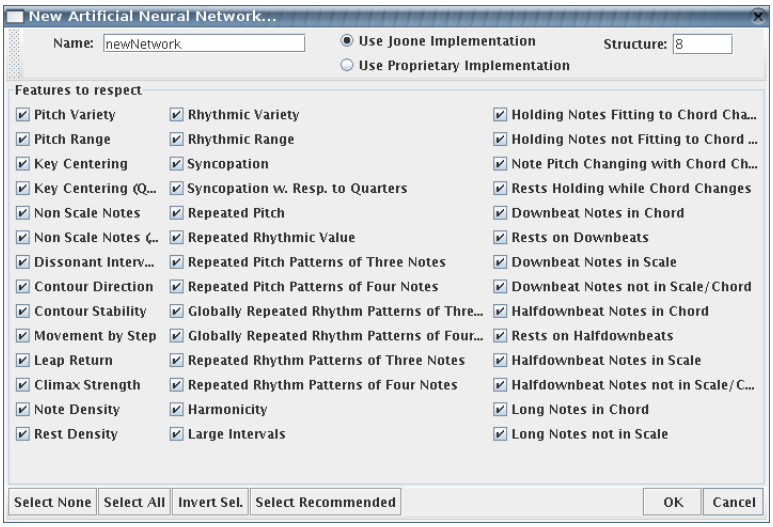


Abbildung A.6: Anlegen eines neuronalen Netzes

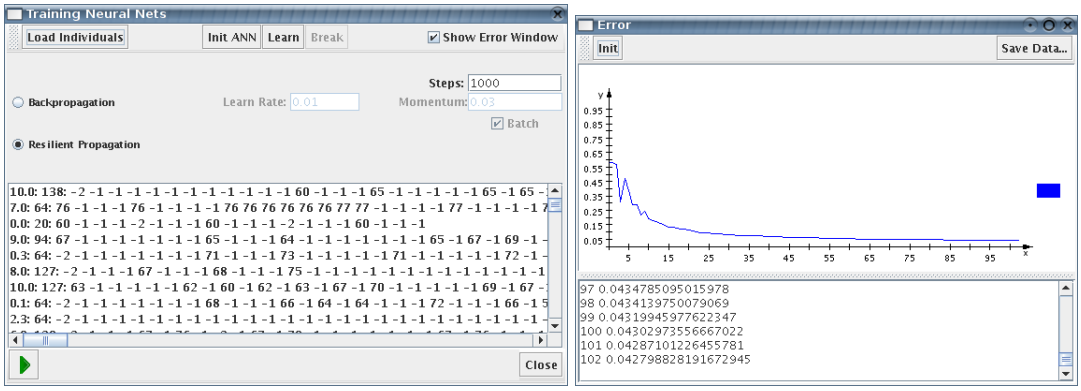


Abbildung A.7: Trainieren eines neuronalen Netzes

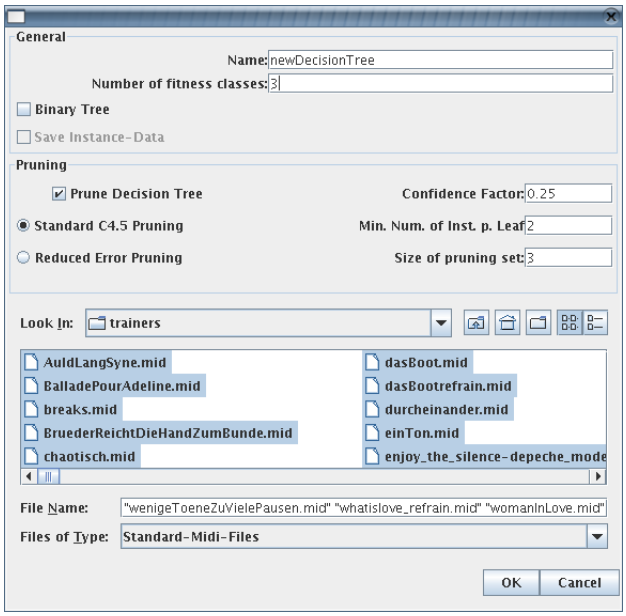


Abbildung A.8: Anlegen eines Entscheidungsbaums

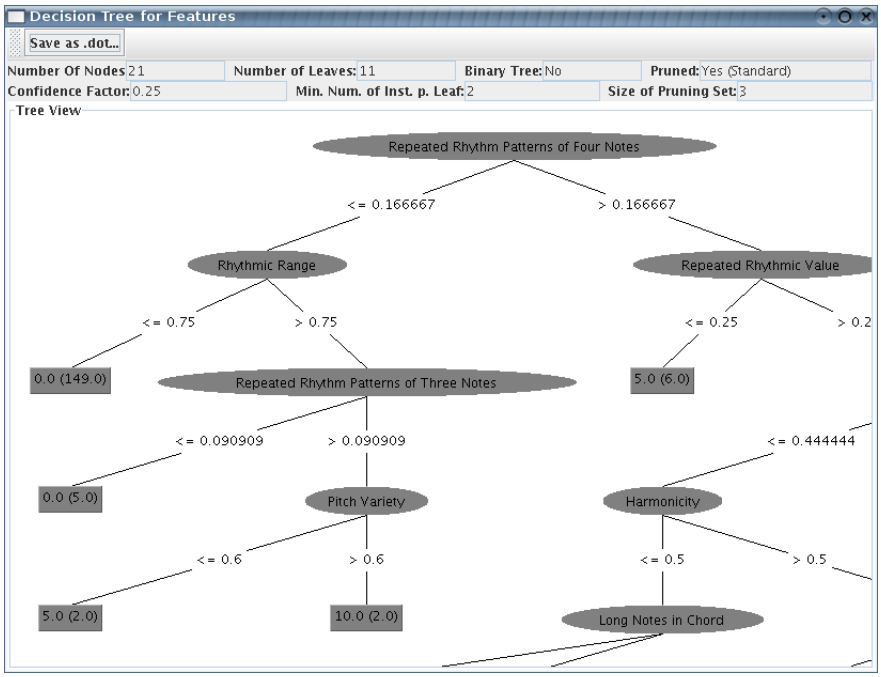


Abbildung A.9: Anzeige eines Entscheidungsbaums

Anhang B

Konfiguration und Start des Systems

Das Programm befindet sich in dem Ordner `MusiCom`. Dieser enthält die folgenden Dateien und Ordner:

`config` Dieser Ordner enthält zwei Konfigurationsdateien, welche weiter unten erklärt werden.

`soundbank` Dieser Ordner enthält die Datei `soundbank-deluxe.gm`. Diese stellt Klangdaten zur Wiedergabe der Melodien zur Verfügung.

`lib` Dieser Ordner stellt die genutzten Bibliotheken zur Verfügung. Dies sind jMusic (Sorensen und Brown, 1998), Weka (Witten und Frank, 2005b), jOpenChart (Müller, 2002) und Joone (Marrone, 2005).

`MusiCom.jar` Diese Datei enthält das eigentliche Programm.

`src` In diesem Ordner findet sich der Quelltext des Programms.

Das Programm wird mit folgendem Befehl aus dem Ordner `MusiCom` heraus gestartet:

```
java -jar MusiCom.jar
```

Es kann sinnvoll sein, wenn mit mehreren Markovketten gearbeitet werden soll, den Parameter `-Xmx400m` einzufügen, was eine Erhöhung des der virtuellen Maschine zur Verfügung stehenden Speichers bewirkt. Der Wert 400m deutet auf 400 Megabyte hin, der real einzusetzende Wert hängt von dem physisch vorhandenen Speicher ab.

Es wird eine Java Runtime Environment benötigt. Diese steht auf `http://java.sun.com` zum Download bereit.

Konfiguration

Im Verzeichnis `config` befinden sich zwei Konfigurationsdateien, welche das Verhalten des Programms maßgeblich beeinflussen.

Die Datei in Abbildung B.1 stellt Optionen zur Verfügung, welche das Verhalten des Systems ändern, allerdings nicht auf die Datenstrukturen Einfluss nehmen. Die Parameter sind die folgenden:

