HOCHSCHULE
HANNOVER
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS
–
Fakultät IV
Wirtschaft und
Informatik

Evolutionäre Algorithmen - Musikkomposition mittels genetischer Algorithmen

Evelyn Elsner

Bachelorarbeit im Studiengang „Mediendesigninformatik“

25. Januar 2021

Autorin: Evelyn Elsner
Matrikelnummer 1475131
Bettina-von-Arnim-Weg 38
30855 Langenhagen
evelyn.elsner@stud.hs-hannover.de

Erstprüfer: Prof. Dr. Ralf Bruns
Hochschule Hannover
Abteilung Informatik, Fakultät IV
ralf.bruns@hs-hannover.de

Zweitprüfer: Prof. Dr. Volker Ahlers
Hochschule Hannover
Abteilung Informatik, Fakultät IV
volker.ahlers@hs-hannover.de

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die eingereichte Bachelorarbeit selbständig und ohne fremde Hilfe verfasst, andere als die von mir angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Werken wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Langenhagen, den 25. Januar 2021 Unterschrift

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Motivation der Arbeit . . . 1
1.2 Aufbau der Arbeit . . . 1

2 Evolutionäre Algorithmen **3**
2.1 Evolutionäre biologische Prozesse . . . 3
2.2 Relevante Grundbegriffe der Genetik im Zusammenhang der genetischen Algorithmen . . . 4
2.2.1 Gen und Genom . . . 4
2.2.2 Genotyp und Phänotyp . . . 4
2.2.3 Allel und Genpool . . . 4
2.2.4 Individuum und Chromosom . . . 4
2.2.5 Population und Generation . . . 4
2.2.6 Eltern und Kinder . . . 5
2.2.7 Genetische Operationen . . . 5
2.3 Phasen des genetischen Algorithmus . . . 5
2.3.1 Codierung . . . 7
2.3.2 Bewertungs- und Fitnessfunktion . . . 9
2.3.3 Selektion . . . 9
2.3.4 Rekombination . . . 11
2.3.5 Mutation . . . 13
2.3.6 Ersetzungsstrategie . . . 15

3 Grundlagen der Musik **17**
3.1 Musikalische Grundbegriffe . . . 17
3.2 Wirkung der Musik . . . 19
3.3 JFugue . . . 20

4 Genetische Algorithmen zur Musikkomposition **23**
4.1 Die Bewertung kreativer Lösungen als Herausforderung . . . 23
4.2 Die menschliche Bewertung - Einsatz interaktiver genetischer Algorithmen 23
4.3 Regelbasierte Fitnessberechnung mittels der gewichteten Summe . . . 24
4.4 Codierung eines Musikstückes . . . 26
4.5 Erkenntnisse aus dem aktuellen Stand der Technik . . . 27

5 Entwurf eines genetischen Algorithmus zur Musikkomposition **28**
5.1 Festgelegte Entwurfsentscheidungen . . . 28
5.2 Codierung . . . 29
5.3 Initialisierung . . . 31
5.4 Fitnessfunktion . . . 32
5.5 Ersetzungsstrategie . . . 36
5.6 Selektionsentwurf . . . 37

5.7 Rekombinationsentwurf . . . 37
5.8 Mutationsentwurf . . . 38
5.9 Das Abspielen der Individuen . . . 39

6 Implementierung des Genetischen Algorithmus zur Musikkomposition 40
6.1 Programmstruktur . . . 40
6.2 Die Initialisierung der Anfangspopulation . . . 42
6.3 Codierung eines Individuums . . . 44
6.4 Durchlaufen des Genetischen Algorithmus . . . 45
6.5 Implementierung der Fitnessberechnung . . . 47
6.6 Sicherung der Eliten . . . 48
6.7 Die Selektion . . . 49
6.8 Die Rekombination . . . 50
6.9 Die Mutation . . . 52
6.10 Individuen als Musikstück hören . . . 55

7 Evaluation 57
7.1 Parametereinstellungen . . . 57
7.1.1 Populations- und Generationengröße . . . 57
7.1.2 Rekombinationswahrscheinlichkeit . . . 60
7.1.3 Mutationswahrscheinlichkeit . . . 62
7.1.4 Elitengröße . . . 63
7.2 Bewertung aller getroffener Entwurfsentscheidungen . . . 63
7.3 Bewertung der erzeugten Musikstücke . . . 65

8 Fazit und Ausblick 68

Anhang IV

Tabellenverzeichnis

1 Gegenüberstellung der direkten Codierung und der Gray-Codierung . . . 8
2 Einige Notenwerte und Pausen gleicher Länge 18
3 Gegenüberstellung der möglichen Werte für die Parameter der Melodie- und Akkord-Individuen 31
4 Gegenüberstellung der Tondauern und ihrem Anteil als Summand der Summe 4.0 eines Taktes 32
5 Konsonanzwerte der einzelnen Intervalle in Halbtonschritten 34
6 Beste Fitnesswerte der Generation 50 in Abhängigkeit der Rekombinationswahrscheinlichkeit 61
7 Fitnesswerte der Generation 50 in Abhängigkeit der Mutationswahrscheinlichkeit 62
8 Fitnesswerte der Generation 50 in Abhängigkeit der Elitengröße 63
9 Ergebnisse der Fintessmerkmale des besten Melodie-Individuums der Anfangs- und Endpopulation 66
10 Ergebnisse der Fintessmerkmale des besten Akkord-Individuums der Anfangs- und Endpopulation 66

Abbildungsverzeichnis

1 Allgemeiner Zyklus eines GA . . . 6
2 Roulette-Rad mit den vier Chromosomen A, B, C und D . . . 10
3 Ein-Punkt-Crossover-Beispiel mit binären Zahlen . . . 12
4 N-Punkt-Crossover-Beispiel mit binären Zahlen . . . 12
5 Uniform-Crossover-Beispiel mit binären Zahlen . . . 13
6 Gleichverteilte Mutation bei binärer und reeller Codierung . . . 14
7 Inversions-Mutation bei binärer und reeller Codierung . . . 14
8 Verschiebe-Mutation bei binärer und reeller Codierung . . . 15
9 Notenschlüssel . . . 17
10 Codieraufbau mit Pausen und unterschiedlichen Tondauern . . . 27
11 Die 36 erlaubten Töne dargestellt auf der Klaviatur . . . 30
12 Die 24 erlaubten Akkord-Töne dargestellt auf der Klaviatur . . . 30
13 Veranschaulichung des entworfenen Zwei-Punkt-Crossovers bei Akkord-Individuen . . . 38
14 Mutationsbeispiel eines Melodie-Individuums . . . 39
15 Klassendiagramm des implementierten GA. . . . 40
16 Fitnessverlauf bei einer Populationsgröße von 500 und einer Generationenzahl von 1.000 . . . 58
17 Fitnessverlauf bei einer Populationsgröße von 1.000 und einer Generationenzahl von 500 . . . 59
18 Fitnessverlauf bei einer Populationsgröße von 10.000 und einer Generationenzahl von 50 . . . 60
19 Fitnessverlauf bei einer Populationsgröße von 10.001, einer Generationenzahl von 50, der Rekombinationswahrscheinlichkeit von 0.7, der Mutationswahrscheinlichkeit von 0.001 und der Elitengröße von 1 . . . 64
20 Musikstück aus dem besten Individuen je Art der Anfangspopulation . . . 66
21 Musikstück aus dem besten Individuen je Art der Endpopulation . . . 66

Listings

1 Die abstrakte Klasse *IndividualFactory*. 42
2 Ausschnitt der Klasse *ChordIndividualFactory*. 43
3 Konstruktor und Attribute der Klasse *Gene*. 44
4 Ausschnitt der abstrakten Klasse *Individual*. 44
5 Ausschnitt der Klasse *MelodyEvolution*. 45
6 Die *evolve*-Methode innerhalb der Klasse *Melodyevolution*. 46
7 Die abstrakte Klasse *FitnessCalc*. 47
8 Die Berechnung des Merkmals *Steigende Konturrichtung* in der Klasse *RaisingContourDirection*. 47
9 Methode *calcWeightedSum* innerhalb der Klasse *MelodyFitnessCalc*. . 48
10 Die Klasse *MelodyElitism*. 49
11 *select*-Methode der Klasse *TournamentSelection*, welche die Melodie-Individuen selektiert. 50
12 *recombinePopulation*-Methode für die Melodie-Individuen innerhalb der abstrakten Klasse *recombination*. 51
13 *recombine*-Methode innerhalb der *OnePointCrossover*-Klasse, die das Ein-Punkt-Crossover implementiert. 51
14 *mutatePopulation*-Methode für die Melodie-Individuen in der abstrakten Klasse *Mutation*. 53
15 *mutateGene*-Methode in der Klasse *MelodyUniformMutation*. 54
16 *mutateNote*-Methode in der Klasse *MelodyUniformMutation*. 54
17 Methode *createGeneticAlgorithmResultMusic* der Klasse *MusicHandler*. 55
18 Methode *playMidiDirectly* in der Klasse *MusicHandler*. 56

1 Einleitung

1.1 Motivation der Arbeit

Eine Lösung für ein gegebenes Problem zu finden, baut auf detailliertes Wissen über dieses Problem auf. Ein Lösungsverfahren zu entwickeln kann Monate oder sogar Jahre großen Aufwand bedeuten, wie auch John H. Holland sagt: [17, S. 1]

> „Living organisms are consummate problem solvers. They exhibit a versatility that puts the best computer programs to shame. This observation is especially galling for computer scientists, who may spend months or years of intellectual effort on an algorithm, whereas organisms come by their abilities through the apparently undirected mechanism of evolution and natural selection.“ [6, S. 66]

Dieses Zitat liefert eine neue Idee den vorliegenden Schwierigkeiten entgegenzuwirken, nämlich auf der Basis der Evolutionsprozesse. Die biologischen Vorgänge können als Vorbilder verwendet werden, die es nachzuahmen gilt. Die Evolutionsprozesse bewirken in der Biologie die Entstehung, die Veränderung und das Aussterben eines Lebewesens. Es entsteht ein Kampf um das Überleben, welchen nur die Stärksten gewinnen. Das Ziel der stetigen Optimierung der Lebewesen entsteht. [19, S. 400-403]
Der Antrieb zum Erreichen eines Optimums ist jedoch nicht nur in der Biologie wiederzufinden. In den unterschiedlichen Gebieten der Kunst wird genauso ein Ideal angestrebt. Sei es nun das Verlangen zum Zeichnen des perfekten Bildes, zum Schreiben eines einfallsreichen Gedichtes oder zum Komponieren eines klangvollen Musikstückes, es wird stets versucht das Bestmögliche aus allem herauszuziehen.
Um diesen Drang näher zu untersuchen, ein optimales Ergebnis im kreativen Bereich nach den Prinzipien der Evolution zu erzielen, entsteht diese Bachelorarbeit. Es werden die Möglichkeiten genetischer Algorithmen (GA) in Bezug auf die Musik untersucht. Die Wahl ist auf den Bereich der Musik gefallen, weil diese oftmals stark konträr die Meinungen verschiedener Hörer spaltet und damit individuell als schön oder unschön betrachtet wird. Zusammen mit der eigenen Erzeugung einer klangvollen Musikkomposition ergibt sich das Ziel der Arbeit.

1.2 Aufbau der Arbeit

Beginnend werden die Grundlagen der GA im Bereich der Biologie erklärend dargestellt, sodass ein Einstieg in die relevante Thematik und die konkreten Begrifflichkeiten erreicht wird. Genauso stehen hier die einzelnen Phasen des Prozesses beschrieben.
Danach folgt eine Einführung in den vorliegenden künstlerischen Bereich, die Musik. Die Musikaspekte, welche im weiteren Verlauf relevant werden, sind hier erläutert. Zusätzlich enthält dieser Abschnitt eine Zusammenführung der musikalischen Bereiche mit denen der Programmierung, indem auf die Musikprogrammierung eingegangen wird.

In Kapitel 4 werden einige Aspekte bisheriger Ansätze zur Musikkomposition mittels der GA aufgezeigt, welche relevante Ideen für den eigenen Entwurf eines solchen liefern. Der Entwurf des eigenen konkreten GA ist dann im darauf folgenden Kapitel detailliert erklärt. Hierbei wird explizit auf alle Prozessphasen eingegangen.
Aufbauend darauf wird weiter auf die konkrete Implementierung des Entwurfs eingegangen und diese schrittweise dargestellt.
Zuletzt wird der entstandene GA untersucht und bewertet. Experimentell werden die Parameter der Implementierung betrachtet, mit dem Ziel das Endergebnis zu optimieren. Die Evaluierung, ob und inwiefern der Algorithmus das Ziel der Arbeit erreicht, folgt, bevor zusammenfassend ein Fazit und ein Ausblick die Arbeit beenden.

2 Evolutionäre Algorithmen

Das folgende Kapitel liefert das Grundverständnis darüber, was ein evolutionärer Algorithmus (EA) und damit im Fokus stehend ein GA ist. Dafür werden konkret die biologischen Prozesse, die relevanten Begriffe der Genetik und detaillierter die einzelnen Schritte eines GA beschrieben.

2.1 Evolutionäre biologische Prozesse

Der Vorgang des Entstehens und des Veränderns der auf der Erde lebenden Arten nennt man Evolution. Zusammen mit dem Beginn des Lebens auf der Erde vor etwa 3,5 Milliarden Jahren kam der Entwicklungsprozess auf. Nur die Lebewesen, die sich bewahren und sich gut an die Umweltbedingungen anpassen, überleben. Der Naturforscher Charles Darwin betitelte diese Fähigkeit 1859 in seinem Werk „On the origin of species by means of natural selection“ als das Prinzip „survival of the fittest“. [19, S. 400-403]
Die vollständige Erbinformation eines Individuums ist in der Desoxyribonukleinsäure (DNS) enthalten. Die Gesamtheit aller Gene, alle Abschnitte der DNS, ist bereits vor der Geburt definiert und ist im Laufe des Lebens nicht änderbar. Durch die Fortpflanzung wird diese Struktur nun an die Nachkommen vererbt. Da Lebewesen, die besser an ihre Umwelt angepasst sind, eine höhere Lebensdauer besitzen und folglich mehr Nachkommen erzeugen können, überleben letztlich die Stärksten. [4, S. 11-17]
Die Grundlage für den Evolutionsprozess bilden die Vorgänge der Mutation, der Rekombination und der Selektion.
Unter der Mutation versteht man eine zufällige Abwandlung des Erbgutes. [10, S. 353]
Bei dem Fortpflanzungsprozess sind hierbei Fehler in der DNS entstanden. [23, S. 9]
Die Rekombination findet bei der Fortpflanzung statt. Die gesamten Erbinformationen beider Elternteile werden bei der sexuellen Paarung in kleine Einzelteile zerlegt, gemischt und anschließend neu kombiniert, sodass erneut eine vollständige Erbinformation, die des Nachkommens, entsteht. Genau können die Stellen der Rekombination nicht lokalisiert werden. Vielmehr ist die Mischung zufällig, folgt gleichzeitig aber auch den Mendelschen Gesetztmäßigkeiten[1].
Der dritte fundamentale Evolutionsprozess ist die Selektion. Sie definiert nach bestimmten Kriterien, welche Individuen zur Fortpflanzung ausgewählt werden und prägt damit den Weg zur Veränderung des Erbgutes. [10, S. 354]

[1] „Von GREGOR MENDEL [Hervorhebung im Original] aufgestellte Vererbungsregeln (Uniformitätsregel, Spaltungsregel und Neukombnationsregel)“ [19, S. 511]

2.2 Relevante Grundbegriffe der Genetik im Zusammenhang der genetischen Algorithmen

2.2.1 Gen und Genom

Ein Gen ist eine Sequenz der DNS und enthält damit einen Teil der Erbinformation. Dieser Abschnitt kann jedoch auch separiert sein. [10, S. 360-362] Betrachtet man die gesamte Menge aller Gene eines Organismus, verwendet man den Begriff Genom. [23, S. 8]

2.2.2 Genotyp und Phänotyp

Der Genotyp ist die vollständige Menge der Gene eines Individuums und damit das Erbbild eines Lebewesens. GA enthalten den Genotypen in Form der kodierten Variablen. Diese Variable dekodiert ist der Phänotyp, also das Erscheinungsbild des Lebewesens. [23, S. 15][10, S. 363]

2.2.3 Allel und Genpool

Ein Allel ist eine bestimmte Ausprägung eines Gens. Es stellt also den Wert des Gens im Genom dar. Betrachtet man bspw. das Gen für die Augenfarbe, so wäre die Farbe Blau ein Allel. Alle Allele der Population zusammen, bezeichnet man als Genpool. [23, S. 8]

2.2.4 Individuum und Chromosom

Ein Individuum ist ein lebender Organismus. Seine Gene werden in einer Menge von Chromosomen getragen. Die Begriffe Individuum und Chromosom werden oft als Synonyme verwendet. [13, S. 2][10, S. 365]

2.2.5 Population und Generation

Die Gesamtheit aller strukturell gleichartigen Lebewesen einer Gattung bildet eine Population. Die Populationsgröße ist zeitlich betrachtet variabel, bedingt durch die Geburt und den Tod einiger Individuen der Art. Unter einer Generation wird die Population zu einem Zeitpunkt verstanden. [13, S. 3]

2.2.6 Eltern und Kinder

Individuen aus der Population, die für die Fortpflanzung sorgen, bezeichnet man als Eltern. Der Begriff Vorfahren kann als Synonym verwendet werden. Die Kinder, bzw. die Nachkommen, sind entsprechend die Lebewesen, welche durch die Reproduktion der Eltern entstehen. [10, S. 364-365]

2.2.7 Genetische Operationen

Um neue Individuen aus bereits existierenden zu reproduzieren, werden genetische Operationen angewandt. Eine Einteilung in zwei Bereiche ist möglich. Die erste Klasse bildet die Rekombination, also die sexuelle Fortpflanzung zweier Lebewesen. Die Mutationen, welche beliebige Genveränderungen durch äußere Wirkungen darstellt, ist die zweite Klasse. [10, S. 372, 398]

2.3 Phasen des genetischen Algorithmus

GA sind neben der genetischen Programmierung (GP), den Evolutionsstrategien (ES) und der evolutionären Programmierung (EP) einer von vier Teilgebieten der EA. Sie werden in der Praxis am häufigsten gebraucht und sind auf Arbeiten von John Holland aus der Mitte der sechziger Jahre zurückzuführen. Während die Entwicklung der verschiedenen Ausprägungsformen der EA anfangs unabhängig voneinander stattfand, existieren letztlich viele Parallelen. Verantwortlich hierfür ist das gemeinsame Vorbild aller, die Evolution. Die EA, folglich auch die GA, orientieren sich nämlich an den biologischen Evolutionsprozessen. [10, S. 353-368]
„[Das Ziel der Evolution] ist es, diejenigen Erbanlagen zu finden, die ein Individuum oder eine Art am besten dazu befähigen, sich im Kampf um das Dasein zu bewähren.“ [16, S. 96]. Die in der Natur vorzufindende Evolution ist somit als Optimierung der Lebewesen zu verstehen. GA stellen damit Optimierungsverfahren dar, bei welchen ein Suchvorgang „[...] im Raum der genetischen Informationen bzw. im Raum der möglichen Erbanlagen [stattfindet].“ [16, S. 96]. [4, S. 11][13, S. 14]
Im Allgemeinen sind Optimierungsprobleme zunächst in unterschiedlichen Bereichen von der Industrie über die Forschung bis hin zur Wirtschaft zu finden. Das Anwendungsspektrum ist breit gefasst und nicht einzugrenzen. Bei den GA werden mit Hilfe des Computers die Prozesse der Evolutionstheorie nachgeahmt, um sich an das gewählte Optimum der Problemlösung anzunähern. Eine mögliche Lösung wird als ein abstraktes Objekt abgebildet und wie ein Lebewesen behandelt. Indem die evolutionären Operatoren, wie z. B. die Mutation, angewandt werden, kann der Organismus verändert, reproduziert und bewertet werden. [23, S. 20]
Nach der Klärung was GA im Allgemeinen sind und welches Ziel sie verfolgen, stellt sich die Frage, wie ein solcher Algorithmus im Detail konzipiert ist.
Zuerst, schon vor dem ersten richtigen Implementierungsschritt, muss zunächst eine

passende Codierung der Chromosomen bestimmt werden. [16, S. 187] Mit der Initialisierung der Population beginnt ein GA. Es werden Individuen, auch Lösungskandidaten genannt, erzeugt, welche zusammen die Anfangspopulation bilden. Üblicherweise werden diese Kandidaten zufällig gewählt, nichtsdestoweniger ist es aber genauso möglich zu Beginn auf die optimalen bekannten Individuen oder Lösungen anderer Optimierungsverfahren zurückzugreifen. [23, S. 24]
Nun wird das Ergebnis der Bewertungsfunktion berechnet. Abhängig von dem zu Beginn definierten Optimierungsproblem ist die Funktion ein Maß für die Güte des Lösungskandidaten. [23, S. 24][10, S. 371]
Weiter wird von dem ermittelten Gütewert eines jeden Lösungskandidaten in dem Selektionsprozess Gebrauch gemacht. Das bedeutet, dass mittels der Fitness bestimmt wird, welche Eltern wie viele Nachkommen generieren.
Die Fortpflanzung selbst geschieht in dem nächsten Schritt, der Rekombination. Indem die Eigenschaften der Eltern neu kombiniert werden, wird die Population durchmischt. Es folgt die Mutation, welche zusätzlich neue Kinder generiert.
Die neue Population entsteht nun, indem die erzeugten Nachkommen erneut bewertet werden. Es wird entweder die gesamte Elterngeneration durch die nächste Generation ersetzt oder lediglich einige Vorfahren werden durch ihre Kinder ausgetauscht.
Dieser Zyklus wiederholt sich bis zur Erfüllung eines Abbruchkriteriums. Ein solches kann sein, dass eine definierte Mindestqualität erlangt ist, eine maximale Generationenanzahl erreicht ist oder dass über einige Generationen hinweg keine Optimierung festzustellen ist. [23, S. 25]

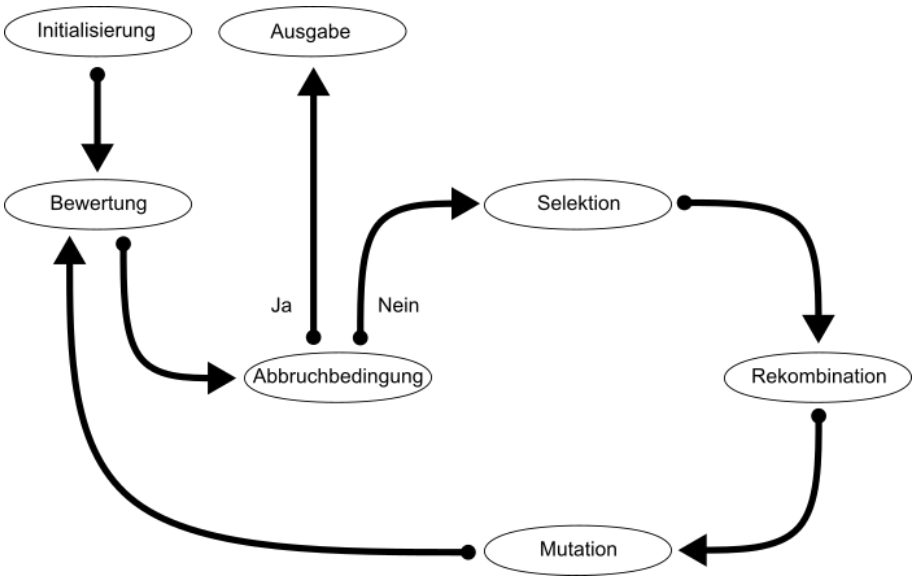


Abbildung 1: Allgemeiner Zyklus eines GA. In Anlehnung an [23, S. 25].

2.3.1 Codierung

Der erste Schritt bei der Entwicklung eines GA ist die Codierung der Individuen. Mit ihr steht und fällt die Entscheidung für den Erfolg oder den Misserfolg des Algorithmus. [10, S. 371] In welcher Form die Chromosomen letztlich verschlüsselt werden, ist abhängig von dem jeweiligen Optimierungsproblem zu wählen. [22, S. 6]
Im Zusammenhang mit den GA wird in der Regel die Binärcodierung verwendet. Hierbei werden alle Chromosomen einer Population als binäre Vektoren aufgefasst. Ein Individuum setzt sich damit aus Nullen und Einsen zusammen. Explizit bedeutet das, dass zunächst die Grundmenge $M := \{0, 1\}$ gegeben ist, sodass ein Chromosom ein binärer Vektor x aus M ist. Ein Individuum ist damit ein Element aus:

$$M^n = \{0, 1\}^n, n \in \mathbb{Z} : n \geq 1$$

Die Darstellung eines binären Vektors ist dann entsprechend:

$$\vec{x} =< x_1, x_2, ..., x_n >$$

Die i-te Position eines Chromosoms $x = <...,x_i,...>$ ist dann das i-te Gen des Individuums. Weiter wird der jeweilige Wert des Gens als Allel bezeichnet. Gene sind damit die Variablen, die Lösungseigenschaften. Um weiter die Informationsmenge bei der Binärcodierung zu steigern, ist es üblich mehrere Bits zu einem Segment zusammenzufassen, um die Eigenschaften darzustellen. [16, S. 189-190][10, S. 5] Das folgende Beispiel verdeutlicht die beschriebenen Zusammenhänge:

Beispiel 1. *Das Chromosom y mit $y =<< 1, 1, 0 >, 1 >$ hat zwei Gene. Das erste Gen, bestehend aus drei Bits, hat das Allel $< 1, 1, 0 >$, das zweite, codiert mit lediglich einem Bit, hat das Allel 1. Die binär-codierten Variablen der Lösung bilden also das Chromosom. Die Position und Länge der Gene innerhalb des Gesamtstrings sind eindeutig. [16, S. 190]*

Die Binärcodierung bringt jedoch ein nicht zu vernachlässigbares Problem mit sich: Eine kleine Veränderung innerhalb der binären Codierung garantiert keine ebenso kleinen Änderungen in einer anderen Verschlüsselungsform. Konkret kann das Problem mit dem folgenden Beispiel verdeutlicht werden:

Beispiel 2. *Mit maximal vier Binärwerten können acht natürliche Zahlen dargestellt werden. Die Codierung der Zahlen 1 bis 8 sieht wie folgt aus:*

- $1 \rightarrow 0001$
- $2 \rightarrow 0010$
- $3 \rightarrow 0011$
- $4 \rightarrow 0100$

- 5 → 0101
- 6 → 0110
- 7 → 0111
- 8 → 1000

Betrachtet man nun die Binärstellen der Zahlen 7 *und* 8*, also* 0111 *und* 1000*, ist zu sehen, dass sich alle vier Stellen voneinander unterscheiden. Alle vier Bitpositionen mussten bei dem Wechsel von der Zahl* 7 *zur* 8 *geändert werden.*

Je größer das n der Zweierpotenz 2^n ist, desto auffälliger ist dieser Effekt. Als mögliche Folge kann es zu Konsequenzen bzgl. der Konvergenz des GA kommen. [16, S. 192-193] Um das beschriebene Problem zu Lösen, wird die Gray-Codierung an Stelle der direkten Codierung verwendet. Dadurch beträgt der Hamming-Abstand[2] zweier benachbarter Wörter gewinnbringend immer den Wert *1*.
Tabelle 1 zeigt die beiden Codierformen gegenübergestellt.

