Mixtape: Analyse und Erstellung

Ähnlichkeitsanalyse von Musik anhand einer praktischen Implementation

Autoren: Curdin Barandun, Stefan Derungs, Gino Paulaitis
Betreuer: Prof. Dr. Josef Joller, Prof. Dr. Ruedi Stoop
Expertin: Léonie Fierz

14. Juni 2013

Aufgabenstellung

Aufgabenstellung
„Mixtape: Analyse und Erstellung“

1. Betreuer

Aufgabenstellung und Betreuer HSR:

- Josef Joller, Dozent für Informatik HSR

2. Ausgangslage, Problembeschreibung

Automatische Musik Klassifikation wird zunehmend ein Thema. Bisher wurden neue CD's von Radiosendern in MP3 konvertiert, mit semantischen Attributen (Musikstil, Spieldauer,…) und auf Musikservern abgespeichert. Die Programmgeneratoren selektieren anschliessend passende Musikstücke und füllen mit Jingles auf.

Dieses Verfahren ist sehr aufwendig und fehleranfällig. Von daher wurden verschiedene Ansätze für eine automatisierte Klassifikation von Musikstücken entwickelt.

3. Aufgabenstellung

Ziel der Arbeit ist zunächst die Erarbeitung einer Übersicht zum Entwicklungsstand "Music Information Retrieval" (MIR). Dieser Teil der Arbeit muss innerhalb von maximal 2 Wochen abgeschlossen sein. Verschiedene Hinweise wurden bereits gegeben und müssen lediglich vertieft werden:

- Neuronaler Ansatz
- Komplexitätsansatz
- Grammatik-Ansatz
- …

Da im Internet verschiedenste Ansätze auch exemplarisch implementiert wurden, müssen in den nächsten zwei Wochen (Woche 3 & 4) die bestehenden Software Implementierungen gesichtet und getestet werden. Dazu ist ein Test-Set von Musikstücken auszuwählen. Die Auswahl kann sich an einer der Publikationen orientieren. Das Test-Set dient insbesondere im Fall des Neuronalen Ansatzes als Trainings-Set. Das muss anschliessend beträchtlich erweitert werden.

4. Zur Durchführung

Die Bearbeitung der Aufgabe setzt eine Einarbeitung in theoretische Grundlagen von Neuronalen Netzen, Komplexitätstheorie (Kolmogoroff, Lempel Ziv [Zip]) und Induktive Grammatiken voraus. Wesentlicher Teil der Einarbeitung ist ferner das Studium der relevanten Literatur (was haben andere auf diesem Gebiet gemacht, an was wird gearbeitet?), gemäss obigem Terminplan.

Solide Programmierkenntnisse und die Bereitschaft, sich rasch neues Domänenwissen anzueignen, werden vorausgesetzt.

Wegen der variablen Breite der Aufgabe eignet sich das Thema gut für eine Bearbeitung durch drei Studierende.

Mit dem Betreuer finden in der Regel wöchentliche Besprechungen statt. Zusätzliche Besprechungen sind nach Bedarf durch die Studierenden zu veranlassen. Besprechungen mit dem Auftraggeber werden nach Bedarf durchgeführt.

Alle Besprechungen sind von den Studenten mit einer Traktandenliste vorzubereiten und die Ergebnisse sind in einem Protokoll zu dokumentieren, das dem Betreuer per E-Mail zugestellt wird.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhalten die Studierenden ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Implementierung und Dokumentation.

5. Dokumentation und Abgabe

Wegen der beabsichtigten Verwendung der Ergebnisse in der Lehre und in weiteren Projekten wird auf Vollständigkeit sowie (sprachliche und grafische) Qualität der Dokumentation erhöhter Wert gelegt.

Die Dokumentation zur Projektplanung und –Verfolgung ist gemäss den Richtlinien der Abteilung Informatik anzufertigen. Die Detailanforderungen an die Dokumentation der Recherche- und Entwicklungsergebnisse werden entsprechend dem konkreten Arbeitsplan festgelegt.

Die Dokumentation ist vollständig auf CD in drei Exemplaren abzugeben.

Neben der Dokumentation sind abzugeben:

- ein Poster zur Präsentation der Arbeit
- alle zum Nachvollziehen der Arbeit notwendigen Ergebnisse und Daten (Quellcode, Buildskripte, Testcode, Testdaten usw.)
- Material für eine Abschlusspräsentation (ca. 20’)

6. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html

22.02.2010	Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung durch die Betreuer
1. Semesterwoche	Kick-off Meeting
2. Semesterwoche	Abgabe der Projektplanung (Entwurf), einschliesslich ggf. zu beschaffender Hardware
4. Semesterwoche	Abgabe eines vorläufigen Technologierechercheberichts und eines detaillierten Vorschlags für einen Arbeitsplan. Festlegung des Projektplans mit dem Betreuer
6. Semesterwoche	Abgabe eines Umsetzungsvorschlags für die Experimentierumgebung (Funktionsumfang, Aufwandsabschätzung) Abstimmung und Festlegung mit dem Betreuer
7. Semesterwoche	Abgabe Anforderungs- und Domainanalyse für die Experimentierumgebung
10. Semesterwoche	Review-Meeting Softwaredesign für die Experimentierumgebung
13. Semesterwoche	Vorstellung der Implementierung (Arbeitsstand)
11.06.2010	Abgabe der Kurzbeschreibung für die Diplomarbeitsbroschüre und des A0-Posters zur Kontrolle an den Betreuer
18.06.2010	Abgabe des Berichtes an den Betreuer (bis 12:00 Uhr) Abgabe des Posters im Studiengangsekretariat

7. Literaturhinweise

Es gibt zurzeit kein Buch zum Thema, aber viele zu den Grundlagen (neuronale Netze, Komplexität, induktive Grammatiken).

8. Beurteilung

Eine erfolgreiche Bachelorarbeit erhält 12 ECTS-Punkten pro Student (1 ECTS Punkt entspricht einer Arbeitsleistung von ca. 25 bis 30 Stunden: 360 Stunden pro Student).
Für die Modulbeschreibung der Studien/Bachelor-Arbeit siehe
https://unterricht.hsr.ch/staticWeb/allModules/10938_M_SAI.html

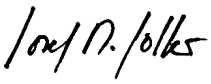
Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/5
2. Berichte (Abstract, Mgmt Summary, techn. u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/5
3. Inhalt*)	3/5

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit mit dem Studenten vereinbart

Im Übrigen gelten die Bestimmungen der Abt. Informatik zur Durchführung von Bachelorarbeiten.

Rapperswil, den

Der verantwortliche Dozent



Josef Joller
Professor für Informatik

Eigenständigkeitserklärung

Wir erklären hiermit,

- dass wir die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt haben, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde.
- dass wir sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben haben.
- dass wir keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt haben.

Rapperswil, 14.06.2013

Curdin Barandun

Stefan Derungs

Gino Paulaitis

Abstract

In dieser Arbeit wird eine Ähnlichkeitsanalyse von Musikstücken mittels der Kompressionsdistanz angestrebt, um daraus ein Pfad für eine Wiedergabeliste zu generieren. Beim Versuch, einen neuen Ansatz für die Berechnung dieser Distanz zu finden, wurde eine eigene, auf dem Lempel-Ziv 77 Kompressionsverfahren basierende Variante umgesetzt.

Auch bei der Auswahl der Musikeigenschaften und deren Umsetzung wurde auf frühere Arbeiten aufgebaut und vor allem im Bereich Tempo und Harmonie versucht, die Verfahren zu optimieren. Dabei ist auch ein Ziel, die Rechenzeit bei der Extraktion von Musikeigenschaften durch eine hochgradig parallelisierte Architektur zu verkürzen.

Für das Problem der Generierung von Wiedergabelisten gab es kaum Arbeiten auf denen man hätte aufbauen können. Dies weil es kaum praktische Umsetzungen von *Music Information Retrieval* Applikationen gibt und die Resultate der Studien grundsätzlich auf den Resultaten von Genreklassifizierungen beruhen. Um einen eigenen Ansatz umzusetzen, musste deshalb zuerst eine Definition für einen optimalen Pfad einer Wiedergabeliste eingeführt werden. Für das Finden dieses Pfades haben wir einen Algorithmus entwickelt, welcher durch Dijkstras Shortest Path Algorithmus inspiriert wurde. Da uns keine Möglichkeit zur Verfügung stand, die Ergebnisse dieser Pfadsuche mit anderen Arbeiten zu vergleichen, müssen Vergleiche mit anderen Methoden auf später verschoben werden.

Management Summary

Ausgangslage

Das Forschungsgebiet *Music Information Retrieval* (MIR) beschäftigt sich mit der Analyse von Musikstücken. Es gibt dafür bereits zahlreiche Dienste, deren Empfehlungen aber oft nicht auf den musiktheoretischen Eigenschaften eines Musikstücks, sondern auf manuell eingetragenen Informationen wie Genre und Interpreten beruhen. Dadurch erhält die Klassifizierung eine unerwünschte subjektive Komponente, da Personen Musikstücke abhängig von musikalischer Bildung und Vorliebe unterschiedlichen Genres zuordnen können. Durch eine feste und meist grobe Kategorisierung der Genres können ausserdem Musikstücke als ähnlich klassifiziert werden, welche beim Hören als nicht ähnlich empfunden werden. Deshalb soll in dieser Arbeit die Möglichkeit untersucht werden, Eigenschaften eines Musikstücks anhand einer Signalanalyse zu bestimmen. Die gewonnenen Charakteristiken sollen dann mittels einer geeigneten Metrik verglichen werden. Als praktische Anwendung wird eine Webapplikation angestrebt, welche mit diesen Verfahren Wiedergabelisten basierend auf individuell festgelegten Ähnlichkeitskriterien generiert.

Vorgehen

Diese Arbeit verschafft zu Beginn einen Überblick über musiktheoretische Eigenschaften und bereits allgemein verfügbare MIR-Werkzeuge (u.a. jAudio, aubio). Für die Extraktion der Musikeigenschaften wurden sowohl existierende Tools eigenen Bedürfnissen angepasst, als auch zusätzliche Verfahren entwickelt. Die Ähnlichkeit der durch die extrahierten Eigenschaften beschriebenen Musikstücke haben wir mittels einer Annäherung an die normalisierte Informationsdistanz (NID) berechnet. Dafür wurde die zugrunde liegende Kolmogorov-Komplexität basierend auf den Grundzügen des LZ77 (Lempel-Ziv Komprimierungsalgorithmus) implementiert. Für das Pattern-Matching wurden Suffix-Arrays eingesetzt, welche auf dem von Sanders und Kärkäinen vorgeschlagenen Skew Algorithmus beruhen. Zur Erzeugung der Wiedergabelisten wurde ein eigenes Verfahren entwickelt, welches die berechneten NIDs benützt und von Dijkstras Shortest-Path Algorithmus inspiriert ist. Das Webinterface zur Bedienung der Applikation baut auf dem Spring Framework auf und benützt HTML5.

Ergebnisse

Die in dieser Arbeit umgesetzte Java-Applikation extrahiert Eigenschaften aus Audiosignalen und berechnet aus dem Vergleich dieser Eigenschaften die NID. Über das Webinterface hat der Anwender die Möglichkeit, ihm zusagende Musikstücke auszuwählen, sowie Ähn-

lichkeitskriterien zu definieren. Mit diesen Einstellungen wird anschliessend eine neue Wiedergabeliste aufgebaut.

Inhaltsverzeichnis

Technischer Bericht

Kapitel 1

Einleitung

Vor dem Aufkommen von CDs und MP3-Playern wurden Wiedergabelisten, wie wir sie heute kennen, auf Kassetten zusammengestellt. Dabei mussten die Musikstücke von verschiedenen Tonbändern auf ein einzelnes kopiert werden. Dies war mit viel Aufwand und Herzblut verbunden. Eine solche Kassette wurde dann stolz als Mixtape bezeichnet. Diese Ära ist längst vorbei, aber die Beliebtheit von Wiedergabelisten ist geblieben. Heute sind wir in der Lage, Wiedergabelisten automatisiert zu erzeugen. Dies ist auch das Ziel der entwickelten Applikation, wodurch der Name als kleiner Tribut an die vergangenen Tage passend scheint.

1.1 Mögliche Umsetzungsideen für diese Arbeit

In diesem Kapitel sollen mögliche Ideen für eine Umsetzung in dieser Bachelorarbeit erläutert werden. Abschliessend nehmen wir dazu Stellung, für welche der Ideen wir uns für die vorliegende Arbeit entschieden haben.

1.1.1 Erkennung von eingebetteten Musikstücken

Heutzutage existieren zahlreiche Remixes und Coverversionen, wo Passagen aus anderen Musikstücken verwendet werden. Oftmals hat man auch das Gefühl, dass zwei scheinbar verschiedene Musikstücke, trotzdem „gleich“ tönen. Man könnte nun den Ansatz ableiten, „Patterns“ oder Ausschnitte aus einem Musikstück zu extrahieren und dann mit einer Musik-Datenbank abzugleichen, um weitere Musikstücke zu finden, die dieselben „Patterns“ enthalten. Ein praktischer Anwendungsfall wäre beispielsweise die Plagiat Erkennung. In der Literatur ist Software zur Plagiat-Suche bereits etabliert. Im Musik-Umfeld ist dies unseres Wissens nach nicht oder nur wenig verbreitet.

1.1.2 Musikstück-Erkennung mittels Singen oder Summen

Bei dieser Idee geht es darum, dass man als Benutzer eine Melodie singen oder summen kann und die Anwendung nach Musikstücken suchen lässt, die mit dieser Melodie übereinstimmen. Beim Prüfen dieser Idee haben wir festgestellt, dass die bereits weit verbreitete mobile Applikation *SoundHound* diese Funktion bereits kennt. Sie ermöglicht dem Benutzer den Namen eines Musikstücks zu finden, indem er einen Ausschnitt aus einer Audio-Aufnahme abspielt oder eine Passage desselben singt oder summt. Im Rahmen dieser Arbeit

könnte man als erstes untersuchen, wie gut die Suche mit verschiedenen Genres funktioniert. Sollten Schwächen bei gewissen Genres entdeckt werden, sollten die Ursachen dieser Schwächen untersucht werden.

1.1.3 Automatisierte Genre-Erkennung mit Benutzer-Eingaben

Auf Grund des wachsenden Datenvolumens an digital verfügbarer Musik ist das Bedürfnis nach automatisierter Klassifizierung von Musikstücken in die jeweiligen Genres allgegenwärtig. Dabei wird weniger nach expliziten Genres kategorisiert, sondern vielmehr wird nach Ähnlichkeiten zwischen den Musikstücken gesucht. Möchte man nun aber dennoch nach Genres kategorisieren, steht man vor der Schwierigkeit, dass diese nicht eindeutig sind. Oftmals ist es selbst für einen geübten Musikkenner schwierig, ein bestimmtes Musikstück mit Gewissheit einem Genre zuzuordnen - nicht zuletzt auch, weil es meist zahlreiche weitere Sub-Genres/Unterarten gibt. Dadurch entsteht das Problem, dass ein Musikstück von verschiedenen Benutzern verschieden eingeordnet werden kann.

Eine Möglichkeit, dieses Problem zu umgehen, bestünde darin, mithilfe einer Software eine automatisierte Einteilung vorzunehmen und anschliessend über Kategorisierungen von Benutzern diese Einteilungen zu perfektionieren. Dieses Verfahren sollte - so die Idee - immer wie besser werden, je mehr Benutzer die jeweiligen Musikstücke einordnen. Der Algorithmus analysiert die Benutzer-Kategorisierungen und bezieht diese in die zukünftigen Einteilungen mit ein. Dadurch erhält die Software mit der Zeit eine immer präzisere „Beschreibung“ eines Genres.

1.1.4 eDJ: Musikwünsche mit automatisierten Überleitungen

Stellen Sie sich vor, Sie sind an einer Hochzeit der DJ. Aus ihrer Sammlung von 30 000 Musikstücken wollen sie nun eine Wiedergabeliste erstellen, welche für die Hochzeit geeignet ist. Sie lassen die Musik durch die Software gruppieren. Aus diesen Gruppen wählen Sie „Lounge Music“ und lassen sich eine Wiedergabeliste generieren. Zusätzlich können an der Hochzeit die Gäste ihre Musikwünsche anbringen. Als erster kommt der Trauzeuge und wünscht ein Musikstück der Heavy Metal Band *Iron Maiden*, weil er damit auf ein besonderes Ereignis während der Schulabschlussfeier des Bräutigams hinweisen möchte. Anschliessend kommt die Oma und wünscht ein Musikstück der deutschen Schlagersängerin *Andrea Berg*. Ausserdem müssen Sie noch das Musikstück spielen, bei dem sich Braut und Bräutigam kennengelernt haben. Diese Musikstücke unterscheiden sich sehr stark in der Musikrichtung.

Nun wäre es ideal, wenn Sie nicht einfach nur ein Musikstück nach dem anderen abspielen müssen, sondern sie einen möglichst „weichen“ Übergang vom einen Musikstück zum anderen machen könnten, indem Sie weitere Musikstücke dazwischen schalten. Diese dazwischen geschalteten Musikstücke sollten sich in der Musikrichtung immer mehr dem nächsten gewünschten Musikstück annähern. An dieser Stelle soll nun ein Stück Software zum Einsatz kommen, welches genau diese Aufgabe erledigt. Über ein Webinterface sollen die Musikwünsche durch die Gäste eingegeben werden. Anschliessend baut sich die Software einen Pfad auf, wie sie zum Ziel kommt - und Sie können in der Zwischenzeit die Party geniessen.