- `mainInstrument` legt das Instrument, mit dem die Melodie gespielt wird, fest. Dies geschieht entsprechend Tabelle 4.2.
- `chordInstrument` legt das Instrument, mit dem die vom Benutzer angelegten Akkorde gespielt werden, fest. Dies geschieht entsprechend Tabelle 4.2.
- `playWithChords` bestimmt, ob die vom Benutzer angelegten Akkorde gespielt werden sollen.
- `pathToData` legt den Pfad fest, zu dem in Dialogboxen zum Laden und Speichern gesprungen werden soll.
- `autoSavePath` bestimmt, wohin Dateien während einer Evolution automatisch gespeichert werden, wenn dies im Programm gewählt wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <comment/>
        <entry key="mainInstrument">0</entry>
        <entry key="chordInstrument">49</entry>
        <entry key="playWithChords">true</entry>
        <entry key="pathToData">../material/midis</entry>
        <entry key="autoSavePath">../material/midis/autosave</entry>
        <entry key="tempdir">/tmp</entry>
        <entry key="bpm">150</entry>
        <entry key="pmidiCall">pmidi -p 128:0 </entry>
        <entry key="timidityCall">timidity </entry>
        <entry key="graphvizCall">dot </entry>
        <entry key="nameOfSynth">Java Sound Synthesizer</entry>
        <entry key="outputmethod">4</entry>
</properties>
```

Abbildung B.1: Die Konfigurationsdatei `config.xml`

- `tempdir` gibt ein temporäres Verzeichnis an. Dies wird nur genutzt, wenn als Inhalt des Parameters `outputmethod` die Werte 1 oder 2 angegeben sind (siehe unten).
- `bpm` gibt die Geschwindigkeit, mit der Melodien abgespielt werden, an.
- `pmidiCall` gibt den Aufruf für das Programm pmidi[1] an. Dies wird genutzt, wenn als Inhalt des Parameters `outputmethod` der Wert 2 angegeben ist.
- `timidityCall` gibt den Aufruf für das Programm timidity[2] an. Dies wird genutzt, wenn als Inhalt des Parameters `outputmethod` der Wert 1 angegeben ist.
- `graphvizCall` gibt den Aufruf für Programme aus dem Paket graphviz[3] an. Diese werden genutzt, um Markovketten erster Ordnung zu visualisieren.
- `nameOfSynth` gibt den Namen des von Java zu nutzenden Synthesizers an. Üblicherweise sollte der hier eingestellte Wert nicht geändert werden.
- `outputMethod` bestimmt die Methode, mit der die Individuen abgespielt werden. Möglich sind Werte zwischen 1 und 5:
  1. Es wird das Programm timidity zum Abspielen der Individuen genutzt. Hierzu wird jeweils eine temporäre Midi-Datei gespeichert, welche dann als Argument dem in `timidityCall` angegeben Aufruf übergeben wird.
  2. Das Programm pmidi wird zum Abspielen der Individuen genutzt. Hierzu wird jeweils eine temporäre Midi-Datei gespeichert, welche als Argument dem in `pmidiCall` angegeben Aufruf übergeben wird.
  3. Die in der Bibliothek jMusic integrierte Abspielmethode wird genutzt. Diese scheint nicht grundsätzlich zu funktionieren.
  4. Eine proprietäre Abspielmethode unter Nutzung des Java Synthesizers wird genutzt. Diese Methode wird empfohlen.

[1] siehe `http://www.parabola.me.uk/alsa/pmidi.html`
[2] siehe `http://timidity.sourceforge.net`
[3] siehe `http://www.graphviz.org`

5. Es wird nichts abgespielt. Diese Methode sollte angewählt werden, falls keine Soundkarte zur Verfügung steht.

Die Optionen der Datei `staticOptions.xml` haben Einfluss auf die Datenstrukturen und damit auch auf die Verarbeitung von gespeicherten Daten. Sie sollten daher wohlüberlegt geändert werden, da gespeicherte Daten möglicherweise nur mit den Einstellungen zu Laden sind, mit denen sie gespeichert worden sind. Die Optionen sind:

- `BeatsPerBar` legt die Anzahl der Viertelschläge je Takt fest.
- `sequenceLength` legt die Länge der Individuen in Takten fest.
- `individualResolution` bestimmt die Auflösung der Individuen.

Weitere Erläuterungen hierzu finden sich in Abschnitt 7.2.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">
<properties>
        <comment>Attention! Changing these values
              makes everything saved unusable!</comment>
        <entry key="BeatsPerBar">4</entry>
        <entry key="sequenceLength">4</entry>
        <entry key="individualResolution">0.25</entry>
</properties>
```

Abbildung B.2: Die Konfigurationsdatei `staticOptions.xml`

Literaturverzeichnis

[Anderson 1996] Anderson, John R.: *Kognitive Psychologie*. Heidelberg: Spektrum Akademischer Verlag, 1996.

[Bäck u. a. 1997] Bäck, Thomas; Fogel, David B.; Michalewicz, Zbigniew (Hrsg.): *Handbook of Evolutionary Computation*. Bristol, UK: Institute of Physics Publishing and Oxford University Press, 1997.

[Beethoven 1824] Beethoven: *Die Sinfonie Nr. 9, d-moll, op. 125*. 1824. – URL `http://beethoven.staatsbibliothek-berlin.de/`. – Zugriffsdatum: 20.09.2005.

[Beierle und Kern-Isberner 2003] Beierle, Christoph; Kern-Isberner, Gabriele: *Methoden wissensbasierter Systeme*. Wiesbaden: Vieweg Verlag, 2003.

[Bentley und Corne 2002] Bentley, Peter J.; Corne, David W. (Hrsg.): *Creative Evolutionary Systems*. San Francisco, USA: Morgan Kaufman Publishers, 2002.

[Beyer u. a. 2005] Beyer, Irmraud; Bickel, Horst; Gropengiesser, Harald; Kluge, Siegfried; Knauer, Berhard; Kronberg, Inge; Krull, Hans-Peter; Lichtner, Hans-Dieter; Scheeweiss, Horst; Ströhle, Gerhard; Tischer, Wolfgang: *Natura – Biologie für Gymnasien*. Stuttgart: Ernst Klett Verlag, 2005.

[Bienert 1967] Bienert, Peter: *Aufbau einer Optimierungsautomatik für drei Parameter*, Technische Universität Berlin, Institut für Mess- und Regelungstechnik, Diplomarbeit, 1967.