Ziffer k	Direkte Codierung	Hamming-Abstand (k, k-1)	Gray-Codierung	Hamming-Abstand (k, k-1)
0	0000	-	0000	-
1	0001	1	0001	1
2	0010	2	0011	1
3	0011	1	0010	1
4	0100	3	0110	1
5	0101	1	0111	1
6	0110	2	0101	1
7	0111	1	0100	1
8	1000	4	1100	1
9	1001	1	1101	1

Tabelle 1: Gegenüberstellung der direkten Codierung und der Gray-Codierung [10, S. 370].

Die Codierung in binärer Form ist jedoch nicht die einzige mögliche Art. Unter anderem kann ein Chromosom und damit die Population als reellwertiger Vektor, als Matrix oder als Baumstruktur verschlüsselt werden, wenn entsprechend die genetischen Operatoren angepasst werden. [10, S. 371]

[2] „Unter dem Hamming-Abstand zweier Codewörter versteht man die Anzahl der Bits, an denen sich diese beiden Codewörter unterscheiden.“ [10, S. 370]

2.3.2 Bewertungs- und Fitnessfunktion

Die Bewertungs- und Fintessfunktion sind relevante Funktionen um die Entscheidung nach Darwins Theorie „survival of the fittest“ bzgl. der Individuen zu treffen. Festzuhalten ist, dass die Formulierung beider Funktionen immer von dem jeweiligen Problem abhängig stattfindet.
Die Aufgabe der Bewertungsfunktion ist es zu ermitteln, wie nah ein Chromosom der Generation an das Optimum der zu lösenden Aufgabe approximiert. Die Güte, Tauglichkeit wird berechnet. Die Fitnessfunktion hingegen beurteilt, wie die Chance des Chromosoms bzgl. der Wahl zur Reproduktion aussieht.
Damit ermitteln beide Funktionen streng betrachtet einen jeweils anderen Wert. [16, S. 195-197] Nichtsdestoweniger werden die Berechnungen oftmals gleichbedeutend behandelt. Dies ist z. B. dann möglich, wenn die besten Individuen in Bezug auf das Optimum genauso die höchste Wahrscheinlichkeit zur Fortpflanzung aufweisen. Dann wird für die Bewertung und für die Fitness dieselbe Funktion verwendet. [13, S. 8]
Die Bewertungs- und Fitnessfunktion können auf verschiedenen Rechenvorschriften basierend ermittelt werden. Ist die Fitness gleichbedeutend mit der Bewertung, so gilt folgender Zusammenhang [16, S. 196][10, S. 371-372]

$$B = F = \sum_{i \leq n} f_i$$

B/F stehen hierbei für die Populationsbewertung/-fitness, n ist die Anzahl der Chromosomen der aktuellen Generation und f_{i} die Bewertung/Fitness des Individuums x_{i}. Damit gilt weiter im Falle der Gleichsetzung beider Begrifflichkeiten:

$$Fit(x_i) = f_i$$

Die GA machen jedoch am häufigsten Gebrauch von der proportionalen Fitness, durch welche die Chance des Individuums sich in den folgenden Generationen fortzupflanzen direkt proportional zu seiner Güte steht:

$$Fit(x_i) = Prop_{fit}(x_i) = \frac{a \cdot f_i}{F}$$

Hierbei wird die Chromosomenfitness proportional zum Verhältnis der Bewertung des Individuums zur Summe der Bewertungen aller Chromosomen der Population berechnet. [16, S. 197]

2.3.3 Selektion

Bei der Selektion werden einige Chromosomen der Generation zur Erzeugung der neuen Population ausgewählt. Die Entscheidung, welche letztlich selektiert werden, unterliegt einigen Kriterien, die das Verhalten des gesamten GA beeinflussen. Grob betrachtet ist es sinnvoll eine Mischung aus der Weiterverwendung bereits gefundenen guten Genmaterials und der Suche neuer erfolgversprechender Erbinformationen auszuführen. [10, S.

384]
Das verfolgte Ziel der Selektion ist es, die Individuen mit einer besseren Fitness auszuwählen, sodass der GA seinem Optimum möglichst nahe kommt. Dazu gibt es verschiedene Selektionsvarianten. [15, S. 154]

2.3.3.1 Roulette-Selektion

Die populärste Selektionsmethode ist die Roulette-Selektion. Oft wird auch der Begriff fitnessproportionale Selektion verwendet. Hierbei wird das Individuum proportional zu seinem Fitnesswert mit einer bestimmten Wahrscheinlichkeit ausgewählt. [10, S. 384-385]
Anschaulich erklärt entspricht damit die Fitness eines jeden Chromosoms einem Sektor auf einem Roulette-Rad. Hat ein Individuum, wie in Abbildung 2 das Chromosom D, eine bessere Tauglichkeit vorzuweisen, so ist entsprechend der zugehörige Scheibenanteil und folglich auch die Wahrscheinlichkeit, dass genau dieses beim Drehen der Scheibe ausgewählt wird, höher. [22, S. 8]

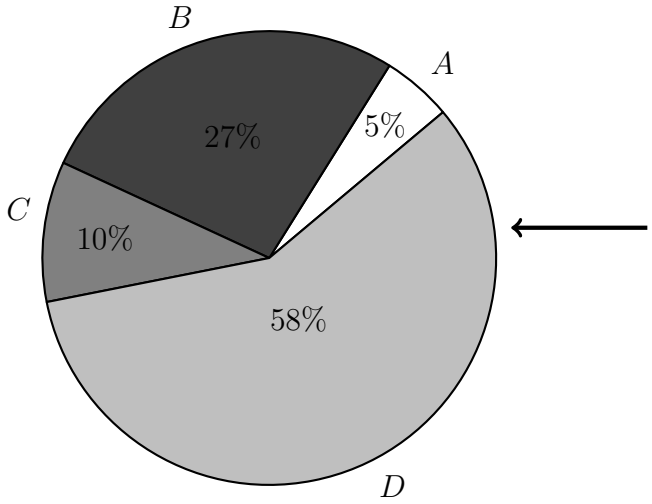


Abbildung 2: Roulette-Rad mit den vier Chromosomen A, B, C und D. In Anlehnung an [22, S. 8].

Ein Nachteil der Roulette-Selektion ist, dass die Häufigkeit des Auswählens eines konkreten Individuums für die neue Population stark von der durch die Sektoren beschriebene Häufigkeit abweichen kann. Beispielhaft erklärt, könnte das Chromosom D in Abbildung 2 beim n-fachen Drehen des Roulette-Rads n mal selektiert werden. Die entstehende Bevölkerung verliert damit die Vielfalt. Gleichzeitig liefert dieser Aspekt aber auch einen Vorteil. Indem ebenso Individuen mit schlechter Fitness selektiert werden können, wird das Risiko des Stoppens im lokalen Optimum und damit das des Nichtfindens des Optimums minimiert. [10, S. 386]

2.3.3.2 Turnier-Selektion

Bei der Turnier-Selektion werden mittels Wettkämpfen die Individuen für die folgende Generation ermittelt. Der Ablauf sieht wie folgt aus:

1. *Zufällige Wahl von k Individuen, aus der Generation, wobei $k \geq 2$ gilt*
2. *Auswahl des Chromosoms der Turniergruppe mit der höchsten Fitness*
3. *Sieger wird Teil der neuen Generation*

[15, S. 155]
Einige Vorteile liefert die Turnier-Selektion. Zum einen ist der Rechenaufwand der Methode gering, da nur die Fitnesswerte der betrachteten Individuen berechnet werden müssen. Zum anderen liefert die Art der Selektion ebenso die Option, gegenteilig ein Chromosom mit einem niedrigen Fitnesswert herauszufiltern. Als Nachteil zu beachten ist, dass das Individuum mit der geringsten Fitness nie ausgewählt werden kann. [10, S. 386-387]

2.3.3.3 Rangbasierte Selektion

Die Chromosomen der nächsten Generation werden bei der rangbasierten Selektion basierend auf ihrem Rang ausgewählt. Hierbei erhält zunächst jedes Chromosom der Generation einen Rang abhängig von seiner Fitness. Je höher die Fitness des Individuums ist, desto größer ist die Ordnung. Man erhält damit eine aufsteigende Sortierung der Individuen beginnend mit Rang 1 bis hin zu Rang n, wobei n gleich der Populationsgröße entspricht. Die Wahrscheinlichkeit eines Chromosoms mit einem niedrigeren Rang, ist damit kleiner als im umgekehrten Fall. [15, S. 155]
Zuletzt werden dann mit Hilfe eines Roulette-Rads, erstellt anhand der ermittelten Wahrscheinlichkeiten, die Chromosomen gewählt. [22, S. 9]
Ein Vorteil dieser Selektionsmethode ist, dass der konkrete Fitnesswert die Auswahlwahrscheinlichkeit des Chromosoms nicht direkt beeinflusst, sondern nur Einfluss auf dessen Rang ausübt, weil nur die Werte relativ zueinander bedeutend sind. Eine Dominanz der Selektion durch Individuen einer besonders guten Güte wird verhindert. [23, S. 75][15, S. 155] Der große Zeitaufwand der Methode ist wiederum nachteilig. [22, S. 9]

2.3.4 Rekombination

Die Rekombination ist die Kreuzung zweier Individuen zur Erzeugung eines neuen Individuums. [4, S. 107] Ob überhaupt eine Kreuzung der zwei selektierten Individuen stattfindet, ist weiter von der Rekombinationswahrscheinlichkeit abhängig. Bei binär codierten GA liegt diese bei zwischen 60% bis 90%. [23, S. 129-130] Des Weiteren wird der genetischen Operation unter Verwendung einer konkreten Rekombinationsvariante eine relevante Bedeutung zugeschrieben. Dies ist darin begründet, dass mit dem Einsatz

einer gezielten Variante eine Generation mit Individuen mit einer durchschnittlich hohen Fitness schnell erzeugt werden kann, schneller als allein durch eine zufällige Kreuzung. [16, S. 198] Die drei bedeutensten Rekombinationsvarianten werden im folgenden erklärt. Alle drei sind sowohl bei binär codierten als auch bei reellwertigen Verschlüsselungen anwendbar. [10, S. 373-375]

2.3.4.1 Ein-Punkt-Crossover

Die Ein-Punkt-Crossover-Variante der Rekombinationsmethoden ist die simpelste Art. Als erstes wird mit Hilfe einer Zufallszahl die Kreuzungsstelle ermittelt, an welcher die Elterngene zerschnitten werden. Anders ausgedrückt, wird die Position bestimmt, ab welcher die Allele der Gene der Eltern neu kombiniert werden. Als Ergebnis sind die entstehenden Kinder der Eltern so aufgebaut, dass das erste Kind zuerst die Werte bis zur Trennstelle des ersten Elternteils und dann den Rest nach der Position des anderen Vorfahrens besitzt. Das zweite Kind enthält entsprechend die restlichen Genabschnitte. [10, S. 373] Abbildung 3 veranschaulicht die Variante:

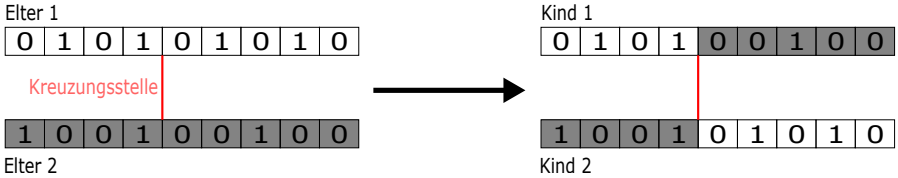


Abbildung 3: Ein-Punkt-Crossover-Beispiel mit binären Zahlen. In Anlehnung an [10, S. 374].

2.3.4.2 N-Punkt-Crossover

Erweitert man die Ein-Punkt-Crossover-Variante insofern, dass an Stelle der einen Kreuzungsstelle nun N solche verwendet werden, so erhält man das N-Punkt-Crossover. Es ist folglich eine direkte Erweiterung der Ein-Punkt-Crossover-Rekombination. Konkret werden hier N Kreuzungsstellen definiert, die die Elterngene teilen. Abwechselnd werden die Abschnitte der Individuen nun getauscht und nicht gewechselt. [10, S. 374]

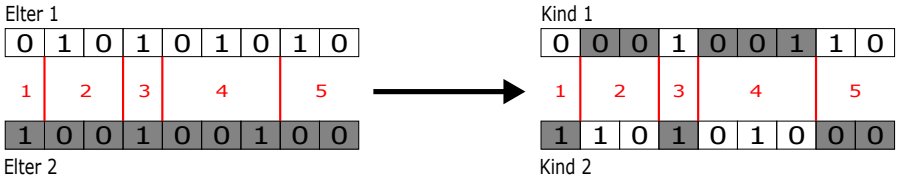


Abbildung 4: N-Punkt-Crossover-Beispiel mit binären Zahlen. In Anlehnung an [10, S. 375].

2.3.4.3 Uniform-Crossover

Bei dem Uniform-Crossover erhält jede Position der Eltern eine Nummer, sodass ein Bitmuster entsteht. Die entsprechenden Nummern werden mittels Zufallszahlen ermittelt. Liegt die Zahl 0 vor, so wird der Wert an der entsprechenden Position des ersten Elternteils an dieselbe Stelle des ersten Kindes übertragen. Bei einer 1 wird er an das zweite Kind gegeben. [10, S. 375]

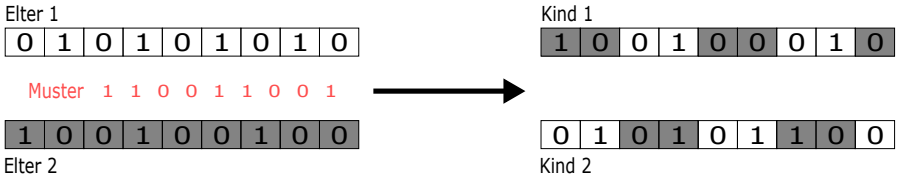


Abbildung 5: Uniform-Crossover-Beispiel mit binären Zahlen. In Anlehnung an [10, S. 375].

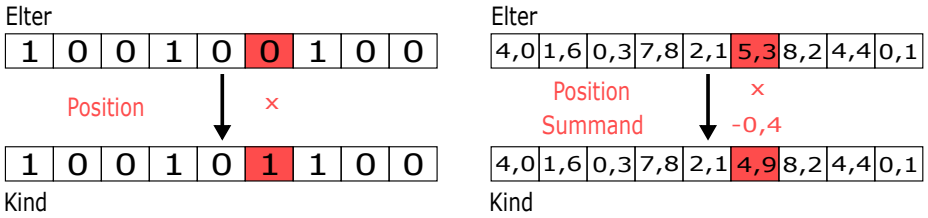
2.3.5 Mutation

Der Mutation wird die Aufgabe zugeschrieben, die zufälligen Veränderungen der Natur innerhalb des GA nachzuahmen. Als genetische Operation findet sie im direkten Vergleich mit der Rekombination relativ selten statt. [10, S. 379] GA besitzen häufig eine Mutationswahrscheinlichkeit von lediglich 0.1% bis 1%. [13, S. 11] Begründet ist dies damit, dass zu häufig mutierte Individuen mit Allelen einer guten Fitness dadurch zunichte gemacht werden können. Nichtsdestoweniger sind Mutationen weiterhin bedeutend. Mittels ihres Einsatzes kann verhindert werden, dass sich eine Population bestehend aus homogenen Individuen entwickelt. [16, S. 98, S. 200] Nur mittels der Mutation ist es möglich neue Allele unterschiedlich der der Startpopulation zu erhalten. [23, S. 9] Lokale Optima im Raum des Suchbereichs des Optimierungsproblems können überwunden werden und damit suboptimale Lösungen umgangen werden. [16, S. 98, S. 200]
Simultan zu der Operation der Rekombination gibt es bei der Mutation ebenso einige Varianten. Im Folgenden wird eine Auswahl an Mutationsformen vorgestellt.

2.3.5.1 Gleichverteilte Mutation

Die gleichverteilte Mutation verläuft so, dass zunächst eine gleichverteilte Zufallszahl bestimmt, welches Elterngen mutiert werden soll. Abhängig von der gewählten Codierungsform des Chromosoms wird weiter verändert. Ist das Individuum binär verschlüsselt, so wird das Bit des ausgewählten Gens invertiert. Bei der reellen Codierung hingegen wird die Mutationsschrittweite dem Wert des Gens hinzuaddiert. Diese entspricht dem durch Wahrscheinlichkeitsverteilung oder zufällig ermittelten Summanden. Die Abbildung 6

zeigt ein Beispiel für eine gleichverteilte Mutation bei der binären und der reellen Codierung.



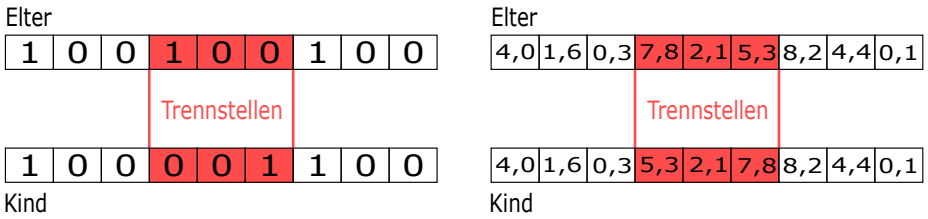
(a) Bei binärer Codierung (b) Bei reeller Codierung

Abbildung 6: Gleichverteilte Mutation bei binärer und reeller Codierung. In Anlehnung an [10, S. 380].

Die Methode kann insofern ausgedehnt werden, dass nicht nur ein Gen mutiert wird, sondern die Entscheidung einzeln für jedes Gen des Chromosoms randomisiert getroffen wird. [10, S. 380-381]

2.3.5.2 Inversions-Mutation

Bei der Inversions-Mutation werden wie bei dem N-Punkt-Crossover mit $N = 2$, zwei zufällige Trennstellen ermittelt. Die Werte innerhalb des so entstandenen Bereichs werden bei dem entstehenden Kind nun in umgekehrter Reihenfolge aufgenommen. Die übrigen Abschnitte werden lediglich übernommen. [10, S. 381-382]

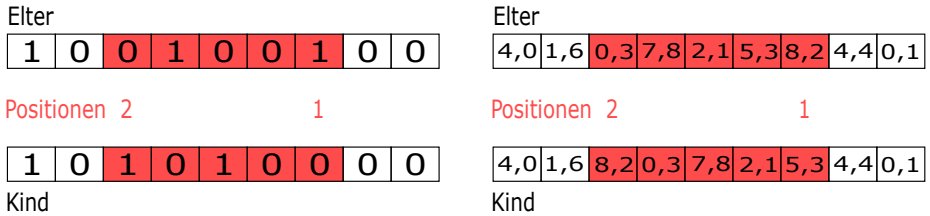


(a) Bei binärer Codierung (b) Bei reeller Codierung

Abbildung 7: Inversions-Mutation bei binärer und reeller Codierung. In Anlehnung an [10, S. 382].

2.3.5.3 Verschiebe-Mutation

Der erste Schritt der Verschiebe-Mutation ist es zwei zufällige Positionen des Elters zu ermitteln. Das Gen an erster Position, der Entnahmestelle, ersetzt den Wert an zweiter Position, der Einfügeposition. Entsprechend verschieben sich die übrigen Werte innerhalb beider Stellen nach links, bzw. nach rechts. [10, S. 382]



(a) Bei binärer Codierung (b) Bei reeller Codierung

Abbildung 8: Verschiebe-Mutation bei binärer und reeller Codierung. In Anlehnung an [10, S. 382].

2.3.6 Ersetzungsstrategie

Mittels der Selektion, der Rekombination und der Mutation ist das Ergebnis die Folgegeneration der Population. Was weiter genauer mit den Ausgangsindividuen, also den Eltern geschieht, wird anhand einer Ersetzungsstrategie definiert, von welcher verschiedene Arten existieren. [16, S. 205]

2.3.6.1 Generationenwechsel

Wird der Generationenwechsel als Ersetzungsstrategie gewählt, bedeutet das einen vollständigen Austausch der aktuellen Population durch die Kinder.
Diese einfachste Form der Ersetzung bringt jedoch den Nachteil mit sich, dass in der Nachfolgegeneration die tauglicheren Individuen der Elterngeneration verloren gehen könnten. Als Folge entsteht ein schlechterer Fitnesswert des besten Individuums und eventuell ebenso eine niedrigere durchschnittliche Güte der gesamten Population.
Ein Vorteil des Generationenwechsels hingegen besteht in der Vermeidung des Vorherrschens vieler Elternchromosomen niedrigerer Fitness in der erzeugten Population. [16, S. 205-206]

2.3.6.2 Elitismus

Der Elitismus ist eine weitere Ersetzungsstrategie. Bei dieser werden die Eliten, also die Individuen bester Fitness der Elterngeneration, unverändert in die nachfolgende Population übernommen. Wie viele solcher Eliten letztlich gewählt werden, ist variabel, jedoch ist der Wert für gewöhnlich Eins.
Der Vorteil des Elitismus gegenüber dem Generationenwechsel ist, dass die gleichbleibende Steigung der Fitnesswerte der folgenden Population gesichert ist, weil die besten Individuen stets in diese übernommen werden.
Nachteilig führt diese Art der Ersetzung dazu, dass die Kinder der besten Individuen die neue Generation dominieren können. Das passiert, wenn der Güteunterschied zwischen

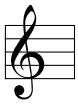
den Eliten und den neuen Chromosomen sehr groß ist. Passiert dies, ist eine mögliche Folge der Stillstand des Evolutionsprozesses. [16, S. 206-207]

3 Grundlagen der Musik

Innerhalb dieses Kapitels werden die wichtigsten musikalischen Grundlagen erklärt, um ein grundlegendes Verständnis über die im folgenden verwendeten Musikaspekte innerhalb des GA zu erhalten. Die Intention hierbei ist nicht, das musiktheoretische Wissen vollständig und in seiner ganzen Komplexität zu erläutern.
Außerdem folgt eine Einführung in die verwendete Open-Source-Programmierbibliothek „JFugue“.

3.1 Musikalische Grundbegriffe

Ein Ton wird mittels einer Note schriftlich definiert. Die Verschriftlichung verschiedener Tonhöhen erfolgt durch die Notenanordnung innerhalb der fünf Notenzeilen, bzw. auch innerhalb der zur Hilfe gezogenen Linien zur Erweiterung des Tonraums.
Je tiefer der Klang ist, desto niedriger ist die Notenposition innerhalb des Notensystems und umgekehrt. Zusätzlich definiert der Notenschlüssel weiter, welche Linie welchen Ton darstellt. Der Violinschlüssel legt den höheren Tonraum fest. Gegenteilig bestimmt der Bassschlüssel die tieferen Töne. [8, S. 37][3, S. 8, S.66]



(a) Violinschlüssel

(b) Bassschlüssel

Abbildung 9: Notenschlüssel.

Die Noten C, D, E, F, G, A, H stellen die sieben Stammtöne, also die Töne der weißen Klaviertasten dar. Um weiter die Klänge der schwarzen Tasten in Notenschrift notieren zu können, verwendet man Vorzeichen und Versetzungszeichen. Diese erhöhen oder erniedrigen den Ausgangston um einen Halbtonschritt, also um den kleinstmöglichen Schritt auf der Klaviatur. Der Unterschied zwischen einem Vorzeichen und einem Versetzungszeichen besteht darin, dass erstere direkt zu Beginn des Notenblattes stehen und für das gesamte Stück gültig sind. Versetzungszeichen hingegen stehen innerhalb des Notensystems vor der Note für die sie gelten und sind bis zum Ende des Taktes einzuhalten. Der Einfachheit halber werden beide Begriffe im weiteren Verlauf gleichbedeutend im Sinne des Versetzungszeichens verwendet, jedoch als Vorzeichen betitelt. Möchte man den Stammton um einen Halbtonschritt erhöhen, notiert man ein Kreuz (♯) vor der Note. Im Falle der Erniedrigung wählt man das Be (♭). Soll die Gültigkeit des Vorzeichens schon vor dem Taktende enden, so kann das mit Hilfe des Auflösungszeichens (♮) erreicht werden. [8, S. 40][5, S. 36-43]
Zur weiteren Einteilung innerhalb hoher und tiefer Tonräume ist der Begriff Oktave

relevant. Nach einem Durchlauf der Stammtöne, wiederholen sie sich zwar in ihrer Reihenfolge, liegen jetzt jedoch eine Oktave höher. [5, S. 36][3, S. 8]
Der Begriff Takt wurde bereits im Zusammenhang mit den Vorzeichen verwendet, was genau ist jedoch ein Takt? Generell strukturiert ein Takt den zeitlichen musikalischen Ablauf. Der Puls der Musik wird in regelmäßige, sich wiederholende Gruppen unterteilt. Ein vertikaler Strich durch die Notenlinien, bezeichnet als Taktstrich, separiert die einzelnen Takte voneinander. Zu Beginn des Musikstückes steht weiter die Taktart definiert. Sie wird als Bruch geschrieben. Der Zähler macht eine Angabe bzgl. der Pulsschlagzahl pro Takt, während der Nenner ihre Notenwerte festlegt. [3, S. 9][5, S. 84-85]
Jede Note weist eine bestimmte Gestalt auf. Beispielsweise kann der Notenkopf ausgefüllt oder hohl sein oder der Notenhals trägt ein bzw. mehrere Fähnchen. Die Kennzeichnung legt die Notenwerte, also das Verhältnis der Tondauern zueinander, fest. Ausgehend von dem längsten möglichen Wert, der ganzen Note, entstehen die kürzeren Werte durch die Halbierung dieser. Halbiert man eine ganze Note, erhält man die halbe, welche wiederum halbiert die Viertelnote bedeutet. Entsprechend der Länge der Noten gibt es genauso Pausen unterschiedlicher Länge. [5, S. 34, S. 48]

Name	Note	Pause
Ganze	𝅝	𝄻
Halbe	𝅗𝅥	𝄼
Viertel	♩	𝄽
Achtel	♪	𝄾

Tabelle 2: Einige Notenwerte und Pausen gleicher Länge. Erstellt nach [5, S. 48].