Je nach verbleibender Zeit für die Bachelorarbeit, könnte man die Idee durch die nachfolgenden Elemente erweitern.

- Die Übergänge am Anfang und Ende eines Musikstücks mit Fade-Effekten (Fade-In und Fade-Out) schöner zu gestalten.
- Musikstück-Übergänge optimaler gestalten. Es soll nicht jedes der Musikstücke zwischen den gewünschten Musikstücken von Anfang bis Ende fertig gespielt werden. Stattdessen soll dort zum nächsten Musikstück gewechselt werden, wo es „gegenüber dem nächsten Musikstück am besten passt“ - natürlich unter Einhaltung gewisser Randparameter (’ein Musikstück muss mindestens x Sekunden spielen’, ’kein Wechsel x Sekunden vor dem Ende eines Musikstücks’, usw.).
- Den Zeit-Aspekt mit einbeziehen. Es soll angegeben werden können, binnen welcher Zeit die Software jeweils ein gewünschtes Musikstück erreichen muss. Wählt man die Zeit knapp, so werden die Übergänge weniger „weich“. Gibt man der Software mehr Zeit, so werden die Übergänge dementsprechend weicher und schöner.
- Der Vorgeschlagene Musikpfad, also die Wiedergabeliste, sollte bearbeitet werden können.
- Die Gruppierung der Musikstücke kann manuell angepasst werden (evtl. unter Einbezug von *Machine learning* Algorithmen).
- Möglichkeit zur Darstellung und Bearbeitung von extrahierten Musikinformationen. Nach der Bearbeitung sollten allfällige Ähnlichkeitsberechnungen und Clusterings mit den editierten Informationen auslöst werden.

1.1.5 Fazit

Der praktische Nutzen, sowie die spannenden Teilprobleme, die eine Implementation des eDJ-Tools mit sich brachte, haben uns schlussendlich von dieser Idee überzeugt. Wir waren uns zwar der grossen Herausforderung, auf die wir uns einliessen, bewusst, doch schien die Möglichkeit auf eine erfolgreiche Umsetzung äusserst verlockend.

1.2 Überblick über Arbeiten im Bereich Musik- und Ähnlichkeitsanalyse

Das Gebiet Music Information Retrieval (MIR), welches sich mit der Musikanalyse befasst, erreichte in den letzten Jahren zunehmende Beliebtheit. Die meisten Ansätze basieren dabei auf maschinellen Lernalgorithmen, um Musik zu klassifizieren. In [47] werden zum Beispiel eine Menge von Eigenschaften definiert, welche in Klangfarben, Textur und Rhythmus aufgeteilt werden. Die Musikstücke werden dann mittels einem statistischen Mustererkennungs-Klassifizierer, wie einem Gaussian Mixture Model, Genres zugeteilt. Dabei wurden relativ gute Resultate erzielt. Es konnten 61% der Musikstücke richtig klassifiziert werden bei 10 Musikgenres.

Noch etwas bessere Resultate wurden mit einer ähnlichen Variante, welche in [35] beschrieben wurde, erzielt. Dabei werden die aus der Spracherkennung beliebten *Mel Frequency Cepstral Coefficients (MFCC)* und ein *Linear Predicting Coding* als Eigenschaften ver-

wendet. Als Klassifizierer wurde ein Hidden Markov Model (HMM) gewählt, weil in [44] gute Resultate bei der Spracherkennung erzielt wurden.

Vergleiche zwischen verschiedenen Klassifizierungsmethoden in [41] zeigten, dass sie die besten Resultate mit neuronalen Netzwerken erzielten, welche eine Genauigkeit von 96% aufweisen. Etwas weniger gut schnitt die Support Vector Machine (SVM) mit einer Genauigkeit von 86% ab. Am schlechtesten waren die k-Nearest Neighbors (k-NN) und k-Means Varianten mit 80% Genauigkeit. Dabei wurden jedoch nur die MFCCs als Features verwendet.

Ein eher neuer Ansatz ist das Vergleichen von Musik über die Kompressionsdistanz. Dabei wurden in [53] die Ähnlichkeit von Musikstücken im MIDI Format mittels der Normalized Compression Distance (NCD) berechnet. Das *CompLearn Toolkit* wird zur Berechnung der NCD vorgeschlagen. Es unterstützt die Standardkompressoren *gzip*, *bzip* und *PPMZ*, um die Distanz zu berechnen und ist universell einsetzbar. Es wird bestätigt, dass die Kompressionsdistanz sehr robuste und detaillierte Ergebnisse liefert. Weiter zeigen sie ihre Nützlichkeit für die Verwendung mit einer SVM zur Klassifizierung auf.

In [56] wird genau dieser Ansatz weiterverfolgt. Dabei wird ein eigener LZ78 basierter String-Kernel implementiert, um die NCD von MFCC Eigenschaften zu berechnen. Diese Distanzen werden daraufhin in einer SVM verwendet, um Musik zu klassifizieren. Sie zeigen dabei, dass mit diesem Verfahren bessere Resultate erzielt werden können, als mit den klassischen Verfahren wie den *Markov Models*.

Einen erst kürzlich publizierten Ansatz wählten die Autoren in [23], in welchem die generierten Noten von einem MIDI Synthesizer als *Musical Words* repräsentiert werden. Durch die Analyse der gespielten Noten sind sie in der Lage, sehr feine Charakteristiken von Musikstücken zu erkennen. Durch diese Grammatik basierte Methode konnten sie bei der Klassifizierung von Musik sehr gute Resultate erzielen.

1.3 Genauer untersuchte Verfahren

1.3.1 Maschinelles Lernen

Während unseren Recherchen war maschinelles Lernen stets ein Thema. Hier schienen besonders zwei Konzepte besonders oft genutzt zu werden, das *Hidden Markov Model* und die *Support Vector Machine*.

Das HMM versucht anhand von gegebenen Sequenzen von Ereignissen, den dahinter versteckten Markov-Prozess zu modellieren. Dies wird in der Praxis zur Klassifizierung genutzt, da die Wahrscheinlichkeit der Zugehörigkeit einer Sequenz zu einer Klasse durch Sequenzen ausgedrückt werden kann.

Die SVM scheint in der Praxis ähnlich eingesetzt zu werden wie das HMM. In ihrer Funktionsweise unterscheidet sie sich aber insofern, dass einzelne Objekte jeweils als Punkte in einer Hyperebene gesetzt werden, wobei Klassen von Objekten mit möglichst grosse Abständen durch Geraden voneinander getrennt sind. Neue Objekte können dann anhand ihrer Position einer der Klassen zugeordnet werden.

Da wir in unserer Arbeit auf überwachtes Lernen verzichten wollten und somit keine Klassifizierung möglich ist, haben wir uns nicht genauer mit HMMs und SVMs beschäftig.

1.4 Resultate und Verbesserungsmöglichkeiten

Wie bereits erwähnt wurde, ist MIR ein durchaus aktives Forschungsgebiet. Verbesserungspotenzial sehen wir nun hauptsächlich im Bezug auf unsere konkrete Problemstellung. Hier verfügt unsere Arbeit durchaus über gewisse Alleinstellungsmerkmale. Die Kombination von Normalized Information Distance (NID), unüberwachtem Lernen und dem Verzicht von MIDI-Files als Input ist uns in dieser Form während unseren Recherchen nicht begegnet. Auch das Pathfinding für die Erstellung einer Wiedergabeliste musste genauer geprüft und mit einer eigenen Lösung realisiert werden.

1.4.1 Musikalische Eigenschaften

Auf die musikalischen Eigenschaften bezogen, besteht in der Erkennung der Grundtöne Potenzial zur Verbesserung. Könnten Grundtöne ähnlich gut erkannt werden, wie diese in MIDI-Files vorhanden sind, so kann durch die NID von Ergebnissen ausgegangen werden, die denen von [45] entsprechen. Hierzu ist jedoch die Erkennung mehrerer Grundtöne nötig, was durchaus eine Herausforderung darstellt. Hierzu haben wir einen eigenen Ansatz entwickelt, dessen Ergebnisse zwar nicht überragen, unserer Meinung nach aber Potenzial hat.

1.4.2 Ähnlichkeitsanalyse

Bei der Ähnlichkeitsanalyse hebt sich vor allem die Kompressionsdistanz ab, welche in [53] und [56] verwendet wurden. Die guten Resultate welche erzielt wurden und die universelle Einsetzbarkeit sind beeindruckend und motivieren, auf diesem Gebiet neue Ansätze zu versuchen. Die Probleme mit der Berechnung durch Standardkompressoren welche in [38] aufgezeigt wurden zeigen auch, wo Optimierungspotential vorhanden ist.

1.4.3 Pathfinding

Zur Generierung von Wiedergabelisten, beziehungsweise die Suche nach einem Pfad zwischen ähnlichen Musikstücken, scheinen nach unserem Vorwissen keine relevanten Arbeiten zu existieren. Diese Problemstellung erinnert allerdings an das *Travelling Salesman Problem (TSP)* oder das *Shortest Path* Problem.

Kapitel 2

Tool- und Framework-Analyse

2.1 Feature Extraction mit jAudio

Die Open-Source Software Suite *jMIR* [5] bietet Möglichkeiten zur Musikanalyse auf Audio-basierte als auch formale Art, es kann kulturelle Informationen aus dem Internet sammeln und hilft bei der Verwaltung von Musik-Sammlungen. Für diese Arbeit war in erster Linie die Audio-basierte Analyse relevant, weshalb die Komponente *jAudio* genauer angeschaut wurde.

jAudio ist für die Extraktion von Low Level und High Level Eigenschaften aus Musikstücken zuständig. Die Software arbeitet sowohl mit einer grafischen Benutzeroberfläche (Abbildung 2.1), als auch Konsolen-basiert und kann auch als Library in eigene Projekte eingebunden werden. Es werden Musikstücke in den Formaten *WAV*, *AIFF*, *AU*, *SND*, sowie *MP3* unterstützt, obwohl letzteres nicht offiziell gelistet wird). Für den Analyseprozess werden über 120 Musik Eigenschaften unterstützt. Als Ausgabeformat der Analyse können ACE-XML[1] und Weka ARFF[21] gewählt werden. Zu jedem Analysevorgang wird ein zusätzliches XML generiert, welches die Beschreibungen zu den in der Analyse verwendeten Eigenschaften enthält.

2.2 Marsyas

Marsyas[1] [8] ist ein freies Framework zur Verarbeitung von Audiosignalen, welches auf Erweiterbarkeit und Anpassbarkeit ausgerichtet ist. Das Framework soll sowohl von Experten als auch von Amateuren verwendbar sein. Zum einen wird dafür eine grafische Benutzerschnittstellen angeboten und zum anderen bietet es eine mächtige API. Mit Marsyas werden einige einfache Tools mitgeliefert, die über ein CLI aufgerufen werden können. Diese sind allerdings nur bedingt nützlich und sind auch eher als Beispielsapplikationen gedacht, welche illustrieren sollen, was mit Marsyas möglich ist.

Das von George Tzanetakis entwickelte Framework scheint ein mächtiges und vielfältiges Framework zu sein. Da es in C/C++ programmiert wurde, eignet es sich allerdings nur bedingt für den Einsatz in Java-Applikationen, da die angebotenen Schnittstellen nur wenig Funktionalität anbieten.

[1] **M**usic **A**nalysis **R**etrieval and **S**ynthesis for **A**udio **S**ignals

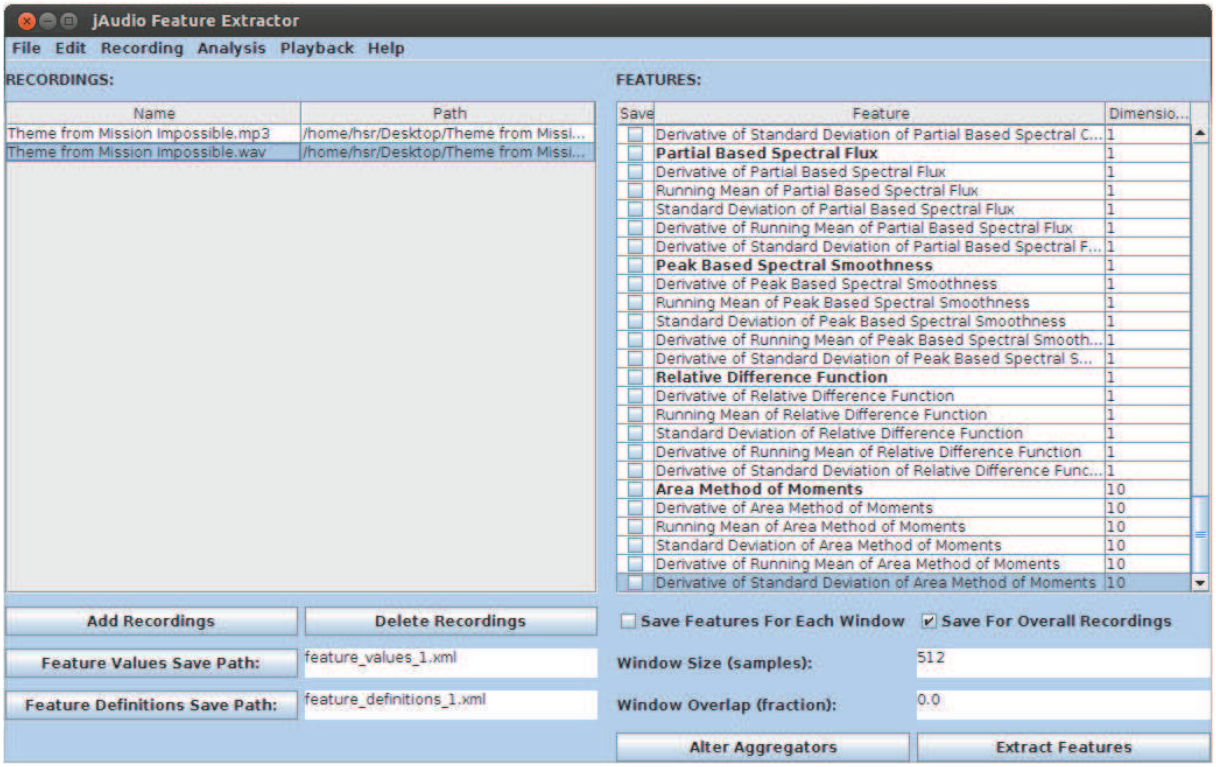


Abbildung 2.1: Benutzeroberfläche von jAudio.

2.3 Das NEMA Projekt

Networked Environment for Music Analysis [9]. Bei diesem multinationalen Projekt *NEMA* handelt es sich um ein Webservice-basiertes Framework. Es wurde vom International Music Information Retrieval Systems Evaluation Laboratory Project entwickelt und soll weltweit von Wissenschaftlern und Ausbildungseinrichtungen verwendet werden können. *NEMA* hat zum Ziel, die Analyse und die Auswertung von Musikdaten zu vereinen, und soll sich bei Bedarf erweitern lassen. Das Framework wurde nicht genauer untersucht, weil einerseits die Webseite seit 2010 nicht mehr aktualisiert wurde und somit die Aktualität des Projekts fraglich ist und andererseits wurde konnte auch keine Zugangsmöglichkeit ausfindig gemacht werden.

2.4 *aubio*

aubio [3] ist ein Tool zur Extraktion von Musikinformationen. Zum Funktionsumfang von *aubio* gehören unter anderem digitale Audiofilter, einen *Phasen Vocoder*, diverse Onset Detection Function sowie diverse Methoden zur Erkennung von Tonhöhen. Es bietet auch eine für diese Arbeit äusserst interessante Implementation des Beat Trackings.

2.5 Weka 3: Data Mining Software in Java

Weka 3 [20] ist eine in Java geschriebene, gut dokumentierte Software-Suite für Data Mining, welche zahlreiche Algorithmen für maschinelles Lernen enthält. Dabei können die Algorithmen entweder über die mitgelieferte grafische Benutzeroberfläche direkt auf Datensätze angewendet werden, oder via API in eigenen Java-Programmen genutzt werden. Die Software verfügt unter anderem über Tools für die Daten-Vorverarbeitung, -Klassifizierung, -Regression[2], -Clustering und -Visualisierung. Für die Analyse können unter anderem Dateien in den Formaten *ARFF*, *CSV* und *XRFF* übergeben werden, wobei vor allem ersteres interessant ist bei der Verwendung in Kombination mit jAudio (siehe Abschnitt 2.1).