[Biles 1994] Biles, John A.: GenJam: A genetic algorithm for generating jazz solos. In: *Proceedings of the International Computer Music Conference (ICMC 1994)*. San Francisco, USA: International Computer Music Association, 1994, S. 131–137.

[Biles 2001] Biles, John A.: *Autonomous GenJam: Eliminating the Fitness Bottleneck by Eliminating Fitness*. 2001. – URL `http://www.it.rit.edu/~jab/GECCO01/index.html`. – Zugriffsdatum: 16.11.2005. – online veröffentlicht.

[Biles 2002] Biles, John A.: GenJam: Evolution of a Jazz Improviser. In: Bentley, Peter J.; Corne, David W. (Hrsg.): *Creative Evolutionary Systems (Bentley und Corne, 2002)*. San Francisco, USA: Morgan Kaufman Publishers, 2002, Kap. 5, S. 165–187.

[Biles 2005] Biles, John A.: Evolutionary Music Tutorial. In: *Tutorial Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, URL `http://www.it.rit.edu/~jab`. – Zugriffsdatum: 16.11.2005, 2005. – Tutorial Proceedings.

[Biles u. a. 1996] Biles, John A.; Anderson, Peter G.; Loggi, Laura W.: Neural network fitness functions for a musical IGA. In: *Proceedings of the Soft Computing Conference (SOCO 1996)*. Reading, UK: ICSC Academic Press, 1996, S. B39–B44. – URL `http://www.it.rit.edu/~jab/SOCO96/SOCO.html`. – Zugriffsdatum: 16.11.2005.

[Bilmes 1998] Bilmes, Jeff A.: *A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models*. Berkeley, USA: International Computer Science Institute, 1998.

[Borgelt u. a. 2003] Borgelt, Christian; Klawonn, Frank; Kruse, Rudolf: *Neuro-Fuzzy-Systeme*. Wiesbaden: Vieweg Verlag, 2003.

[Bray u. a. 2004] Bray, Tim; Paoli, Jean; Sperberg-McQueen, C. M.; Maler, Eve; Yergeau, François: *Extensible Markup Language (XML) 1.0 (Third Edition)*. 2004. – URL `http://www.w3.org/TR/2004/REC-xml-20040204/`. – Zugriffsdatum: 04.04.2006.

[Burton 1998] Burton, Anthony R.: *A Hybrid Neuro Genetic Pattern Evolution System Applied to Musical Composition*. Guildford, Surrey, England, University of Surrey, School of Electronic Engineering, Information Technology and Mathematics, Dissertation, 1998. – URL `http://www.tony-b.freeuk.com/phd.html`. – Zugriffsdatum: 16.11.2005.

[Burton und Vladimirova 1997a] Burton, Anthony R.; Vladimirova, Tanya: *Applications of Genetic Techniques to Musical Composition*. 1997. – URL `http://www.tony-b.freeuk.com/phd.html`. – Zugriffsdatum: 16.11.2005. – Submitted to the Computer Music Journal.

[Burton und Vladimirova 1997b] Burton, Anthony R.; Vladimirova, Tanya: Genetic algorithm utilising neural network fitness evaluation for musical composition. In: *Proceedings of the Interational Conference on Genetic Algorithms and Artificial Neural Networks*. Wien, Österreich: Springer-Verlag, 1997, S. 220–224.

[Cannam u. a. 2005] Cannam, Chris; Bown, Richard; Laurent, Guillaume; McIntyre, D. M.: *Rosegarden*. Software. 2005. – URL `http://www.rosegardenmusic.com`. – Zugriffsdatum: 21.10.2005.

[Chen und Miikkulainen 2001] Chen, Chun-Chi J.; Miikkulainen, Risto: Creating Melodies with Evolving Recurrent Neural Networks. In: *Proceedings of International Joint Conference on Neural Networks (IJCNN 2001)*. Washington DC, USA, 2001.

[Crawford u. a. 2001] Crawford, Tim; Iliopoulos, Costas S.; Winder, Russel; Yu, Haifeng: Approximate Musical Evolution. In: *Computers and the Humanities* 35 (2001), S. 55–64.

[Deb 1989] Deb, Kalyanmoy: *Genetic Algorithms in Multimodal Function Optimization*, University of Alabama, Diplomarbeit, 1989.

[Drosdowski 1990] Drosdowski, Günther (Hrsg.): *Duden – Das große Wörterbuch der deutschen Sprache in zehn Bänden*. Bd. 05 – Fremdwörterbuch. 5. Auflage. Mannheim: Bibliographisches Institut, 1990.

[Feller 1968] Feller, William: *An Introduction to Probability Theory and Its Applications*. Bd. 1. New York, USA: John Wiley & Sons, Inc., 1968.

[Fogel 1962] Fogel, Lawrence J.: Autonomous automata. In: *Industrial Research* 4 (1962), S. 14–19.