Das folgende Beispiel verdeutlicht den Zusammenhang zwischen einem Takt und den Notenwerten:

Beispiel. *Die zu Beginn des Notenblattes definierte Taktart des $\frac{4}{4}$-Taktes legt fest, dass innerhalb eines Taktes vier Noten der Länge $\frac{1}{4}$ erlaubt sind. [8, S. 39]*

Die Notenwerte bezeichnen allerdings nur die relative Dauer einer Note. Das bedeutet, dass lediglich Aussagen, wie bspw. die erste Note ist doppelt so schnell wie die folgende, gemacht werden können. Ebenso sieht es bei den Pausenwerten aus. Um aber nichtsdestoweniger absolut messbare Notendauern zu erhalten, kommt ein weiterer musikalischer Begriff ins Spiel: Das Tempo. Tempoangaben stehen zu Beginn eines Stückes. Zuerst steht ein Notenwert, welcher meist dem Taktnenner entspricht. Dann folgt eine Zahl, die die Anzahl der Schläge pro Minute (BPM) abbildet.
Neben der messbaren Tempoangabe, existieren weiter Tempobezeichnungen als „[...] Ausdruck eines subjektiven Tempoempfindens [...]“ [5, S. 80]. Allgemein erfolgen die Bezeichnungen in der italienischen Sprache. [5, S. 79-81]
Ein weiterer Begriff in dem Bereich der Musik ist der Intervallbegriff. Ein Intervall stellt

den Abstand zwischen zwei Tönen dar. Es ist möglich beide Töne in Melodie, also nacheinander, oder gleichzeitig als Akkord zu spielen. [5, S. 113] Anstatt von links nach rechts, wie nacheinander gespielte Töne, sind Akkordtöne übereinander positioniert im Notenblatt notiert. [5, S. 35] Nach den lateinischen Ordnungszahlen werden Intervalle bezeichnet. Die ersten acht Intervalle lauten: Prime, Sekunde, Terz, Quarte, Quinte, Sexte, Septime, Oktave. [5, S. 113]
Mehr als zwei Töne können im Zusammenklang als Akkord gespielt werden. Erklingen drei Töne gleichzeitig, wird der Akkord als Dreiklang bezeichnet. [21, S. 165] Generell bilden ein Grundton, gefolgt von einer Terz und anschließend von einer Quinte, wieder vom Grundton aus betrachtet, einen Dreiklang. [3, S. 17] Folglich gibt es viele mögliche Dreiklänge, weiter wird jedoch nur auf den Dur- und Moll-Dreiklang eingegangen. Von unten nach oben betrachtet, bilden eine große Terz (vier Halbtonschritte) gefolgt von einer kleinen Terz (drei Halbtoschritte) den Dur-Dreiklang. Dreht man die Reihenfolge der zwei Intervalle um, erhält man einen Moll-Dreiklang. [5, S. 164]

3.2 Wirkung der Musik

Musik kann zahlreiche unterschiedliche Emotionen in den Zuhörern auslösen. Sie kann einen depressiven Charakter, wie Chopins Trauermarsch, aufweisen, gegenteilig sehr lebensfreudig sein, wie z. B. das Folklied „My Bonnie is over the ocean“ oder ganz andere Empfindungen hervorrufen. Um genau diese zahlreichen Emotionen zu bewirken, kann auf einige musikalische Mittel als Werkzeug zurückgegriffen werden. [12, S. 42-43]
Ein Mittel, um einen speziellen Musikcharakter hervorzubringen ist der Verlauf der Melodie. Unter dieser wird die gestaltende Aneinanderreihung der Töne verstanden. [5, S. 149-150] Die Form des Melodiebogens hat Einfluss auf die Wirkung auf den Menschen. Ist der Bogen bspw. symmetrisch, ist das Ende des Komponierten zu erwarten, viele kurze Melodiebögen derselben Richtung oder ein sehr langer Bogen hingegen, üben wiederum einen anderen Einfluss auf den Zuhörer aus.
Eng im Zusammenhang mit der Melodie stehen die Intervalle, welche sequentiell geordnet eine Melodie erzeugen. Grob betrachtet, kann man sagen, dass jeder Intervall einen Grundcharakter besitzt. Der Trauermarsch Chopins wirkt als Beispiel unter anderem durch den Einsatz vieler Primen (zwei identische Töne) gerade so bedrückend. [12, S. 42-43]
Weiter ist oftmals auch die Größe des Intervalls für die Wirkung ausschlaggebend. Töne mit einem großen Abstand zueinander, können relativ schnell irritierend wirken, da sie dem Gesetz der Ähnlichkeit widersprechen. [18]
Die Zusammenklänge lassen sich weiter grob kategorisieren in konsonante und dissonante Intervalle. Der Klang zweier oder mehrerer Töne ist konsonant, wenn ihr Klang harmonisch und reibungslos wirkt. Dissonant bedeutet gegenteilig einen spannungsvollen, unharmonischen Klang zu hören.
Eine eindeutige Einordnung der Intervalle in eine der beiden Kategorien lässt sich jedoch nicht immer ohne weiteres vornehmen, jedoch kann man bspw. grob sagen, dass alle Primen und Oktaven konsonante Intervalle sind, während alle Septimen dissonant

sind. [5, S. 119][21, S. 159, S. 346]
Ebenso der Dur- und Moll-Dreiklang sind Konsonanzen. [5, S. 164] Weiter erzeugen beide Dreiklangarten eine unterschiedlichen Stimmung des Hörers. Im allgemeinen erweckt ein Moll-Dreiklang eine traurige Musikcharakteristik, wohingegen ein Dur-Dreiklang gegenteilig fröhlich wirkt. [12, S. 41-44]
Das Tempo beeinflusst weiter unsere Emotionen. Ob wir ein Tempo als schnell und erregend oder doch eher traurig und ruhiger wahrnehmen, steht im engen Zusammenhang mit dem Tempo unseres Herzschlages. 70 Mal pro Minute schlägt das Erwachsenenherz. Tempi, die nun schneller als 70BPM sind, erzeugen erstere Wirkung, langsamere hingegen letztere. [12, S. 45]
Außerdem gibt es den Einsatz von Mustern in der Musik. Wegen des Gesetztes der Ähnlichkeit sind die Zuhörer fähig solche zu erkennen und sehen diese sogar entgegen. Arbeitet man folglich mit Mustern, unterstützt man die Erwartungshaltung des Menschen. [18]
Zuletzt steht noch der Rhythmus in enger Verbindung zu den Mustern, weil bei diesem oft die Neigung zur Wiederholung von Gleichem oder Ähnlichem besteht. [21, S. 576] „Rhythmus kann man definieren als proportionierte und wiederholte zeitliche Abfolge von Tönen.“ [12, S. 45] Anordnungen, welche rhythmisch organisiert sind, haben den Vorteil der Einprägsamkeit und üben damit generell einen Einfluss auf die menschlichen Emotionen beim Hören der Musik aus. [12, S. 45]

3.3 JFugue

„JFugue“ ist eine Open-Source-Programmierbibliothek, welche im Jahre 2002 erstmals von David Koelle veröffentlicht wurde. „JFugue 5.0“, die aktuelle Version, erschien 2015.[3]
Die Bibliothek ermöglicht die einfache Musikprogrammierung in der Programmiersprache „Java“ ohne dass der Gebrauch komplexer „MIDI“ [4]-Dateien nötig ist. [24]
Die Musikprogrammierung mittels „JFugue“ ist dank des „MusicStrings“ sehr simpel. Das formatierte String-Objekt enthält musikalische Anweisungen. Nach dem Parsen dieses und der Objekterzeugung durch „JFugue“, ist es möglich Musik zu generieren und abzuspielen. Genauer stellt jeder Zeichensatz, separiert durch Leerzeichen, z. B. eine Note, einen Akkord oder einen Stimmenwechsel dar. Ein solcher Zeichensatz wird auch als Token bezeichnet. Die Groß- und Kleinschreibung ist hierbei unbedeutend. [9, S. 21-22]
Die im weiteren Verlauf der Arbeit verwendeten Elemente bzgl. des „MusicStrings“ werden nun kurz erklärt. Um die Einheitlichkeit aller Token zu gewähren, wird jeweils die frei festgelegte Schreibweise genannt.

[3] David Koelle. Download JFugue 5.0. Get JFugue and start creating programs that sing!. 2002-2020, besucht am: 12.01.2021. URL: http://www.jfugue.org/download.html.
[4] „Abk. für **M**usical **I**nstrument **D**igital **I**nterface [Hervorhebung im Original] (Schnittstelle für digitale Musikinstrumente), seit 1983 genormte Verbindung zwischen elektronischen Instrumenten und dem Computer [...].“ [21, S. 424]

(1) *Note*
Das Token des „MusicStrings“ beginnt mit der Note, die großgeschrieben wird. Die möglichen Noten sind: C, D, E, F, G, A, B. Die Darstellung der Noten in numerischer Form unterstützt „JFugue“ ebenso, was jedoch an dieser Stelle nicht weiter relevant ist. [9, S. 22-23]

(2) *Vorzeichen*
Notenvorzeichen können nach der Note selbst angegeben werden. Das Zeichen $\sharp$ steht für ein Kreuz, $\flat$ für ein Be. Steht keines der beiden Zeichen an zweiter Stelle, wird die Tonhöhe nicht weiter verändert. [9, S. 23]

(3) *Oktave*
Es folgt die numerische Oktavenangabe. Werte von 1 bis 10 sind möglich. Der definierte Standardwert bei einzelnen Noten ist die 5 und bei Akkorden die 3. In dieser Arbeit wird der Standardwert nichtsdestoweniger immer notiert. [9, S. 23]

(4) *Tondauer*
Den Abschluss des Tokens bildet die Angabe der Tondauer mit Hilfe eines Buchstabens. „JFugue“ unterstützt Tondauern von einer ganzen Note bis zu der Einhundertachtundzwanzigstelnote. Im Weiteren sind jedoch nur die folgenden vier Tondauern wichtig: Ganze (w), Halbe (h), Viertel (q), Achtel (i). Die viertel Note ist der Standardwert, wird aber auch immer angegeben. [9, S. 26]

(5) *Akkord*
Um einen Akkord zu spielen liefert „JFugue“ eine Vielzahl von solchen vordefinierten mit, die gleichzeitig eigenständig formuliert werden können. Dann werden die zusammen erklingenden Noten mit einem + verbunden. In der Arbeit werden alle Akkorde händisch gebildet. [9, S. 24]

(6) *Tempo*
Das Tempo kann ebenso definiert werden. In dieser Arbeit wird jedoch auf den Standardwert von 120BPM zurückgegriffen, was dem allegro (italien.: lustig, heiter) Tempo entspricht. [9, S. 36][21, S. 23]

(7) *Instrument*
Eine Auswahl von 128 verschiedenen Instrumenten besteht. Das Standardinstrument, das Klavier, wird folgend als einziges Instrument verwendet, weshalb der „MusicString“ nicht erweitert wird. [9, S. 31-33]

Zusätzlich zu den genannten Elementen des „MusicStrings“ liefert die Programmierbibliothek noch viele weitere mit.
Um die programmierte Musik abspielen zu können, wird die Klasse „Player“ mitgeliefert. [9, S. 63]
Obwohl das ursprüngliche Ziel „JFugues“ ist auf MIDI-Dateien bei der Musikprogrammierung zu verzichten, liefert die Open-Source-Programmierbibliothek einige Funktionalitäten mit, um mit diesen Dateien arbeiten zu können. Dadurch ist es möglich die

große bereits existierende Menge an „MIDI“ -Dateien genauso weiter in „JFugue“ zu nutzen, indem u. a. „MIDIs“ kreiert oder abgespielt werden können. [9, S. 71]

4 Genetische Algorithmen zur Musikkomposition

Das Vorhaben Musik mittels eines GA zu komponieren, ist bereits angegangen worden. Das folgende Kapitel beschreibt einige der bereits existierenden Konzepte genauer und liefert damit einen groben Überblick über den bisherigen Stand der Technik.
Konkret wird zuerst auf die Schwierigkeit der Bewertung von Musik eingegangen. Es folgen zwei verschiedene Methoden zur Fitnessbewertung und die Darstellung bisheriger Problemcodierungsansätze. Die Schlussfolgerung der Ergebnisse beendet das Kapitel.

4.1 Die Bewertung kreativer Lösungen als Herausforderung

GA sollen für beliebige Optimierungsprobleme beste definierte Lösungen finden. [23, S. 20] Auf Grund ihrer Leistungsfähigkeit in verschiedensten großen Problemräumen, ist es naheliegend diese ebenso auf kreative Bereiche, wie die der Kunst und der Musik anzuwenden. Eine algorithmische Fintessfunktion muss als feste Voraussetzung für die Durchführung eines GA existieren. [1, S. 131-135] Und genau hierbei liegt das Problem bei Anwendungen im musikalischen Bereich.
Die bestehende Behauptung die Schönheit der Musik könne nicht in Regeln festgehalten werden, gilt es nämlich mittels geeigneter Güteermittlungen zu entkräften. Es muss ein Weg gefunden werden, wie der ästhetische Aspekt der Musik algorithmisch formuliert werden kann. [14, S. 55] Allerdings stellt es eine enorme Herausforderung dar explizite Funktionen für die Fitnessbewertung von Musikstücken zu definieren, weil hier für gewöhnlich die menschlichen Sinne zur Bewertung eingesetzt werden. Denn genau diese Sinne ermöglichen es uns mit z. B. Eindrücken und Vorlieben bzgl. der Musik eine Aussage über ihre Güte zu treffen.

4.2 Die menschliche Bewertung - Einsatz interaktiver genetischer Algorithmen

Ist die Schwierigkeit eine algorithmische Fitnessbestimmung für ein zu lösendes Problem zu finden kaum bis gar nicht überwindbar, so wird als Ausweg eine menschliche subjektive Bewertung herangezogen. Es wird Gebrauch von interaktiven GA gemacht.
Interaktive GA benötigen existierende Personen, die die Individuen bewerten. Der Mensch bestimmt die Fitnesswerte und ist damit ein Teil des gesamten Optimierungsprozesses. Eine Mensch-Maschinen-Interaktion findet hier statt, um das Ziel einer optimalen Population zu erreichen.
Diese Art der Fitnessbewertung bringt jedoch einige Nachteile gegenüber der berechenbaren Gütebestimmung mit sich. Zum einen können nur kleinere Populationen und insgesamt auch weniger Generationen betrachtet werden. Der Mensch besitzt einfach nicht die Fähigkeit die Tauglichkeitsbeurteilung so schnell wie eine Maschine vorzunehmen. Zum anderen gibt es zusammenhängend mit der Anzahl der Individuen betrachtet das

Problem der Nutzerermüdung. Er kann nämlich nicht zu viele Individuen bewerten, da er dafür seine gesamte Aufmerksamkeit benötigt, was auf die Dauer sehr anstrengend wird.
Wiederum vorteilhaft an der interaktiven Fitnessbewertung ist, dass nur wenige Generationen erzeugt werden müssen, um qualitativ gute Ergebnisse zu erhalten. [7, S. 10-18] Das Finden der Lösung innerhalb des Problemraumes wird hierbei außerdem stark von dem menschlichen Sinn für Ästhetik geleitet, was wohl mit der bedeutenste Grund für den Einsatz interaktiver Beurteilungen ist. [1, S. 131]
„GenJam“, 1994 von John A. Biles entwickelt, ist ein konkretes Beispiel für den Einsatz interaktiver GA. Es stellt ein Model dar, welches mit Hilfe eines GA, Jazz-Stücke komponiert. Die Idee ist es einen Jazzschüler zu imitieren, welcher versucht ein Stück zu improvisieren. Die Kreationen bewertet der Mensch in der Rolle seines Mentors. Er ist damit für die interaktive Fitnessbeurteilung zuständig. [1] „Es erhält als Vorgabe einen rhythmischen Rahmen, eine Akkordfolge sowie eine automatisch (durch Fremdsoftware) erzeugte Begleitung aus Klavier, Bass und Schlagzeug im MIDI-Format.“ [8, S. 52].
Wie genau macht das Programm „GenJam“ nun Gebrauch der interaktiven Fitnessbestimmung? Zuerst definiert das Model zwei Populationen. Auf der einen Seite ist das die Population gebildet aus den zu verwaltenden Takten, wobei eine Tonfolge einen Takt bestimmt. Diese wird als Taktpopulation bezeichnet. Andererseits ist die zweite Population, die Phrasenpopulation, entsprechend die Aufstellung der Takte.
Mit dieser Grundlage kann der GA mit der Bewertung der Individuen, also der Takt- und Phrasenpopulation beginnen. Die Entscheidung, ob das Stück als gut oder schlecht bewertet wird, trifft der Mentor, der Mensch. Bereits während des durchgängigen Abspielens der Improvisation entscheidet er, ob ihm das Gehörte gefällt oder nicht. Er drückt entsprechend für ein gutes Empfinden den Buchstaben „g“ („good“) und ansonsten das „b“ („bad“). Die positive Bewertung inkrementiert damit den Fitnesswert, die negative dekrementiert diesen. Abhängig von dem Moment, in welchem die Taste gedrückt wird, bewertet der Mensch außerdem entweder die eine oder andere beider Populationen. [8, S. 52] John A. Biles führt an dieser Stelle jedoch den Begriff „Fitness Bottleneck“, zu deutsch „Fitness Flaschenhals“, ein. Wie bereits oben beschrieben, ist der Mensch nämlich als Engpass anzusehen, durch welchen die Populationsgröße und die Anzahl der Generationen eingeschränkt werden. [1]
Natürlich implementiert das Programm „GenJam“ ebenso die weiteren Schritte des GA, wie unter anderem die Mutation, worauf in dieser Arbeit jedoch nicht tiefer eingegangen wird, weil an dieser Stelle die Art der Fitnessbewertung den Fokus erhalten soll.

4.3 Regelbasierte Fitnessberechnung mittels der gewichteten Summe

Weiter existiert bereits der objektive Ansatz die Fitness der Individuen mittels einer gewichteten Summe zu berechnen. Hier werden zunächst einzelne qualitative Teilaspekte betrachtet und Werte für diese bestimmt, anschließend summiert man diese unter

Berücksichtigung ihres Gewichtes zu der Gesamtfitness auf.
Es wird damit konkret versucht die menschlichen Sinne mittels subjektiver Regeln innerhalb eines Algorithmus zu ersetzen, was eine mögliche Lösung für die in 4.1 beschriebene Herausforderung darstellt.
Es folgt eine Beschreibung bisheriger Ansätze der gewichteten Summe.
George Papadopoulos und Geraint Wiggins haben ein System zur Erzeugung von Jazzmusik konzipiert, bei welchem zu vorgegebenen Akkorden eine Melodie entwickelt werden soll. [8, S. 53]
Ein Fokus der Arbeit besteht darin, die im Allgemeinen subjektive Bewertung der Melodien nun objektiv zu ermitteln. Objektivität bedeutet in diesem Sinne jedoch nicht, dass auf die „richtige“ oder „universelle“ Art und Weise Melodien erzeugt werden, sondern vielmehr, dass ein beständiger und neutraler Weg für das Vorhaben gefunden wird. [18]
Papadopoulus und Wiggins nutzen für dieses Ziel die gewichtete Summe. Im ersten Schritt definieren sie acht Merkmale, die bei den Chromosomen der Population betrachtet werden. Jedem Merkmal wird dabei ein Gewicht zugeteilt. Einmal wird dafür die Größe der Intervalle betrachtet. Weiter geht es mit der Musterwiederholung basierend auf dem Tonhöhenverlauf, dem Fortbestehen von Noten bei Akkordwechseln, der Ähnlichkeit zu einem Tonhöhenverlauf und zur Geschwindigkeit der Melodie. Außerdem wird bewertet, welche Note mittels eines Schlages betont wird. Die erste und dritte Note im Falle eines vier Viertel Taktes werden insofern betrachtet, ob z. B. an der Position des Schlages eine Pause steht. Merkmal acht ist die Bewertung der Länge der Note, was der Nutzer des Systems selber entscheidet. Abhängig davon, inwiefern genau das jeweilige Merkmal als positiv, bzw. als negativ wahrzunehmen ist, erfolgt die Gewichtung. Beispielsweise wird im Allgemeinen ein zu großes Intervall als störend in Bezug auf den Melodiefluss empfunden, weshalb die Bewertung negativ mit dem Wert -20 ausfällt. Soeine Beurteilung findet bei allen acht Kriterien statt.
Nachdem sämtliche Teilbewertungen abgeschlossen sind, wird die Gesamtfitness anhand der gewichteten Summe berechnet. [18][8, S. 54-55]
Michael Towsey, Andrew R. Brown, Susan Wright und Joachim Diederich formulieren in ihrer Arbeit „Towards melodic extension using genetic algorithms“ einen sehr ähnlichen Ansatz zu dem von George Papadopoulos und Geraint Wiggins. Sie gehen der Frage nach, ob es es qualitative Merkmale und melodische Eigenschaften gibt, die hilfreich bei dem Finden guter Melodien sind? Dafür erläutern sie ebenso einige Merkmale, genau 21, die betrachtet werden können, um diese anschließend bspw. bei der Berechnung der gewichteten Summe einzusetzen. [14, S. 54-57] Es folgt eine Aufzählung einiger Merkmale, um einen Überblick über die Gesamtheit erhalten zu können:

(1) *Tonhöhenbereich*
Die Amplitude der Melodie wird ermittelt mittels der Division des vorliegenden Tonhöhenbereichs (höchste Note subtrahiert mit der tiefsten Note) durch den maximalen erwünschten. [14, S. 57]

(2) *Dissonante Intervalle*
Zuerst erfolgt die separate Dissonanzbewertung der Intervalle, wobei jedem Intervall ein Gewicht zugeteilt wird. So wird z. B. eine kleine Sekunde (ein Halbton-

schritt) mit Null bewertet, während eine große Septime (elf Halbtonschritte) den Wert 1 zugeteilt bekommt und damit als stärker dissonant beurteilt wird. Errechnet wird das Ergebnis dann aus der Summe aller Intervalldissonanzen als Zähler und der Anzahl aller Intervalle als Nenner. [14, S. 57]

(3) *Konturverlauf*
Die Tendenz der Melodie zu steigen oder zu fallen wird betrachtet. Der Zähler entspricht der Summe aller steigender bzw. fallender Intervalle und der Nenner ist die Summe aller dieser. [14, S. 58]

(4) *Rhythmische Vielfalt*
Der rhythmische Zusammenhang steht im Fokus. Es wird betrachtet, wie viele verschiedene Notenwerte aller möglichen verwendet werden. Der Zähler ist die Zahl verwendeter verschiedener Notenwerte und wird geteilt durch die Anzahl aller. [14, S. 58]

4.4 Codierung eines Musikstückes

Noch vor der Bestimmung der Fitness eines Musikstückes, ist es relevant, wie die Codierung des Optimierungsproblems aussieht. Es stellt sich die Frage, wie die Musik im Computer notiert werden kann, sodass diese innerhalb des GA genutzt werden kann?
Handelt es sich um ein Klavierstück, so ist die erste Möglichkeit die insgesamt 88 verschiedenen Klaviertasten von eins bis 88 durchzunummerieren. Die Null stellt eine Pause dar. Weiter wird die Tondauer als Vielfaches der minimalen Zeiteinheit codiert. Mit diesem Verfahren ist es nicht nur möglich einstimmige Musikstücke abzubilden, sondern ebenso mehrstimmige. An Stelle lediglich einer Reihe von diesen ganzzahligen Paaren, also den Noten und deren Dauern, werden mehrere solcher Längen parallelisiert, um die Mehstimmigkeit wiederzugeben. [20, S. 3036]
Die numerische Darstellung der Noten wurde bereits konkreter in dem Paper „A genetic algorithm for composing music“ angewandt. Eine kurze aus vier $\frac{4}{4}$-Takten bestehende Komposition ist hier codiert in einem Zahlen-Array. Ausgehend von den Grundannahmen, die Referenznote sei das $C4$, eine Notenweite von zwei Oktaven, also von 14 Noten (lediglich die weißen Tasten der Klaviatur) stehen zur Verfügung und eine Pause wird als 0 verschlüsselt, ergibt sich zuerst das Intervall $[0, 14]$ für die möglichen Codierungszahlen. Weiter fehlt nun die Tondauer. Die Achtel-Note wird als kleinstmögliche Tondauer definiert. Ein Takt kann damit aus maximal acht Noten bestehen, eine Komposition aus maximal 32. Das Array besitzt somit eine Länge von 32. Genauso vergrößert sich das vorherige Intervall nun auf $[0, 15]$. Die Zahl 15 steht nämlich für jede längere Tondauer als die der kleinsten von einem Achtel. Die Abbildung 10 zeigt ein Beispiel für eine mögliche Komposition unter den genannten Regeln. Die unter den Noten stehenden Zahlen sind als Einträge an den Indizes 0 bis 31 des Codierungs-Arrays zu verstehen.[11, S. 163-166]

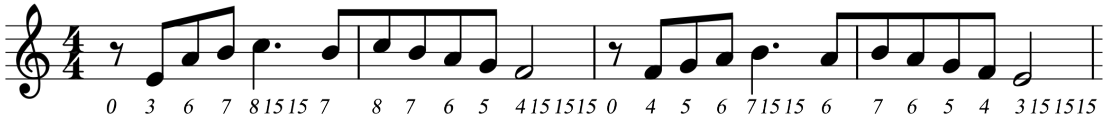


Abbildung 10: Codieraufbau mit Pausen und unterschiedlichen Tondauern. In Anlehnung an [11, S. 165].

Ein weiterer möglicher Codierungsansatz ist es das Musikstück als Zeichenkettenreihe darzustellen. Dabei werden unter anderem die Noten als Buchstaben und die Tondauern als Zahlen codiert. Das Vorzeichen Be (♭) entspricht einem − und das Kreuz (♯) einem +. Außerdem können dann mittels anderer Zeichen Musikelemente vom Tempo über die Oktave bis hin zu dem Vortragsziel verschlüsselt werden. [20, S. 3036]

4.5 Erkenntnisse aus dem aktuellen Stand der Technik

Die in diesem Kapitel erläuterten Ansätze bzgl. der Herangehensweise für die Implementierung eines GA zur Musikkomposition liefern bereits eine Reihe an möglichen Ideen für die genaue Umsetzung eines eigenen Algorithmus.
Zunächst liefern die zwei Problemcodierungmethoden konkrete Herangehensweisen an die Schwierigkeit Musik algorithmisch darzustellen. Insbesondere der Ansatz der Zeichenkettencodierung ist für die weitere Arbeit relevant und wurde ausgearbeitet. Nichtsdestoweniger werden auch die genannten Konzepte eines festen erlaubten Notenintervalls oder einer minimalen Tondauer keineswegs verworfen, sondern passend umformuliert. Damit sind beide Ansätze als Orientierung und Inspiraton für die eigene später erklärte Codierung anzusehen.
Ähnlich verhält es sich mit der Ermittlung der Fitness. Die Problematik innerhalb kreativer Einsatzbereiche der GA gilt es als Herausforderung zu meistern. Das bedeutet, dass zwar die interaktive Fitnessbewertung interessant für einen möglichen Güteansatz ist, allerdings soll das Ziel der Arbeit eine objektive Bestimmung sein. Dafür liefern vor allem die genannten Merkmale in 4.3 hilfreiche Aspekte für die spätere Formulierung einer gewichteten Summe als Fitnessberechnung.