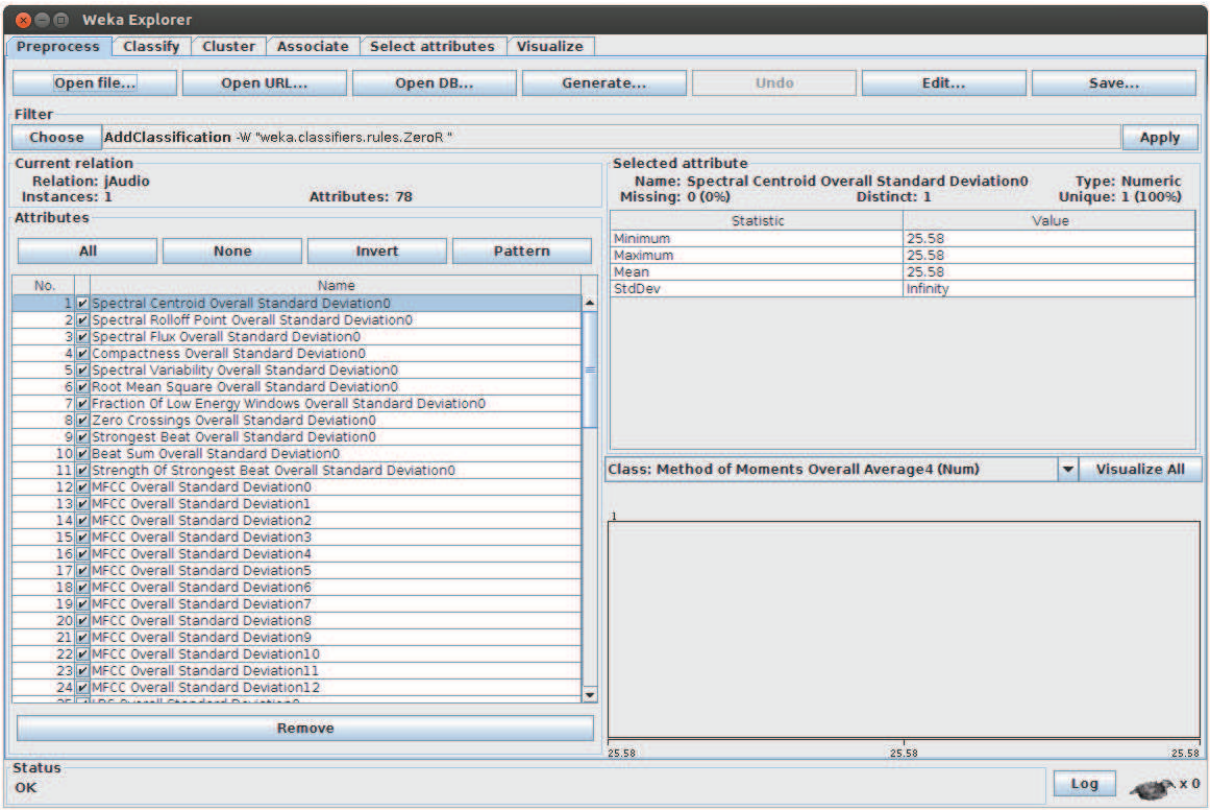


Abbildung 2.2: Benutzeroberfläche von Weka 3.

2.6 ELKI

Environment for Developing KDD-Applications Supported by Index-Structures [4]. ELKI ist eine quelloffene und gut dokumentierte Library für Data Mining und Data Management. Ungleich zu vielen anderen Data Mining Frameworks, wie Weka 3 oder YALE, ist in ELKI das Data Mining und Data Management voneinander separiert. ELKI ist offen für beliebige Datentypen und Distanz- oder Gleichheits-Funktionen. Ihr fundamentaler Ansatz ist die

[2] **Regression** Ein statistisches Analyseverfahren, bei dem die Beziehung zwischen Variablen abgeschätzt wird. Es wird versucht, eine Beziehung zwischen einer abhängigen und einer oder mehreren unabhängigen Variablen festzustellen.

Unabhängigkeit von File Parsern, Datenbanken, Datentypen, Distanzen und Distanzfunktionen sowie Data Mining Algorithmen. Helferklassen für algebraische und analytische Berechnungen werden in der Regel für alle Algorithmen angeboten.

Es werden sehr generische Interfaces zur Verfügung gestellt, um beispielsweise eigene Datentypen und deren Repräsentation in Form von Vektoren zu implementieren. Sehr interessant sind auch die Schnittstellen für eigene Distanzfunktionen und eigene Algorithmen. Einige Algorithmen, wie das *k-Means Clustering* oder der *Expectation Maximization* Algorithmus, werden bereits mitgeliefert.

Allerdings sind die Implementationen nicht zur Einbindung als Library in eigene Projekte gedacht, sondern nur zu Forschungs- und Analysezwecken. Sie sind allerdings immer noch schneller als Weka 3. Eigene Algorithmen Implementationen können einfach in das MiniGui eingebunden werden und in Kombination mit bereits verfügbaren Implementationen getestet werden. ELKI wird noch aktiv entwickelt, weshalb die Hersteller für keine stabile API garantieren können und sie deshalb von einer Verwendung als Library abraten.

2.7 KNIME

Konstanz Information Miner [7]. KNIME ist eine professionelle, quelloffene Software zur interaktiven Datenanalyse. Sie stellt mehrere hundert Module zur Datenintegration, Datentransformation, Datenanalyse und Datenvisualisierung bereit. Module können per *Drag and Drop* in den Workflow hineingezogen und mit anderen Modulen verbunden werden. Teilweise ist etwas Konfigurationsaufwand nötig, um die jeweiligen Module verwenden zu können.

Die Core Architektur erlaubt die Verarbeitung von grossen Datenmengen, welche nur durch die Festplattengrösse limitiert werden. Zudem besteht die Möglichkeit, Open Source Verfahren und Algorithmen von Drittherstellern wie beispielsweise Weka 3 einzubinden. KNIME ist in Java entwickelt und als Eclipse Plugin verfügbar. Auf der Webseite steht eine umfangreiche Dokumentation mit Beispiel-Applikationen und Tutorials bereit.

2.8 Informatiosnvisualisierung mit JUNG

Java Universal Network/Graph Framework [6]. JUNG ist eine Open Source Software Bibliothek in Java, welche eine erweiterbare Sprache für das Modellieren, die Analyse und die Visualisierung von Daten bereitstellt. Die Visualisierungen werden dabei als Graph oder Netzwerk dargestellt. Ausserdem bietet es Möglichkeiten zur Darstellung von Entities und deren Beziehungen an, zum Beispiel mittels gerichteten oder ungerichteten Graphen, multimodalen Graphen, Graphen mit parallelen Kanten und Hypergraphen. Zudem bietet JUNG einen Mechanismus für das Annotieren von Entities und von Beziehungen mit Metadaten.

Die aktuelle Distribution beinhaltet eine Vielzahl von Algorithmen für Graphen Theorie, Data Mining, Analyse von Sozialen Netzwerken, sowie Routinen für Clustering, Decomposition, Optimierung, Random Graph Generierung, statistische Analyse und die Berechnung von Netzwerk Distanzen und Flows. Ausserdem wird ein Visualisierungs-Framework angeboten, mit dessen Hilfe Werkezeuge für die interaktive Untersuchung von vernetzten Daten erstellt werden können. Zudem werden Filter-Mechanismen unterstützt.

Bei der Visualisierung werden eine Vielzahl von unterstützenden Funktionen für die Arbeit mit Kanten und Knoten angeboten: *Drag and Drop* der Kanten und Knoten, Manipulation der Darstellung (Farbe, Kantendicke, etc.) und editieren der Eigenschaften. Je nach Umsetzungsart und Funktionsumfang einer grafischen Benutzeroberfläche für Mixtape wäre JUNG somit eine interessante Komponente, da sie eine einfache Benutzer-Interaktion ermöglicht.

JUNG bietet eine umfangreiche API, um eigene Graphen zu generieren und darzustellen. In den Demo Applikationen auf der Hersteller-Webseite können der Funktionsumfang des JUNG Visualisierungs-Frameworks getestet werden. Die Funktionsweise von JUNG ist dank der API, den verfügbaren Tutorials und den Demo Applikationen gut ersichtlich und gut dokumentiert.

2.9 Musiksammlungen zur Verwendung in der MIR-Forschung

2.9.1 GTZAN Genre Collection

Die Musiksammlung [10] wurde für eine renommierte Arbeit [47] im Bereich Genre Klassifizierung zusammengestellt. Die Sammlung enthält keine Titel und gewährt keine Copyright Berechtigung. Trotzdem wird sie zu Vergleichszwecken angeboten. Falls Arbeitsresultate veröffentlicht werden, welche mithilfe dieser Musiksammlung erzielt worden sind, muss vorgängig der Autor George Tzanetakis kontaktiert werden.

2.9.2 RWC Music Database

Die Musiksammlung [17] ist nicht frei von Copyright Beschränkungen, kann aber nach einer Registrierung und einer geringen Gebühr für Forschungszwecke verwendet werden. Die Sammlung enthält über 300 Musikstücke, welche sich gemäss Hersteller in der Qualität stark unterscheiden können. Teilweise wurden Musikstücke auch in Genres klassifiziert. Die Datenbank wurde von der RWC Music Database Arbeitsgruppe der Real World Computing Partnership aus Japan zusammengestellt. Letzterer gehören auch alle Rechte.

2.9.3 MAGNATAGATUNE

Diese Musiksammlung [13] wurde ebenfalls für MIR-Forschungszwecke zusammengestellt und ist besonders für Tagging-Aufgaben geeignet Die Musikstücke wurden mit manuell erfassten Annotationen ergänzt und liegen als MP3 (16kHZ, 32kBps, Mono) vor. Die Musikstücke wurden ausserdem mit Informationen über die Musik-Struktur und die musikalischen Eigenschaften, wie Rhythmus, Tonhöhe und Klangfarbe versehen.

Kapitel 3

Grundlagen der Hörwahrnehmung

Dieser Abschnitt soll eine kurze Einführung in Grundlagen der Hörwahrnehmung geben. Hierzu soll in einem ersten Schritt ein Einblick in die Anatomie des menschlichen Gehörs verschaffen werden, darauf aufbauend werden anschliessend die Hauptbereiche der Psychoakustik vorgestellt. Dies ist insofern wertvoll, da doch einige der heutzutage meistverwendetsten Features auf diesen Grundlagen beruhen und nicht etwa mathematischer oder physikalischer Natur sind, wie zum Beispiel die MFCCs.

3.1 Anatomie des menschlichen Gehörs

Die Cochlea (Hörschnecke) ist Teil des Innenohrs und zuständig für die Hörwahrnehmung. Die tatsächliche Umsetzung von Schallwellen in Nervenimpulse geschieht durch Haarzellen auf dem Cortischen Organ, welches sich auf der Basilarmembran befindet. Die Haarzellen sind in vier Reihen angeordnet, welche sich in ihren Eigenschaften unterscheiden.

Die äusseren drei Reihen dienen als sogenannter cochleärer Verstärker, was durch die direkt vom Frequenzspektrum gesteuerte Längenverstellung der einzelnen Haarzellen ermöglicht wird. Hierdurch werden Schwingungen an den Resonanzstellen etwa tausendfach verstärkt, während Schwingungen jenseits dieser Resonanzstellen stark gedämpft werden.

Die Weiterverarbeitung geschieht nun durch die innere Reihe, deren Haarzellen jeweils auf eine spezielle Frequenz spezialisiert sind. Die einzelnen Frequenzen des zuvor durch den cochleären Verstärker zerlegten Klangs reizen nun die jeweiligen Haarzellen. Dieser Reiz löst ein elektrisches Signal aus, welches dem eigentlichen Nervenimpuls entspricht.

3.2 Tonheit

Die tatsächliche Höhe eines Tons ist durch die Frequenz des physikalischen Signals gegeben und wird üblicherweise in der SI-Einheit Hertz (Hz) angegeben. Die Anatomie und Funktionsweise des menschlichen Gehörs hat zur Folge, dass Unterschiede zwischen zwei Frequenzen nicht linear wahrgenommen werden. Die vom Menschen tatsächlich empfundene Tonhöhe wird in der Psychoakustik als Tonheit bezeichnet.

Die Tonheit einer Frequenz steht in einem linearen Zusammenhang mit dem Abstand zwischen dem Erregungsmaximum der Basilarmembran und des Helicotrema. Beim ovalen Fenster ist die Basilarmembran schmal und dick, wird jedoch mit abnehmendem Abstand

zum Helicotrema immer breiter und dünner. Dies führt zu einer hohen Eigenfrequenz am ovalen Fenster und zu einer tiefen Eigenfrequenz am Helicotrema. Nun schwingt die Basilarmembran dort am stärksten, wo sie mit einer gewissen Frequenz am besten schwingen kann. Die zuvor durchgeführte Verstärkung durch den cochleären Verstärker führt zu einer sehr starken, gut abgrenzbaren Erregung in einem sehr kleinen Gebiet der inneren Haarzellen. Hinzu kommt noch eine laterale Hemmung, wodurch stark erregte Neuronen leicht erregte benachbarte Neuronen hemmen.

Die Nervenzellen auf der Basilarmembran sind in 24 Bereiche unterteilt, welche 1.3 Millimeter breit sind. Alle Nervenimpulse, die innerhalb eines dieser Bereiche entstehen, werden gemeinsam verarbeitet. Diese 24 Bereiche entsprechen den sogenannten Frequenzgruppen, die vom menschlichen Gehör gemeinsam ausgewertet werden. Unterhalb von 500 Hertz sind diese in etwa 100 Hertz grosse Bereiche eingeteilt. Oberhalb dieser Grenze besteht zwischen den einzelnen Gruppen ein Verhältnis von 1.19, was einer kleinen Terz entspricht. Somit kann eine Frequenzgruppe als ein dementsprechend breiter Bandpass angesehen werden. Zur Einteilung von Frequenzen in Frequenzgruppen kann die *Mel-Skala* (bzw. *Bark-Skala* oder *ERB-Skala*) verwendet werden.

Im Weiteren ist zu erwähnen, dass eine innere Haarzelle maximal 4000-mal pro Sekunde feuern kann. Dies würde eine sinnvolle Analyse auf Frequenzen von maximal 4 kHz erlauben. Nun macht sich allerdings das Zentralnervensystem die Phasenkodierung zu Nutze. Werden an einer Haarzelle Wellenberge in unterschiedlichen Zeitabständen erkannt, so wird der grösste gemeinsame Teiler dieser Zeitabstände als tatsächlicher Abstand zwischen den Wellenbergen genutzt.

3.3 Lautheit

Ähnlich wie die Tonhöhe entspricht auch die Lautstärke nicht der tatsächlich wahrgenommenen Lautstärke eines Tones. Physikalisch ist die Lautstärke einzig vom Schalldruck abhängig. Die empfundene Lautstärke ist hingegen vom Frequenzbereich, der Schalldauer sowie der Bandbreite abhängig. In der Psychoakustik wird die wahrgenommene Lautstärke als Lautheit bezeichnet.

Grundsätzlich liegt der hörbare Bereich von Frequenzen in etwa zwischen 20 Hz und 20 kHz. Das menschliche Gehör reagiert am empfindlichsten auf Frequenzen im Bereich zwischen 2 und 5 kHz. In diesem Bereich sind bereits sehr leise Töne hörbar. Ab einer Frequenz von 10 kHz muss der Schallpegel deutlich erhöht werden, um die Hörwahrnehmung zu ermöglichen. Dies gilt auch für sehr niedrige Frequenzen unterhalb von etwa 100 Hz.

Das menschliche Gehör mittelt den Schalldruck über ein Intervall von 200 Millisekunden. Dies hat zur Folge, dass beispielsweise ein Ton, der während 100 Millisekunden erklingt, leiser wirkt, als derselbe Ton, der während 200 Millisekunden erklingt. Ab 200 Millisekunden ist allerdings kein Unterschied mehr wahrnehmbar. Gleichzeit nimmt allerdings die Stärke eines Nervenimpulses ab, wenn eine Haarzelle über längere Zeit erregt wird. Wird eine kurze Erholungsphase eingelegt, so feuert die Haarzelle wieder stärker.

3.4 Tonigkeit

Die Tonigkeit beschreibt die wahrgenommene Ähnlichkeit zwischen zwei Klängen im Verhältnis 1:2. So kann beispielsweise ein Musikstück problemlos wiedererkannt werden, obwohl es eine Oktave höher oder tiefer gespielt wird. Wird hierfür ein anderes Verhältnis gewählt, fällt dies sogleich bedeuten schwerer. Zurückzuführen ist dies wohl auf die Zusammensetzung eines Klanges aus Harmonischen.

Kapitel 4

Harmonie und Klangfarbe

In diesem Abschnitt soll das Zusammenspiel vom Klängen, sowie deren Komponenten, etwas genauer untersucht werden. Zuallererst sollten jedoch noch einige für diesen Kontext relevante Grundbegriffe geklärt werden.

4.1 Klänge und Geräusche

Oftmals werden Klänge fälschlicherweise mit Geräuschen gleichgesetzt. Trotz der offensichtlichen Gemeinsamkeiten als Schallereignis beschreiben die beiden Begriffe genau gegenteilige Effekte.

Ein Klang ist definiert durch eine beliebige Grundfrequenz, sowie ganzzahlige Vielfache dieser Grundfrequenz, die Oberfrequenzen. Im Zusammenhang mit Schall spricht man hierbei auch oft von Grundton und Obertönen. Aufgrund der gemeinsamen Verhältnisse untereinander werden Grundfrequenz und Oberfrequenzen auch als Harmonische bezeichnet. Dabei ist darauf zu achten, dass die erste Harmonische nicht etwa dem ersten Oberton entspricht, sondern dem Grundton. Sowohl die Anzahl, als auch die Amplituden der Harmonischen, können beliebig variieren. So ist es auch möglich, dass Harmonische einfach vollständig fehlen. Dies gilt selbst für die Grundfrequenz. Die Höhe eines Klanges wird jeweils durch den Grundton vorgegeben, auch dann, wenn der Grundton an sich nur sehr leise oder gar nicht vorhanden ist.