[Fogel u. a. 1966] Fogel, Lawrence J.; Owen, Alvin J.; Walsh, Michael J.: *Artificial Intelligence Through Simulated Evolution*. New York, USA: John Wiley & Sons, Inc., 1966.

[Gartland-Jones 2003] Gartland-Jones, Andrew: MusicBlox: a Real-time Algorithmic Composition System Incorporating a Distributed Interactive Genetic Algorithm. In: *Proceedings of the Conference on Evolutionary Computing*, 2003.

[Gartland-Jones und Copley 2003] Gartland-Jones, Andrew; Copley, Peter: The Suitability of Genetic Algorithms for Musical Composition. In: *Contemporary Music Review* 22 (2003), Nr. 3, S. 43–55.

[Gerdes u. a. 2004] Gerdes, Ingrid; Klawonn, Frank; Kruse, Rudolph: *Evolutionäre Algorithmen*. Wiesbaden: Vieweg Verlag, 2004.

[Glover 1989] Glover, Fred: Tabu Search Part 1. In: *ORSA Journal on Computing* 1 (1989), Nr. 3, S. 190–206.

[Gottlieb 2000] Gottlieb, Jens: *Evolutionary Algorithms for Constrained Optimization Problems*. Aachen: Shaker Verlag, 2000.

[Grachten u. a. 2004] Grachten, Maarten; Arcos, Josep-Lluís; Mántaras, Ramon L. de: Melodic Similarity: Looking for a good abstraction level. In: *Proceedings of the International Conference on Music Information Retrieval (ISMIR)*. Barcelona, Spanien, 2004.

[Grossberg und Carpenter 1992] Grossberg, Stephen; Carpenter, Gail A.: A Massively Parallel Architecture for a Self-Organising Neural Pattern Recognition Machine. In: *Neural Networks – Theoretical Foundations and Analysis*. New Jersey, USA: IEEE Press, 1992, S. 147–208.

[Haupt und Haupt 2000] Haupt, Randy L.; Haupt, Sue E.: The creative use of genetic algorithms. In: *IEEE Potentials* (2000), April/May, S. 26–29.

[Hildebrand 2002] Hildebrand, Lars: *Asymmetrische Evolutionsstrategien*, Fachbereich Informatik der Universität Dortmund, Dissertation, 2002.

[Hildebrand 2004a] Hildebrand, Lars: *Vorlesung: Grundlagen und Anwendungen der Computational Intelligence I: Evolutionäre Algorithmen*. Universität Dortmund. 2004. – URL `http://ls1-www.cs.uni-dortmund.de/~hildebra/Vorlesungen/GACIIIEA/GACI2-gr.html`. – Zugriffsdatum: 10.11.2005. – Folien zur Vorlesung.

[Hildebrand 2004b] Hildebrand, Lars: *Vorlesung: Grundlagen und Anwendungen der Computational Intelligence I: Künstliche Neuronale Netze*. Universität Dortmund. 2004. – URL `http://lrb.informatik.uni-dortmund.de/~hildebra/GACI1-gr.html`. – Zugriffsdatum: 28.10.2005. – Folien zur Vorlesung.

[Holland 1975] Holland, John H.: *Adaption in Natural and Artificial Systems*. Michigan, USA: University of Michigan Press, 1975.

[Hopfield 1982] Hopfield, John J.: Neural networks and physical systems with emergent collective computational abilities. In: *Proceedings of the National Academy of Sciences*, 1982, S. 2554–2558.

[Huisinga und Meerbach 2005] Huisinga, Wilhelm; Meerbach, Eike: *Markov Chains for Everybody – An Introduction to the theory of discrete time Markov chains on countable state spaces.* Berlin: Fachbereich Mathematik und Informatik Freie Universität Berlin, January 2005.

[Igel und Hüsken 2003] Igel, Christian; Hüsken, Michael: Empirical evaluation of the improved Rprop learning algorithms. In: *Neurocomputing* 50 (2003), S. 105–123.

[Iosifescu 1980] Iosifescu, Marius: *Finite Markov Processes and Their Applications*. Bukarest, Romänien: John Wiley & Sons, Inc., 1980.

[Jacob 1995] Jacob, B.L.: Composing with genetic algorithms. In: *Proceedings of the International Computer Music Conference*. San Francisco, USA: International Computer Music Association, September 1995, S. 452–455.

[Jansen 2004] Jansen, Thomas: *Evolutionäre Algorithmen*. Universität Dortmund, 2004. – Skript zur Vorlesung.

[Järveläinen 2000] Järveläinen, Hanna: *Algorithmic musical composition*. April 2000. – URL `http://www.tml.tkk.fi/Studies/Tik-111.080/2000/papers/hanna/alco.pdf`. – Zugriffsdatum: 16.11.2005.

[Johanson und Poli 1998] Johanson, Brad; Poli, Riccardo: GP-Music: An Interactive Genetic Programming System for Music Generation with Automated Fitness Raters / Stanford University, University of Birmingham. 1998 (CSRP-98-13). – Forschungsbericht.