5 Entwurf eines genetischen Algorithmus zur Musikkomposition

Der konkrete Entwurf eines GA zur Musikkomposition und die dabei getroffenen Entscheidungen sind Thema des folgenden Kapitels. Auf die einzelnen Teilschritte zur Konzipierung eines GA, wie z. B. auf die Codierung oder auf den Operator die Mutation, wird hierbei genauer eingegangen, sodass zum Ende des Abschnittes das Verständnis über die Herangehensweise und den Umfang des Algorithmus vollständig dargelegt wurde.

5.1 Festgelegte Entwurfsentscheidungen

Das Ziel des entworfenen GA ist es, eine kurze fröhliche und harmonische Musiksequenz von wenigen Sekunden für das Klavier zu erzeugen. Dafür wurden zu Beginn einige frei definierte Entwurfsentscheidungen getroffen.
Das Klavierstück ist mehrstimmig. Das bedeutet an dieser Stelle, dass eine Melodie im Violinschlüssel zusammen mit Akkorden im Bassschlüssel gespielt wird. Somit gibt es zwei Arten von Individuen: Das Melodie-Individuum und das Akkord-Individuum.
Das Melodie-Individuum, die erste Stimme, wird vollständig unabhängig von den Akkord-Individuen erzeugt und entwickelt. Es kann damit während des Durchlaufens des GA für sich alleine behandelt werden.
Etwas anders verhält es sich bei den Akkorden. Direkt relevant anzumerken ist, dass Dreiklänge erzeugt werden sollen, weshalb die Begriffe Akkord und Dreiklang im weiteren Verlauf bedeutungsgleich verwendet werden, obwohl dies musikalisch betrachtet nicht ganz richtig ist. Jedem Akkord-Individuum wird nämlich eine Melodie zugeordnet, welche dieses dann in gewisser Weise untermalt. Ein Dreiklang selbst kann nämlich nur im Zusammenklang mit einem Melodieton erklingen, weil es als zweite Stimme eine hierarchisch tiefere Position einnimmt. Somit müssen die Akkorde zum einen für sich selbst stehend harmonisch, zum anderen in der Kombination mit dem Melodieton passend sein.
Es gilt, dass die Akkorde als freiwillige Erweiterung anzusehen sind. Eine Melodie kann ohne eine zweite Stimme existieren, umgekehrt gilt dies jedoch nicht. Sinn und Zweck der Mehrstimmigkeit ist es im Gesamten das komponierte Musikstück etwas komplexer zu gestalten.
Die Idee dafür basiert auf den eingeführten zwei Populationen innerhalb des Models „GenJam“, beschrieben in dem Kapitel 4.4. Anders werden nun aber nicht eine Takt- und Phrasenpopulation, also nicht mehr eine Teilmenge und die Gesamtheit, sondern jeweils zwei verschiedene Mengen mit unterschiedlichen Relationen zueinander betrachtet.
Weitere Ausgangsannahmen sind, dass die Melodie zum ersten Erzeugungszeitpunkt aus genau fünf $\frac{4}{4}$-Takten bestehen muss. Jeder Takt muss vollständig mit Tönen gewisser Notendauern gefüllt sein. Damit ist der Einsatz von Pausen an dieser Stelle nicht erlaubt.

Die Anzahl der Akkorde ist dann aufbauend auf dem zugehörigen Melodie-Individuum der Endpopulation. Die möglichen Töne der jeweiligen Art liegen innerhalb eines jeweils definierten Intervalls auf der Klaviatur. Sowohl die weißen, als auch die schwarzen Tasten werden berücksichtigt und eine Tondauer wird aus einem Bereich zwischen einer minimalen und maximalen solchen bestimmt.
Auf Basis dieser Grundentscheidungen arbeiten die folgenden konkreteren Festlegungen und Prinzipien.

5.2 Codierung

Der Entwurf eines GA beginnt mit der Definition der Codierung. Um die Musik sinnvoll darzustellen, wird ein Individuen in dieser Arbeit als String, genauer als „MusicString“ der Open-Source-Programmierbibliothek „JFugue“ dargestellt. Der Ideenansatz zur Codierung mittels einer Zeichenkette ist dem Kapitel 4.4 entnommen. Als neu kann der unterstützende Einsatz „JFugues“ betrachtet werden. Die Bibliothek liefert nämlich bereits genau solche Strings, verwendet aber teils andere Zeichen, um dasselbe auszudrücken.
Des Weiteren ist anzumerken, dass die Codierung in erster Linie konkret für das Klavier als Zielinstrument konzipiert und für dieses ausgearbeitet ist. Allerdings können einige der folgenden Ansätze genauso auf andere Instrumente übertragen werden, solange sich die Notenschriften beider Instrumente gleichen.
Zunächst wird ein Chromosom als eine Melodie innerhalb des Violinschlüssels definiert, bei welcher maximal ein Ton gleichzeitig erklingt. Jeder Takt der zu Beginn erzeugten Anfangspopulation bestehend aus fünf Takten der Taktart $\frac{4}{4}$, setzt sich wiederum aus einer unterschiedlichen Anzahl an Noten, abhängig von der Tondauer, zusammen. Weiter ist eine Musiknote als ein Gen zu interpretieren. Damit ergibt sich der Zusammenhang, dass genauso wie die DNS die Erbinformation des Individuums ist, alle Noten der fünf Takte aneinandergereiht das Musikstück bilden.
Ein Gen besitzt die vier Parameter Notenname, Vorzeichen, Tondauer und Oktave. Die gültigen Allele der ersten drei genannten Parameter sind bereits in dem Kapitel JFugue erklärt. Während die Darstellung des Notennamens und der Tondauer gleich dem Ansatz in Kapitel 4.4 sind, codiert „JFugue“ das Vorzeichen unterschiedlich mit dem Zeichen $\flat$, bzw. $\sharp$.
Der vierte Parameter, die Oktave, verhält sich anders zu den Dreien, weil das zur Verfügung stehende Intervall bzgl. der Oktavenwahl „JFugues“ auf die drei Oktaven 4, 5 und 6 minimiert wird. Im Gesamten betrachtet, ist ein Gen somit eine von 36 möglichen Tonhöhen. Die Anzahl ergibt sich aus den verschiedenen Kombinationen der sieben Notennamen mit den drei möglichen Vorzeichen innerhalb der drei Oktaven.

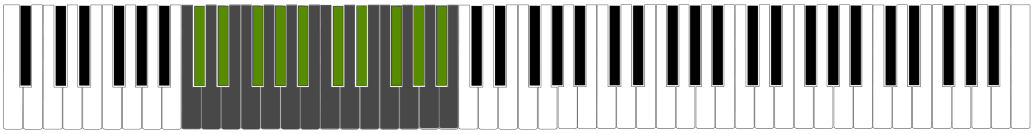
Abbildung 11: Die 36 erlaubten Töne dargestellt auf der Klaviatur.

Das bedeutet, dass ein Gen mittels der Konkatenation der Parameter in der Reihenfolge Notenname, Vorzeichen (nur bei Kreuz oder Be), Oktave und Tondauer abgebildet wird. Durch das Zusammenfügen aller Gene eines Individuums wird der Repräsentant codiert.

Beispiel. *A♯4q A♯4i G4q C♭5i G♭4q A♭4w G♯5h A♯5q D5q F♯6w E♯5h A♭4i G♯4i E5q stellt die Codierung eines Individuums dar. Das Individuum besteht aus 14 Genen. Das erste Gen hat das Allel A♯4q und das letzte Gen besitzt das Allel E5q.*

Wichtig zu verstehen ist, dass ein Genotyp dekodiert nicht immer eindeutig einen Phänotypen darstellt. Die Menge aller möglichen Genotypen ist größer als die aller Erscheinungsbilder. Dies ist bedingt durch die möglichen Vorzeichen. Beispielsweise entsprechen so der erhöhte Ton $C\sharp$ und das erniedrigte $D\flat$ in derselben Oktave betrachtet letztlich einen und denselben Ton und Phänotypen.
Die Codierung wird nun um die Möglichkeit erweitert ein mehrstimmiges Musikstück zu erzeugen. Eine zweite Stimme innerhalb des Bassschlüssels ergänzt die oben beschriebene erste Melodiestimme. Es wird zusätzlich zur ersten Population bestehend aus den Melodien, die Akkord-Population erzeugt. Dabei erhält jedes Akkord-Individuum je ein Melodie-Chromosom aus der ersten Endpopulation zufällig zugeordnet. Die Anzahl der Akkorde, die ein Dreiklangs-Individuum bilden, ist aufbauend auf dem Ergebnis der Endpopulation der Melodie zufallsbasiert und wird genauer in dem Abschnitt 5.3 beschrieben.
Da die Akkorde im tieferen Tonraum liegen, sind die möglichen Oktaven nun die Oktave 3 und 4. Das Intervall des gültigen Tonumfangs ist um eine Oktave verkleinert und umfasst nun 24 mögliche Töne.

Abbildung 12: Die 24 erlaubten Akkord-Töne dargestellt auf der Klaviatur.



Ähnlich zu dem Ansatz eines gültigen Notenintervalls im Kapitel Codierung eines Musikstückes, gibt es diesen damit auch in diesem Entwurf. Die artenabhängigen Oktaven bestimmen die Lage und den Umfang der jeweiligen erlaubten Individuums-Töne.
Anders als bisher, erklingen jeweils drei Töne im Zusammenklang, danach erst folgt der

nächste Akkord. Jeweils drei Gene, konkateniert mittels der „JFugue“-Schreibweise für Akkorde, dem +, bilden damit einen Dreiklang. Dieser schallt stets für die Dauer einer halben Note.
Außerdem kann ein Dreiklang immer nur zusammen mit einem Melodieton beginnend erklingen. Beide Arten müssen allerdings nicht dieselbe Tondauer besitzen. Es ist lediglich relevant, dass für beide zur selben Zeit die Töne erklingen.

Beispiel. *G♭3h+D♯4h+D4h F♭4h+E♭3h+E4h E♯3h+D4h+C3h B4h+B♭3h+B♯3h stellt die Codierung eines Akkord-Individuums dar. Das Individuum besteht aus zwölf Genen mit derselben Tondauer. Drei Gene zusammen erzeugen einen Dreiklänge, hier insgesamt vier Stück.*

Um den Lösungsraum besser abschätzen zu können, stellt die Tabelle 3 die möglichen Werte bzgl. der vier Parameter vergleichend für die beiden Arten gegenüber.

Parameter	Melodie-Werte	Akkord-Werte
Notenname	C, D, E, F, G, A, B	C, D, E, F, G, A, B
Vorzeichen	♭, ♯	♭, ♯
Oktave	4, 5, 6	3, 4
Tondauer	w, h, q, i	h

Tabelle 3: Gegenüberstellung der möglichen Werte für die Parameter der Melodie- und Akkord-Individuen.

Die codierten Melodie-Individuen bilden zusammen erklingend mit den in Abhängigkeit erzeugten Akkord-Chromosomen ein Musikstück. Es wird als eine Kombination beider Arten repräsentiet, was in der Notenschrift einem zweistimmigen Stück entspricht, wobei die erste Stimme im Violin- und die zweite im Bassschlüssel geschrieben steht.

5.3 Initialisierung

Nach der Definition der Codierung eines Individuums kann die Anfangspopulation initialisiert werden. Die einzelnen Parameter der Gene werden zufällig aus dem gültigen Wertebereich bestimmt.
Bedeutend bei der Erzeugung der Melodiepopulation ist, dass die fünf $\frac{4}{4}$-Takte der Anfangspopulation vollständig mit konkreten Tondauern, das heißt ohne Pausen gefüllt sind. Dafür werden diese, die mittels des „MusicStrings“ als w, h, q und i codiert sind, zunächst in Zahlenwerte umgewandelt, entsprechend ihrem Anteil eines $\frac{4}{4}$-Taktes. Dadurch können die Berechnungen zur Aufsummierung der Tondauern der einzelnen Gene bis zum gefüllten Takt, was dem double-Wert von 4.0 entspricht, durchgeführt werden. Dabei nimmt die Dauer einen Anteil als Summand des $\frac{4}{4}$-Taktes ein.

Tondauer	Anteil eines $\frac{4}{4}$-Taktes
Ganze	4.0
Halbe	2.0
Viertel	1.0
Achtel	0.5

Tabelle 4: Gegenüberstellung der Tondauern und ihrem Anteil als Summand der Summe 4.0 eines Taktes.

Beispiel. *Eine ganze Note (*4.0*) füllt bereits den Takt, während eine halbe Note (*2.0*) nur summiert mit einer solchen zweiten, zwei folgenden Vierteln (*$1.0+1.0$*), vier Achtel-Noten (*$0.5+0.5+0.5+0.5$*) oder einer Viertel und zwei Achteln (*$0.1+0.5+0.5$*) den Takt vervollständigt.*

Die erste Melodie-Initialisierungsphase erzeugt mit diesem Vorgehen Individuen, deren DNS mindestens aus fünf (nur ganze Tondauern) Genen und maximal aus 40 (nur achtel Tondauern) solchen besteht.
Nach dem Durchlaufen des GA bzgl. der Melodiepopulation, wird die Akkord-Anfangspopulation in Abhängigkeit von dessen finalem Ergebnis initialisiert. Die Populationsgröße bleibt unverändert. Genauso wie bereits bei der ersten Initialisierungsphase ist die Parameterdefinition zufallsbasiert innerhalb des gültigen Bereichs, mit dem Unterschied, dass der vierte Parameter, die Tondauer, immer den Wert h besitzt. Durch den Zufall bedingt, ist es möglich, dass ein Akkord anstatt wie vorgesehen ein Dreiklang, lediglich ein Zusammenklang zweier verschiedener Noten ist oder im schlimmsten Fall sogar nur eine einzelne Note darstellt und damit im musikalischen Sinne keinen Akkord mehr bildet.
Jedes Akkord-Individuum wird mit einem Melodie-Individuum gepaart. Einer zufällig ermittelten Teilmenge der Gene des zugehörigen Melodie-Individuums wird jeweils ein Dreiklang zugeordnet. Die Wahrscheinlichkeit, dass eine Melodienote für den Zusammenklang mit einem Akkord selektiert wird, beträgt 50%. Somit werden zu der gegebenen separaten Melodienote drei Gene eines Akkord-Individuums erzeugt.
Insgesamt entstehen in beiden Initialisierungsphasen so viele Chromosomen, wie der Parameter der Populationsgröße angibt und diese bilden zusammen die Anfangspopulation der jeweiligen Stimme.

5.4 Fitnessfunktion

Der GA dieser Arbeit soll als Ergebnis eine kurze Klaviermusik produzieren, die eine hohe Fitness und Güte aufweist. Wichtig hierbei ist, dass die Fitness objektiv bestimmt wird. Das bedeutet, dass interaktive und damit subjektive Bewertungen, wie es bspw. das Model „GenJam“, beschrieben in dem Kapitel 4.2, macht, nicht erlaubt sind. Außerdem besteht eine weitere Schwierigkeit darin, keine musikalische Vorlage nachahmen

zu wollen, sondern etwas vollständig Neues komponieren zu wollen, wodurch die Fitness nicht orientiert an einer Mustervorlage berechnet werden kann.
Die aktive, erfreute und harmonische Grundstimmung des Stücks wurde bereits in dem Kapitel 5.1 als Ziel beschrieben. Zielorientiert wird damit ein algorithmischer Weg konzipiert, welcher die Frage, welches der erzeugten Musikstücke besser ist, beantworten kann. Einige der Wirkweisen unterschiedlicher musikalischer Eigenschaften des Kapitels 3.2 werden zunächst als Merkmale definiert. Der Zusammenhang aller quantifizierbarer Aspekte verschiedener emotionaler Bedeutungen wird mit einer gewichteten Summe ermittelt. Daraus ergibt sich letztlich die fachliche Güte.
Es wird damit zuerst die Güte der Individuen hinsichtlich einzelner konkreter musikalischer Merkmale berechnet. Die Genotypen werden untersucht nach dem Erfolg bzgl. einer Charakteristik. Da das Stück fröhlich sein soll, ist es sinnvoll ein schnelleres Tempo zu wählen, was mittels kürzerer Tondauern erreicht wird. Ebenso die Intervalle wirken sich stark auf die Stimmung der Musik aus. Zu große Intervalle gilt es schlechter zu bewerten, wegen ihrer irritierenden Wirkung auf den Hörer, was genauso für Dissonanzen gilt. Steigende Melodiebögen sind gleichzusetzten mit einer im positiven Sinne wachsenden Stimmung, musikalische Muster sind einprägsam und Dur-Dreiklänge wirken heiter. Diese im Kapitel Wirkung der Musik ermittelten Zusammenhänge werden folgend konkreter definiert. Das Ergebnis eines jeden Merkmals multipliziert mit einem Gewichtungsfaktor ist ein Summand der gewichteten Summe.
Weiter anzumerken ist, dass die Güteermittlung der Melodie-Individuen vollständig unabhängig von der der Akkorden ist. Das heißt, dass bei der Betrachtung der Merkmale bzgl. der Melodien keinerlei Eigenschaften der Akkorde bekannt sein müssen, um diese zu berechnen. Die Güte der Chromosomen ersterer Stimme wird zudem bereits vor der Initialisierung der Akkord-Anfangspopulation ermittelt.
Bei den Akkord-Individuen wiederum liegt wegen ihrer unterstützenden Funktion für ihre zugehörige Melodie keine vollständig losgelöste Fitnessberechnung vor. Stattdessen wird hier ein Mittelmaß gewählt. Die Hälfte der Merkmale für die Dreiklänge bezieht die Melodie mit ein, die zweite Hälfte allerdings ist dann ebenso ganz unabhängig von der anderen Art. Einerseits soll damit konkret erreicht werden, dass beide Arten im Zusammenklang harmonisch wirken und vor allem zusammengehörig. Andererseits muss die Bassstimme auch für sich alleine wohlgeformt klingen, denn nur so kann sich ein positives Gesamtstück bilden.
Generell gilt damit, dass ein fitnesstechnisch gutes Melodie-Individuum zusammen mit einem solchen guten Dreiklang ein genauso gutes Musikstück ergeben, wie eine Melodie mit einer niedrigen Güte, aber einem Akkord identischer Fitness, wie beim ersten Fall. Da die erste Stimme jedoch einen Großteil des Musikstückes ausmacht, ist es nur sinnvoll genauso hier eine hohe Fitness anzustreben. Die Harmonik der Melodie dient dann als Grundlage dafür harmonische Dreiklänge als Erweiterung auszuarbeiten.
Nun wird detaillierter auf den Berechnungsentwurf der gewichteten Summe eingegangen. Die ermittelten Werte aller Merkmale und auch der finalen Summe liegen innerhalb des Intervalls $[0.0, 1.0]$, wobei die Null das schlechteste mögliche Ergebnis ist.
Die ausgewählten Merkmale der Melodie-Individuen, die als Ziel die angestrebte fröhliche Wirkung erzeugen sollen, sind bereits in den Arbeiten von Michael Towsey und

seinen Partnern [14] aufgegriffen. Zusätzlich ist hier ein weiteres neues entworfen. Es folgt ein Überblick der Merkmale:

(1) *Wiederholtes rhythmisches Muster*
Es wird geprüft, wie viele aufeinanderfolgende Töne gleicher Tondauern ein Individuum verfügt im Verhältnis zur Gesamtanzahl der Intervalle. Das Optimum ist erreicht, wenn alle Gene des Individuums denselben Wert für den Parameter die Tondauer aufweisen. [14, S. 59]

(2) *Geschwindigkeit*
Mit Hilfe des „JFugue“ Standardtempos von 120BPM kann die absolut messbare Notendauer bestimmt werden. Je kürzer die Tondauer ist, desto schneller ist das Stück. Besteht die gesamte Melodie aus Achteln, ist das Optimum dieses Merkmals erreicht.

(3) *Steigende Konturrichtung*
Das Verhältnis zwischen der Anzahl der steigenden Intervalle zur Gesamtzahl aller Intervalle wird bestimmt. Ist jeder Ton i tiefer als sein Folgeton $i + 1$, liegt eine konstant steigende Melodiekurve vor und ergibt die maximale Bewertung. [14, S. 58]

(4) *Höhepunktstärke*
Die Anzahl des Vorkommens des höchsten Tons wird gezählt. Kommt dieser nur einmal vor, wird der melodische Höhepunkt am deutlichsten wahrgenommen, und mit eins bewertet. [14, S. 58]

(5) *Wiederholte Tonhöhe*
Berechnet wird, wie viele Töne derselben Höhe aufeinander folgen in Relation zur Anzahl aller Intervalle. Optimal ist ein Individuum, wenn alle seine Gene dieselbe Tonhöhe besitzen. [14, S. 59]

(6) *Konsonante Intervalle*
Bei diesem Merkmal wird das betrachtete Intervall auf seinen Konsonanzwert überprüft und in das Verhältnis zur Anzahl aller Intervalle gesetzt. Die Bewertung bzgl. der Konsonanz stellt die invertierten Werte der Dissonanzbewertung innerhalb Towseys Arbeit dar und ist in der Tabelle 5 dargestellt. [14, S. 57-58]

Intervall (in Halbtonschritten)	Konsonanzwert
0, 1, 2, 3, 4, 5, 7, 8, 9, 12	1.0
10	0.5
6, 11, $\geq$ 13	0.0

Tabelle 5: Konsonanzwerte der einzelnen Intervalle in Halbtonschritten. In Anlehnung an [14, S. 57-58].

Nachdem die Berechnung aller Melodie-Merkmale beendet ist, fehlt nun die Entscheidung, wie viel Gewicht zur Gesamtsumme ein einzelnes solches beiträgt. Dafür ist es sinnvoll diese gegenübergestellt zu betrachten.
Zu Beginn kann man die sechs Merkmale in zwei Kategorien unterteilen. Die erste Gruppe umfasst die zwei Merkmale (1) *Wiederholtes rhythmisches Muster* und (2) *Geschwindigkeit* und konzentriert sich damit auf den Genparameter die Tondauer. Ihr gemeinsames Optimum ist erreicht, wenn alle Gene des Individuums für die Tondauer den Wert i besitzen. Nichtsdestoweniger können beide Merkmale aber auch unterschiedliche eigenständige Optima besitzen. Somit arbeiten diese nicht zwingend zusammen, aber ebenso nicht gegeneinander, weshalb ihnen sinnvoller Weise dasselbe Gewicht zuzuweisen ist.
Die übrigen vier Merkmale bilden die zweite Kategorie. Diese konzentriert sich auf die Tonhöhe, bestehend aus den drei Genparametern Notenname, Vorzeichen und Oktave. Kommt innerhalb der betrachteten Melodie lediglich einmal der höchste Ton vor, ist das das gemeinsame bestmögliche Ergebnis der Merkmale (3) *Steigende Konturrichtung* und (4) *Höhepunktstärke*. Dieses Optimum widerspricht gleichzeitig aber auch zum Teil dem Maximum des Merkmals (5) *Wiederholte Tonhöhe*. Bei diesem sind die Tonhöhen nämlich alle idealerweise dieselben. Das bedeutet, dass ein Individuum, welches einen Idealwert bzgl. der Eigenschaft (4) hat, nie optimal bzgl. Merkmal (5) sein kann. Gleichermaßen verhält es sich zwischen (3) und (5). Folglich werden die Merkmale (3) und (4) gleich gewichtet, jedoch nur halb so stark wie das Merkmal (5). Zuletzt wird die Berechnung (6) *Konsonante Intervalle* der Kategorie gewichtet. Das Merkmal arbeitet in keiner Weise konträr zu einem Weiteren. Es ist möglich, dass die Intervalle optimal konsonant sind und gleichzeitig das Optimum von (3), (4), aber ebenso von (5) (alle Intervalle bestehen aus Null Halbtonschritten) bilden. Aufgrund der Vielfalt des Merkmals nimmt dieses das meiste Gewicht der gewichteten Summe ein.
Die zweite Kategorie besteht nun aus doppelt so vielen Merkmalen wie die erste. Um trotzdem ein Gleichgewicht zwischen beiden Gruppen herzustellen, wird beiden jeweils die Hälfte des erlaubten Gesamtgewichtes von 100% zugeordnet. Die jeweiligen 50% sind nun pro Kategorie auf die zwei, bzw. die vier Merkmale gemäß der Feststellungen aufzuteilen.
Damit ergibt sich anhand der getroffenen Aussagen die folgende Fitnessfunktion eines Melodie-Individuums $F_m(x_i)$:

$$F_m(x_i) = (1)_{x_i} \cdot 0.25 + (2)_{x_i} \cdot 0.25 + (3)_{x_i} \cdot 0.025 + (4)_{x_i} \cdot 0.025 + (5)_{x_i} \cdot 0.05 + (6)_{x_i} \cdot 0.4$$

Die betrachteten Merkmale der Akkord-Individuen sind vier weitere eigenständig definierte, welche die Musiksequenz optimieren sollen:

(7) *Melodienoten-Vorkommen*
Jeder Akkord erklingt gleichzeitig mit einer zugeordneten Melodienote. Ist eine Dreiklangsnote gleich dieser Melodienote, unabhängig der Oktave, ist das Optimum erreicht. Kommt die Melodienote häufiger vor, erhält der Akkord eine Bewertung von 0.5, bei keiner Übereinstimmung den Wert 0.0

(8) *Melodienoten-Position innerhalb des Akkords*
Betrachtet wird die Position der Note innerhalb des Dreiklangs, die der Melodie-

note entspricht. Kommt die Melodienote gar nicht vor oder liegt an der zweiten oder dritten Stelle von oben betrachtet, so ist das das Pessimum. Eine Bewertung von 1.0 erhält der Dreiklang, wenn die relevante Note genau einmal und gleichzeitig an höchster Position innerhalb des Akkords auftaucht. Ansonsten erfolgt eine Bewertung von 0.5.