Entspricht ein Schallereignis nicht den Anforderungen eines Klanges, so wird es als Geräusch bezeichnet. Ein Geräusch, dessen Namen bereits auf Rauschen hindeutet, ist somit ein chaotisches Schallereignis, dem keine Tonhöhe zugeordnet werden kann.

4.2 Harmonie

Melodie und Rhythmus sind Begriffe, unter denen sich die meisten konkret etwas vorstellen können. Die Definition von Harmonie scheint hingegen weitgehend unbekannt zu sein. Dies obwohl der Begriff Harmonie, also Einklang, an sich bereits stark auf seine Bedeutung hinweist. Die Harmonik beschäftigt sich nämlich in seiner reinsten Form mit dem Zusammenspiel mehrerer Klänge zu einem einzigen Zeitpunkt. So stellt die Harmonie in gewisser Weise das Gegenstück zum Rhythmus dar, welcher die zeitliche Komponente eines Musikstückes beschreibt. Aus diesem Grund wird die Harmonik auch als *vertikale Komponente der*

Musik und die Rhythmik als *horizontale Komponente der Musik* bezeichnet.

4.3 Klangfarbe

Im Gegensatz zur Harmonik, welche das Zusammenspiel mehrerer Klänge beschreibt, beschränkt sich die Klangfarbe auf die Komponenten eines einzelnen Klanges. Die Eigenschaften der Komponenten geben einem Klang die Charakteristik, die Farbe, und lassen somit Klänge der gleichen Höhe verschieden klingen. Somit ist es also hauptsächlich die Klangfarbe, die es ermöglicht, Instrumente voneinander zu unterscheiden. Dem entsprechend sind es die Materialien, sowie die Bauform von Gegenständen, die einen Einfluss auf die Klangfarbe haben.

Betrachtet man das Spektrum eines Klanges, so sind es das Vorhandensein (bzw. Fehlen), sowie die Amplituden der einzelnen Harmonischen, welche substanziellen Einfluss auf die Klangfarbe haben. Aber auch Unregelmässigkeiten eines Klanges beeinflussen diesen, wie zum Beispiel der Rauschanteil oder die Inharmonizität zwischen den Harmonischen, ohne die ein Klang unnatürlich wirkt.

Kapitel 5

Rhythmus-basierte Feature Extraction

5.1 Musiktheorie Begriffe

Um den Rhythmus-Begriff in der Musik besser verstehen zu können, werden einige Begriffe eingeführt, welche dem *The Oxford Dictionary of Music* [16] entlehnt sind. Teilweise wurde unser Verständnis der Begriffe durch die Dokumentationen zu *aubio* [3] und jAudio [5] ergänzt. **Für diesen Abschnitt dienen diese Quellen als Grundlage.**

Impuls Kann als gespielte Note oder als ungespielte Note (Pause) wahrgenommen werden.

Grundschlag[1] Wird durch gleichmässige Impulse im Zeitverlauf charakterisiert. Es beschreibt sich zeitlich regelmässig wiederholende Laute [2]. Eng verwandt mit dem Grundschlag ist der **Beat**[3]. Der Beat lässt sich in verschiedene Unterarten unterteilen [4]. Im allgemeinen Sprachgebrauch werden Beat und Grundschlag oft auch als Synonym verwendet. Gemäss Definition aus [16] gibt es jedoch einen subtilen Unterschied. Der Grundschlag gibt lediglich die Geschwindigkeit der einzelnen Impulse im Zeitverlauf wieder. Der Beat hingegen sagt aus, welcher Art diese Impulse sind. Das folgende Beispiel soll den Unterschied verdeutlichen. Der Dirigent gibt mit seinem Dirigentenstab durch jede Handbewegung den Grundschlag vor. Mit der Richtung der Handbewegung (beispielsweise von unten nach oben) beschreibt der Dirigent den Beat.

Abbildung 5.1: Grundschläge im Verlauf der Zeit schematisch dargestellt.

Takt[5] Bezeichnet eine Gruppierung von mehreren Grundschlägen. Dabei werden die Grundschläge durch bestimmte Notenwerte charakterisiert.

[1] *engl.* pulse
[2] Aus [16], Pulse
[3] *engl.* beat
[4] Downbeat, Upbeat, On-Beat und Off-Beat, Backbeat, Crossbeat
[5] *engl.* bar

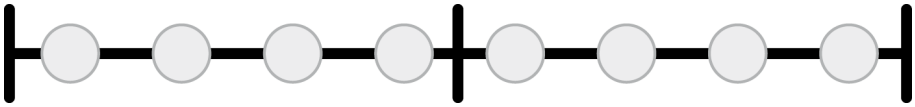


Abbildung 5.2: Der Takt unterteilt die Grundschläge mittels Taktstrichen in Gruppen.

Metrum[6] Beschreibt ein Muster, wie die einzelnen Grundschläge betont werden. Diese Muster folgt einer gewissen Gesetzmässigkeit innerhalb eines Rhythmus. Das Metrum wird allerdings oft mit dem Takt gleichgesetzt, da der Takt eher ein Notationselement ist.

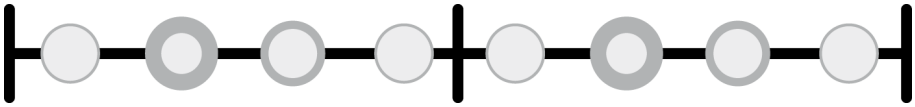


Abbildung 5.3: Das Metrum in diesem Bild wird wie folgt beschrieben: unbetont, stark betont, leicht betont, unbetont.

Rhythmus[7] Beschreibt nun die Struktur, die auf den vorgenannten Komponenten aufbaut. Er beschreibt die zeitliche Struktur der Töne - also konkret, wann ein Ton und wann eine Pause gespielt wird.

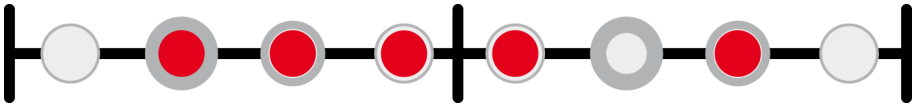


Abbildung 5.4: Rot markierte Grundschläge beschreiben gespielte Noten, während die übrigen Grundschläge Pausen sind.

Synkope Die Synkope ist ein musikalisches Stilelement, welches ein Beat um ein bestimmtes Mass vor oder nach seiner eigentlichen Position verschiebt [16]. Synkopen werden oft an Stellen eingesetzt, wo die schweren Beats keine Betonung erhalten. Dadurch kann zusätzliche rhythmische Spannung erzeugt werden ([24], S.143). Bei der Implementation von Algorithmen zur Erkennung von Rhythmen bedeutet das allerdings, dass man sich dieses Stilmittels bewusst sein muss und mit Unregelmässigkeiten im Erkennungsmuster rechnen muss.

Tempo und Beats Per Minute (BPM) Das Tempo beschreibt die Regularität des Grundschlags. Es beschreibt ein Mass für die Geschwindigkeit eines Musikstücks, also wie schnell ein Musikstück gespielt wird. Als Zählmass für das Tempo werden in der Regel die Anzahl Grundschläge oder Impulse pro Minute (*engl. Beats Per Minute, BPM*) angegeben.

[6] *engl.* meter
[7] *engl.* rhythm

5.2 Wissenschaftliche Arbeiten zur Rhythmus-Erkennung

In diesem Abschnitt werden verschiedene Verfahren möglichst kurz und ohne mathematische Details beschrieben. Für mehr Details zu den Berechnungen sei auf die entsprechenden Arbeiten verwiesen.

5.2.1 Inter-Onset Intervalle

In der Musiktheorie werden starke Beats, sogenannte *Downbeats*, als Zeitpunkte beschrieben, an welchen mehrere metrische Eigenschaften zusammenfallen und sich wiederholen [16]. In der Literatur werden verschiedene Ansätze verfolgt, um Onset[8] Eigenschaften zu analysieren.

- Brown 1993 [25]: Auftretensfrequenz von Noten ist höher bei starken metrischen Zeitpunkten. Aus daraus berechneten Zeitdifferenzen sollten Patterns abgeleitet werden können. Die Autokorrelationsfunktion wird verwendet, um Periodizitäten in den *Inter Onset Intervals* zu erkennen.
- Meudic 2002 [40]: basiert auf Brown und erweitert die Theorie auf weitere Eigenschaften. Es werden Onset Eigenschaften verwendet sowie Zeitindizes von Onsets, welche mit Beat-Indizes übereinstimmen.

5.2.2 Gouyon und Herrera: Klassifierung in Zweier- und Dreiertakt

Die beiden Autoren haben in Ihrer Arbeit [54] zum Ziel, das Metrum eines Musikstücks automatisiert zu bestimmen. Dabei reduzieren Sie das Modell soweit, dass Sie nur in zwei Metrum-Klassen einteilen, nämlich in den Zweier- *(duple)* und den Dreiertakt *(triple)*. Sie konzentrieren sich ausschliesslich auf Beat Deskriptoren und vernachlässigen die *Angriffs Eigenschaften* [9]. Begründet wird diese Vereinfachung wie folgt. Liegt ein Musikstück in geschriebener Form als Note oder als MIDI vor, kann daraus das Metrum exakt abgelesen werden. Hat man hingegen ein Audiosignal, so kann, gemäss der Meinung der beiden Autoren, daraus nicht eindeutig und unumstritten das Metrum bestimmt werden. Es kann beispielsweise nicht eindeutig unterschieden werden zwischen Beats, die in Zweier-, Vierer- oder Achter-Gruppen eingeteilt wurden. Eine Einteilung in Zweier- und Dreiertakte ist aber eindeutig voneinander unterscheidbar. Nachfolgend werden die Kernpunkte des Verfahrens kurz beschrieben. Für eine genauere Beschreibung sei an dieser Stelle auf den Artikel [54] verwiesen.

[8] Frei übersetzt: Angriffszeitpunkt einer Musiknote.
[9] *engl.* onset features

Berechnung der Frame Deskriptoren Das Audiosignal wird in Frames von 20ms aufgeteilt und anschliessend werden auf diesen Frames die folgenden Low-Level Eigenschaften extrahiert:

- f1: Energie[10]
- f2: Spectral Flatness
- f3: Energie in der oberen Hälfte des ersten Bark-Bands (etwa 50-100 Hz)

Berechnung der Beat Segment Deskriptoren Beat-Grenzen werden Frame Indizes zugeordnet und für jedes Beat werden drei Regionen definiert:

- R0: Gesamtes Beat-Segment, neu-zentriert um den Beat-Index
- R1: Die 120ms Region um den Beat-Index
- R2: Der Rest des Beat-Segments (also $R_0 \cap \overline{R1}$)

Vier Beat Deskriptoren werden wie folgt definiert:

- Standardabweichung von f1 über R0
- Durchschnitt von f2 und f3 über R1
- *Temporal Centroid* über R0

Diese Deskriptoren werden normalisiert, indem der Durchschnitt abgezogen und durch die Standardabweichung geteilt wird. Jeder musikalische Auszug[11] wird durch vier Zeitsequenzen repräsentiert, deren Länge mit der Anzahl Beats im jeweiligen Auszug übereinstimmen.

Ermittlung der Periodizität Um Periodizitäten zu ermitteln, wird die normalisierte Autokorrelation für jede Sequenz berechnet. Peaks in einer Deskriptor-Autokorrelation weisen auf Verzögerungen hin, für welche dieser Deskriptor Wiederholungen innerhalb der Sequenz aufzeigt.

Berechnung der entscheidungstragenden Eigenschaften Die Autoren definieren ein Kriterium M für die Entscheidungsfindung, ob ein musikalischer Auszug ein Zweiklang oder ein Dreiklang ist. Die Definition besagt, dass je positiver M ist, desto eher repräsentiert es einen Zweiklang und je negativer M ist, desto eher repräsentiert es ein Dreiklang. Für jeden Deskriptor existiert ein solches M, welches eine reelle Zahl ist.

Klassifikation Wie bereits erwähnt, werden musikalische Auszüge durch vier Eigenschaften beschrieben, nämlich das M Kriterium für die vier Beat Deskriptoren. Um die Klassifizierung zu machen, wenden die Autoren eine Pattern-Erkennung mittels *Discriminant Analysis* an.

[10] *engl.* Energy
[11] *engl.* musical excerpt

Ergebnisse und weitere Arbeit

Die Autoren haben in ihren Versuchen festgestellt, dass mithilfe des *Temporal Centroids* gute Klassifizierungsergebnisse erzielt werden können. Um relevante Eigenschaften auswählen zu können, muss eine repräsentative Eigenschaft für jeden Cluster ausgewählt werden. Mit einem dem k-NN Algorithmus ähnlichen Verfahren wurden dabei gute Ergebnisse erzielt, weil damit und mit den M Kriterium die Dimensionen der Eigenschaften reduziert werden können und somit auch die Cluster-Berechnungszeiten verkürzt werden können. Sie weisen allerdings darauf hin, dass die Reduktion der Eigenschaften nicht zu stark ausgenutzt werden sollte.

Dieses Verfahren von Gouyon und Herrera dient dazu, verschiedene Beat Gruppierungen zu erkennen und auszuwählen. Es dient aber nicht dazu, zu bestimmen, welches der Beats in einer Gruppe, das erste ist (wie beispielsweise welches das „eins“ bei „eins-zwei-drei-eins-zwei-drei-...“ ist). Für die zukünftige Weiterarbeit zählen die Autoren zwei Punkte auf. Einerseits müsste die Datenbank mit Audio-Material viel grösser sein, um die Allgemeingültigkeit und Skalierbarkeit des Algorithmus untermauern zu können. Andererseits kann die Definition vom Kriterium M noch weiter ausgereift werden.

5.2.3 Oliveira et. al.: Echtzeit Tempo und Beat Tracking System

Die Autoren präsentieren in ihrer Arbeit [34] ein System, welches in Echtzeit das Tempo und den Beat eines Musikstücks ermittelt. Echtzeit bedeutet dabei, dass die Musikstücke nicht auf Windows aufgeteilt werden und darauf dann die Werte berechnet werden, sondern die Autoren verfolgen hier einen Ansatz, der seine Ursprünge im *BeatRoot* System [29] hat.

In *BeatRoot* verarbeiten mehrere sogenannte parallel-laufende *Agenten* ein Musikstück-Input sequentiell und vergleichen sogenannte Hypothesen bezüglich Tempo und Beats - nach dem Motto „der bessere/wahrscheinlichere gewinnt“. Die Autoren erweitern diese Strategie, indem sie einen kausalen Entscheidungsprozess einbauen. Begründet wird dieser Ansatz unter anderem mit dem geringeren Rechenaufwand und mit der Unstetigkeit der Windows zueinander. Des Weiteren soll eine höhere Robustheit gegenüber rauschbehaftetem Input erreicht werden. Nachfolgend werden die Kernpunkte des Verfahrens kurz beschrieben. Für eine genauere Beschreibung sei an dieser Stelle auf den Artikel [34] verwiesen.

Extraktion der Audio Eigenschaften Es wird mit der musikalischen Eigenschaft *Spectral Flux* gearbeitet. Um die Onset Detection Function zu glätten und Fehlerkennungen zu reduzieren, wird auf die extrahierten Werte einen Butterworth Tiefpassfilter angewendet.

Pre-Tracking Als Vorverarbeitungsschritt wird das System mit einem *Induction Window* mit einer Länge von fünf Sekunden initialisiert. Während der Pre-Tracking Phase (auch Induktionsphase genannt) werden keine Ergebnisse (Beat/Tempo) ausgegeben, sondern an deren Ende werden die getroffenen Hypothesen zu Perioden, Phasen und Partitur ans Beat Tracking Modul übergeben. Die dreiteilige Induktionsphase ist wie folgt aufgebaut:

Period Hypotheses Induction Als erstes wird basierend auf der Autokorrelation von Spectral Flux eine fortlaufende Periodizitätsfunktion berechnet.

Phase Hypotheses Selection Für jede Periodenhypothese wird eine isochrone Beat-Sequenz mit konstanter Periodenlänge für jede mögliche Phase, welche die gleichen Länge wie das *Induction Window* hat, ermittelt.

Agents Setup Die originale Partitur wird aufgeteilt und den entsprechenden Perioden-Phasen-Hypothesen übergeben. Dabei wird darauf geachtet, dass die Partitur der Summe der Zeitabweichungen zwischen den Trainingstemplates der Beat-Elemente und den lokalen Maxima im *Spectral Flux* entspricht. Anschliessend werden die Partituren entsprechend den metrischen Beziehungen zwischen jedem Perioden-Hypothesenpaar aktualisiert.

Beat Tracking Die in der vorherigen Phase berechneten Hypothesen werden nun dazu verwendet, die Beat Agenten für das Beat Tracking zu initialisieren. Das Beat Tracking findet in Echtzeit statt. Dabei werden die eingelesenen *Spectral Flux* Werte dazu verwendet, Tempo- und Zeitabweichungen laufend zu überwachen.