[Jourdain 2001] Jourdain, Robert: *Das wohltemperierte Gehirn*. Heidelberg: Spektrum Verlag, 2001.

[Kirkpatrick u. a. 1983] Kirkpatrick, Scott; Gelatt, C. D.; Vecchi, M. P.: Optimization by Simulated Annealing. In: *Science* 220, 4598 (1983), S. 671–680.

[Kohonen 2001] Kohonen, Teuvo: *Springer Series in Information Sciences*. Bd. 30: *Self-Organizing Maps*. 3. Auflage. Berlin: Springer Verlag, 2001.

[Koza 1992] Koza, John R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge: MIT Press, 1992.

[Marrone 2005] Marrone, Paolo: *Joone – Java Object Oriented Neural Engine*. Software. 2005. – URL `http://www.jooneworld.com`. – Zugriffsdatum: 04.04.2005.

[McCulloch und Pitts 1943] McCulloch, Warren; Pitts, Walter: A Logical Calculus of the Ideas Immanent in Nervous Activity. In: *Bulletin of Mathematical Biophysics* 9 (1943), S. 127–147.

[McIntyre 2005] McIntyre, Michael: *Rosegarden Companion*. Bomots, 2005.

[Metropolis u. a. 1953] Metropolis, N.; Rosenbluth, A.; Teller, A.; Teller, E.: Equations of state calculations by fast computing machines. In: *Journal of Chemical Physics* 21 (1953), S. 1087–1091.

[MIDI Manufacturers Association 1995] MIDI Manufacturers Association: *MIDI 1.0 Specification Message Summary, Expanded Status Bytes List, Summary of Control Change Messages (Data Bytes)*. 1995. – URL `http://www.midi.org/about-midi/table1.shtml`. – Zugriffsdatum: 20.10.2005.

[MIDI Manufacturers Association 2001] MIDI Manufacturers Association: *Complete MIDI 1.0 Detailed Specification*. 2001. – URL `http://www.midi.org/about-midi/docorder.shtml`. – Zugriffsdatum: 16.11.2005.

[Minsky 1981] Minsky, Marvin: Music, Mind, and Meaning. In: *Computer Music Journal* 5 (1981), Nr. 3. – URL `http://web.media.mit.edu/~minsky/papers/MusicMindMeaning.html`. – Zugriffsdatum: 21.09.2005.

[Miranda 2003a] Miranda, Eduardo R.: On the evolution of music in a society of self-taught digital creatures. In: *Digital Creativity* 14 (2003), Nr. 1, S. 29–42.

[Miranda 2003b] Miranda, Eduardo R.: On the Music of Emergent Behaviour: What can Evolutionary Computation Bring to the Musician? In: *Leonardo* 36 (2003), Nr. 1, S. 55–58.

[Miranda 2004] Miranda, Eduardo R.: At the Crossroads of Evolutionary Computation and Music: Self-Programming Synthesizers, Swarm Orchestras and the Origins of Melody. In: *Evolutionary Computation* 12 (2004), Nr. 2, S. 137–158.

[Miranda u. a. 2004] Miranda, Eduardo R.; Burraston, Dave; Edmonds, Ernest; Livingstone, Dan: Cellular Automata in MIDI-based Computer Music. In: *Proceedings of the International Computer Music Conference (ICMC2004)*, 2004.

[Miranda u. a. 2003] Miranda, Eduardo R.; Kirby, Simon; Todd, Peter M.: On Computational Models of the Evolution of Music: From the Origins of Musical Taste to the Emergence of Grammars. In: *Contemporary Music Review* 22 (2003), Nr. 3, S. 91–111.

[Miranda 1993] Miranda, Eduardo R.: Cellular Automata Music: An Interdisciplinary Project. In: *Interface/Journal of New Music Research* 22 (1993), S. 3–21.

[Miranda 2001] Miranda, Eduardo R.: *Composing Music with Computers*. Elsevier/Focal Press, 2001.

[Moroni u. a. 2002] Moroni, Artemis; Manzolli, Jônatas; Zuben, Fernando V.; Gudwin, Ricardo: Vox Populi: Evolutionary Computation for Music Evolution. In: Bentley, Peter J.; Corne, David W. (Hrsg.): *Creative Evolutionary Systems (Bentley und Corne, 2002)*. San Francisco, USA: Morgan Kaufman Publishers, 2002, Kap. 7, S. 205–221.

[Motte 1993] Motte, Diether de La: *Melodie*. DTV, 1993.

[Motte 2002] Motte, Diether de La: *Musikalische Analyse*. Bärenreiter Verlag, 2002.

[Motte 2004] Motte, Diether de La: *Harmonielehre*. DT V, 2004.