(9) *Verschiedene Noten*
Das Merkmal bewertet, ob der Akkord im tatsächlichen Sinne ein Dreiklang ist. Das bedeutet, dass das Optimum erreicht ist, wenn alle drei Noten verschieden sind. Sind zwei Noten oder sogar drei Noten dieselben, ist die Bewertung 0.0.

(10) *Dur-Grad*
Es wird geprüft, ob der Dreiklang ein Dur-Dreiklang ist. Ist das Intervall von der tiefsten zur mittleren Note vier Halbtonschritte groß und das darüber liegende drei solche, liegt ein Dur-Dreiklang vor und wird mit dem Maximalwert bewertet. Stimmt lediglich ein Intervall, bekommt der Dreiklang den Wert 0.5, ansonsten den Wert 0.0.

Diese vier Merkmale müssen nun ebenso einen Gewichtungsfaktor zugeteilt bekommen. Zunächst lassen sich zwei Kategorien einteilen. Jede der beiden Kategorie wird wieder im Gesamten mit 50% gewichtet.
Die erste konzentriert sich auf die Verknüpfung der Melodie mit den Akkorden und beinhaltet die Merkmale (7) und (8). Gemeinsam bewerten beide Merkmale unter anderem das Nichtvorhandensein der Melodienote innerhalb eines Dreiklangs mit 0.0. Der genauere Vergleich zeigt weiter, dass das Merkmal (7)*Melodienoten-Vorkommen* fast schon als eine Teilmenge von (8)*Melodienoten-Position innerhalb des Akkords* aufgefasst werden kann. Beide Eigenschaften weisen viele Übereinstimmungen auf, welche in (8) dann noch genauer spezifiziert werden mittels der Position innerhalb des Akkords. Da es somit schwieriger ist das Optimum von Eigenschaft (8) zu erreichen, wird dieser ein stärkeres Gewicht und zwar $\frac{3}{4}$ des zur Verfügung stehenden 50% zugeteilt.
Die zweite Kategorie umfasst die Merkmale (9)*Verschiedene Noten* und (10)*Dur-Grad* und fokussiert sich lediglich auf den Dreiklang selbst. Wie es auch schon bei der ersten Kategorie der Fall war, konkretisiert die eine Eigenschaft weiter die andere. Und zwar ist beispielsweise das Optimum von (9) nur ein vorausgesetzter Teilaspekt der maximalen Bewertung von (10), denn nur wenn alle drei Noten verschieden sind, können überhaupt erst die zwei geforderten Intervallschritte auftauchen. Das strengere Kriterium (10) erhält deshalb auch hier dasselbe größere Gewicht wie (8).
Die Fitnessfunktion eines Akkord-Individuums $F_a(x_i)$ sieht damit wie folgt aus:

$$F_a(x_i) = (7)_{x_i} \cdot 0.125 + (8)_{x_i} \cdot 0.375 + (9)_{x_i} \cdot 0.125 + (10)_{x_i} \cdot 0.375$$

5.5 Ersetzungsstrategie

Um zu gewährleisten, dass in jeder folgenden Generation das beste Individuum mindestens genauso tauglich, gemäß des Fitnesswertes ist, wie das der Elterngeneration,

wird der Elitismus eingesetzt. Die Ersetzungsstrategie verhindert, dass die besten Chromosomen durch Zufall verloren gehen. Die Anzahl von zwei Eliten wird unverändert zum Teil der neuen Population. Damit weicht die Größe der Elitenzahl um eins von dem in Kapitel 2.3.6.2 genannten üblichen Wert ab. Diese Entscheidung dient lediglich der Einfachheit, da durch die an der empfohlenen Zahl nächste gerade Elitengröße die übergebene Population für die Rekombination weiterhin gerade bleibt nach Abzug der Eliten, wodurch wiederum überhaupt erst Individuen rekombiniert werden können.

5.6 Selektionsentwurf

Die Auswahl der Individuen, welche die neue Generation bilden, erfolgt mittels der Turnier-Selektion. Vor allem wegen ihrer Effizienz in Bezug auf den Rechenaufwand und der nicht vorhandenen Vorarbeit, ist diese Methode gewählt worden.
Die Turniergröße ist mit zwei Individuen so klein wie möglich gewählt. Der Selektionsdruck, also das Maß für die Präferenz guter Chromosomen gegenüber schlechteren, ist damit sehr niedrig. Zum einen bedeutet das, dass zwar die Anzahl der bleibenden schlechteren Individuen in der Population größer ist, als bei einem hohen Selektionsdruck, gleichzeitig jedoch nimmt die Vielfalt der Population nicht ab. [2, S. 31] Doch aufgrund des Einsatzes des Elitismus, wird der mögliche auftretende Nachteil eines zu geringen Selektionsdrucks vor dem eines zu hoch gewählten bevorzugt.
Insgesamt finden so viele Turniere statt, wie die Population groß ist nach der Subtraktion der zwei Eliten. Beträgt die Populationsgröße n, so werden $n - 2$ Wettkämpfe durchgeführt.
Als Ergebnis liefert der Algorithmus eine Liste aller Sieger, die nun zusammen mit den vorher geretteten Eliten die Elternindividuen für die Rekombination und Mutation bilden.

5.7 Rekombinationsentwurf

Die Rekombination findet zu einer 75 prozentigen Wahrscheinlichkeit statt. Das entspricht dem Mittelwert aus den im Kapitel 2.3.4 genannten üblichem Bereich. Zwar gilt der dortige Wert für die binäre Codierung, nichtsdestoweniger wird dieser bei der in der Arbeit definierten Verschlüsselungsform zur Orientierung übernommen.
Der Rekombinationsoperator rekombiniert stets zwei Individuen. Das bedeutet, dass eine gerade Populationsgröße nötig ist.
Die Gene der Melodie-Individuen werden mittels des Ein-Punkt-Crossover gekreuzt. Die Kreuzungsstelle wird bei demjenigen Chromosom ermittelt, welches eine kürzere DNS aufweist. Damit wird verhindert, dass der Index außerhalb dieser liegt und keine Rekombination erfolgt. Die Gene der beiden Kinder entstehen weiter nach dem in Kapitel Ein-Punkt-Crossover beschriebenen Verfahren.
Bei den Akkord-Individuen ist die Wahl der Rekombinationsvariante in Abhängigkeit der DNS-Länge beider Eltern entworfen. Weisen beide dieselbe Anzahl an Genen auf,

unterscheidet sich die Kreuzung hier nicht von der der Melodie-Chromosomen. Hat ein Elternteil jedoch mehr Gene, muss sichergestellt werden, dass je ein Kind eine genauso lange Erbinformation besitzt wie das zugeordnete Elter. Dieser weitere Schritt ist nötig, weil die Kinder anstatt eine neue Melodienote erhalten, zu welcher sie erklingen dürfen, die des zugeordneten Elters übernehmen. Bei der Überschreitung der erlaubten Länge würde damit versucht werden, einen Dreiklang an einer Position zu spielen, an welcher kein Melodieton enthalten ist, was der Codierung widerspricht. Deshalb wird in diesem Fall die Zwei-Punkt-Crossover-Variante eingesetzt.

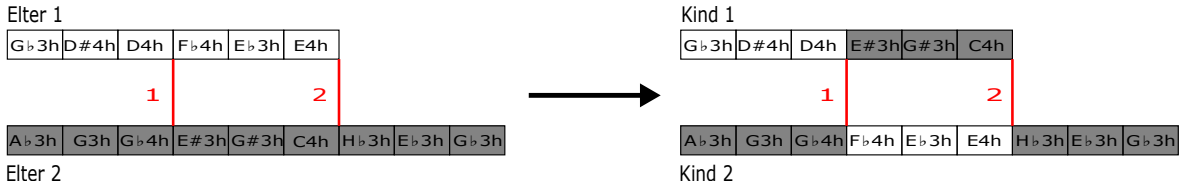


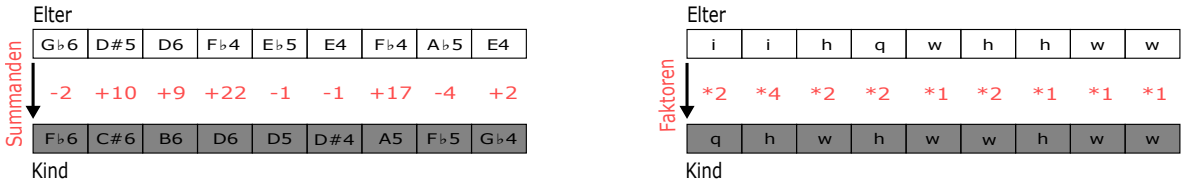
Abbildung 13: Veranschaulichung des entworfenen Zwei-Punkt-Crossovers bei Akkord-Individuen.

Nachdem wie bisher auch die erste Kreuzungsstelle ermittelt wurde, wird der zweite Punkt gleich der Länge des kürzeren DNS-Strings gesetzt. Die Gene, die damit nach dem ersten Rekombinieren die Ursprungslänge des anfangs kürzeren Elternteils überschreiten, werden zurück an das Ende der Erbstruktur des zum zweiten Elter gehörenden Kindes geordnet.
Als Abschluss beider Varianten wird eine Liste gefüllt mit den Kindern zurückgegeben. Diese enthält dieselbe Anzahl neuer Chromosomen, wie die zu Beginn als Parameter übergebene Elterngeneration.

5.8 Mutationsentwurf

Dem Mutationalgorithmus wird zunächst die Liste der aktuellen Population übergeben, was dem Rückgabewert der Rekombination entspricht. Der Mutationsoperator mutiert jedes einzelne Gen eines jeden Individuums zu einer ein prozentigen Wahrscheinlichkeit. Die Mutationswahrscheinlichkeit entspricht dem genannten üblichen Maximum des Kapitels Mutation. Bewusst wird der Wert größer gesetzt und nicht nur auf ein einzelnes zufällig gewähltes Gen des Chromosoms, sondern auf alle seiner Gene, angewandt. Einer homogenen Population soll damit stärker entgegengewirkt werden.
Nach der Auswahl des zu mutierenden Gens findet die Gleichverteilte Mutation statt. Zum einen wird die Tonhöhe mutiert. Dafür wird zufällig eine Mutationsschrittweite ermittelt. Diese muss die Bedingung erfüllen, dass sie als Summand summiert mit dem Wert der Tonhöhe den Bereich der erlaubten 36 Tonhöhen bei einem Melodie-Individuum und die 24 Töne eines Akkord-Chromosoms nicht verlässt. Ist der Zufallswert somit zu groß oder auch zu klein, wird dieser so lange neu gewählt, bis die Einschränkung umsetzbar ist.

Die Tondauer lediglich der Melodie-Individuen wird zusätzlich mutiert. Dafür wird die Tondauer des Gens, anstatt wie gewöhnlich mit einem Summanden summiert, nun mit einem Faktor multipliziert. Der Faktor wird auch so häufig zufällig ermittelt, bis sich dieser multipliziert mit der Tondauer innerhalb des Bereichs von einer Maximaldauer einer ganzen Note und einem Minimalwert einer achtel Note befindet.
Die folgende Abbildung 14 stellt ein Beispiel einer Mutation für ein Melodie-Individuum dar, wobei sowohl die Tonhöhen-Mutation, als auch die der Tondauer gezeigt werden.



(a) Tonhöhen-Mutation.

(b) Tondauer-Mutation.

Abbildung 14: Mutationsbeispiel eines Melodie-Individuums.

Das mutierte Gen bilden die veränderte Tonhöhe und Dauer zusammen, bzw. die mutierte Tonhöhe selbst in der zweiten Musikstimme.
Nachdem nun jedes Gen des Individuums entweder mutiert oder unverändert vorliegt, liegt die neue DNS vor, die das mutierte Chromosom repräsentiert. Mutiert bedeutet an dieser Stelle nicht, dass zwingend ein Gen verändert wurde, sondern lediglich, dass die Entscheidung für oder wider die Mutation mittels der Wahrscheinlichkeit getroffen wurde. Der Algorithmus arbeitet auf jedem Individuum der übergebenen Population und liefert abschließend die mutierte Generation als Liste zurück.

5.9 Das Abspielen der Individuen

Die erzeugten Individuen stellen jeweils getrennt entweder eine kurze Melodie oder eine Akkordfolge und zusammen ein Musikstück dar, welche anzuhören sind.
In dieser Arbeit werden lediglich die besten Individuen bzgl. ihres Fitnesswertes abgespielt. Zum einen erklingt dieses der jeweiligen Anfangspopulation und der Endpopulation separat. Dafür wird jeweils die als *String* vorliegende DNS in einer MIDI-Datei gespeichert. Weiter ist es sinnvoll dann ebenso beide Individuen-Arten zusammen in einer MIDI-Datei zu speichern und zu hören, da das finale Ergebnis des Algorithmus ein zweistimmiges Musikstück ist.
Insgesamt erklingen damit sechs kurze Stücke, wobei vier dieser lediglich als Zwischenergebnisse der separaten Stimmen zu betrachten sind. Die letzten beiden stellen die direkte Gegenüberstellung der Anfangs- und Endpopulation beider Arten zusammen dar. Der Vergleich vom ersten zum letzten generierten Musikstückes ist also zu hören.

6 Implementierung des Genetischen Algorithmus zur Musikkomposition

Das folgende Kapitel zeigt die Implementierung der im Kapitel 5 getroffenen Entwurfsentscheidungen auf. Der Programmcode ist in der Programmiersprache Java geschrieben.

6.1 Programmstruktur

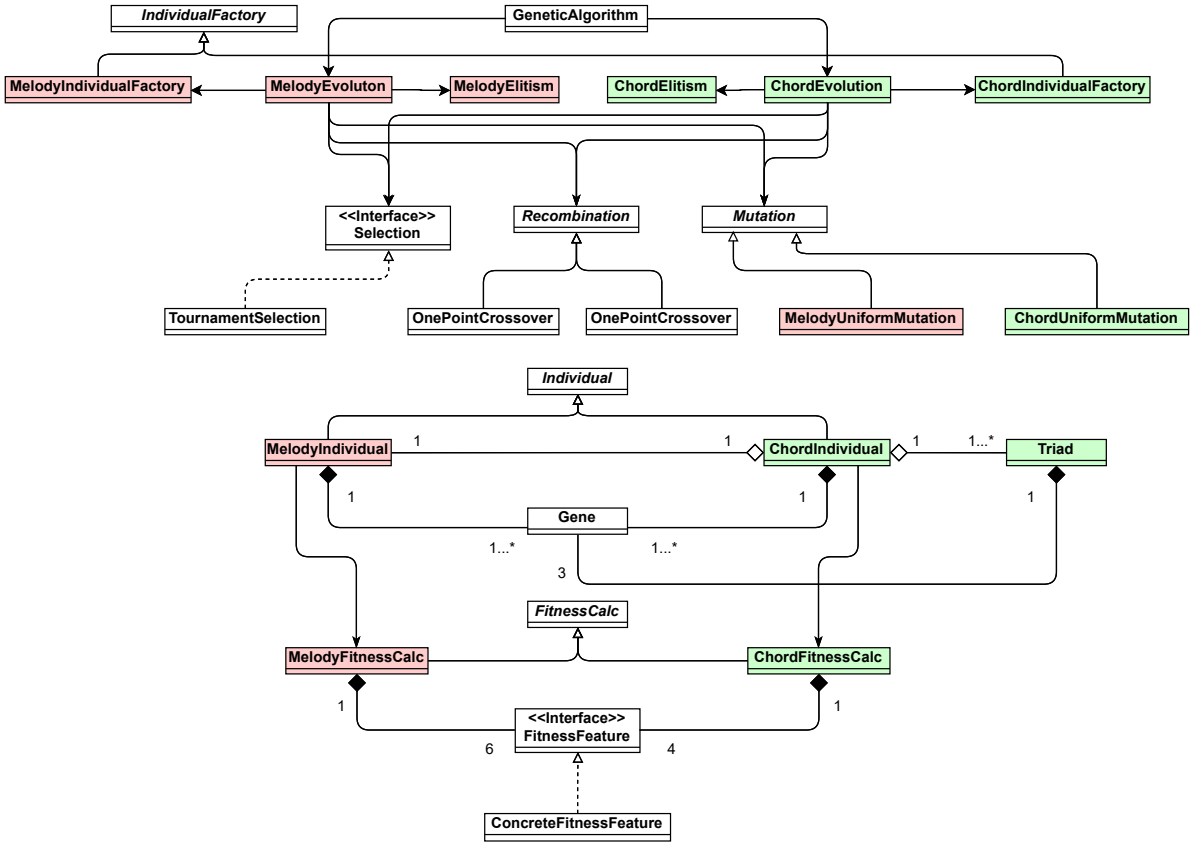


Abbildung 15: Klassendiagramm des implementierten GA.

Abbildung 15 stellt das Klassendiagramm des implementierten GA dar. Die rötlich gefärbten Klassen enthalten den relevanten Programmcode für die Melodie-Individuen, wohingegen die grünen den der Akkord-Individuen codieren. Wichtig anzumerken ist, dass diese beiden Kategorien sich in vielen Aspekten ähneln, wie z. B. der Reihenfolge des Ablaufs des GA. Nichtsdestoweniger ist eine Unterscheidung relevant, weil es sich um zwei verschiedene Arten von Individuen handelt und die Erweiterung um die Akkord-Individuen separat behandelt werden muss. Ist ein Element weiß, bedeutet das, dass die

Klasse, bzw. das Interface, relevanten Code für beide Arten implementiert. Es folgt eine kurze Erklärung der einzelnen Klassendiagrammelemente und deren Funktionalität:

- **GeneticAlgorithm**: Die Klasse stellt den Einstiegspunkt dar. Sie handhabt zuerst den Aufruf des Durchführens des GA für die Melodie-Individuen und danach den der Akkord-Individuen. Außerdem definiert sie die Parameter der Populationsgröße, der Elitengröße und der Anzahl der zu durchlaufenden Generationen.
- **IndividualFactory**: Die abstrakte Klasse gibt das zu implementierende Verhalten für die Subklassen an.
  - **MelodyIndividualFactory/ ChordIndividualFactory**: Es wird jeweils die Anfangspopulation initialisiert.
- **MelodyEvolution/ ChordEvolution**: In den Klassen befinden sich jeweils die Implementierung des GA. Es werden unter anderem die Stationen des Elitismus, der Selektion und die der Operatoren Rekombination und Mutation durchgeführt. Genauer wird hier ebenso die Wahl der konkreten Variante letzterer drei festgelegt.
- **MelodyElitism/ ChordElitism**: Die Klasse implementiert die Ersetzungsstrategie den Elitismus.
- **Selection**: Das zu implementierende Verhalten der Klasse *TournamentSelection* wird innerhalb des Interfaces vorgegeben.

  TournamentSelection: Die Klasse implementiert die Turnier-Selektion.
- **Recombination**: Als abstrakte Klasse implementiert sie den ersten Schritt der Rekombination und dient weiter als Vorlage für die gewählten Rekombinationsvarianten.
  - **OnePointCrossover**: Das Ein-Punkt-Crossover ist hier implementiert.
  - **TwoPointCrossover**: Die Klasse enthält den Code für den Zwei-Punkt-Crossover.
- **Mutation**: Die abstrakte Klasse führt einen Teil der Mutation aus und liefert weiter die Schnittstelle für die gleichverteilte Mutation.
  - **MelodyUniformmutation/ ChordUniformMutation**: Die Klassen implementieren jeweils das konkrete Verhalten der gleichverteilten Mutation.
- **Individual**: Das Strukturelement, die abstrakte Klasse *Individual*, enthält zum einen Attribute und Methoden, die den konkreten Klassen *MelodyIndividual* und *Chordindividual* vererbt werden und zum anderen gibt sie ein Verhalten für diese vor. Wichtige Attribute sind der Fitnesswert und eine Liste aller Gene des Individuums.
  - **MelodyIndividual/ ChordIndividual**: Die Subklassen stellen jeweils die konkreten Individuen dar und codieren ihr spezifisches Verhalten.

    * **ChordIndividual**: Ein Objekt der Klasse *ChordIndividual* aggregiert jeweils ein Objekt der Klasse *MelodyIndividual*. Außerdem besitzt ein *ChordIndividual* ein oder mehrere Objekte der Klasse *Triad*.

- **Triad**: Die Klasse codiert einen Dreiklang, wobei jedes Klassenobjekt aus genau drei Genen besteht.
- **FitnessCalc**: Die abstrakte Klasse enthält einen Teil der Fitnessbestimmung eines Individuums und gibt weiter ein Verhalten zur Gütebestimmung vor.
  - **MelodyFitnessCalc/ ChordFitnessCalc**: Beide Klassen erben jeweils die Implementierungen der Oberklasse und definieren weiter speziell das vorgeschriebene Verhalten. Je ein Objekt der Klasse *MelodyFitnessCalc*, bzw. *ChordFitnessCalc*, besteht aus sechs Objekten, bzw. aus Vieren, die das Interface *FitnessCalc* implementieren.
- **FitnessFeature**: Das Interface dient mit den Schnittstellen-Methoden als Strukturelement.
  - **ConcreteFitnessCalc**: Die Klasse implementiert das konkrete Verhalten des Interfaces und steht stellvertretend für alle in Kapitel 5.4 beschriebenen Merkmale.

6.2 Die Initialisierung der Anfangspopulation

Listing 1: Die abstrakte Klasse *IndividualFactory*.

```
1  public class IndividualFactory{
2      public static final String[] NOTES =
3          new String[] { "C", "D", "E", "F", "G", "A", "B" };
4      public static final String[] DURATIONS =
5          new String[] {"w", "h", "q", "i"};
6
7      public abstract Individual[] createPopulation(int populationSize);
8
9      public abstract ArrayList<Gene> createGenes();
10 }
```

Die Initialisierungsphase findet in den drei *Factory*-Klassen statt. Zunächst definiert die abstrakte Oberklasse *IndividualFactory* die beiden konstanten Listen *NOTES* und *DURATION* (Z. 2-5), welche die gültigen Werte der zwei entsprechenden Parameter eines Gens eines Individuums beinhalten. Weiter stehen hier die Grundfunktionen zum Erzeugen einer Anfangspopulation abstrakt deklariert.
Die genaue Implementierung wird nun in Listing 2 verdeutlicht. Die *createPopulation*-Methode erzeugt so viele Individuen, wie es die übergebene Populationsgröße festlegt, was innerhalb der *for*-Schleife in 16f. in Listing implementiet ist. Im Falle eines

Akkord-Individuums wird zusätzlich definiert, welche Gene des zugehörigen Melodie-Individuums zusammen mit einem Dreiklang erklingen sollen (Z. 14f.).
Die Objekterzeugung eines beliebigen Individuums benötigt als Konstruktorparameter eine Liste aller erzeugten zugehörigen Gene. Dafür ist die Methode *createGenes* zuständig. Die Gene werden entsprechend der Kriterien in Kapitel 5.2 und 5.3 erzeugt. In den Zeilen 24ff. ist ein Ausschnitt der Methodenimplementierung für die Akkorde abgebildet. Die Variable *chordCount* (Z. 27) enthält die Anzahl der Noten der Melodie-Population, die einen Dreiklang erhalten sollen. Dieser muss verdreifacht werden, um pro Akkord-Individuum drei Gene für eine zugehörige Melodienote zu erzeugen.

Listing 2: Ausschnitt der Klasse *ChordIndividualFactory*.

```
1  public class ChordIndividualFactory extends IndividualFactory{
2      private static MelodyIndividual[] melodyPopulation =
3          new MelodyIndividual[GeneticAlgorithm.POPULATION_SIZE];
4
5      public ChordIndividualFactory(MelodyIndividual[] population) {
6          melodyPopulation = population;
7      }
8
9      @Override
10     public ChordIndividual[] createPopulation(int populationSize) {
11         ChordIndividual[] individuals =
12             new ChordIndividual[populationSize];
13         for(int i = 0; i < populationSize; i++) {
14             melodyPopulation[i].setNotesWhichGetAChord(GeneHandler
15                 .defineNotesWhichGetAChord(melodyPopulation[i]));
16             ChordIndividual individual =
17                 new ChordIndividual(createGenes(), melodyPopulation[i]);
18             individuals[i] = individual;
19         }
20         return individuals;
21     }
22
23     @Override
24     public ArrayList<Gene> createGenes() {
25         //...
26
27         for(int i = 0; i < (3 * chordCount); i++) {
28             /* Erzeugung einer randomisierten Akkordnote und Kreierung
29             eines neuen Gen-Objektes */
30         }
31         return genes;
32     }
33 }
```

6.3 Codierung eines Individuums

Nach dem Erklären des Ablaufs der Initialsierungsphase, ist es ebenso sinnvoll auf die Codierung eines Gens und das daraus resultierende Individuum zu schauen.
Listing 3 zeigt einen Ausschnitt der Klasse *Gene*. Die vier Attribute eines Gens werden bei der Objekterzeugung in Form von Konstruktorparametern übergeben und gesetzt. Innerhalb des Konstruktors wird zusätzlich die Methode *createGeneString* aufgerufen (Z. 15), welche aus den Attributen einen „JFugue“-kompatiblen „MusicString“, wie z. B. $A\sharp 4q$ erzeugt.

```
1  public class Gene implements Cloneable{
2      private String name = null;
3      private int octave = 0;
4      private String accidental = null;
5      private String duration = null;
6
7      //...
8
9      public Gene(String name, int octave, String accidental,
10         String duration) {
11         this.name = name;
12         this.octave = octave;
13         this.accidental = accidental;
14         this.duration = duration;
15         createGeneString();
16     }
17
18     //...
19 }
```

Listing 3: Konstruktor und Attribute der Klasse *Gene*.

Nach der Erzeugung aller Gene eines Individuums wird dieses selbst produziert. Die abstrakte Klasse *Individual* implementiert ein Individuum, wie Listing 4 zeigt. Bei der Objekterzeugung wird dem Konstruktor eine Liste aller Gene übergeben. Ähnlich wie eben folgt die Bildung der vollständigen Erbinformation im Sinne „JFugues“ durch den Aufruf der Methode *createIndividualDNAString*.
Zusätzlich besitzt ein jedes Individuum ein Fitness-Attribut und eine *calcFitness*-Methode, innerhalb welcher die Berechnung der Variablen stattfindet.

```
public abstract class Individual implements Cloneable{
    protected ArrayList<Gene> genes;
    protected double fitness = 0.0;
    //...

    public Individual(ArrayList<Gene> genes) {
        this.genes = genes;
        createIndividualDNAString();
```

```
9       }
10
11      protected abstract void createIndividualDNAString();
12
13      public abstract void calcFitness();
14
15      //...
16  }
```

Listing 4: Ausschnitt der abstrakten Klasse *Individual*.