Agenten Ausführung Jeder der in der vorangegangenen Phase initialisierten Beat Agenten wird Voraussagen aufgrund der Input-Daten machen und propagieren. Dabei wird er bezüglich Beat-Positionen und Tempo alternative Hypothesen aufstellen. Anschliessend wird jede Hypothese unter Berücksichtigung der Abweichung (also Fehler) gegenüber dem lokalen Maximum im betrachteten Daten-Frame ausgewertet. Die Auswertung geschieht dabei in einem zweistufigen Toleranzbereich. Die beiden Toleranzbereiche sind aufgeteilt in einen inneren Bereich, der kurze Perioden- und Phasen Abweichungen bearbeitet, und einen äusseren (asymmetrischen) Bereich, der plötzliche Tempo-Veränderungen erkennen kann (zum Beispiel, wenn ein Musikstück abrupt verlangsamt wird). Abhängig davon wo Abweichungen gefunden werden, werden entweder die Periode und Phase eines Agenten korrigiert oder der Agent generiert drei untergeordnete Agenten, um weitere Hypothesen zu berechnen. Deshalb wird dies auch als *kausaler* Ansatz bezeichnet. Die Agenten werden terminiert, wenn bestimmte Kriterien eintreffen.

Hauptagent Ein Hauptagent überprüft fortlaufend die Daten aller Agenten, um den besten Agenten pro Daten-Frame ermitteln zu können.

Nicht-Kausale Version Während der kausale Ansatz in jedem Zeit-Frame die Beats vom *aktuell* besten Agent ermittelt, wird im nicht-kausalen Ansatz nur der *letzte* beste Agent berücksichtigt. Im Gegensatz zur kausalen Version wird hier nach dem Pre-Tracking und dem Agent Setup die Analyse wieder vom Anfang des Musikstücks gemacht.

Ergebnisse und weitere Arbeit

Die Autoren schreiben, dass Ihr Verfahren entweder das Tempo richtig ermittelt, oder andernfalls eine Fehlerquote von 80% Fehler auf metrischer Stufe hat. Des Weiteren erzielt die nicht-kausale Version leicht schlechtere Ergebnisse. Des Weiteren scheint das Verfahren einerseits nicht zu stark von der korrekten Tempo Abschätzung abzuhängen und andererseits die Fähigkeit zu haben, sich von Fehlern erholen zu können. Verglichen mit anderen Verfahren kommen die Autoren zum Schluss, dass ihr Verfahren ziemlich gute Ergebnisse lie-

fert. Sie sehen allerdings noch Verbesserungspotential im Induktionsprozess. Ansatzpunkte könnten sein:

- Untersuchungen über die benötigte Datenmenge anstellen, die für den Induktionsprozess benötigt wird.
- Die Zuverlässigkeit der Schätzungen verbessern.
- Die Robustheit gegenüber Rauschen erhöhen.
- Den Induktionsprozess nicht nur vom Beginn des Audio-Inputs an, sondern von beliebigen Teilstücken der Audio-Daten aus zu starten.

5.3 Tempo Extraktion mit *aubio*

Im Abschnitt 2.4 wurde *aubio* bereits kurz vorgestellt. Aufgrund von Tests wurde festgestellt, dass das Beat Tracking ziemlich gute Ergebnisse liefert, weshalb beschlossen wurde, es für diese Arbeit zu verwenden. Wie aber bereits erwähnt, ist *aubio* in C geschrieben, weshalb die benötigten Klassen erst in Java umgeschrieben werden mussten. Nachfolgend soll nun auf die Funktionsweise des Tempo Extraktionsvorgangs eingegangen werden.

Zu Beginn wird das Musikstück in *Windows* fixer Grösse aufgeteilt. Aufgrund der fixen Grösse der Windows wird die eigentliche Struktur eines Musikstücks nicht berücksichtigt. Dadurch kann es vorkommen, dass das Musikstück an „ungünstigsten“ Stellen, wie zum Beispiel mitten in einer sogenannten *Angriffsphase*[12], aufgeteilt wird, wodurch eine Onset Detection Function (ODF) in diesem Fall die Angriffsphase nicht erkennen würde. Um diese Problematik zu umgehen, kommen zu den Windows sogenannte *Offsets* oder *Hops* zum Einsatz, welche bewirken, dass sich die Windows nun überlagern. In der Literatur [54, 50, 47] trifft man die Konfiguration oft so an, wie es in Abbildung 5.5 dargestellt wird. Die Offset-Grösse wird so gewählt, dass sie die Hälfte oder ein Viertel der Window-Länge ausmacht.

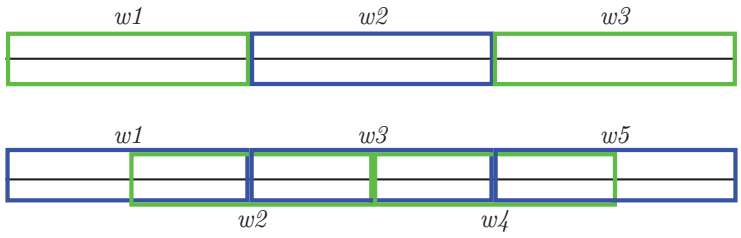


Abbildung 5.5: Vergleich der Segmentierung mit und ohne Offsets.

Der Beat Tracking Prozess startet mit einem Phase Vocoder, der das Audiosignal in seine Frequenzbestandteile zerlegt. Dabei wird das Signal vorgängig noch in Frequenz und Zeit

[12] Ein Ton lässt sich in vier Phasen aufteilen: Angriff, Dämpfung, Sustain, Ausklingen *(engl. attack, decay, sustain, release)*.

skaliert. Die Skalierung bzw. Gewichtung, wie es *aubio* nennt, wird mittels einer *Hanning*-Fensterfunktion erreicht. Der Phase Vocoder behält immer ein „Lookback“ zum vorhergehenden Window, welches für die Verarbeitung des aktuellen Windows miteinbezogen wird. Dies wird vermutlich gemacht, um die Ergebnisse zwischen benachbarten Windows zu glätten. Das Spektrum aus dem Phase Vocoder wird nun einer ODF übergeben, dessen Ergebnis später einer Peak Picking Funktion übergeben wird, welche den Angriffspunkt ermittelt.

Unsere Versuche haben gezeigt, dass je nach Genre und je nach verwendeter ODF beim Endergebnis unterschiedlich gute Tempo-Angaben erzielt werden. Die Ergebnisse des *Peak Pickers* werden dem Beat-Tracker übergeben, welcher mittels Autokorrelation und verschiedener Schwellenwerte die Anzahl Beats Per Minute [BPM] berechnet.

Die Software *Sonic Visualiser*, welche ebenfalls open source ist, integriert mittels Plugin die *aubio* Library. Mit dieser Software wurde mittels qualitativen Versuchen die Leistungsfähigkeit von *aubio* anhand ausgewählter Musikstücke untersucht. Dabei wurde festgestellt, dass bereits mit den Standard-Einstellungen je nach Musikstück bei der Beat-Erkennung relativ gute Ergebnisse erzielt werden. Enthält ein Musikstück klare und einfache Muster in Bezug auf Perkussionsinstrumente, so ist die Beat-Erkennungsrate sehr hoch und sehr zuverlässig. Fehlen allerdings Perkussionsinstrumente oder enthält das Musikstück viele wechselnde oder sich überlagernde Muster, so leidet die Qualität der erkannten Beats. Dies wurde insbesondere bei der getesteten klassischen Musik festgestellt.

5.3.1 Zielsetzung mit *aubio*

Aufgrund der vorwiegend guten Experimentergebnisse wurde beschlossen, *aubio* genauer zu analysieren. Da Mixtape Java-basiert ist, mussten erst die benötigten C-Dateien in Java umgeschrieben werden. Dabei mussten insbesondere auf auf zwei Dinge geachtet werden. Einerseits enthält C vorzeichenbehaftete und vorzeichenlose primitive Datentypen, wohingegen Java nur vorzeichenbehaftete Datentypen kennt. Andererseits mussten Zeiger und Referenzen bei der Übersetzung korrekt dereferenziert werden, was aufgrund der Komplexität und der Kopplungen der Klassen nicht immer ganz einfach war.

Um das Problem der vorzeichenlosen Datentypen zu lösen, standen zwei Optionen zur Verfügung:

- Datentypen mit dem doppelten Wertebereich verwenden (beispielsweise *long* anstatt *int*) und dabei darauf achten, dass keine negativen Werte erlaubt sind.
- Datentypen mit gleichem Wertumfang arbeiten (für einen *unsigned int* aus C einen *int* in Java verwenden) und mit dem Risiko eines Überlaufs rechnen.

Nach einer näheren Untersuchung des verwendeten C-Codes wurde beschlossen, die letztere Variante zu verwenden. Einerseits sollen dadurch die Ressourcen des ohnehin schon ausgelasteten System gespart werden. Ausserdem wurden im C-Quellcode auch keine Bereiche ausgemacht, wo ein Überlauf-Problem auftreten könnte.

Da *aubio* bereits relativ stabile Ergebnisse erzielt, stellte sich die Frage, ob und wie dieses Ergebnis noch verbessert werden könnte. Ansatzpunkte für eine mögliche Verbesserung sind folgende:

- Der grundlegende Algorithmus für die Tempo- bzw. Beat-Extraktion verbessern, sei es durch Beschleunigung oder Modifikation des Verfahrens an sich oder durch modifizieren von fixierten Parametern und Grenzwerten.

- Weitere *Onset Detection Functions* implementieren und deren Auswirkung auf die Ergebnisse vergleichen.

Für diese Arbeit wurde beschlossen, vor allem den ersten Ansatzpunkt zu verfolgen, da zum einen *aubio* bereits einige ODF anbietet. Zum anderen fehlten bei den recherchierten ODF auch oft wichtige Details, um die Funktionen anhand des jeweiligen Papers [36, 26] implementieren zu können.

5.3.2 Beschreibung der Verfahrensmodifikation

Standardmässig verwendet *aubio* jeweils nur eine einzelne ODF für die Tempo-Extraktion. Wie zuvor erwähnt wurde, liefert dieses Verfahren mit einer ODF bereits gute Ergebnisse. Deshalb stellte sich nun die Frage, wie es sich verhalten würde, wenn die Ergebnisse mehrerer Tempo-Extraktionen, welche mit verschiedenen ODF durchgeführt werden, kombiniert werden. Daher wurde beschlossen, das Tempo-Extraktionsverfahren mehrfach durchzuführen, jedoch jeweils mit einer anderen ODF. Da die einzelnen Extraktionsvorgänge voneinander unabhängig sind, bietet es sich hier optimal an, die Berechnungen nach Möglichkeit parallelisiert durchzuführen.

An dieser Stelle wird darauf hingewiesen, dass während dem Extraktionsprozess Schwankungen innerhalb des „theoretisch“[13] gleichen Tempos bei den extrahierten Beat-Werte auftreten können. Einerseits können diese Unregelmässigkeiten durch Synkopen (siehe Abschnitt 5.1) entstehen, welche vom Beat Tracking nicht erkannt werden. Andererseits können diese Schwankungen auch durch mathematische Ungenauigkeiten im Algorithmus durch oder Fehler im Byte-Code eines Musikstücks verursacht werden.

Nach der Extraktion der Ergebnisse müssen diese zusammengeführt werden. Hierfür stehen verschiedene Möglichkeiten zur Verfügung. Im Folgenden sollen einige Verfahren erläutert werden.

5.3.3 Ergebnisauswertung

Ergebnisauswertung mittels statistischen Funktionen

Eine erster möglichst einfacher Versuch die Ergebnisse zusammenzuführen, wurde mit der Anwendung von statistischen Methoden unternommen. Dabei wurde die Aussagekraft der beiden folgenden Funktionen untersucht.

- Mittelwert mit Standardabweichung: dieser doch sehr simple Ansatz liefert bereits relativ gute Ergebnisse. Zusammen mit der Standardabweichung hat man dann auch eine gute Aussage, wie zuverlässig der Mittelwert ist.

- Median: der Vorteil dieser Funktion liegt theoretisch in der höheren Stabilität gegenüber Ausreissern im Vergleich zum Mittelwert. In unseren Tests haben wir allerdings festgestellt, dass der Median nur in den seltensten Fällen signifikant andere Werte liefert als der Mittelwert.

Beide Funktionen haben allerdings einen entscheidenden Nachteil. Sie fassen die extrahierten Beat-Werte zu einem einzigen Wert zusammen. Besteht ein Musikstück nun aber aus

[13] Damit ist gemeint, dass der Zuhörer kein Unterschied im Grundtakt des Musikstücks wahrnimmt.

mehreren verschiedenen Tempi, so würde diese Information verloren gehen. Aus diesem Grund kommen diese Funktionen für die Ergebnisauswertung nicht in Frage.

Ergebnisauswertung mittels Clustering

Ausgangslage Bei diesem Auswertungsverfahren wird davon ausgegangen, dass jeder extrahierte Beat, über alle ODF hinweg betrachtet, grundsätzlich gleichberechtigt ist. *aubio* bietet die Eigenschaft *Confidence* an, welche vermutlich eine Aussage über die Zuverlässigkeit bzw. Richtigkeit der extrahierten Beats machen soll. Aufgrund der fehlenden Dokumentation in *aubio* zu dieser Eigenschaft wissen wir das aber nicht mit Sicherheit.

Für den Umgang mit der *Confidence* muss beachtet werden, dass je nach verwendeter ODF die Werte sehr unterschiedlich sind. Die Tatsache, dass die *Confidence* einen Null-Wert enthält, wenn der Beat Tracking Algorithmus das Tempo für ein Window nicht eindeutig bestimmen kann, stellt ebenfalls einen Nachteil dar. Enthält ein Musikstück nämlich eine längere Pause, so lässt sich mit diesem Algorithmus nicht unterscheiden, ob die Null-Werte wegen der fehlenden *Confidence* auftreten, oder ob es tatsächlich Pausen sind. Dieses Verhalten ist unserer Ansicht nach jedoch im Widerspruch zur eigentlichen Aufgabe der *Confidence*, so wie wir sie verstanden haben.

Eine sinnvollere Implementation der *Confidence* wäre unserer Ansicht nach, wenn die *Confidence* Werte ungleich Null zurückliefert, auch wenn der dazu extrahierte Beat ein Null-Wert ist. Schliesslich sollten auch Pausen zuverlässig erkannt und eine Zuverlässigkeitsaussage darüber gemacht werden.

Trotz den genannten Nachteile wurde beschlossen, nicht vollständig auf den Einsatz der *Confidence* zu verzichten, da sie doch eine Möglichkeit zur Gewichtung der Beats bietet. Allerdings wurden in der Anfangsphase in Versuchen die Werte genau untersucht, um mit den verschiedenen Wertebereiche und deren Auswirkungen vertraut zu werden.

Beschreibung des Auswertungsverfahrens Als erstes werden die Ergebnisse $E_{1...n}$ aus den einzelnen ODF ganzzahlig gerundet und zu einer Gesamtergebnismenge E_{Gesamt} zusammengefügt. Die Rundung der Beat-Werte wird damit begründet, dass der Zuhörer nicht zwischen Tempo Werten, die sich nur in den Nachkommastellen unterscheiden, differenzieren kann. Die Werte in E_{Gesamt} werden nun in einen Clustering Algorithmus übergegeben. Da es bei diesem Verfahren darum geht, herauszufinden welche Tempi in einem Musikstück vorhanden sind, ist bei der Wahl des Clustering Algorithmus zu beachten, dass beim Initialisieren des Clusterings nicht angeben werden muss, wie viele Cluster zu bilden sind. Aus diesem Grund wurde der *Expectation Maximization* Algorithmus für das Clustering ausgewählt. Die Anzahl zu bildender Cluster wird zur Laufzeit aufgrund eines festgelegten Grenzwertes ermittelt. In unserem Fall wird der Grenzwert durch die Standardabweichung beschrieben. Das bedeutet, dass Werte innerhalb eines Clusters maximal um den Betrag der Standardabweichung vom Clustermittelpunkt entfernt sein dürfen. Mittels qualitativen Versuchen wurde einen Wert von fünf BPM als Grenzwert ermittelt und festgesetzt. Im Eigenexperiment hat sich gezeigt, dass mit diesem Grenzwert kaum Unterschiede in den Tempi wahrgenommen werden können und die Musikstücke somit als ähnlich eingestuft werden können. Dieser Grenzwert stellt allerdings eindeutig eine subjektive Komponente dar und hängt mitunter von den musikalischen Kenntnissen des Zuhörers und dessen

auditivem Wahrnehmungsvermögen [14] ab.

Wie in Abschnitt 5.3.2 bereits erwähnt wurde, können während dem Beat Tracking Schwankungen auftreten. Die Auswertung mittels Clustering bietet den Vorteil, dass diese Ausreisser keine spürbare Auswirkung auf das Ergebnis haben.

Die Daten (sogenannte Instanzen), die dem Clustering übergeben werden, können entsprechend ihrer Relevanz gewichtet werden. Es wurden zwei verschiedene Ansätze getestet:

Verfahren ohne Gewichtung der Instanzen Mit diesem Ansatz sollen alle Beat-Werte gleichberechtigt behandelt werden. Da der eingesetzte Clustering Algorithmus jedoch eine Gewichtung der Instanzen voraussetzt, werden hier alle Instanzen mit dem Wert 1 gewichtet.