[Mozart 1793] Mozart, Wolfgang A.: *Anleitung so viel Walzer oder Schleifer mit zwei Würfeln zu componiren so viel man will ohne musikalisch zu seyn noch etwas von der Composition zu verstehen (KV Anh. 294d)*. Berlin–Amsterdam: J.J. Hummel, 1793.

[Müller 2002] Müller, Sebastian: *JOpenChart Library and Toolkit Version 0.94*. Software. 2002. – URL `http://jopenchart.sourceforge.net`. – Zugriffsdatum: 01.05.2006.

[Papadopoulos und Wiggins 1999] Papadopoulos, George; Wiggins, Geraint: AI Methods for Algorithmic Composition: A Survey, a Critical View and Future Prospects. In: *Symposium on AI and Scientific Creativity (AISB'99): Symposium on Musical Creativity*, 1999, S. 110–117.

[Pazos u. a. 1999] Pazos, Alejandro; Riego, A. S. del; Dorado, Julian; Cardalda, J.J. R.: Genetic Music Compositor. In: *Congress on Evolutionary Computation (CEC'99)* Bd. 2, 1999, S. 885–890.

[de la Puente u. a. 2002] Puente, Alfonso O. de la; Alfonso, Rafael S.; Moreno, Manuel A.: Automatic composition of music by means of Grammatical Evolution. In: *Proceedings of the International Conference on APL: Array Processing Languages: Lore, Problems, and Applications (APL'02)*. Madrid, Spanien, 2002, S. 148–155.

[Putnam 1994] Putnam, Jeffrey B.: Genetic Programming of Music / New Mexico Institute of Mining and Technology. Socorro, New Mexico: Institute of Mining and Technology, 1994. – Forschungsbericht.

[Quinlan 1986] Quinlan, John R.: Induction of decision trees. In: *Machine Learning* 1 (1986), Nr. 1, S. 81–106.

[Quinlan 1992] Quinlan, John R.: Learning with continuous classes. In: *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence*, 1992, S. 343–348.

[Quinlan 1993] Quinlan, John R.: *Programs for machine learning*. San Francisco: Morgan Kaufmann, 1993.

[Rechenberg 1965] Rechenberg, Ingo: *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment Library Translation 1122. 1965.

[Riedmiller 1994] Riedmiller, Martin: Rprop – Implementation Details / University of Karlsruhe. Januar 1994. – Forschungsbericht.

[Riedmiller und Braun 1992] Riedmiller, Martin; Braun, Heinrich: RPROP – A Fast Adaptive Learning Algorithm. In: *Proceedings of the International Symposium on Computer and Information Sciences*. Antalya, Turkey, 1992.

[Riedmiller und Braun 1993] Riedmiller, Martin; Braun, Heinrich: A direct adaptive method for faster backpropagation learning: the Rprop algorithm. In: *Proceedings of the International Conference on Neural Networks*. San Francisco, USA, 1993.

[Roederer 2000] Roederer, Juan G.: *Physikalische und psychoakustische Grundlagen der Musik*. Berlin: Springer Verlag, 2000.

[Rojas 1996] Rojas, Raúl: *Neural Networks – A Systematic Introduction*. Berlin: Springer Verlag, 1996.

[Rosenblatt 1958] Rosenblatt, Frank: The Perceptron: a Probabilistic Model for Information Storage and Organization in the Brain. In: *Psychological Review* 65 (1958), S. 386–408.

[Rudolph 1994] Rudolph, Günter: An Evolutionary Algorithm for Integer Programming. In: Davidor, Yuval; Schwefel, Hans-Paul; Männer, Reinhard (Hrsg.): *Parallel Problem Solving from Nature – PPSN III*. Berlin: Springer, 1994, S. 139–148.

[Russell und Norvig 2003] Russell, Stuart; Norvig, Peter: *Artificial Intelligence – A modern Approach*. Berkeley, USA: Prentice Hall, 2003.

[Santos u. a. 2000] Santos, Antonino; Arcay, Bernardino; Dorado, Julian; Romero, Juan; Rodriguez, Jose: Evolutionary Computation Systems for Musical Composition. In: *Proceedings of the International Conference Acoustic and Music: Theory and Applications (AMTA 2000)* Bd. 1, 2000, S. 97–102. – http://galileo.dc.fi.udc.es/cm.

[Schira 2003] Schira, Josef: *Statistische Methoden der VWL und BWL – Theorie und Praxis*. München: Pearson Studium, 2003.

[Schwefel 1965] Schwefel, Hans-Paul: *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*, Technische Universität Berlin, Hermann Föttinger Institut für Strömungsmechanik, Diplomarbeit, 1965.

[Schwefel 1995] Schwefel, Hans-Paul: *Evolution and Optimum Seeking*. New York, USA: John Wiley & Sons, Inc., 1995.

[Sorensen und Brown 1998] Sorensen, Andrew; Brown, Andrew: *jMusic – Music Composition in Java*. 1998. – URL `http://jmusic.ci.qut.edu.au/`. – Zugriffsdatum: 15.11.2005.