6.4 Durchlaufen des Genetischen Algorithmus

Der konkrete Prozess des GA der Melodie-Individuen, bzw. der Akkord-Individuen, beginnt mit der Initialisierung und der Erzeugung einer Anfangspopulation (Z. 2-5). Danach wird die Selektionsvariante gesetzt und die Operatoren Rekombination und Mutation werden gewählt (Z. 7-9). Mit dem Aufruf der Methode *evolve* in Zeile 11 startet der eigentliche Evolutionsprozess. Die Methode liefert die finale Population nach Beendigung des GA zurück.

```
1   public static MelodyIndividual[] startGeneticAlgorithm() {
2       IndividualFactory individualFactory = new MelodyIndividualFactory();
3       MelodyIndividual[] melodyPopulation =
4           (MelodyIndividual[]) individualFactory
5           .createPopulation(GeneticAlgorithm.POPULATION_SIZE);
6
7       selection = new TournamentSelection();
8       recombination = new OnePointCrossover();
9       mutation = new MelodyUniformMutation();
10
11      melodyPopulation = evolve(melodyPopulation);
12
13      return melodyPopulation;
14  }
```

Listing 5: Ausschnitt der Klasse *MelodyEvolution*.

Weiter geht es mit der Betrachtung des genauen Verhaltens der *evolve*-Methode. Listing 6 zeigt den Aufbau des Evolutionsprozesses eines Melodie-Individuums. Zuerst wird die Güte der übergebenen Anfangspopulation bewertet (Z. 5). In der *while*-Schleife ab Zeile 11 entsteht die neue Generation. Dafür werden zunächst die Eliten gesichert und separat zwischengespeichert (Z. 14-17). Es folgt die Tournament-Selektion, die gewählte Rekombinationvariante und die gleichverteilte Mutation (Zeile 20-35). Abschließend werden in Zeile 28f. beide Bestandteile der neuen Gesamtpopulation, also die Eliten und die erzeugten Kinder zu der neuen Generation vereinigt, sodass erneut die Fitness bestimmt werden kann (Z. 32). Die Generation wird inkrementiert (Z. 34) und der Schleifendurchlauf beginnt von Neuem. Insgesamt beginnt der genannte Prozess wieder erneut, solange

kein Individuum der aktuellen Population eine optimale Fitness von 1.0 aufweist und die Generation kleiner der maximal definierten ist. In dieser Arbeit ist die maximale Generationenzahl auf den Wert 100 gesetzt. Ob die erste Bedingung hierbei erfüllt ist, wird schon vor dem Erreichen der Schleife in Zeile 9 und dann bei der Nichterfüllung als letzter Schritt innerhalb dieser in Zeile 36 für jede entwickelte Generation überprüft.

```
1  public static MelodyIndividual[] evolve(MelodyIndividual[] population){
2          //...
3
4          // Fitnessberechnung der Anfangspopulation
5          evaluate(population);
6
7          nextGeneration = population;
8
9          double bestFitness = getFittest(population).getFitness();
10
11         while(bestFitness != 1.0 &&
12             generation < GeneticAlgorithm.GENERATION) {
13             // Eliten retten
14             elite = MelodyElitism
15                 .saveElite(nextGeneration,
16                 GeneticAlgorithm.ELITE_SIZE);
17             elite = cloneIndividuals(elite);
18
19             // Selektion, Rekombination und Mutation durchfuehren
20             generationPart = (MelodyIndividual[]) selection
21                 .select(nextGeneration, 2);
22             generationPart = recombination
23                 .recombinePopulation(generationPart);
24             generationPart = mutation
25                     .mutatePopulation(generationPart);
26
27             // Zusammenfuegen der Eliten und der Kinder
28             nextGeneration =
29                 mergeGenerationParts(elite, generationPart);
30
31             // Fitnessberechnung der aktuellem Generation
32             evaluate(nextGeneration);
33
34             generation++;
35
36             bestFitness = getFittest(nextGeneration).getFitness();
37         }
38         return nextGeneration;
39 }
```

Listing 6: Die *evolve*-Methode innerhalb der Klasse *Melodyevolution*.

6.5 Implementierung der Fitnessberechnung

Die Berechnung der Fitness ist innerhalb der abstrakten Klasse *FitnessCalc*, bzw. innerhalb dessen Subklassen *MelodyFitnessCalc* und *ChordFitnessCalc* lokalisiert. Für ein jedes Individuum wird die Methode *calcFitness* (Z. 2-5) aufgerufen. Diese koordiniert zuerst die Berechnung der einzelnen Gütemerkmale (Z. 3), gefolgt von der Bestimmung der gewichteten Summe (Z. 4). Zurückgegeben wird der final berechnete *double*-Fitnesswert.

```
1  public abstract class FitnessCalc {
2      public double calcFitness(Individual individual) {
3          calcFeatures(individual);
4          return calcWeightedSum();
5      }
6
7      public abstract void calcFeatures(Individual individual);
8
9      public abstract double calcWeightedSum();
10 }
```

Listing 7: Die abstrakte Klasse *FitnessCalc*.

Die Methode *calcFeatures* ruft die einzelnen *calcFeature*-Methoden für die jeweilige Individuum-Art auf. Eine Eigenschaftsermittlung zeigt Listing 8 mit der vollständigen Implementierung des Fitnessmerkmals *Steigende Konturrichtung* innerhalb der Klasse *RaisingContourDirection*. Diese ist eine der zehn codierten Merkmalsklassen, welche eine Position der Stellvertreterklasse *ConcreteFitnessCalc* innerhalb des Klassendiagramms in dem Kapitel Programmstruktur einnimmt.
Die Merkmalsberechnung startet mit einer *for*-Schleife, welche beim ersten Gen des Chromosoms beginnt und bis hin zum vorletzten Gen dessen die Indexposition erhöht. In jedem Durchlauf wird das aktuelle Gen mit dem folgenden verglichen in Bezug auf dessen Tonhöhe. Für den Vergleich selbst werden zwei Methoden der Hilfsklasse *GeneHandler* verwendet. Erstere liefert einen Genstring getrimmt um die Tondauer zurück (Z. 6-8, Z. 11-13), sodass mittels der zweiten Methode in Zeile 10 und 15 eine Tonhöhe, dargestellt als *Integer*-Wert, ermittelt werden kann. Unter der Bedingung, dass die erste Note tiefer ist als die zweite, wird ein Zähler erhöht (Z. 16-18). Damit speichert die Variable *raisingCount* nach Verlassen der *for*-Schleife die Anzahl der Noten, die höher als dessen direkte Vorgängernote erklingen. Abschließend wird die Variable durch die Anzahl der vorkommenden Intervalle im Individuum dividiert, um ein Ergebnis im Intervall $[0.0, 1.0]$ zurückzuliefern (Z. 20-21).

```
1  public class RaisingContourDirection implements FitnessFeature{
2      @Override
3      public double calcFeature(Individual individual) {
4          int raisingCount = 0;
5          for(int i = 0; i < individual.getGenes().size() - 1; i++) {
```

```
6                 String firstTrimmedString = GeneHandler
7                     .trimGeneStringByDuration(individual.getGenes()
8                     .get(i).getGeneString());
9                 int firstDirectionValue = GeneHandler
10                    .getNoteDirectionValue(firstTrimmedString, false);
11                String secondTrimmedString = GeneHandler
12                    .trimGeneStringByDuration(individual.getGenes()
13                    .get(i + 1).getGeneString());
14                int secondDirectionValue = GeneHandler
15                    .getNoteDirectionValue(secondTrimmedString, false);
16                if(firstDirectionValue < secondDirectionValue) {
17                    raisingCount++;
18                }
19            }
20            return ((double) raisingCount /
21                (individual.getGenes().size() - 1));
22        }
23    }
```

Listing 8: Die Berechnung des Merkmals *Steigende Konturrichtung* in der Klasse *RaisingContourDirection*.

Mit Beendigung der Merkmalsberechnungen beginnt die Ermittlung der gewichteten Summe. Listing 9 bildet die *calcWeightedSum*-Methode für ein Melodie-Individuum ab, welche gemäß der definierten Fitnessfunktion innerhalb des Kapitels Fitnessfunktion implementiert ist.

```
1 @Override
2 public double calcWeightedSum() {
3     return (calcRepeatedRhythmPattern * 0.25 + calcNoteSpeed * 0.25
4         + calcRaisingContourDirection * 0.025
5         + calcClimaxStrength * 0.025 + calcRepeatedPitch * 0.05
6         + calcConsonantInterval * 0.4 );
7 }
```

Listing 9: Methode *calcWeightedSum* innerhalb der Klasse *MelodyFitnessCalc*.

6.6 Sicherung der Eliten

Die Ersetzungsstrategie, der Elitismus, ist innerhalb der Klasse *MelodyElitism*, bzw. für ein Akkord-Indiviuum in *ChordElitism*, implementiert. Zu Beginn wird eine Kopie der übergebenen Population in der Liste *generation* erstellt (Z. 5-10), auf welcher weiter gearbeitet wird. In einer *for*-Schleife wird das beste Individuum der Generation bzgl. seiner Fitness und dessen Position ermittelt (Z. 18-24), sodass dieses der separat angelegten Elitenliste hinzugefügt (Z. 27) und anschließend aus der Kopie gelöscht werden kann (Z. 28). Der Prozess findet so oft statt, wie der zweite Methodenparameter n, die Anzahl der zu sichernden Eliten, groß ist. Zurückgegeben wird abschließend eine Liste

aller Eliten.

```
1  public class MelodyElitism{
2      public static MelodyIndividual[] saveElite(
3          MelodyIndividual[] population, int n) {
4          MelodyIndividual[] elite = new MelodyIndividual[n];
5          ArrayList<MelodyIndividual> generation =
6              new ArrayList<MelodyIndividual>();
7          // Uebergebene Population in separater Liste speichern
8          for(int i = 0; i < population.length; i++) {
9              generation.add(population[i]);
10         }
11         int index = -1;
12         // Fuer die Laenge n der Elitenliste
13         for(int count = 0; count < n; count++) {
14             MelodyIndividual bestIndividual = generation.get(0);
15             int individualPosition = 0;
16             /* Fitnesstechnisch bestes Individuum der Generation und
17             dessen Position ermitteln */
18             for(int i = 0; i < generation.size(); i++) {
19                 if(bestIndividual.getFitness() < generation.get(i)
20                     .getFitness()) {
21                     bestIndividual = generation.get(i);
22                     individualPosition = i;
23                 }
24             }
25             index++;
26             // Bestes Individuum der Eliten-Liste hinzufuegen
27             elite[index] = bestIndividual;
28             generation.remove(individualPosition);
29         }
30         return elite;
31     }
32 }
```

Listing 10: Die Klasse *MelodyElitism*.

6.7 Die Selektion

Der Schritt der Turnier-Selektion findet in der Klasse *TournamentSelection* statt. Die zuständige Methode *select* erhält als Parameter die Population, sowie die Turniergröße k. Für die Größe der Population abzüglich der Elitengröße werden k Individuen zufällig aus der Population als Turnierkandidaten gewählt (Z. 10-16). Die Fitness aller Kandidaten wird verglichen. Das Individuum, welches den besten Fitnesswert aufweist, gewinnt die aktuelle Turnierrunde (Z. 19-23). Es wird in der Liste der neuen selektierten Population gespeichert(Z. 26) und eine neue Turnierrunde beginnt. Als Ergebnis wird die neue Population bestehend aus den Siegern aller Turniere zurückgegeben.

```java
@Override
public MelodyIndividual[] select(MelodyIndividual[] population, int k){
    MelodyIndividual[] newPopulation = new MelodyIndividual[population
        .length - GeneticAlgorithm.ELITE_SIZE];
    Random random = new Random();
    MelodyIndividual[] candidates = new MelodyIndividual[k];
    MelodyIndividual winner = null;
    /* Fuer die Groesse der Population abzueglich der Elitengroesse
    Turniere durchfuehren */
    for(int i = 0; i < newPopulation.length; i++) {
        /* k Individuen der Population zufaellig als Kandidaten fuer
        ein Turnier waehlen */
        for(int c = 0; c < k; c++) {
            candidates[c] = population[random.nextInt(population
                .length)];
        }
        winner = (MelodyIndividual) candidates[0];
        // Den Gewinner des aktuellen Turniers ermitteln
        for(int j = 0; j < candidates.length - 1; j++) {
            if(winner.getFitness() <= candidates[j + 1].getFitness()) {
                winner = (MelodyIndividual) candidates[j + 1];
            }
        }
        try {
            winner = (MelodyIndividual) winner.clone();
            newPopulation[i] = winner;
        }
        catch(CloneNotSupportedException exception) {
            System.out.println("It is not allowed to clone this
            object.");
        }
    }
    return newPopulation;
}
```

Listing 11: *select*-Methode der Klasse *TournamentSelection*, welche die Melodie-Individuen selektiert.

6.8 Die Rekombination

Der Einstiegspunkt der Rekombination befindet sich innerhalb der abstrakten Klasse *Recombination*. Listing 12 zeigt die Methode *recombinePopulation* für die Melodie-Individuen. Innerhalb dieser wird zuerst geprüft, ob der übergebene Parameter, die Melodiepopulation, eine gerade Anzahl an Individuen besitzt (Z. 6-9), da nur dann jeweils zwei Chromosomen rekombiniert werden können. Ist die Bedingung gegeben, entscheidet weiter die Rekombinationswahrscheinlichkeit darüber, ob die Gene der momentan betrachteten Elternindividuen rekombiniert werden (Z. 16-21). Andernfalls werden diese

unverändert Teil der nächsten Generation (Z. 22-25).

```
1  public abstract class Recombination {
2      public static final double RECOMBINATION_RATE = 0.75;
3
4      public MelodyIndividual[] recombinePopulation(MelodyIndividual[]
5          population) {
6          if(population.length % 2 != 0) {
7              throw new IllegalArgumentException("The population must
8                  consist of an even number of individuals.");
9          }
10         MelodyIndividual[] nextGeneration = new MelodyIndividual[
11             population.length];
12         /* Fuer die Population mittels der Rekombinationswahrschein-
13         lichkeit bestimmen, ob je zwei Individuen rekombiniert werden
14         sollen */
15         for(int i = 0; i < population.length - 1; i += 2) {
16             if(Math.random() <= RECOMBINATION_RATE) {
17                 MelodyIndividual[] children = new MelodyIndividual[2];
18                 children = recombine(population[i], population[i + 1]);
19                 nextGeneration[i] = children[0];
20                 nextGeneration[i + 1] = children[1];
21             }
22             else {
23                 nextGeneration[i] = population[i];
24                 nextGeneration[i + 1] = population[i + 1];
25             }
26         }
27         return nextGeneration;
28     }
29
30     //...
31 }
```

Listing 12: *recombinePopulation*-Methode für die Melodie-Individuen innerhalb der abstrakten Klasse *recombination*.

Sollen die beiden Eltern-Individuen rekombiniert werden, so wird der Algorithmus der entsprechenden Rekombinationsvariante mittels des Aufrufs der *recombine*-Methode (Z. 18) gestartet. Das Ein-Punkt-Crossover für die Individuen innerhalb des Violinschlüssels ist im folgenden Listing 13 codiert.

```
1  public class OnePointCrossover extends Recombination {
2      @Override
3      protected MelodyIndividual[] recombine(MelodyIndividual parent1,
4          MelodyIndividual parent2) {
5          // Kreuzungsstelle ermitteln
6          int crossingPoint = getCrossingPoint(parent1, parent2);
7          ArrayList<Gene> genesChild1 = new ArrayList<Gene>();
8          ArrayList<Gene> genesChild2 = new ArrayList<Gene>();
```

```
9          Gene geneChild1 = null;
10         Gene geneChild2 = null;
11         // Gene gemaess des Eins-Punkt-Crossover rekombinieren
12         for(int i = 0; i < crossingPoint; i++) {
13             geneChild1 = buildGene(parent1, geneChild1, i);
14             genesChild1.add(geneChild1);
15             geneChild2 = buildGene(parent2, geneChild2, i);
16             genesChild2.add(geneChild2);
17         }
18         for(int i = crossingPoint; i < parent2.getGenes().size(); i++){
19             geneChild1 = buildGene(parent2, geneChild1, i);
20             genesChild1.add(geneChild1);
21         }
22         for(int i = crossingPoint; i < parent1.getGenes().size(); i++){
23             geneChild2 = buildGene(parent1, geneChild2, i);
24             genesChild2.add(geneChild2);
25         }
26         // Kinder mit den entstandenen DNS erzeugen und zurueckgeben
27         MelodyIndividual child1 = new MelodyIndividual(genesChild1);
28         MelodyIndividual child2 = new MelodyIndividual(genesChild2);
29         MelodyIndividual[] children = new MelodyIndividual[2];
30         children[0] = child1;
31         children[1] = child2;
32         return children;
33     }
34
35     //...
36 }
```

Listing 13: *recombine*-Methode innerhalb der *OnePointCrossover*-Klasse, die das Ein-Punkt-Crossover implementiert.

Zu Beginn wird mit Hilfe des Aufrufs der Methode *getCrossingPoint* in Zeile 6 der Kreuzungspunkt bestimmt. Dieser wird innerhalb des kürzeren DNS-Strings beider Eltern ermittelt, sodass der Wert nicht außerhalb der Länge der Erbinformation eines der beiden Individuen liegt. Nun werden die Gene in gemäß des beschriebenen Verfahrens in dem Kapitel Ein-Punkt-Crossover neu zusammengefügt (Z. 12-25), sodass mittels den entstandenen DNS zwei neue Kinder erzeugt (Z. 29f.) und zurückgegeben werden können.

6.9 Die Mutation

Der Mutationsschritt beginnt in der in Listing 14 abgebildeten Klasse *Mutation*. Die Methode *mutatePopulation*, welche hier für die Melodie-Individuen codiert abgebildet ist, betrachtet jedes einzelne Gen eines jeden Individuums der als Parameter übergebenen Melodiepopulation und entscheidet, ob es mutiert wird oder nicht (Z. 8-22). Die Mutationswahrscheinlichkeit ist als Konstante der Klasse definiert (Z. 2). Nachdem die

Entscheidung und der Ablauf der Mutation für alle Gene eines Chromosoms abgeschlossen ist, wird ein neues Melodie-Individuum mit der gewandelten DNS erzeugt (Z. 24). Abschließend liefert die Methode eine mutierte Population derselben Größe, wie die der anfangs übergebenen Generation, zurück.

```
1  public abstract class Mutation {
2      public static final double MUTATION_RATE = 0.01;
3
4      public MelodyIndividual[] mutatePopulation(MelodyIndividual[]
5          population) {
6          MelodyIndividual[] mutatedPopulation = new MelodyIndividual
7              [population.length];
8          for(int i = 0; i < population.length; i++) {
9              ArrayList<Gene> mutatedGenes = new ArrayList<Gene>();
10             // Jedes einzelne Gen des Individuums wird betrachtet
11             for(int j = 0; j < population[i].getGenes().size(); j++) {
12                 // Das Gen mit der Wahrscheinlickeit von 1% mutieren
13                 if(Math.random() <= MUTATION_RATE) {
14                     Gene mutantGene = mutateGene(population[i]
15                         .getGenes().get(j));
16                     mutatedGenes.add(j, mutantGene);
17                 }
18                 else {
19                     mutatedGenes.add(j, population[i].getGenes()
20                         .get(j));
21                 }
22             }
23             // Neues Individuum mit den mutierten Genen erzeugen
24             mutatedPopulation[i] = new MelodyIndividual(mutatedGenes);
25         }
26         return mutatedPopulation;
27     }
28
29     //...
30 }
```

Listing 14: *mutatePopulation*-Methode für die Melodie-Individuen in der abstrakten Klasse *Mutation*.

Soll ein Gen mutiert werden, so findet die gleichverteilte Mutation selbst innerhalb der Klasse *MelodyUniformMutation*, bzw. für die Akkord-Individuen in ihrer dafür entsprechenden Klasse statt. Listing 15 stellt die *mutateGene*-Methode innerhalb ersterer dar. Diese ruft beginnend die beiden Methoden *mutateNote* und *mutateDuration* auf (Z. 4-5), innerhalb welcher das explizite Verhalten der Veränderung implementiert ist. Wichtig anzumerken ist, dass *mutateDuration* nicht für Akkord-Individuen implementiert ist, da diese immer Tondauern einer halben Note besitzen. Das gewandelte Gen liegt danach als *String* vor und wird so weiterverarbeitet (Z. 6f.), dass letztlich ein neues Gen-Objekt erzeugt werden kann (Z. 9-11). Dieses wird in Zeile 12 zurückgegeben.

```
@Override
protected Gene mutateGene(Gene gene) {
    // Notenhoehe und Tondauer mutieren und konkatenieren
    String mutantCode = mutateNote(gene) + mutateDuration(gene
        .getDuration());
    String[] mutantElements = GeneHandler
        .getGeneElementsSeparately(mutantCode);
    // Mutiertes Gen-Objekt erzeugen und zurueckliefern
    Gene mutant = new Gene(mutantElements[0],
        Integer.parseInt(mutantElements[1]), mutantElements[2],
        mutantElements[3]);
    return mutant;
}
```

Listing 15: *mutateGene*-Methode in der Klasse *MelodyUniformMutation*.

Weiter interessant ist die exakte Implementierung der *mutateGene*-Methode. Diese ist für beide Individuum-Arten ziemlich ähnlich und unterscheidet sich lediglich in den definierten Grenzen des gültigen Bereichs für die Tondauern, da beide einen verschiedenen aufweisen.
Listing 16 bildet die Methode für ein Melodie-Individuum ab. Im ersten Schritt wird die Tondauer des übergebenen Gens entfernt (Z. 3f.) und der entsprechende Tonhöhen-*Integer*-Wert wird mittels der Hilfsklasse *GeneHandler* ermittelt (Z. 5f.). Nun wird in Zeile 8 der Summand randomisiert erzeugt. Das Zufallsintervall entspricht der doppelten Größe des Tonhöhenumfangs der jeweiligen Individuum-Art, gleichverteilt in den negativen und positiven Wertebereich. Hier wird damit eine Zufallszahl aus dem Intervall $[-36, 36]$ bestimmt. Die Summe des Summanden und dem Tonhöhenwert wird berechnet (Z. 9) und auf die Gültigkeit bzgl. der 36 Tonhöhen überprüft, sodass bei einem Wert außerhalb des Bereichs die beiden letzten Schritte solange wiederholt werden, bis der definierte Rahmen eingehalten ist (Z. 12-15). Abschließend wird der Zahlenwert der Tonhöhe in einen *String* umgewandelt (Z. 16f.) und als Ergebnis zurückgegeben (Z. 18).

```
private String mutateNote(Gene gene) {
    // Die Tondauer von dem GeneString entfernen
    String noteWithoutDuration = GeneHandler
        .trimGeneStringByDuration(gene.getGeneString());
    int noteValue = GeneHandler
        .getNoteDirectionValue(noteWithoutDuration, false);
    // Summanden zufaellig ermitteln
    int summand = (int)(Math.random() * 72) + (-36);
    int mutantValue = noteValue + summand;
    /* Zufaelligen Summanden solange suchen, bis die mutierte
    Tonhoeheinnerhalb des gueltigen definierten Bereichs liegt */
    while(mutantValue > 35 || mutantValue < 0) {
        summand = (int)(Math.random() * 72) + (-36);
        mutantValue = noteValue + summand;
    }
    String mutantCode = GeneHandler
```

```
17          .getNoteByDirectionValue(mutantValue, false);
18      return mutantCode;
19  }
```

Listing 16: *mutateNote*-Methode in der Klasse *MelodyUniformMutation*.

6.10 Individuen als Musikstück hören

Nachdem der GA im Gesamten beendet ist, ist es interessant zu hören, wie die erzeugten Individuen klingen. Dafür ist die Hilfsklasse *MusicHandler* implementiert. Diese codiert eine Reihe an Methoden, um die Chromosomen mittels „JFugue“ als Musikstück abspeichern und spielen zu können.
Das Listing 18 zeigt mit der Methode *createGeneticAlgorithmResultMusic*, wie dieser Prozess, für die Melodie-Individuen gemeinsam mit den Dreiklängen erklingend, aussieht. Zu Beginn wird ein *Pattern*-Objekt erstellt, welchem dann in der ersten Stimme die DNS als *String* des fittesten Individuums der Melodie-Endpopulation und in der zweiten dieselbe der Akkorde hinzugefügt wird (Z. 4-6). Innerhalb des *try*-Blocks wird dann versucht das Muster in der MIDI-Datei „Endpopulation.mid“ zu speichern (Z. 11f.), sodass diese Datei letztlich abgespielt werden kann (Z. 16f.).

```
1  public static void createGeneticAlgorithmResultMusic() {
2      //...
3
4      Pattern lastPattern = new Pattern();
5      lastPattern.add("V0 " + lastBestMelodyIndividualDna);
6      lastPattern.add("V1 " + lastBestChordIndividualDna);
7
8      try {
9          //...
10
11         // Pattern in MIDI-Files speichern
12         MidiFileManager.savePatternToMidi(lastPattern,
13             new File("Endpopulation.mid"));
14
15         // MIDIs abspielen
16         playMidiDirectly(new File("Endpopulation.mid"),
17             "Endpopulation.mid");
18     }
19     catch (Exception e) {
20         System.out.println("Error when saving to MIDI file");
21     }
22 }
```

Listing 17: Methode *createGeneticAlgorithmResultMusic* der Klasse *MusicHandler*.

In der *playMidiDirectly*-Methode derselben Klasse ist der Code für das eigentliche Abspielen des Musikstückes implementiert. Anfangs wird mittels der als Parameter über-

gebenen MIDI-Datei ein Sequenz-Objekt erzeugt (Z. 4). Das Objekt *player* spielt dann diese Sequenz (Z. 6).

```
1 public static void playMidiDirectly(File file, String fileName) throws
2     IOException, InvalidMidiDataException{
3     System.out.println("Now playing: " + fileName);
4     Sequence sequence = MidiSystem.getSequence(file);
5     Player player = new Player();
6     player.play(sequence);
7 }
```

Listing 18: Methode *playMidiDirectly* in der Klasse *MusicHandler*.

Einige der hier erzeugten Anfangs- und Endpopulationen sind im Anhang beispielhaft in der Notenschrift dargestellt.

7 Evaluation

Das Kapitel der Evaluation dient dazu die getroffenen Entscheidungen bzgl. der Parameterwerte zu bewerten und zu begründen. Dafür werden schrittweise die unterschiedlichen Ergebnisse basierend auf den vorgenommenen Einstellungen betrachtet, sodass abschließend ein Gesamturteil entsteht.
Final wird evaluiert, inwiefern die Generierung eines Musikstückes erfolgreich ist.