Verfahren mit Gewichtung der Instanzen Bei diesem Ansatz wurden die berechneten *Confidences* für die Gewichtung der Instanzen eingesetzt. Wie bereits erwähnt, sind die Wertebereiche der *Confidences* sehr unterschiedlich, weshalb sie vorgängig normalisiert werden, so dass deren Wertebereich zwischen 0 und 1 zu liegen kommt.

Aufgrund der Tatsache, dass die normalisierten Werte der *Confidences* meist in der Nähe von 1 liegen, liefern beide Ansätze vergleichbare Resultate. Da der Ansatz mit Gewichtung der Instanzen minim bessere Resultate liefert und weil er durch die *Confidence*-abhängige Gewichtung etwas weniger willkürlich erschient, wurde dieser Ansatz implementiert.

Das Clustering wird mit Weka 3 durchgeführt. Durch diese Abhängigkeit ist es nicht möglich, das Verfahren sauber in die Architektur von Mixtape zu integrieren. Dies führt dazu, dass das Clustering nicht parallel zum Tempo Extraktionsprozess durchgeführt werden kann. Dadurch können die laufend extrahierten Beat-Werte pro Window nicht nach und nach dem Clustering hinzugefügt werden, sondern der gesamte Clustering Prozess kann erst starten, nachdem alle Windows extrahiert worden sind. Dies führt zu einem massiven Leistungseinbruch im gesamten Analyseprozess.

5.3.4 Ergebnisauswertung mittels Kompressionsdistanz

In diesem Verfahren werden die extrahierten Beat-Werte pro Window betrachtet. Die Tempo Extraktion der Windows kann parallelisiert durchgeführt werden. Sobald alle Werte der einzelnen Tempo Extraktionen mit den verschiedenen ODF vorliegen, werden die Werte mit den jeweiligen *Confidences* gewichtet und auf 5 BPM gerundet.

$$\mathrm{B}eat\mathrm{W}indow_i = \mathrm{R}ound\left(\sum_{k=1}^{Count(\mathrm{ODF})} \frac{Tempo_k * \frac{Confidence_k}{Max(Confidence[k])}}{\sum_{m=1}^{Count(\mathrm{ODF})} Confidences_i} * 0.2\right) * 5$$

Der Vorteil dieses Verfahrens gegenüber dem Clustering Ansatz ist der, dass die Berechnungsschritte nach der Tempo Extraktion viel weniger Ressourcen benötigen und somit die Analyse massiv beschleunigt werden kann.

[14] Mit dem auditiven Wahrnehmungsvermögen ist nicht das Hören an sich gemeint, sondern vielmehr die Fähigkeit, das Gehörte zu interpretieren. Im oben beschriebenen Fall wäre das die Fähigkeit des Zuhörers, Tempounterschiede in Musiksignalen zu erkennen und zu klassifizieren.

5.3.5 Ergebnisauswertung mittels Gauss-Glocken-Funktion

Dieses Verfahren wird nur in der Theorie erläutert und wurde aus Zeitgründen nicht implementiert. Dadurch ist auch nicht sichergestellt, dass dieses Verfahren überhaupt brauchbare Ergebnisse liefert. Der Vollständigkeit halber soll es aber dennoch beschrieben werden.

Wie in den vorherigen Verfahren beschrieben wurde, werden auch hier die Tempi über alle Windows und alle ODF extrahiert. Ebenso werden die *Confidences* auf Werte zwischen 0 und 1 normalisiert. Nun werden die extrahierten Werte in ein Koordinatensystem mit einem Wertebereich zwischen 0 und 200 auf der x-Achse und zwischen 0 und 1 auf der y-Achse eingetragen. Dabei wird die x-Koordinate durch den Beat-Wert bestimmt und die y-Koordinate durch die *Confidence*. Auf der *Confidence* wird allerdings noch eine Gauss-Glocken-Funktion angewendet, deren Zentrum in der x-Koordinate des aktuellen Beat-Wertes liegt. Dadurch erhalten Beats, die häufiger vorkommen, auf der x-Koordinate höhere Werte, als solche, die selten oder nie vorkommen.

Am Ende stehen zwei Möglichkeiten zur Verfügung. Zum einen stellen die eingetragenen Werte im Koordinatensystem eine Funktion dar, auf welche eine Peak-Detection angewendet werden kann, um absolute Beat-Werte zu erhalten und diese für die Ähnlichkeitsanalyse heranzuziehen. Müssen hingegen keine absoluten Beat-Werte bekannt sein, könnte auch das bestimmte Integral eingesetzt werden. Bekanntlich kann das Integral grafisch als Fläche unterhalb einer Funktion interpretiert werden. Werden nun zwei Musikstücke miteinander verglichen, ist die Ähnlichkeit umso grösser, umso weniger sich die beiden Flächen unterscheiden.

Kapitel 6

Spektrale Features

Spektrale Features beschreiben die Form und Eigenschaften eines Signals. Sie werden fensterbasiert, also auf kurze Ausschnitte des Signals, berechnet. So kann der zeitliche Verlauf der Eigenschaften analysiert und verglichen werden. Da diese Eigenschaften einen starken Bezug zu den Klangfarben haben, werden sie in [30, 31] für die Identifikation und Klassifikation von Musikinstrumenten verwendet.

Ein wichtiges Features ist dabei der *Spectral Centroid* [52]. Beim *Spectral Centroid* wird berechnet, wo das Zentrum des Spektrums liegt. Dadurch kann eine Aussage darüber gemacht werden, wo die dominierenden Frequenzen konzentriert sind. Ein hoher Wert bedeutet also, dass mehr hohe Frequenzen (Oberton Anteil) vorhanden sind. Diese Eigenschaft wird auch als „Helligkeit“ eines Musikstücks bezeichnet und hat einen starken Bezug zu den Klangfarben (Timbre).

Eine Variation davon ist das *Median of Sepctrum*. Ergebnisse von [57] haben jedoch gezeigt dass für die Klangfarben das *Spectral Centroid* besser geeignet ist.

Eine weiteres wichtiges Feature ist die *Spectral Skewness*. Sie zeigt auf, wie symmetrisch die Energie um den *Spectral Centroid* verteilt ist. In [30] wird sie eingesetzt, um zu bestimmen ob das Spektrum höhenlastig oder basslastig ist.

Um die Umgebung um den *Spectral Centroid* noch genauer zu analysieren, wird auch gerne die *Spectral Kurtosis* berechnet wie zum Beispiel in den Arbeiten [52]. Diese beschreibt, wie steil oder flach das Spektrum um den *Spectral Centroid* abfällt.

Spektrale Features werden in fast allen untersuchten Audioanalyseverfahren verwendet, da sie einfach zu extrahierende Eigenschaften sind und weil damit schon viele Erfahrungen gesammelt wurden.

6.1 Wahrnehmung - Mel Frequency Cepstrum

Das Mel Frequency Cepstrum (MFC) ist eine Repräsentation für das Energiespektrum eines Tons in einem kurzen Zeitraum [37]. Die MFCC, welche dieses Spektrum charakterisieren, werden vor allem im Bereich der Spracherkennung eingesetzt, wo schon sehr gute Resultate erzielt wurden [39] und deshalb auch sehr gerne für die Musikanalyse benutzt werden. Die Frequenzen werden dabei dem menschlichen Gehör entsprechend angepasst. Dies geschieht mit der Transformation ins Mel-Spektrum, welche das Ohr abbilden soll. Frequenzen, welche das menschliche Ohr nicht mehr direkt bestimmen kann (> 1 KHz), werden approximiert. Dabei werden höhere Frequenzen (höhere Töne) nicht mehr linear, son-

dern logarithmisch als höher empfunden. Durch das Simulieren dieser Eigenschaft mit den Mel-Bändern können Aussagen darüber gemacht werden, wie ein Audiosignal wirklich vom Menschen wahrgenommen wird.

Einige Arbeiten bauen nur auf den MFCCs auf, wie zum Beispiel in [41, 56]. Die Berechnung der MFCCs geschieht grob auf folgende Weise:

1. Zerlege das Signal in einzelne Frames
2. Finde das Amplituden-Spektrum
3. Logarithmieren
4. Ins Mel - Spektrum konvertieren
5. Wende die Diskrete Kosinustransformation an (DCT)

Die Diskrete Kosinustransformation transformiert ähnlich wie die diskreten Fouriertransformation ein zeitdiskretes Signal in den Frequenzbereich.

Kapitel 7

Harmonische Eigenschaften

7.1 Grundton

Die Erkennung des Grundtons bildet die Basis der Extraktion aller weiteren harmonischen Eigenschaften. Hinzu kommt, dass wir die Grundtöne als entscheidenden Faktor in Bezug auf die Ähnlichkeitsanalyse betrachten. Diese Annahme basiert auf den herausragenden Erfolgen von [45], dessen Ähnlichkeitsanalyse ebenfalls auf der *NCD* aufbaut. Hierbei ist zu beachten, dass für den erwähnten Ansatz keine Extraktion von Eigenschaften nötig ist, da *MIDI*-Files als Input verwendet wurde. *MIDI*-Files wiederum beschreiben in erster Linie die zu spielenden Grundtöne. Aus diesen Gründen wurde dem entsprechend viel Zeit und Aufwand in die Erkennung des Grundtons eines Klanges investiert.

Zur Zeit existieren bereits einige etablierte Ansätze zur Erkennung des Grundtons, wie zum Beispiel die *Autokorrelation* oder das *Harmonic Product Spectrum*, welche in dieser Arbeit genauer untersucht wurden [43]. Zwar lieferten diese Ansätze äusserst solide Ergebnisse für die Analyse einzelner Klänge, erklangen jedoch mehrere gleichzeitig, konnte keiner dieser Ansätze überzeugen. Inspiriert durch die Arbeit von [51] zur Erkennung mehrerer Grundtöne, wurde schlussendlich ein eigener Algorithmus umgesetzt. Dieser sucht ebenfalls ebenfalls nach der bestmöglichen Kombination von Grundtonkandidaten, unterscheidet sich jedoch völlig in der Wahl dieser Kandidaten.

Da in unserem Fall aufgrund der Ähnlichkeitsanalyse sowieso eine Quantisierung der Ergebnisse nötig gewesen wäre, konnte dies bereits in die Umsetzung des Algorithmus miteinbezogen werden. Als Quantisierungsschritte wurden die Frequenzen der üblichen 88 Klaviertasten verwendet, welche sogleich als Kandidaten für mögliche Grundtöne dienen. Dies ermöglicht, dass auf eine aufwändige Suche nach möglichen Grundtonkandidaten im Frequenzspektrum verzichtet werden konnte.

In einem weiteren Schritt wird nun pro definiertem Kandidat jeweils ein Prototyp seines Spektrums erzeugt. Hierfür wird an denjenigen Stellen, an denen eine Harmonische des Grundtonkandidaten auftreten kann, eine Gausssche Glockenkurve eingesetzt, die die Verteilung der Frequenzen um den jeweiligen Punkt möglichst gut modellieren soll.

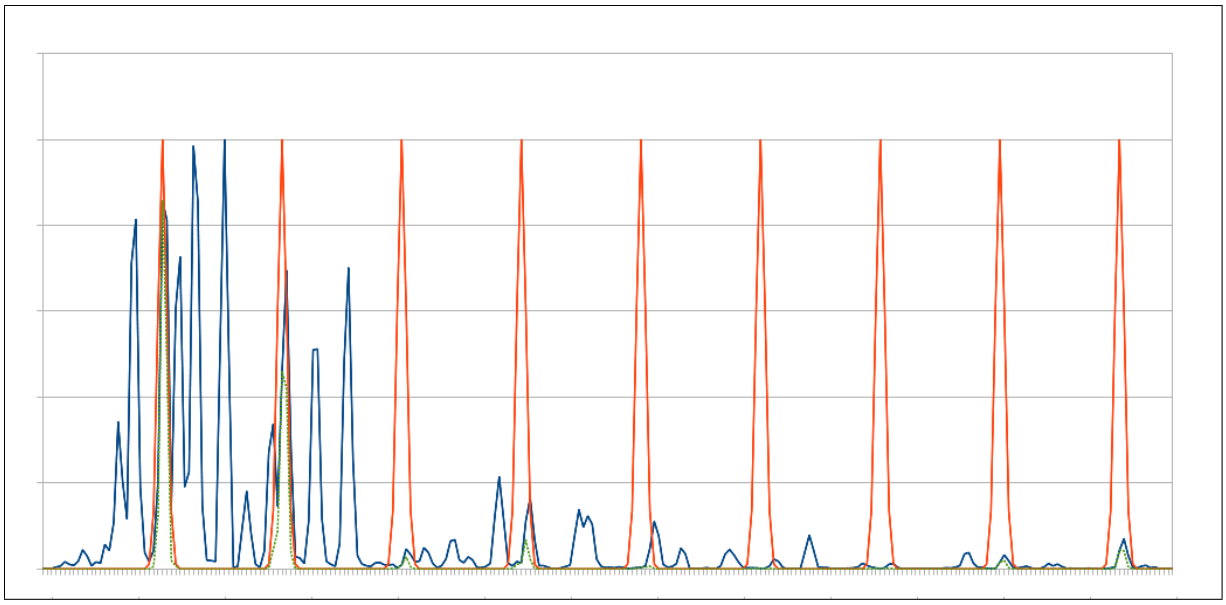


Abbildung 7.1: Beispielhaftes Frequenzspektrum mit Prototype und deren Multiplikation

Zur Erkennung eines Grundtons werden die Prototypen p_i mit dem gegeben Spektrum s verglichen. Daraus resultiert nun die Fitness F_i des jeweiligen Kandidaten.

$$F_i = \frac{\sum_{x=0} s(x)p_i(x)}{\sum_{x=0} p_i(x)}$$

Hierbei entspricht $\sum_{x=0} p_i(x)$ dem Potenzial des Prototyps. Dieses Potenzial muss in die Berechnung miteinbezogen werden, da Grundtonkandidaten, die tiefere Grundtöne repräsentieren, über mehr Harmonische verfügen und somit stets eine höhere Fitness als höhere Kandidaten mit weniger Harmonischen erreichen würden.

Nach Abschluss dieser Berechnungen wird nun der Kandidat mit der höchsten Fitness als Grundton gewählt. Ausgehend von dessen Prototyp wird nun die Fitness der Kombination mit den übrigen Prototypen berechnet. Die Kombination $p_{i,j}$ zweier Prototypen p_i und p_j wird wie folgt erreicht:

$$p_{i,j}(x) = max(p_i(x), p_j(x)).$$

Die resultierenden Prototypen werden nun wiederum auf ihre Fitness überprüft. Derjenige Grundtonkandidat, dessen Kombination mit dem zuvor erkannten Grundton die grösste Steigerung der Fitness verursacht, wird ebenfalls zu den erkannten Grundtönen hinzugefügt. Dieser Vorgang wird nun solange wiederholt, bis die Abbruchbedingung erfüllt ist. Als Abbruchbedingung wird die Steigerung der Fitness im Verhältnis zur zuvor berechneten Fitness berücksichtig. Sinkt dieses Verhältnis unter einen bestimmten Schwellwert, so wird nicht nach weiteren Grundtönen gesucht. Als Schwellwert wird zurzeit 0.05 verwendet, was einer benötigten Steigerung von 5% entspricht.

7.2 Harmonische

Sobald der Grundton bekannt ist, kann die Erkennung der Harmonischen recht einfach durgeführt werden. Bekanntlich sind die Harmonischen jeweils ganzzahlige Vielfache des Grundtons, womit die zu erwartenden Positionen im Spektrum bereits vorgegeben sind. Durch Inharmonizität kann hierbei allerdings nicht von perfekten ganzzahligen Vielfachen ausgegangen werden. Aus diesem Grund wird jeweils in einem kleinen Intervall um die erwartete Position einer Harmonischen nach der höchsten Amplitude gesucht. Dieser Punkt entspricht dann der tatsächlichen Position der Harmonischen. Wurde kein Maximum gefunden (bzw. befindet sich das Maximum auf den Grenzen des Abweichungsintervalls), so wird davon ausgegangen, dass die gesuchte Harmonische fehlt, da zumindest ihre Amplitude zu klein für eine Erkennung ist.

7.3 Inharmonizität

Wie bereits in den Grundlagen zur Klangfarbe erwähnt wurde, spielt die Inharmonizität der Harmonischen eine Rolle im Bezug auf die Charakteristiken eines Klanges. Die Inharmonizität kann somit genutzt werden, um Klänge und Instrumente zu identifizieren.

Die Berechnung der Inharmonizität gemäss [48] baut auf den Harmonischen auf und entspricht grundsätzlich einem Mass für die Abweichung der Harmonischen von den jeweiligen Idealpositionen.

$$Inharmonicity = \frac{2}{f_0} \frac{\sum\limits_{h} | f(h) - h f_0 | a^2(h)}{\sum\limits_{h} a^2(h)}$$

7.4 Energieverhältnis zwischen geraden und ungeraden Harmonischen

Die Form des Spektrums eines Klanges hat Einfluss auf dessen Klangfarbe. Obwohl grundsätzlich beliebige Formen möglich sind, so gibt es doch Eigenschaften, welche auf eine gewisse Klasse von Klängen hindeuten kann. Das Energieverhältnis zwischen geraden und ungeraden Harmonischen gehört zu diesen Eigenschaften und basiert auf der Beobachtung, dass das Verhältnis zwischen der Amplituden aller geraden und aller ungeraden Harmonischen bei ähnlichen Instrumenten ebenfalls ähnlich ist.