[Sun Developer Network 2005] Sun Developer Network: *JavaTM 2 Platform Standard Edition 5.0 API Specification*. 2005. – URL `http://java.sun.com/j2se/1.5.0/docs/api/`. – Zugriffsdatum: 15.11.2005.

[Sun Microsystems Inc. 2001] Sun Microsystems Inc.: *Java Sound Programmer Guide*. 2001. – URL `http://java.sun.com/j2se/1.4.2/docs/guide/sound/programmer_guide/`. – Zugriffsdatum: 15.11.2005.

[Towsey u. a. 2000] Towsey, Michael; Brown, Andrew; Wright, Susan; Diederich, Joachim: Towards Melodic Extension Using Genetic Algorithms. In: Brown, A. R.; Wilding, R. (Hrsg.): *Proceedings of Interfaces: The Australian Computer Music Conference*, 2000, S. 85–91.

[Unemi 1998] Unemi, Tatsuo: A design of genetic encoding for breeding short musical pieces. In: *ALife VIII: Workshop Proceedings*. Tokyo, Japan, 1998, S. 25–31. – URL `http://www.intlab.soka.ac.jp/~unemi/`. – Zugriffsdatum: 16.11.2005.

[Unemi und Nakada 2001] Unemi, Tatsuo; Nakada, Eiichi: A tool for composing short music pieces by means of breeding. In: *Proceedings of the Systems, Man, and Cybernetics Conference*. Tokyo, Japan, 2001.

[Waschka II 2001] Waschka II, Rodney: Theories of Evolutionary Algorithms and a 'New Simplicity' Opera: Making Sappho's Breath. In: *Artificial Life* 8 (2001), March, Nr. 1, S. 79–86. – URL `http://galileo.cincom.unical.it/esg/Music/workshop/workshop.htm`. – Zugriffsdatum: 16.11.2005. – Workshop: Artificial Life Models for Musical Applications of the Conference on Artificial Life.

[Wegener 2002] Wegener, Ingo: *Evolutionäre Algorithmen*. Universität Dortmund, 2002. – Skript zur Vorlesung.

[Werner und Todd 1998] Werner, Gregory M.; Todd, Peter M.: Frankensteinian methods for evolutionary music composition. In: Griffith, Niall; Todd, Peter (Hrsg.): *Musical Networks: Parallel Distributed Perception and Performance*. Cambridge, USA: MIT Press/Bradford Books, 1998, S. 313–339. – MIT Press/Bradford Books.

[Widrow und Hoff 1960] Widrow, Bernard; Hoff, Marcian: Adaptive switching circuits. In: *IRE WESCON Convention Record*. New York, USA, 1960, S. 96–104.

[Wiggins und Papadopoulos 1998] Wiggins, Geraint; Papadopoulos, George: A genetic Algorithm for the generation of Jazz Melodies. In: *Proceedings of the Finnish Conference on Artificial Intelligence (STeP '98)*. Jyväskylä, Finnland, 1998. – URL `http://www.soi.city.ac.uk/~geraint/papers/STeP98.pdf`. – Zugriffsdatum: 17.11.2005.

[Wiggins u. a. 1999] Wiggins, Geraint; Papadopoulos, George; Phon-Amnuaisuk, Somnuk: Evolutionary Methods for Musical Composition. In: *International Journal of Computing Anticipatory Systems* 4 (1999), S. . – URL `http://www.dai.ed.ac.uk/daidb/papers/documents/rp882.html`. – Zugriffsdatum: 16.11.2005.

[Witten und Frank 2005a] Witten, Ian H.; Frank, Eibe: *Data Mining*. San Francisco: Elsevier Inc., 2005.

[Witten und Frank 2005b] Witten, Ian H.; Frank, Eibe: *Weka 3 – Data Mining with Open Source Machine Learning Software in Java*. Software. 2005. – URL `http://www.cs.waikato.ac.nz/ml/weka/`. – Zugriffsdatum: 10.04.2006.

[Wrobel u. a. 2000] Wrobel, Stefan; Morik, Katharina; Joachims, Thorsten: Maschinelles Lernen und Data Mining. In: Görz, Günther; Rollinger, Claus-Rainer; Schneeberger, Josef (Hrsg.): *Handbuch der Künstlichen Intelligenz*. Oldenbourg Verlag, 2000, Kap. 14, S. 517–597.

[Xenakis 1971] Xenakis, Iannis: *Formalized Music – Thought and Mathematics in Composition*. Bloomington, Indiana, USA: Indiana University Press, 1971.

[Zell 1994] Zell, Andreas: *Simulation neuronaler Netze*. Oldenbourg Verlag, 1994.

[Zielinsky 2000] Zielinsky, Gregor: *Die neue virtuelle MIDI/Audio Technik*. mitp-Verlag, 2000.

Index