7.1 Parametereinstellungen

Der implementierte GA weist eine Reihe veränderbarer Parameter auf. Alle Parametereinstellungen verfolgen das in Kapitel 5.1 beschriebene Gesamtziel.
Als erstes ist eine Populationsgröße für die Erzeugung der Anfangspopulation zu wählen. Diese multipliziert mit dem Parameter der Generationenanzahl, ergibt die Gesamtanzahl an erzeugten Individuen während eines Durchlaufs des Algorithmus. Des Weiteren ist es möglich, die drei Parameter Elitengröße, Rekombinations- und Mutationswahrscheinlichkeit innerhalb der verschiedenen Phasen der Implementierung zu verändern.

7.1.1 Populations- und Generationengröße

Der erste Parameter, die Populationsgröße, wird bereits in der Initialisierungsphase der Anfangspopulation benötigt. Genauso muss beginnend festgelegt werden, wie viele Generationen der GA durchlaufen soll.
Um den Einfluss der beiden Parameter *Population* und *Generation* auf den Verlauf des Algorithmus bzgl. der Fitnesswerte überprüfen zu können, werden zunächst einige Voraussetzungen dargelegt. Zum einen werden nur diese beiden Werte variiert, alle anderen möglichen Einstellungen, wie z. B. die Rekombinationswahrscheinlichkeit, bleiben unverändert entsprechend den Werten des Kapitels Entwurf eines genetischen Algorithmus zur Musikkomposition. Zum anderen werden die Werte für die bessere Vergleichbarkeit lediglich so verändert, dass immer dieselbe Gesamtzahl an Individuen erzeugt wird.
Anfangs werden die Werte der Populationsgröße auf 1.000 Individuen und die der Generationenzahl auf 500 fix gesetzt. Damit werden insgesamt 500.000 Individuen pro Art erzeugt. Vergleichend dazu werden die Zahlen im nächsten Schritt dem jeweils anderen Parameter zugeordnet.
Die untenstehenden Liniendiagramme 16 und 17 zeigen die Auswirkungen des Genannten in direkter Gegenüberstellung. Die Werte stellen hierbei jeweils das arithmetische Mittel über zehn Durchläufe des GA hinweg dar.
Vergleichend festzustellen ist, dass sich die Güterwerte der Melodie-Individuen relativ unabhängig von den getroffenen Einstellungen verhalten. Ob nun die Gesamtpopulation von 500.000 Individuen durch eine größere Populationsgröße oder eine solche Generationenanzahl erreicht wird, hat keinen bedeutsamen Einfluss auf die erste Stimme. Der maximale Fitnesswert, durchschnittlich betrachtet, beginnt bei einem Wert von etwa 0.5

und steigt bis hin zu 0.95. Die Werte der besten Melodie-Individuen der jeweiligen Generation sind etwas höher, was hauptsächlich bei kleinen Generationenzahlen bemerkbar ist.
Die Fitnesswerte der Akkord-Individuen stellen einen deutlicheren Unterschied abhängig der Parametergrößen dar. Bei einer höheren Populationszahl und einer dafür niedrigeren Generationenanzahl weisen die fittesten Chromosomen bereits zu Beginn eine nennenswertere Güte auf. Die besten Individuen haben statt eines Anfangswertes von 0.43 und eines Endwertes von 0.77 nun Gütewerte um 0.47 und 0.85. Während sich dieser Aspekt bei der betrachteten durchschnittlichen Akkord-Fitness hier noch nicht bemerkbar macht, ist spätestens ab etwa der Generation 50 auch hier eine deutliche Optimierung festzustellen. Anstatt eines Endwertes von etwa 0.76, haben die Chromosomen hier nun eine durchschnittliche Fitness um 0.83 herum.

Abbildung 16: Fitnessverlauf bei einer Populationsgröße von 500 und einer Generationenzahl von 1.000.

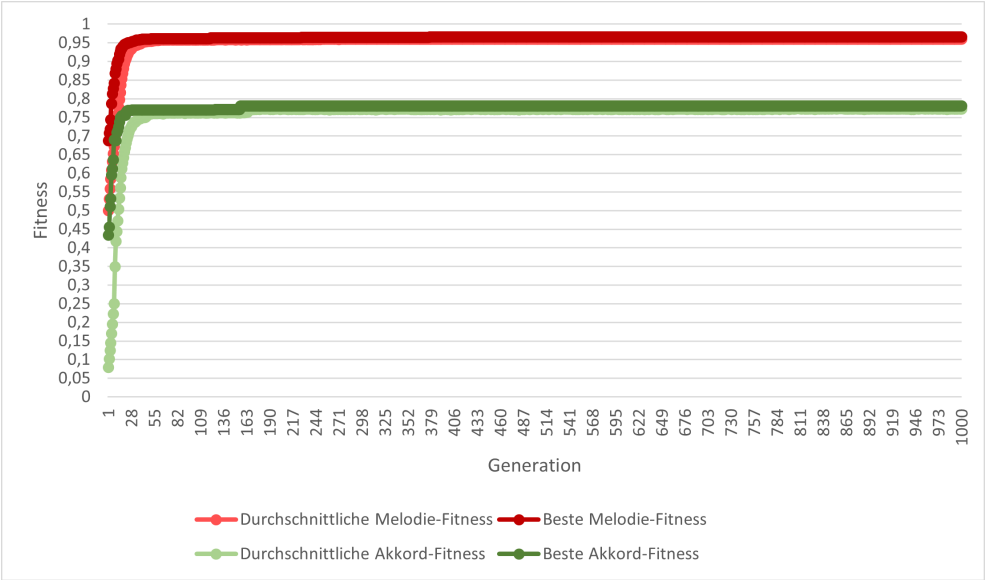
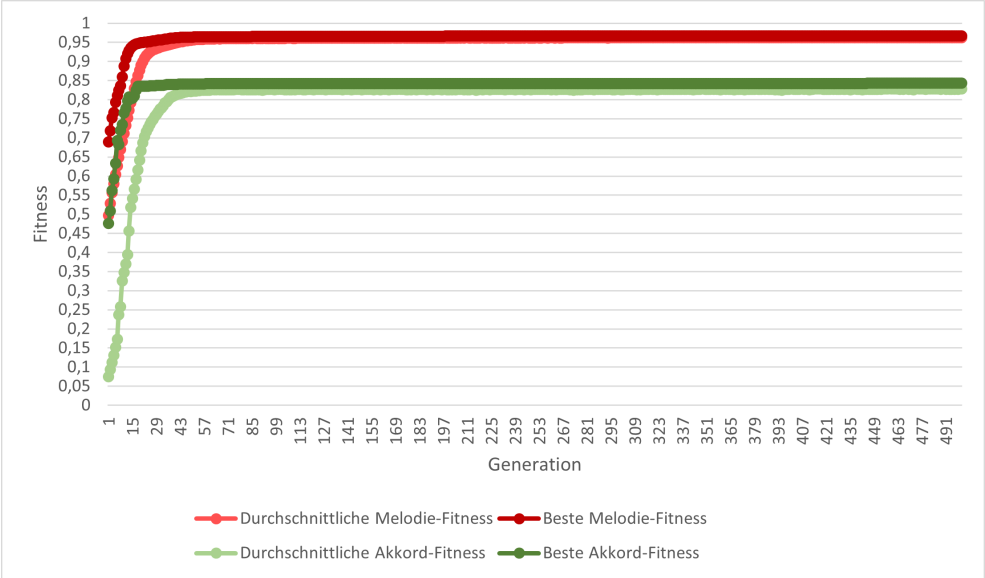


Abbildung 17: Fitnessverlauf bei einer Populationsgröße von 1.000 und einer Generationenzahl von 500.



Im Gesamten ergibt sich, dass die Einstellung einer kleineren Generationenzahl und dafür einer größeren Population lediglich einen positiven Effekt auf die besten Akkord-Individuen ausübt und in keiner Weise die Werte negativ verändert. Eine größere Variabilität direkt zu Beginn des Algorithmus wirkt sich damit positiv auf die Güte aus. Außerdem ist auffällig, dass der GA bei beiden Arten bereits nach wenigen Generationen keine verbesserten Ergebnisse mehr liefert und dass die durchschnittliche Fitness dann ähnlich oder gleich der besten Güte ist. Dies lässt auf eine mangelnde Variabilität in der Population schließen.
Um nun die obigen Mangel in die richtige Richtung hin zu korrigieren, wird in der Grafik 18 die bisher systematische Anzahl an erzeugten Individuen pro Art von 500.000 nun in die Faktoren der Populationsgröße von 10.000 und der Generationenzahl von 50 aufgeteilt. Die Verzehnfachung des ersten Parameters führt direkt zu Beginn eine größere Variabilität ein. Gleichzeitig fallen mit der Betrachtung weniger Generationen die Ergebnisse ohne eine bedeutende Verbesserung weg.

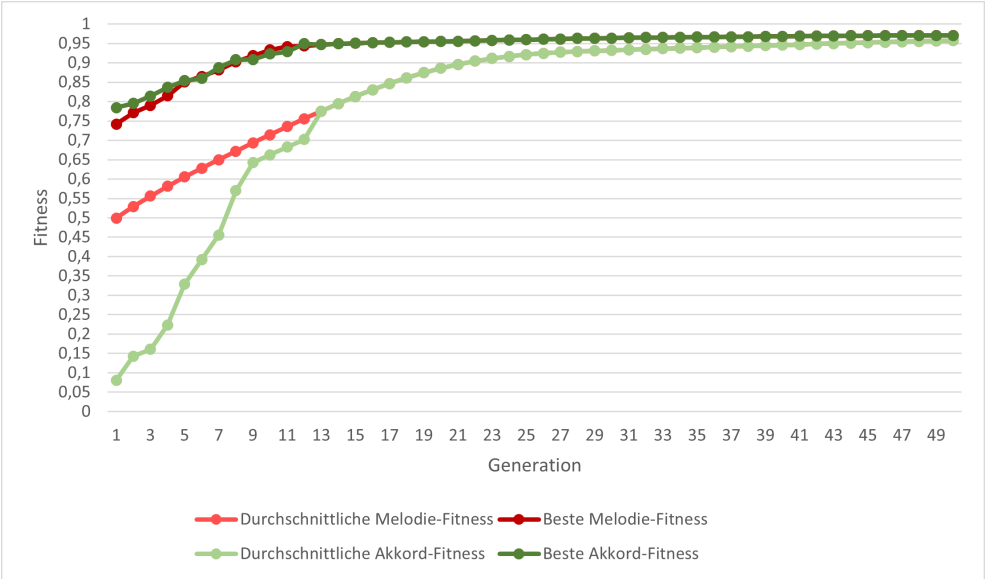


Abbildung 18: Fitnessverlauf bei einer Populationsgröße von 10.000 und einer Generationenzahl von 50.

Des Weiteren interessant ist der Vergleich der Fitness beider Arten miteinander. Deutlich ist, dass die durchschnittliche Güte der Akkorde wesentlich schlechtere Werte liefert als die der Melodie-Chromosomen. Gleichzeitig erzielt die zweite Stimme jedoch stellenweise auch bessere beste Werte, bezogen auf Abbildung 18. Der Abstand zwischen den durchschnittlichen und besten Werten ist damit bei den Akkorden wesentlich größer. Dies ist mit dem Vergleich der Bewertung der einzelnen Fitnessmerkmale beider Stimmen zu erklären. Zwar liegen die Ergebnisse der Merkmale beider Arten innerhalb des Intervalls $[0.0, 1.0]$, doch die Melodie-Eigenschaften können im Gegensatz zu denen der Akkorde zu jedem Berechnungszeitpunkt beliebige Werte annehmen. Die Akkord-Merkmale werden gemäß des Kapitels 5.4 nur mit einem von maximal drei verschiedenen Werten bewertet. Die möglichen Zahlen sind hierbei 0.0, 0.5 und 1.0. Im Falle des Merkmals *Verschiedene Noten* entfällt sogar die mögliche Bewertung mit 0.5. Zwischenwerte von z. B. 0.9 sind hier noch nicht möglich, was hingegen bei einer Melodie bereits funktioniert. Als Folge weisen die Akkorde stärkere Extrema entweder in die bessere oder schlechtere Güterichtung auf. Zu Beginn besteht die Akkordpopulation somit aus überwiegend eher schlechter bewerteten Individuen. Unter ihnen müssen sich gleichzeitig aber auch einige sehr gute Dreiklänge bewegen, was der hohe beste Fitnesswert bereits zu Beginn bei Generation 1 sowie die Tatsache der schnellen Verbesserung innerhalb weniger Generationen der durchschnittlichen Fitness aussagen.

7.1.2 Rekombinationswahrscheinlichkeit

Die Rekombinationswahrscheinlichkeit ist ein weiterer Faktor mit einem starken Einfluss auf die Fitness der Individuen. Auf den Ergebnissen des vorherigen Kapitels 7.1.1 ba-

sierend, soll nun die getroffene Entscheidung der verwendeteten Wahrscheinlichkeit von 0.75 für eine Rekombination beurteilt werden, welche das Mittel des in der Literatur empfohlenen Bereichs darstellt. [23, S. 129-130]
Orientiert an der Literaturrichtlinie wird das Verhalten des implementierten GA für Wahrscheinlichkeiten innerhalb des Intervalls $[0.5, 1.0]$ in 0.1 Schritten und zusätzlich für die verwendete Wahrscheinlichkeit von 0.75 überprüft, womit zum einen der empfohlene Bereich betrachtet wird, zum anderen Werte, die sich außerhalb dieses befinden. Es wird jeweils der beste Gütewert der Individuen-Art der Generation über zehn Durchläufe hinweg in Abhängigkeit von der Rekombinationswahrscheinlichkeit berechnet. Relevant ist die Fitness der letzten Generation 50. Die folgende Tabelle 6 stellt die Ergebnisse der Melodie- und Akkord-Chromosomen gegenübergestellt dar. Alle Werte sind auf vier Dezimalstellen gerundet.

Rekombinationswahrscheinlichkeit	Beste Melodie-Fitness	Beste Akkord-Fitness
0.5	0.9702	0.9702
0.6	0.9711	0.9711
0.7	0.9715	0.9715
0.75	0.9709	0.9709
0.8	0.9710	0.9710
0.9	0.9708	0.9708
1.0	0.9707	0.9707

Tabelle 6: Beste Fitnesswerte der Generation 50 in Abhängigkeit der Rekombinationswahrscheinlichkeit.

Auffällig ist zu Beginn, dass die Fitnesswerte beider Individuen-Arten dieselben sind für alle sieben überprüften Wahrscheinlichkeiten. Damit sind die Ergebnisse der Melodie und der Akkorde im Endeffekt gleichwertig, unabhängig von den Gütewerten der Individuen der einzelnen Art in der entsprechenden Startpopulation.
Des Weiteren ist die Wirkweise der Wahrscheinlichkeit auf die Güte erkennbar. Obwohl die Veränderungen der Fitnesswerte lediglich ab dem Tausendstellbereich stattfinden, ist diese zu beachten. Die Werte steigen vom Anfang bis einschließlich zur Wahrscheinlichkeit von 0.7 an, wonach diese erneut stets schrittweise fallen, wird der Wert von 0.75 außen vor gelassen. Laut der Daten erreicht ein Individuum damit den maximalen Fitnesswert von 0.9715, wenn der Rekombinationsoperator ein solches mit einer 70 prozentigen Wahrscheinlichkeit rekombiniert.
Die Wahl der Mitte aus dem literaturempfohlenen Bereich von 75% ist weiter nicht der optimale Wert. Es ist bemerkbar, dass dieser sogar eine schlechtere Fitness liefert als der direkt folgende Wert 0.8. Allerdings bewegt sich hier der Unterschied lediglich im Zehntausendstelbereich und wird deshalb nicht weiter beachtet. Gleichzeitig ist die getroffene Entscheidung aber auch nicht die schlechteste, betrachtet man bspw. die Ergebnisse des Wertes 0.5.
Zusammenfassend ist somit festzuhalten, dass die 75 prozentige Rekombinationswahr-

scheinlichkeit ein guter Orientierungspunkt ist, allerdings besser durch einen leicht niedrigeren Prozentsatz wie dem von 70% ersetzt werden sollte, um bessere Fitnesswerte zu erzielen.

7.1.3 Mutationswahrscheinlichkeit

Der zweite Operator, die Mutation, wird nun evaluiert. Der implementierte Algorithmus arbeitet mit einer Mutationswahrscheinlichkeit von 1%, was der oberen Grenze der in der Literatur empfohlenen Wahrscheinlichkeit entspricht. [13, S. 11] Die anderen Parameterwerte bleiben wie gehabt unverändert.
Um zu überprüfen, wie gut die getroffene Wahl gegenüber niedrigeren der im empfohlenen Intervall liegenden Wahrscheinlichkeiten ist, ist die untenstehende Tabelle erstellt worden. Zusätzlich werden die Ergebnisse der halbierten, bzw. verdoppelten, unteren, bzw. oberen beschriebenen Grenzen untersucht. Nach demselben Prinzip wie im vorherigen Kapitel 7.1.2 sind die besten Fitnesswerte hier berechnet worden. Es wurde lediglich anstelle der Rekombinationswahrscheinlichkeit die der Mutation variiert.

Mutationswahrscheinlichkeit	Beste Melodie-Fitness	Beste Akkord-Fitness
0.0005	0.9705	0.9705
0.001	0.9712	0.9712
0.0025	0.9706	0.9706
0.005	0.9701	0.9701
0.0075	0.9706	0.9706
0.01	0.9706	0.9706
0.02	0.9705	0.9705

Tabelle 7: Fitnesswerte der Generation 50 in Abhängigkeit der Mutationswahrscheinlichkeit.

Genauso wie bereits in der Tabelle 6 sind auch hier die Fitnesswerte beider Arten dieselben und die Unterschiede der verschiedenen Mutationswahrscheinlichkeiten bewegen sich ebenso maximal im Tausendstelbereich.
Ein explizites Schema nach dem die Gütewerte zustande kommen, ist nicht zu erkennen. Auffällig ist aber, dass die beiden Werte, welche das literaturempfohlene Intervall nicht mehr einschließt, dieselbe zweitniedrigste Fitnesszahl von 0.9705 erzielen. 0.9701 ist die schlechteste ermittelte Fitness bei der mittleren Wahrscheinlichkeit von 0.005. Das beste Ergebnis erreicht hingegen die 0.1 prozentige Mutation. Gemäß der Daten ist damit nicht die gewählte obere Intervallgrenze von 1%, sondern die unterste von 0.1% die erfolgversprechendste Mutationswahrscheinlichkeit.

7.1.4 Elitengröße

Die Elitengröße ist bewusst, wie in Kapitel 5.5 beschrieben ist, gewählt. Nichtsdestoweniger ist es interessant, das Verhalten des GA in Bezug auf die Fitness der besten Individuen beider Arten der letzten Generation 50 zu untersuchen.
In der Tabelle 8 sind dafür die entsprechenden Güteergebnisse für ein Elite-Chromosom weniger und eins mehr als die gewählte Anzahl von 2 gegenübergestellt. Zu beachten ist hierbei, dass die gesetzte Populationsgröße von 10.000 Individuen pro Art entsprechend angepasst werden muss. Die Summe der Eliten addiert mit der Populationsgröße muss stets eine gerade Zahl sein, was der Entwurf des GA voraussetzt. Zusätzlich muss diese, der Vergleichbarkeit halber, gleich dem Wert 10.002 entsprechen, sodass die Parametergrößen gemäß des Abschnittes 7.1.1 weiter eingehalten werden können. Konkret ist die Populationsgröße damit 10.001 groß, wenn nur es nur ein Elite-Individuem gibt, 9.999 groß wiederum bei drei Eliten.

Elitengröße	Beste Melodie-Fitness	Beste Akkord-Fitness
1	0.9711	0.9711
2	0.9706	0.9706
3	0.9704	0.9704

Tabelle 8: Fitnesswerte der Generation 50 in Abhängigkeit der Elitengröße.

Als erstes sind erneut die Tatsache derselben Fitnesswerte beider Arten, sowie die minimalen Änderungen anzumerken. Weiter interessant ist, dass die Gütewerte mit steigender Elitengröße sinken. Den Werten nach ist damit die übliche empfohlene Elitengröße von 1 am besten geeignet.

7.2 Bewertung aller getroffener Entwurfsentscheidungen

Nachdem nun die fünf Parameter, zwei in Kombination und drei unabhängig voneinander, überprüft sind, ist es abschließend sinnvoll diese gemeinsam im Algorithmus arbeitend zu beurteilen. Weiter soll entschieden werden, ob die Entwurfsentscheidungen bzgl. des implementierten GA bereits optimal sind oder sich verbessern lassen.
Konkret stellt die Grafik 18 bereits die Ergebnisse der Fitnesswerte des bestehenden Algorithmus dar. Die beste Kombination der Populationsgröße und Generationenzahl ist hier vereint implementiert. Die anderen Parameterwerte sind an dieser Stelle noch die entsprechenden Werte der Unterkapitel von 5. Diese erwiesen sich nach den vorherigen Überprüfungen als nicht optimal.
Liefern nun die einzelnen besten Parameterwerte ebenso gemeinsam eine bessere Fitness der Individuen? Um dies zu überprüfen, sind die finalen Werte der Güte mit den jeweiligen optimalen Parameterwerten gemäß der Kapitel 7.1.2, 7.1.3 und 7.1.4 erneut gemittelt über zehn Durchläufe berechnet und grafisch dargestellt. Zu beachten ist an

dieser Stelle, dass die Populationsgröße um ein Individuum, bedingt durch die reduzierte Elitenzahl, vergrößert ist, womit sich der in Abschnitt 7.1.1 erfolgversprechendste ermittelte Parameterwert verändert. Allerdings sollte diese Veränderung gemäß der dortigen Feststellungen lediglich einen positiven Effekt auf den Algorithmus ausüben, sodass dieser Aspekt nicht weiter relevant zu beachten ist.

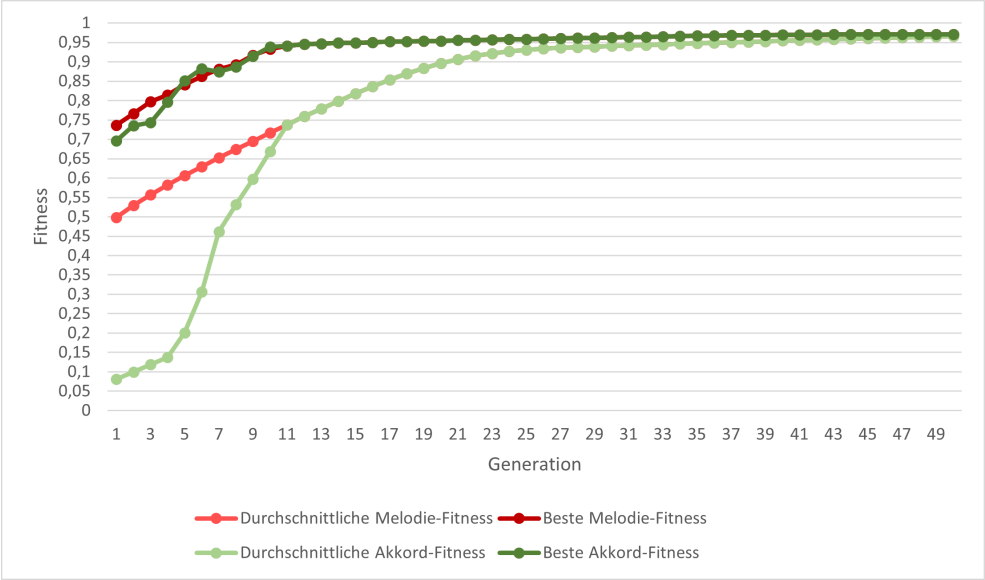


Abbildung 19: Fitnessverlauf bei einer Populationsgröße von 10001, einer Generationenzahl von 50, der Rekombinationswahrscheinlichkeit von 0.7, der Mutationswahrscheinlichkeit von 0.001 und der Elitengröße von 1.

Im Vergleich beider Liniendiagramme miteinander sind die Fitnesswerte der Melodie-Individuen unverändert. Sowohl die durchschnittlichen als auch die besten Werte haben sich nicht verbessert, allerdings positiv anzumerken ebenso nicht verschlechtert. Die Einstellungen bzgl. der Parameter sind in Bezug auf die erste Stimme somit nicht gewinnbringend.
Die Fitnesswerte der Akkord-Individuen hingegen haben sich leicht verändert, allerdings vorweggenommen, nicht unbedingt in die geplante Richtung. Direkt zu Beginn bei der ersten Generation ist zu erkennen, dass die beste Güte der zweiten Stimme jetzt schlechter ist, als zuvor, mit einem Wert von nur noch 0.7 anstatt etwa 0.79. Die Dreiklänge sind nun niedriger bewertet als die besten Melodie-Individuen, was vorher der umgekehrte Fall war. Im Verlauf der Generationen wird diese anfangs schlechtere Güte wieder wett gemacht. Die Ergebnisse der Generation 50 der besten Akkorde beider Grafiken sind identisch. Die durchschnittliche Fitness betrachtet ist ebenso der Anfangswert derselbe. Positiv ist dann wiederum die minimal bessere Güte dieser in der letzten Generation. Hier entspricht diese dem Wert der besten Akkord-Chromosomen, an Stelle leicht unter diesen zu liegen, wie zuvor.
Generell entsprechen die Ergebnisse mit den neuen Parametern nicht den Erwartungen.

Anstatt als Minimum die besten Gütewerte der beiden Arten zu optimieren, was die verschiedenen Parameter im einzelnen bereits erreicht haben, sind diese unverändert. Nur die durchschnittliche Akkord-Fitness weist jetzt geringfügig bessere Gütewerte auf. Das bedeutet, dass in der letzten Generation nun mehr gute Dreiklänge als zuvor vorhanden sind. Weiter wurde aber mit der Reduzierung sowohl der Rekombinations- als auch Mutationswahrscheinlichkeit die Akkord-Population der ersten Generation verschlechtert. Die hier nicht außer Acht zu lassende schlechtere Fitness der zweiten Stimme lässt auf anfangs zu wenig gewinnbringende Rekombinationen, welche wenige, aber erfolglose Genveränderungen durch die Mutation durchliefen, schließen. Die erniedrigte Elitenanzahl übt hingegen auf diese Tatsache keinen Einfluss aus. Außerdem ist anhand der Daten generell kein negativer Einfluss der veränderten Elitengröße festzustellen, denn der Nachteil des Elitismus, eine Dominanz der besten Individuen in einer der beiden Generationen, ist nicht erkennbar, was durch die Erniedrigung anstelle der Erhöhung der Anzahl durchaus zu erwarten war.
Final festzuhalten ist somit, dass der implementierte GA mittels der anfänglichen Parameterwerte besser arbeitet als mit der Kombination aller separat ermittelter Parameterwerte. Die verschiedenen Parameter wirken damit zusammen und können nicht wie vorgenommen vollständig im Alleingang untersucht werden.