Berechnet wird dieses Verhältnis wie folgt:

$$\mathrm{R} = \frac{\sum\limits_{o=1}^{\mathrm{O}} a_o^2}{\sum\limits_{e=0}^{\mathrm{E}} a_e^2}$$

Wobei O die Menge der ungeraden und E die Menge der geraden Harmonischen bezeichnet.

7.5 Tristimulus

Der Tristimulus gehört eher zu den Exoten unter den musikalischen Eigenschaften. Seine Ursprünge hat der Tristimulus in der Farbenlehre, in welcher er genutzt wird, um die Mischung dreier Hauptfarben zu einer beliebigen Farbe zu beschreiben. Analog dazu beschreibt der Tristimulus in der Musik die Mischung der Harmonischen zu einer Klangfarbe.

Der Tristimulus ist also ein 3-Tupel, welches die Verteilung der Harmonischen, und somit die Klangfarbe, beschreibt. *T1* ist das Verhältnis der ersten Harmonischen zu allen Harmonischen.

$$T1 = \frac{a_1}{\sum_{h=1}^{\mathrm{H}} a_h}$$

T2 ist das Verhältnis der zweiten, dritten und vierten Harmonischen zu allen Harmonischen.

$$T2 = \frac{a_2 + a_3 + a_4}{\sum_{h=1}^{\mathrm{H}} a_h}$$

T2 ist das Verhältnis der restlichen Harmonischen zu allen Harmonischen.

$$T3 = \frac{\sum_{h=5}^{\mathrm{H}} a_h}{\sum_{h=1}^{\mathrm{H}} a_h}$$

Kapitel 8

Distanzberechnung

Motiviert durch die guten Resultate, welche in [56, 53] erzielt wurden, haben wir die Distanzberechnung mittels Kompression genauer betrachtet. Besonders interessant dabei ist, dass die Kompressionsdistanz kein Wissen über die Daten benötigt, welche sie vergleicht. Dies macht sie sehr universell einsetzbar.

8.1 Kompressionsdistanz

Die Kompressionsdistanz basiert stark auf der Kolmogorov-Komplexität einer Zeichenkette. Deshalb wird hier etwas genauer darauf eingegangen.

Bei der Kolmogorov-Komplexität wird unterschieden zwischen der Komplexität einer Zeichenkette x und der bedingten Komplexität einer Zeichenkette x relativ zu einer Zeichenkette y.

Dabei ist die Komplexität K(x) einer Zeichenkette x die Länge des kürzesten binären Programms x*, welches x auf einem universellen Computer, wie zum Beispiel einer Turing Maschine, ausgeben kann. Die bedingte Kolmogorov-Komplexität K(x|y) ist die Länge des kürzesten Programms welches x mit y als Input berechnet.

Daraus lässt sich die Information I(y:x) über x, welche sich in y befindet, herleiten als

$$\mathrm{I}(y:x) = \mathrm{K}(x) - \mathrm{K}(x|y*) \tag{I}$$

Interessanterweise zeigen Resultate in [32], dass es eine Konstante c ≥ 0 gibt (unabhängig von x und y), welche besagt, dass innerhalb dieser Konstanten

$$\mathrm{K}(x,y) = \mathrm{K}(x) + \mathrm{K}(y|x*) = \mathrm{K}(y) + \mathrm{K}(x|y*) \tag{II}$$

woraus folgt, dass innerhalb eines Terms c

$$\mathrm{I}(x:y) = \mathrm{I}(y:x) \tag{III}$$

gilt.

Die Notation K(x,y) aus (II) wird dabei benutzt für die Länge des kürzesten Programms welches x und y ausgibt.

Die Teilaussage K(x,y) = K(x) + K(y|x*) aus (II) lässt sich aus der Kettenregel der Kolmogorov-Komplexität herleiten, welche eine Analogie zur Kettenregel der Informations-Entropie darstellt:

$$\mathrm{H}(x, y) = \mathrm{H}(x) + \mathrm{H}(y|x) \tag{IV}$$

Sie besagt, dass die kombinierte Zufälligkeit von zwei Sequenzen die Summe der Zufälligkeit der einen Sequenz zusammen mit der übrigbleibenden Zufälligkeit der anderen Sequenz ist. Mit diesen Eigenschaften ist es möglich die Informationsdistanz zwischen zwei Objekten zu berechnen, indem man die geteilte Information vergleicht.

Formell gesehen ist die Informationsdistanz die Länge des kürzesten binären Programm E(x,y) welches x von y berechnet und umgekehrt. Resultate in [33] zeigen dass die Informationsdistanz bis auf einen zusätzlichen Term O(log max$\big(\mathrm{K}(y|x), \mathrm{K}(x|y)\big)$

$$\mathrm{E}(x, y) = max\left(\mathrm{K}(y|x), \mathrm{K}(x|y)\right) \tag{V}$$

ist.

Diese Distanz muss jedoch noch normalisiert werden um verschiedene Objekte vernünftig miteinander zu vergleichen. Dazu werden in [42] verschiedene Möglichkeit aufgezeigt, wie eine NID basierend auf der Kolmogorov-Komplexität berechnet werden kann. Ein erster Ansatz besagt, wenn zwei Sequenzen x und y gegeben sind, definiere die Funktion d(x,y) mit

$$d(x, y) = \frac{(\mathrm{K}(x|y) + \mathrm{K}(y|x))}{\mathrm{K}(x, y)}. \tag{VI}$$

Der Bruch kann mit (I) und (II) umgeschrieben werden zu

$$d(x, y) = \frac{(\mathrm{K}(y) + \mathrm{K}(x|y)) - ((\mathrm{K}y) - \mathrm{K}(y|x)))}{\mathrm{K}(y) + \mathrm{K}(x|y)},$$

woraus

$$d(x, y) = 1 - \frac{\mathrm{I}(x : y)}{\mathrm{K}(x, y)} \tag{VII}$$

folgt.

I(x:y) wird dabei als arithmetische Transinformation ("gegenseitige Information") bezeichnet. Dies, weil in (III) gezeigt wurde, dass bis zu einem konstanten Term gilt:

$$\mathrm{I}(x : y) = \mathrm{I}(y : x)$$

Eine präzisere Möglichkeit, welche auch in in [42] beschrieben wurde, besagt für zwei gegebene Sequenzen x und y

$$d(x,y) = \frac{\max\big(\mathrm{K}(x|y*), \mathrm{K}(y|x*)\big)}{\max\big(\mathrm{K}(x), \mathrm{K}(y)\big)} \tag{VIII}$$

was wie folgt interpretiert werden kann: Falls K(y) > K(x), dann gilt

$$d(x,y) = \frac{\mathrm{K}(y) - \mathrm{I}(x:y)}{\mathrm{K}(y)} = 1 - \frac{\mathrm{I}(x:y)}{\mathrm{K}(y)} \tag{IX}$$

Mit den vorgestellten Formeln kann eine normalisierte Distanz zwischen [0,1] berechnet werden, welche die Anforderungen an eine Metrik erfüllen [42].

Da die Kolmogorov-Komplexität nicht berechenbar ist, muss ein Weg gefunden werden um diese zu approximieren. Dafür bietet es sich an, einen Kompressor C zu verwenden und die Länge der komprimierten Zeichenkette als dessen Komplexität zu werten. Die bedingte Komplexität K(x|y) kann mit einem Standardkompressor jedoch nicht berechnet werden. Da bis auf eine logarithmischen Präzision K(x,y) = K(yx) gilt, verwenden die Authoren von [42] eine Annäherung basierend auf (II), welche besagt dass gilt:

$$\mathrm{K}c(x|y) = \mathrm{K}c(xy) - \mathrm{K}c(y). \tag{X}$$

Kc(xy) ist dabei die Kompression über den zusammengesetzten String xy. Aus dieser Eigenschaft und mit Hilfe von (III) leiten die Authoren die Normalized Compression Distance (NCD)

$$\mathrm{NCD}(x,y) = \frac{\mathrm{C}(xy) - \min\big(\mathrm{C}(x), \mathrm{C}(y)\big)}{\max\big(\mathrm{C}(x), \mathrm{C}(y)\big)} \tag{XI}$$

ab.

Damit kann die Distanz mittels Standardkompressoren berechnet werden. Jedoch muss man bei Standardkompressoren einige Sachen beachten, um gute Resultate zu erhalten. Bei verschiedenen Tests [38]] mit den Kompressionsalgorithmen *gzip* und *bzip2* zeigte sich, dass die Resultate sehr gut sind, solange die Zeichenkette innerhalb der vorgesehenen Fenstergrösse der Algorithmen bleiben. *bzip2* arbeitet mit Blockgrössen von 900 Kilobytes (100 Kilobytes mit *fast* Option). Übersteigt die Objektgrösse die Blockgrösse, so wird das Objekt aufgeteilt. Dadurch wird die Distanz stark verfälscht und unbrauchbar. Bei *gzip* entsteht ein Problem mit der *sliding Windowsize*, welche 32 Kilobytes beträgt. Wird diese grösse Überschritten wird auch hier die Distanz so stark verfälscht, dass sie unbrauchbar wird. Als Alternative wird PPMZ getestet, welche eine unendliche Windowsize erlaubt und durchwegs sehr gute Resultate liefert.

Wird also ein Standardkompressor verwendet, muss bei der Berechnung von C(xy) (Kompression über beide Objekte) unbedingt darauf geachtet werden, dass die Windowsize nicht überschritten wird.

Weiter muss in Betracht gezogen werden, dass bei den meisten Standardkompressoren noch weiter optimiert wird, was die eigentliche Komplexität der Zeichenkette verfälschen

kann. Zudem wäre es schön, die bedingte Kolmogorov-Komplexität K(x|y) besser zu approximieren, als mit der in (IX) gezeigten Methode.

Um auf diese Probleme eingehen zu können, haben wir den Lempel-Ziv 77 Algorithmus genauer betrachtet.

8.1.1 LZ77 basierende Distanzberechnung

Der LZ77 Algorithmus benutzt ein gleitendes Fenster konstanter Grösse (sliding Window) als Vorschaupuffer. Der Inhalt des Vorschaupuffers ist dabei mit dem Wörterbuch zu komprimieren, wobei sich im Wörterbuch bereits komprimierte Zeichenketten befinden. Der komprimierte Teil wird dann ins Wörterbuch aufgenommen.

Komprimierte Zeichenketten werden als Tripel ausgegeben. Ein solches Tripel baut sich wie folgt auf:

(Index im Wörterbuch, Länge der zu komprimierenden Daten, nachfolgendes)

Das nachfolgende Zeichen ist nötig, um Zeichen zu komprimieren, welche sich nicht im Wörterbuch befinden. Diese werden mit Position und Länge 0 angegeben, wudurch nur das nachfolgende Zeichen eingefügt wird. Also ein Tripel der Form (0,0,<zeichen>).

Ein Beispiel mit dem Wort “blablub“, wobei die erste Kolonne das Wörterbuch ist, die zweite der Vorschaupuffer und die dritte der Output Tripel:

0	1	2	3	4	5	6	7			
								blablub	(0,0,b)	kein Match
							b	lablub	(0,0,l)	kein Match
						b	l	ablub	(0,0,a)	kein Match
					b	l	a	blub	(5,2,u)	2 Matches ab Pos 5
		b	l	a	b	l	u	b	(1,1,-)	1 Match ab Pos 1
	b	l	a	b	l	u	b		(-,-,-)	Vorschaupuffer leer

Abbildung 8.1: Lempel-Ziv 77 Kompression an einem Beispiel

Solche Tripel werden dann mit einer bestimmten Bitzahl kodiert. Die einfachste Variante ist eine konstante Grösse für den Index und die Länge zu wählen. Dies könnte zum Beispiel 12 Bit für den Index und 4 Bit für die Länge sein. Dies ist auch abhängig davon, wie gross das Wörterbuch ist. Bei grösserem Wörterbuch sind grössere Indices nötig und längere Übereinstimmungen möglich, was mehr Bits beansprucht.

In der Realität wird jedoch meistens eine Huffmann Codierung über die Länge und die Indices gemacht, um die Kompression noch weiter zu optimieren.

Diese Optimierungen in der zweiten Phase und die festen Fenstergrössen, welche bei den Standardkompressoren zur Zeitoptimierung verwendet werden, sind nach unseren Recherchen problematisch. Sie können die reine Komplexität einer Zeichenkette stark verfäl-

schen. Deshalb haben wir uns entschlossen, für die Distanzberechnung eine eigene Variante des LZ77 Algorithmus zu verwenden.

8.1.2 Angepasster LZ77 Algorithmus zur Distanzberechnung

Wie bereits erwähnt wurde, lässt sich aus der Länge der komprimierten Zeichenkette deren Komplexität ableiten. Die Länge der Zeichenkette hängt wiederum von der Anzahl der Tripel ab welche benötigt werden, um die Zeichenkette zu kodieren, und mit wie viel Bit die Tripeleinträge kodiert werden.

Da die Indexangaben in den Tripel immer relativ zu den sich gerade im Wörterbuch befindenden Zeichen sind, sagen übereinstimmende Längen- und Indexangaben nichts über die Gleichheit der Tripel aus. Somit würde eine Huffmann Codierung über diese Werte die reine Komplexität der Zeichenkette verfälschen.

Deshalb behandeln wir die Tripel so, als würden die Einträge mit fester Bitlänge kodiert. Dies erlaubt uns die Komplexität direkt aus der Anzahl Tripel abzuleiten. Dadurch wird das tatsächliche Kodieren der Tripel obsolet.

Um nun so ein Tripel zu generieren, muss die bestmögliche Übereinstimmung im Wörterbuch gefunden werden. Dafür ist die Wahl einer geeignete Datenstruktur für das Wörterbuch essentiell. Vorgeschlagen werden zum Beispiel Binary Maps und Tries. Eine immer beliebtere Datenstruktur für das Pattern Matching sind Suffix Trees oder Suffix Arrays. Diese scheinen auch für unser Problem gut geeignet zu sein.

Ein Suffix Tree ist dabei ein Baum mit allen möglichen Suffixe einer Zeichenkette. Typischerweise sind die Äste lexikographisch sortiert.

Beispiel mit der Zeichenkette “blablub“:

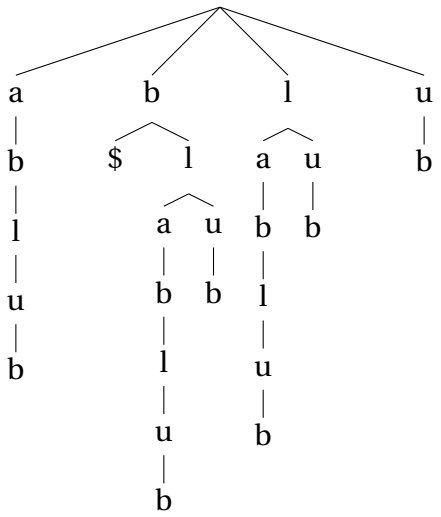


Abbildung 8.2: Suffix Tree für den String blablub

Aus diesem Baum lassen sich nun alle möglichen Suffixe einer Zeichenkette schnell finden, weshalb diese Struktur für das Pattern Matching sehr geeignet ist.

Ein Suffix Array ist grundsätzlich ein Suffix Tree in kompakter Form. In [55] wird gezeigt dass ein Suffix Array im Gegensatz zu einem Suffix Tree dreimal weniger Speicher benötigt. Wie der Suffix Tree kann auch das Suffix Array zum Beispiel mit dem Skew Algorithmus [55] in linearer Zeit aufgebaut werden.

Ein Suffix Array (SA) ist dabei ein lexikographisch sortiertes Array mit den Indices aller Suffixe einer Zeichenkette. Dieselbe Zeichenkette wie im Beispiel 8.2

index	0	1	2	3	4	5	6
values	b	l	a	b	l	u	b

hat folgenden Suffix Array

Suffix Array	2	6	0	3	1	4	5

Abbildung 8.3: Suffix Array des Strings blablub

Auf den ersten Wert eines Suffix S_i, welcher lexikographisch der i-kleinste ist, kann somit folgendermassen zugegriffen werden

$$S_i[0] = values[SA[i]];$$

der zweite Wert wäre dann

$$S_i[1] = values[SA[i] + 1];$$

usw...

Somit ist der Wert SA[0] im Suffix Array der Startindex des lexikographisch kleinesten Suffix. Im Suffix Tree entspricht dies dem ersten Ast (ganz links). Eine sehr beliebte Erweiterung des Suffix Arrays ist das Array mit den Longest Common Prefixes (LCP). Auf den Suffix Tree bezogen ist dies die Anzahl der gemeinsamen Vorfahren. Wichtig ist, dass sich das LCP Array auf das Suffix Array bezieht.