7.3 Bewertung der erzeugten Musikstücke

Neben den Parametereinstellungen ist außerdem die Evaluierung des eigentlichen Ziels, dem Erzeugen eines Musikstückes, interessant. Es soll bewertet werden, inwiefern der GA erfolgreich eine kurze Musik, gemäß der in Kapitel 5.4 definierten Merkmale, aus einer zufällig generierten Anfangspopulation erzeugt.
Dafür werden folgend das Musikstück bestehend aus dem besten Melodie- und Akkord-Individuen der Anfangspopulation mit denselben der Endpopulation, also der Individuen der Generation 50, verglichen. Die Codierung der vier betrachteten Individuen sieht zunächst wie folgt aus:

Anfangsmelodie-Codierung. *G♯6w C♯6w G♯5w C5w E5w*

Anfangsakkord-Codierung. *E♭3h+E♭4h+E4h F♯3h+D4h+F4h*

Endmelodie-Codierung. *F5q F5q F5q F5q F5q F5q F5q F5q F5q F5q C♯6q E♭6q*

Endakkord-Codierung. *B♭3h+D4h+F4h*

Die Abbildungen 20 und 21 stellen die codierten Ergebnisse bestehend aus je zwei Individuen in der Notenschrift dar.

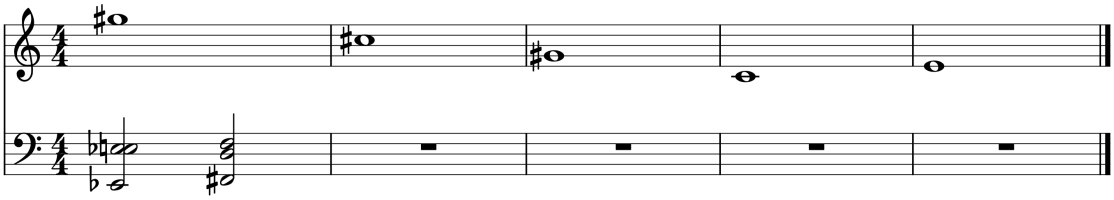


Abbildung 20: Musikstück aus dem besten Individuen je Art der Anfangspopulation.

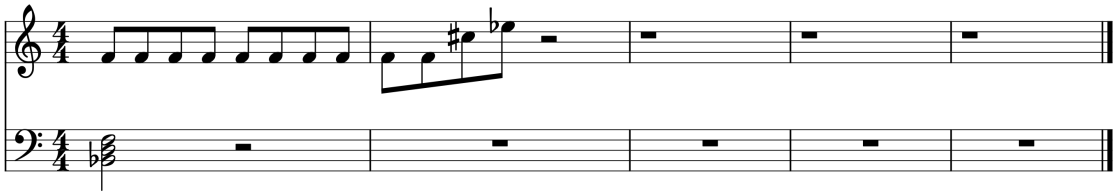


Abbildung 21: Musikstück aus dem besten Individuen je Art der Endpopulation.

Die explizite Bedeutung dieser codierten und dargestellten Noten wird genau in den Tabellen 9 und 10 verdeutlicht. Anhand der Noten werden nämlich die Merkmale der jeweiligen Individuen-Art berechnet, sodass eine Gegenüberstellung der Ergebnisse der Anfangs- zur Endpopulation möglich ist. Die Fitnessmerkmale sind hier gleich der Nummer aus Abschnitt 5.4 benannt und die Ergebnisse sind auf maximal vier Nachkommastellen gerundet.

Fitnessmerkmal	Erreichter Anfangsfitnesswert	Erreichter Endfitnesswert
(1)	1.0	1.0
(2)	0.125	0.5
(3)	0.25	0.1818
(4)	1.0	1.0
(5)	0.0	0.8182
(6)	1.0	1.0

Tabelle 9: Ergebnisse der Fintessmerkmale des besten Melodie-Individuums der Anfangs- und Endpopulation.

Fitnessmerkmal	Erreichter Anfangsfitnesswert	Erreichter Endfitnesswert
(7)	0.0	1.0
(8)	0.0	1.0
(9)	1.0	1.0
(10)	0.25	1.0

Tabelle 10: Ergebnisse der Fintessmerkmale des besten Akkord-Individuums der Anfangs- und Endpopulation.

Bei der Betrachtung der Werte ist zunächst deutlich, dass innerhalb beider Arten die Anfangswerte, bis auf eine Ausnahme, stets kleiner oder gleich der Endwerte sind. Damit optimiert der Algorithmus die zufälligen Anfangsstücke im Laufe der Generationen. Die einzige Ausnahme, bei der keine Verbesserung stattfindet, liegt zwischen den besten Melodie-Individuen bei dem Merkmal (3)*Steigende Konturrichtung* vor. Der Ursprungswert von 0.25 sinkt auf 0.1818. Allerdings ist diese Tatsache eine logische Schlussfolgerung aus der Verbesserung des Merkmals (5) *Wiederholte Tonhöhe*, wie es in Kapitel 5.4 beschrieben steht.
Um nun eine Aussage über die finalen Fitnesswerte der vier betrachteten Individuen treffen zu können, ist die gewichtete Summe je Individuum berechnet. Die Berechnung der Anfangs- und Endgüte der Melodien sieht wie folgt aus:

$$\begin{aligned} F_m(x_0) &= 1.0 \cdot 0.25 + 0.125 \cdot 0.25 + 0.25 \cdot 0.025 + 1.0 \cdot 0.025 + 0.0 \cdot 0.05 + 1.0 \cdot 0.4 \\ &= 0.7125 \end{aligned}$$

$$\begin{aligned} F_m(x_{50}) &= 1.0 \cdot 0.25 + 0.5 \cdot 0.25 + 0.1818 \cdot 0.025 + 1.0 \cdot 0.025 + 0.8182 \cdot 0.05 + 1.0 \cdot 0.4 \\ &= 0.8455 \end{aligned}$$

Eine Verbesserung der Melodie-Fitness nach dem Durchlaufen des GA ist erkennbar. Weiter kann das Ergebnis des Endchromosoms zusätzlich mit der durchschnittlichen besten Fitness aus der Grafik 18 verglichen werden. Die erzeugte Melodie weist zwar bereits einen ordentlich hohen Gütewert mit 0.8455 auf, liegt damit jedoch noch unter der dortigen Zahl von 0.95. Diese liegt somit noch unter dem ermittelten besten Durchschnittswert der finalen Melodien. Weiter werden ebenso die Fitnesswerte der Akkorde berechnet:

$$\begin{aligned} F_a(x_0) &= 0.0 \cdot 0.125 + 0.0 \cdot 0.375 + 1.0 \cdot 0.125 + 0.5 \cdot 0.375 \\ &= 0.3125 \end{aligned}$$

$$\begin{aligned} F_a(x_{50}) &= 1.0 \cdot 0.125 + 1.0 \cdot 0.375 + 1.0 \cdot 0.125 + 1.0 \cdot 0.375 \\ &= 1.0 \end{aligned}$$

Die Akkord-Fitness verbessert sich signifikant. Die Diskrepanz zwischen der Güte der zu Beginn besten zweiten Stimme und dieser finalen ist wesentlich höher als es bei den Melodien der Fall ist. Erneut ist der Vergleich zur Abbildung 18 herangezogen, sodass diese Feststellung damit zusätzlich unterstützt wird. Im Gegensatz zur Endmelodie erreicht das Akkord-Chromosom der fünfzigsten Generation sogar das Optimum mit einem Fitnesswert von 1.0. Zusammenfassend ist bei beiden Arten eine deutliche Verbesserung der finalen Fitness ausgehend von der Anfangsgüte festzustellen. Das bedeutet, dass der implementierte GA gemäß der getroffenen Definitionen eines guten Musikstückes zielführend funktioniert.

8 Fazit und Ausblick

Diese Bachelorarbeit hat sich ausführlich mit der Erstellung eines GA zur Komposition eines Musikstückes beschäftigt.
Das Kapitel 4 hat aufgezeigt, welche bisherigen Ansätze, aber auch Herausforderungen diesbezüglich existieren. Neben den beschriebenen Codierungsansätzen, ist der Unterschied zwischen einer menschlichen subjektiven und regelbasierten objektiven Fitnessbestimmung deutlich geworden. Weiter wäre es interessant zu untersuchen, ob und wenn ja, welche anderen Möglichkeiten existieren, eine Fitnessbewertung für Musikstücke oder auch andere künstlerische Bereiche vorzunehmen.
Basierend auf diesen Ansätzen konnte ein eigener GA entworfen und implementiert werden. Die Codierung verwendet einige bereits bekannte Aspekt, wie die in Form eines Strings. Bereichert wird die bisherige mögliche Herangehensweise mittels der Verwendung „JFugues“. Damit ist die erfolgreiche Überbrückung von der reinen Musik zur Musikprogrammierung mit wenig Aufwand gelungen. Allerdings ist in dieser Bachelorarbeit bei weitem noch nicht die gesamte Bandbreite der Programmierbibliothek ausgenutzt worden, was in Zukunft angegangen werden sollte. Komplexere Musik bestehend aus z. B. mehreren verschiedenen Instrumenten, unterschiedlichen Tempi oder anderweitigen musikalischen Feinheiten, kann damit codiert werden.
Gleichzeitig ist die zu überwindende Schwierigkeit der algorithmischen Fitnessbewertung eines Musikstückes deutlich geworden. Die Relevanz der objektiven Beurteilung hat sich herauskristallisiert. Genau dieser Aspekt hat einen großen Fokus bei der Konzeptentwicklung erhalten. Mit der Definition verschiedener Regeln, die dann zusammen in einer gewichteten Summer berechnet werden, ist ein Lösungsansatz für dieses Problem entwickelt worden. Die Anzahl der betrachteten Merkmale beider Individuen-Arten kann jedoch noch ausgearbeitet werden, sodass ein größeres Repertoire verschiedenster Merkmale existiert, wodurch noch konkreter definiert werden kann, was ein gutes Musikstück ausmacht. Zusätzlich wäre es hier sinnvoll auch einen engeren Bezug zwischen den beiden Arten, der Melodie und den Akkorden, herzustellen. Ein Weg die Gesamtfitness beider Arten zusammen zu bestimmen wäre ebenso sinnvoll. Damit könnte die zweite Stimme nicht nur als Erweiterung wie bisher, sondern gleichwertig mit der Melodiestimme behandelt werden. Eine bessere Harmonisierung beider Stimmen des Musikstückes könnte hieraus folgen.
Inwiefern der GA nun gute Ergebnisse im Sinne des Kapitels 5.1 liefert, ist anschließend evaluiert worden. Die Gegenüberstellung der besten Individuen einer beliebigen Anfangs- und Endpopulation hat deutlich gezeigt, dass das Ziel mit dem vorliegenden GA erreicht wurde. Im Sinne der definierten Regeln findet eine Optimierung des Ausgangsproblems statt, sodass erfolgreich ein kurzes Musikstück entsteht, welches beim Hörer fröhliche Empfindungen auslöst.
Allerdings ist genauso nicht außer Acht zu lassen, dass die bisherigen Parameterwahlen noch zu verbessern sind. Nachdem experimentell die besten Einstellungen bzgl. der einzelnen unterschiedlichen Parameter unabhängig voneinander getroffen worden sind, lieferte die Vereinigung aller dieser besten Werte unerwartet keine besseren Individuen,

sogar schlechtere. Folglich ist das Zusammenspiel aller Parameter relevant. Das Finden der besten Kombination dieser aller wäre eine zukunftsorientierte Möglichkeit, den implementierten GA weiter zu verbessern.
Abschließend ist die Komposition eines Musikstückes mit Hilfe eines GA als interessantes und vor allem breit zu erkundendes Anwendungsgebiet anzusehen. Die bisherigen Ansätze und Ideen laden dazu ein, das Gebiet weiter zu erforschen. Doch bereits jetzt ist klar, dass die richtige Vereinigung der drei unterschiedlichen Themengebiete, die der Biologie, der Kunst und der Informatik etwas ästhetisch Wirksames hervorbringen kann und damit einen enormen Reiz besitzt.

Literatur

[1] John A. Biles. Genjam: A genetic algorithm for generating jazz solos. In *Proceedings oft the International Computer Music Conference (ICMC 1994)*, pages 131–137, San Francisco, USA, 1994. International Computer Music Association.

[2] Harald Feltl. Ein Genetischer Algorithmus für das Generalized Assignment Problem. Diplomarbeit, Technische Universität Wien, Wien, 2003.

[3] Lisl Hammaleser. *Übungsprogramm Musiklehre compact: Definitionen - Aufgaben - Tests*. Schott Music GmbH & Co. KG, Mainz, 1982. URL: https://www.ebook.de/de/product/1327900/lisl_hammaleser_uebungsprogramm_musiklehre.html.

[4] Jochen Heistermann. *Genetische Algorithmen: Theorie und Praxis evolutionärer Optimierung*, volume 9. Teubner-Texte zur Informatik, Leipzig, 1994. URL: https://www.ebook.de/de/product/21053686/jochen_heistermann_genetische_algorithmen.html.

[5] Christopher Hempel. *Neue Allgemeine Musiklehre: Mit Fragen und Aufgaben zur Selbstkontrolle*, volume 6. Schott Music GmbH & Co. KG, Mainz, 2008. URL: https://www.ebook.de/de/product/1334137/christoph_hempel_neue_allgemeine_musiklehre.html.

[6] John H. Holland. Genetic algorithms. *Scientific American*, 267(1):66–73, 1992. URL: http://www.jstor.org/stable/24939139.

[7] Mario Kaulmann. Anwendung interaktiver evolutionärer Algorithmen zur Erzeugung von Schlagzeugrhythmen. Masterarbeit, Technische Hochschule Brandenburg, Brandenburg, 2019.

[8] Roman Klinger. Komposition von Musik mit Methoden der Computational Intelligence. Diplomarbeit, Universität Dortmund, Dortmund, 2006.

[9] David Koelle. *The CompleteGuide to JFugue. Programming Music in JavaTM*, volume 1. 2008. URL: http://www.jfugue.org/4/jfbmrkklprpp/TheCompleteGuideToJFugue-v1.pdf.

[10] Wolfram-Manfred Lippe. *Soft-Computing mit Neuronalen Netzen, Fuzzy-Logic und Evolutionären Algorithmen*. Springer-Verlag, Berlin Heidelberg, 2006. URL: https://www.ebook.de/de/product/25411585/wolfram_manfred_lippe_soft_computing.html, doi:10.1007/3-540-29274-8.

[11] Dragan Matić. A genetic algorithm for composing music. *Yugoslav Journal of Operations Research*, 20(1):157–177, 2010.

[12] Jasmin Reuter. Wahrnehmung und Wirkung von Musik im Film. Anforderungen im genrespezifischen Kontext. Diplomarbeit, Hochschule der Medien, Stuttgart, 2007.

[13] Bianca Selzam. Genetische Algorithmen. Projektgruppe 431 – Metaheuristiken, Universität Dortmund, Dortmund, 2004. URL: https://ls11-www.cs.tu-dortmund.de/lehre/SoSe03/PG431/Ausarbeitungen/GA_Selzam.pdf.

[14] Michael Towsey und Andrew R. Brown und Susan Wright und Joachim Diederich. Towards melodic extension using genetic algorithms. *Educational Technology & Society*, 4:54–65, 2001.

[15] Maik Buttelmann und Boris Lohmann. Optimierung mit genetischen algorithmen und eine anwendung zur modellreduktion (optimization with genetic algorithms and an application for model reduction). *at - Automatisierungstechnik*, 52(4):151 – 163, 2004. doi:https://doi.org/10.1524/auto.52.4.151.29416.

[16] Eberhard Schöneburg und Frank Heinzmann und Sven Fedderson. *Genetische Algorithmen und Evolutionsstragtegien: Eine Einführung in Theorie und Praxis der simulierten Evolution*, volume 1. Addison-Wesley, Bonn, 1994.

[17] Ingrid Gerdes und Frank Klawonn und Rudolf Kruse. *Evolutionäre Algorithmen. Genetische Algorithmen-Strategien und Optimierungsverfahren-Beispielanwendungen*, volume 1. Vieweg & Sohn Verlag, Wiesbaden, 2004. URL: https://www.ebook.de/de/product/1315011/ingrid_gerdes_frank_klawonn_rudolf_kruse_klaus_klawonn_evolutionaere_algorithmen.html.

[18] George Papadopoulos und Geraint Wiggins. A genetic algorithm for the generation of jazz melodies. *Proceedings of the Finnish Conference on Artificial Intelligence*, 1998.

[19] Andrea Becker und Inga Bokerlmann und Hans-Peter Krull und Mirko Schäfer. *Natura. Biologie für Gymnasien. Oberstufe*. Ernst Klett Verlag, Stuttgart, 2012. URL: https://www.ebook.de/de/product/18955481/natura_biologie_fuer_gymnasien_schuelerbuch_mit_cd_rom_11_12_schuljahr.html.

[20] Manuel Alfonseca und Manuel Cebrian und Alfonso Ortega. A simple genetic algorithm for music generation by means of algorithmic information theory. In *2007 IEEE Congress on Evolutionary Computation*, pages 3035–3042, 2007. doi:10.1109/CEC.2007.4424858.

[21] Wiebke Alf und Prof. Dr. Regine Allgayer-Kaufmann und Dr. Frank P. Bär und u. a. *Der Brockhaus Musik. Komponisten, Interpreten, Sachbegriffe*, volume 3. F. A. Brockhaus GmbH, Mannheim, 2006.

[22] Qendresa Mehmeti und Vladimir Amrein und Marius Kulms und Nick Macin und Christian Loenser und Dmitri Bogonos und Manuel Waidhas. *Genetische Algorithmen*. Wirtschaftsmathematisches Projekt zur Numerik im WS 2013/14, Technische Universität Dortmund, Dortmund, 2014.

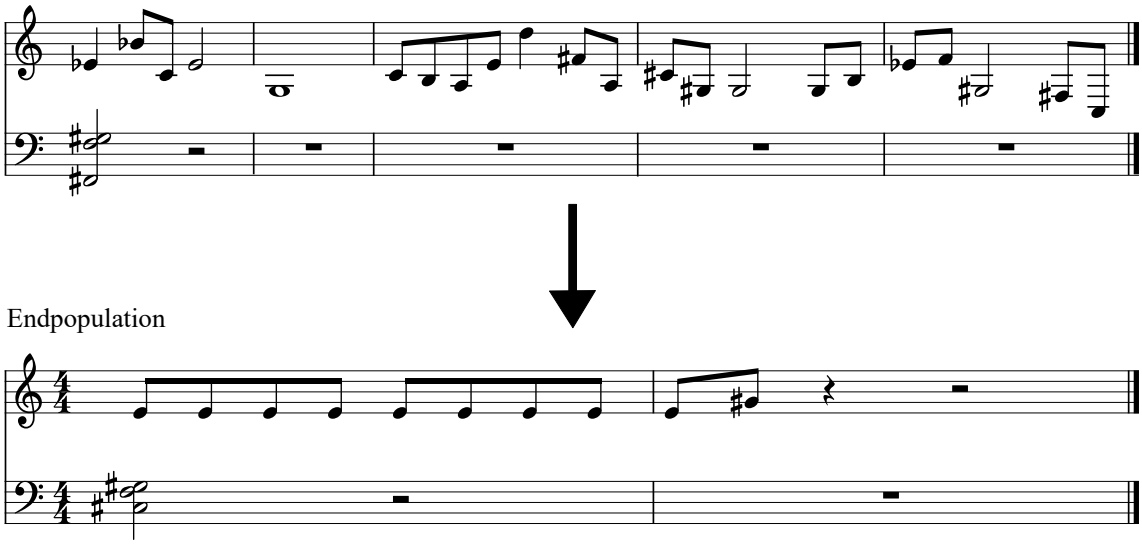
[23] Karsten Weickert. *Evolutionäre Algorithmen*, volume 3. Springer Vieweg, Wiesbaden, 2015. URL: https://www.ebook.de/de/product/24198849/karsten_weicker_evolutionaere_algorithmen.html, doi:10.1007/978-3-658-09958-9.

[24] Wikipedia. JFugue, besucht am: 12.12.2020. URL: https://en.wikipedia.org/wiki/JFugue.

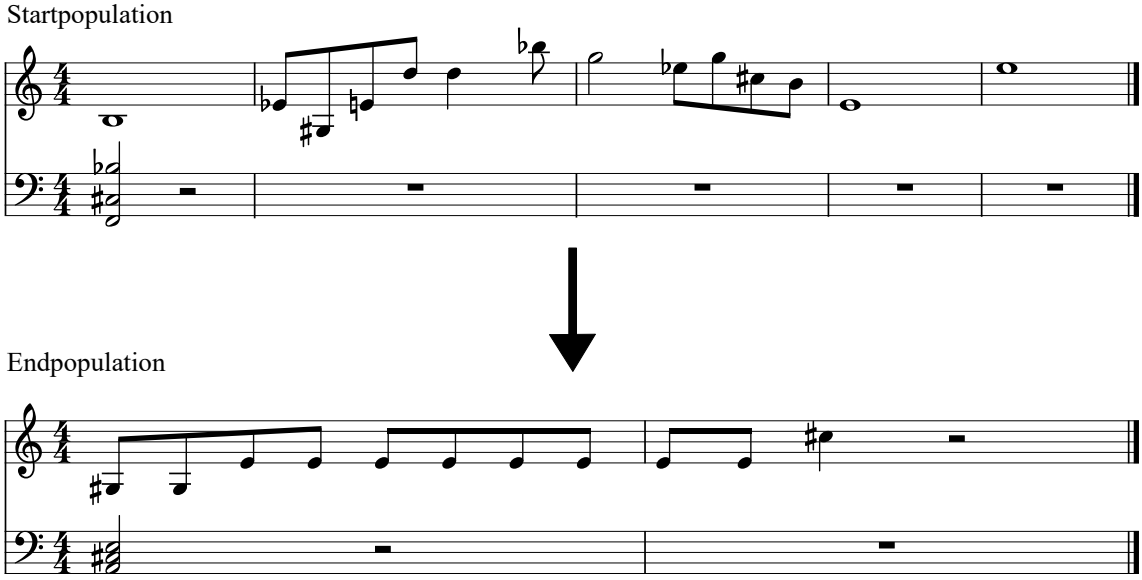
Anhang

Einige Beispiele für erzeugte Anfangs- und Endpopulationen sind hier in der Notenschrift dargestellt.

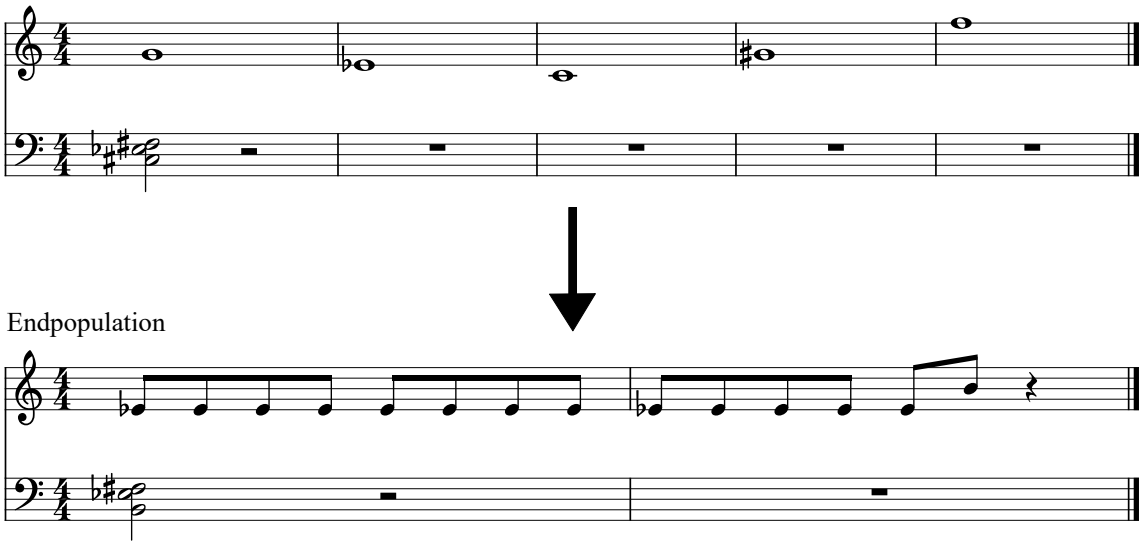
Startpopulation



Startpopulation

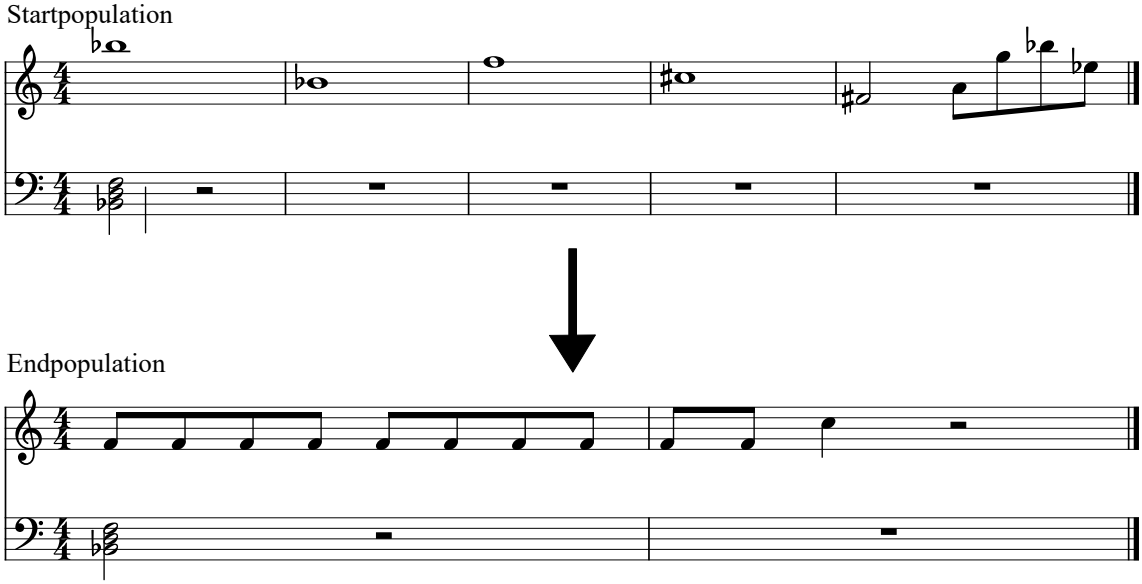


Startpopulation



Endpopulation

Startpopulation



Endpopulation