Betrachten wir dazu nochmals das Suffix Array aus Beispiel 8.3

Suffix Array	2	6	0	3	1	4	5

Das LCP Array dazu sieht folgendermassen aus

LCP	0	0	1	2	0	1	0

Es besagt, wie viel gemeinsame Vorfahren (Zeichen) zwei nachfolgende Indices im Suffix Array haben. Der *Longest Common Prefix* des i-ten Index im Suffix Array mit dem i+1-ten Index ist also

$$\mathrm{LCP}(\mathrm{SA}[i], \mathrm{SA}[i+1]) = \mathrm{LCP}[i+1]$$

Zusammen mit dem LCP Array kann so aus einem Suffix Array einfach und in linearer Zeit ein Suffix Tree aufgebaut werden.

Aufgrund des minimierten Speicherverbrauchs und der einfachen Datenstruktur im Vergleich zu einem Suffix Tree haben wir uns für die Verwendung eines Suffix Arrays entschieden, um das Pattern Matching im Wörterbuch durchzuführen.

Um nun die Anzahl Tripel herauszufinden, gehen wir etwas anders vor als dies im LZ77 Algorithmus geschieht. Normalerweise wird, wie beim Abschnitt über LZ77 gezeigt, der Inhalt des Vorschaupuffers mit dem Wörterbuch komprimiert, welches schon komprimierte Zeichenketten beinhaltet. Danach wird der komprimierte Teil ins Wörterbuch aufgenommen. Ein solches Vorgehen wird auch in [22] für Suffix Arrays vorgeschlagen. Jedoch muss so nach jedem Kompressionsschritt das Suffix Array neu über das Wörterbuch aufgebaut werden. Dies führt vor allem bei wenig Redundanz zu hohen Performance-Einbussen. Deshalb haben wir eine andere Möglichkeit gesucht.

Anstatt für die Kompression die beste Übereinstimmung in bereits komprimierten Zeichen zu suchen, suchen wir die beste Übereinstimmung in noch kommenden Zeichen. Genauer betrachtet bedeutet dies, dass das Wörterbuch immer kleiner wird, im Gegensatz zum originalen Algorithmus, wo das Wörterbuch jeweils um den komprimierten Teil des Vorschaupuffers vergrössert wird. Wichtig ist bei dieser Methode, dass beachtet wird, dass eine Zeichenkette nicht mit sich selbst komprimiert werden darf.

Zur Erläuterung ein Beispiel mit der Zeichenkette S = “aaablablub“. Es wird die Schreibweise S[i] verwendet für den Suffix mit Startindex i in der Zeichenkette S.

Als erstes geht es darum, die längste Übereinstimmung zu finden. Theoretisch ist die längste Übereinstimmung des Suffix S[0] der Suffix S[1] da

$$\mathrm{S}[0] = aaablablub$$

$$\mathrm{S}[1] = aablablub$$

zwei Übereinstimmungen (matches) hat. Dies ist jedoch nicht korrekt, da sich die zwei Suffix überschneiden, wodurch das zweite Zeichen in S mit sich selbst komprimiert würde. Deshalb ist für den zu komprimierenden Suffix S[i] nur Suffix S[y] legitim, für welchen gilt

$$y \geq i + matches$$

Deshalb ist die längste Übereinstimmung nur eins.

Der nächste zu komprimierende Suffix ist gleich wie beim LZ77 Algorithmus, der Suffix an der Position S[oldPos + matches + 1]. Das +1 kommt eigentlich daher, dass in einem LZ77 Tripel auch noch das nachfolgende Zeichen dazukommt. Dies behalten wir bei, da sonst keine Übereinstimmung gleich gewertet würde wie eine Übereinstimmung von eins. Auf das Beispiel bezogen ist der nächste Suffix Index i = 0 + 1 + 1, also S[2]. Die beste Übereinstimmung ist dabei S[5] mit drei Übereinstimmungen. Also ist der nächste Suffix, welcher betrachtet werden muss, S[5]. Dieses Verfahren wird fortgeführt, bis alle Zeichen komprimiert wurden.

Jeder Kompressionsschritt hat dabei einen LZ77 Tripel als Output. Da wir, wie bereits erwähnt, die Tripel so behandeln, als würden sie mit einer festen Bitzahl kodiert, zählen wir einfach die benötigten Tripel. Daraus folgt dass

$$K(S) = T(S) \tag{XII}$$

wobei T(S) die Anzahl Tripel ist, welche benötigt werden, um S zu komprimieren.

Für die bedingte Komplexität $K(S_1|S_2)$ wird dasselbe Konzept verwendet. Mit derselben Methode wie bei der Berechnung der Komplexität K(S) wird in S_1 die beste Übereinstimmung gesucht. Zusätzlich wird noch in S_2 die beste Übereinstimmung gesucht. Jedoch mit dem Unterschied, dass jeweils komplett in S_2 gesucht werden kann, ohne die Restriktionen, welche bei der Selbstkompression beachtet werden müssen. Die Länge L_m der besseren Übereinstimmung

$$L_m = \max matches(S_1), matches(S_2) \tag{XIII}$$

wird zu einem Tripel. Wobei wie vorhin die Komplexität von der Anzahl Tripel abgeleitet wird

$$K(S_1|S_2) = T(S_1|S_2), \tag{XIV}$$

wobei $T(S_1|S_2)$ die Anzahl Tripel ist, welche benötigt werden, um S_1 zu komprimieren, mit S_2 als Input.

Damit leitet sich aus der vorgeschlagenen Distanzberechnung (VIII) mit Hilfe von (XII) und (XIV) folgende Formel ab

$$d(S_1, S_2) = \frac{\max T(S_1|S_2), T(S_2|S_1)}{\max T(S_1), T(S_2)} \tag{XV}$$

Diese wird nun für die Distanzberechnung eingesetzt.

Kapitel 9

Wiedergabelisten Generierung

9.1 Ausgangslage

Wird einer Wiedergabeliste ein Musikstück hinzugefügt, so geht es darum, einen möglichst optimalen Weg vom letzten Stück in der Wiedergabeliste zum hinzugefügten zu finden. Werden mehrere Musikstücke hinzugefügt, muss vom letzten Stück aus ein optimaler Weg gefunden werden, welcher alle gewünschten Stücke beinhaltet. Als Metrik für die Ähnlichkeit zweier Musikstücke wird die zuvor berechnete NID verwendet.

9.2 Was ist ein optimaler Weg

Die erste Frage, welche für die Wiedergabelistengenerierung beantwortet werden muss, ist, was im Sinne einer Wiedergabeliste einen optimalen Weg darstellt. Dafür wurde angenommen, dass ein optimaler Weg im Sinne einer Wiedergabeliste durch möglichst sanfte Übergänge zwischen aufeinanderfolgenden Musikstücken definiert ist. Die „Sanftheit“ eines Überganges wird dabei durch die berechnete NID zwischen zwei Stücken definiert. Der „sanfteste“ Pfad ist dabei der Pfad, welcher über alle Stücke in der Wiedergabeliste gesehen die kleinsten Übergänge hat.

9.3 Abbildung auf bekannte Probleme

Für die Wiedergabelisten Generierung wurden verschiedene Verfahren untersucht, auf welche das Wegfindungsproblem für gewünschte Musikstücke abgebildet werden könnte.

Ein erster Kandidat dafür war das *TSP*. Beim *TSP* geht es um das Problem, bei einer Anzahl von Städten den kürzesten Pfad zu finden, der alle Städte genau einmal besucht.

Es schien, als würde dies ziemlich genau auf das Problem passen, einen Weg zu finden welcher alle gewünschten Musikstücke beinhaltet. Jedoch gibt es einige Probleme, die es schwierig machen, das Wiedergabelisten Generierungsproblem auf das *TSP* abzubilden.

Ein Problem ist, dass die Wiedergabeliste nicht nur aus den gewünschten Stücken bestehen soll, sondern mit passenden Stücken aus der Musikbibliothek aufgefüllt wird. Da beim *TSP* jede Stadt besucht wird, kann nicht einfach die komplette Bibliothek benutzt werden. Es müsste eine Vorauswahl getroffen werden, um die zusätzlichen Stücke zu bestimmen, welche für das *TSP* benutzt werden sollen.

Bei dieser Vorauswahl stellt sich die schwierige Frage, wie viele Stücke ausgewählt werden sollen und aufgrund welcher Kriterien dies geschehen soll. Ein Ansatz, der in Betracht gezogen wurde, um die möglichen Kandidaten auszuwählen, war das Clustering Verfahren. Dazu wurde zu Testzwecken ein einfaches hierarchisches Clustering mit SLINK [46] implementiert. Dabei stellte sich sehr schnell heraus, dass es sehr schwierig ist, mit einem hierarchischen Clustering für alle möglichen Kombinationen von Musikwünschen gute Grenzwertkriterien zu finden, sodass aus den Resultaten ein vernünftiger Pfad im Sinne einer Wiedergabeliste generiert werden kann.

Eine Alternative wäre ein Verfahren mit überwachtem Lernalgorithmus für die Auswahl der zusätzlichen Musikstücke. Die Umsetzung einer solchen Lösung kombiniert mit dem *TSP* hätte jedoch den Umfang dieser Arbeit gesprengt. Deshalb musste eine einfachere Lösung gefunden werden.

Um das Problem mit mehreren gewünschten Musikstücken zu vereinfachen, haben wir die Wegfindung auf den Pfad zwischen zwei Musikstücken reduziert. Dies erreichen wir, indem wir die gewünschten Musikstücke im Vorhinein so sortieren, dass gilt

$$\forall S_i \in \mathbb{W}$$

$$NID(S_i, S_a) \leq NID(S_y, S_a) \forall y \in \mathbb{W} \backslash \mathbb{C} \tag{I}$$

wobei $\mathbb{W}$ die Menge aller Musikwünsche ist, S_a der Vorgänger des Musikstücks S_i und $\mathbb{C}$ die Menge von S_a und seinen Vorgängern ist.

Nun kann jeweils der Pfad zwischen S_a und S_i als einzelne Wiedergabeliste betrachtet werden.

Wie bereits erwähnt wurde, haben wir einen optimale Wiedergabeliste durch die „Sanftheit“ der Übergänge definiert. Nach dieser Definition ist der optimalste Pfad zwischen zwei gewünschten Musikstücken der Weg mit den jeweils kleinsten NID's zwischen aufeinanderfolgenden Stücken.

Dieses Problem lässt sich auch mit dem *Shortest Path Problem* vergleichen, wobei der kürzeste Pfad zwischen zwei gewünschten Musikstücken gesucht wird und die Übergangsdistanzen (NID) die Kantengewichte sind.

Bei genauerer Betrachtung des *Dijkstras Shortest Path* Algorithmus mussten wir jedoch feststellen, dass die NID als Kantengewichte nicht sehr geeignet ist, um einen optimalen Pfad im Sinne einer Wiedergabeliste zu generieren.

Zur Erläuterung ein Beispiel: Die NID zwischen zwei einzelnen Features ist immer zwischen 0 und 1. Nehmen wir nun an, dass sich alle ausgerechneten Distanzen zwischen 0.5 und 0.9 bewegen. Auf das Kantengewicht bezogen bedeutet das, dass jede Kante mindestens ein Gewicht von 0.5 und maximal ein Gewicht von 0.9 besitzt. Im Extremfall wäre dann bei einem Pfad vom gewünschten Musikstück A zum nächsten gewünschten Musikstück F die Distanz zwischen A und F gleich 0.9. Ein möglicher Graph mit den Musikwünschen A und F und den zusätzlichen Kandidaten B, C, D und E könnte dabei folgendermassen aussehen:

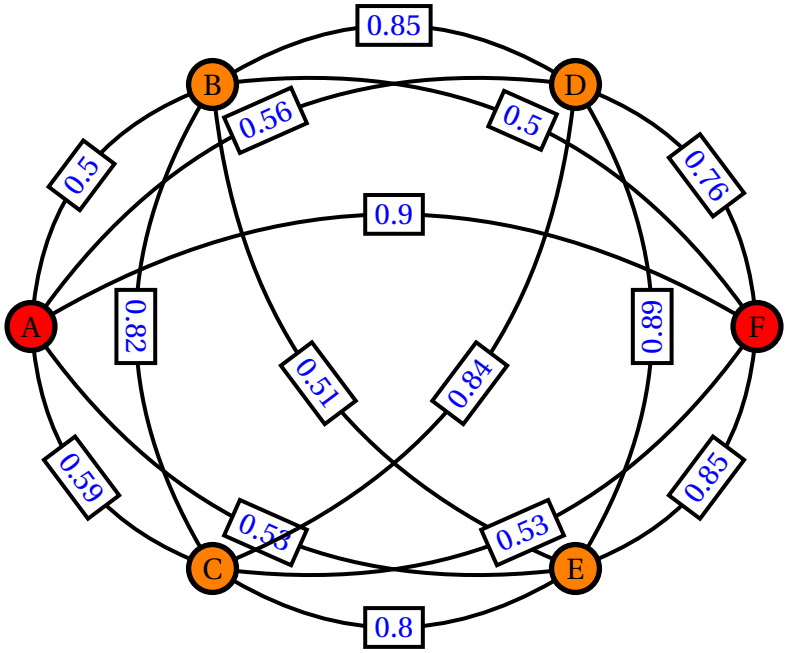


Abbildung 9.1: Musikwünsche (rot) und Musikstücke in Bibliothek (orange) mit Distanzen

Man kann erkennen, dass es keinen kürzeren Pfad A ⇒ F gibt als den direkten. Es ist aber schnell ersichtlich, dass dies nicht der „sanfteste“ und somit nicht der optimalste Pfad im Sinne einer Wiedergabeliste ist. Dies wäre der Pfad A ⇒ B ⇒ F: Dies, weil es der Pfad ist, mit den jeweils kleinsten Übergängen, wie am folgenden Graph zu sehen ist:

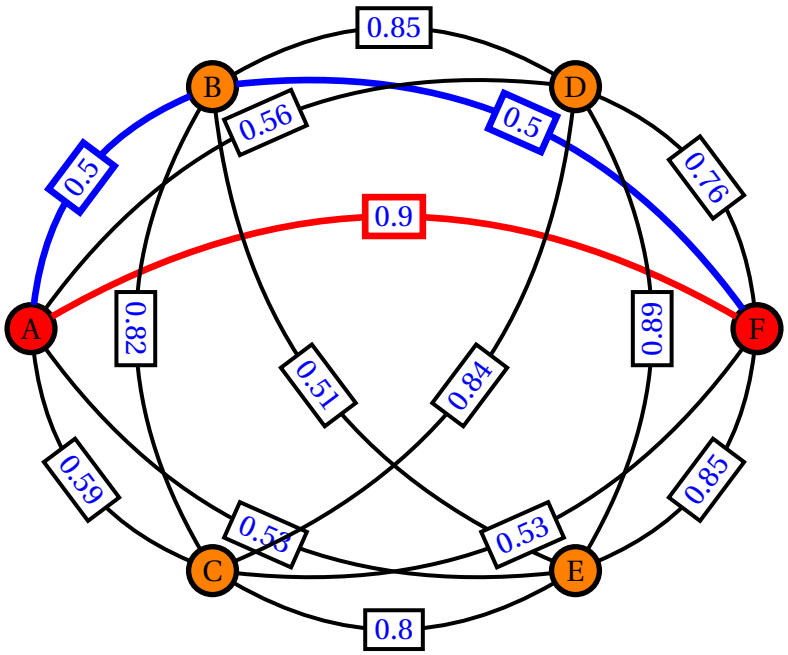


Abbildung 9.2: Shortest Path (rot) und Sanftester Pfad (blau)

Wobei der rote Pfad der Shortest Path und der blaue Pfad der „sanfteste Pfad“ ist.

Mit dieser Erkenntnis konnten wir das Verfahren an die Bedürfnisse eines Wiedergabelistenpfades anpassen.

9.4 Sanfter Pfad

Um für die Wiedergabeliste zwischen zwei gewünschten Musikstücken A und F die „sanften“ Übergänge zu finden, sollte ein Nachfolger jeweils das Musikstück mit der kleinsten Distanz sein. Ist A das erste Stück in der Wiedergabeliste und F das letzte, so sollte für die eingefügten Stücke gelten:

$$\forall S_c \in \mathbb{P}:$$

$$NID(S_c, S_a) \leq NID(S_d, S_a) \forall d \in \mathbb{B} \tag{II}$$

wobei $\mathbb{P}$ die Menge aller Musikstücke in der Wiedergabeliste ist, S_a der Vorgänger des sich in der Wiedergabeliste befindenden Musikstücks S_c und $\mathbb{B}$ die Menge aller Musikstücke in der Bibliothek.

Dabei muss jedoch beachtet werden, dass man sich bei der Auswahl von zusätzlichen Stücken aus der Bibliothek nicht weiter vom gewünschten Stück F, zu welchem man die Übergänge generieren möchte, entfernt. Zudem sollte die Möglichkeit gegeben sein, keine Duplikate in der Wiedergabeliste zuzulassen. Deshalb wird (II) zu

$$\forall S_c \in \mathbb{P}:$$

$$NID(S_c, S_F) \leq NID(S_a, S_F) \bigwedge NID(S_c, S_a) \leq NID(S_d, S_a) \forall d \in \mathbb{B} \backslash \mathbb{P} \tag{III}$$

erweitert, wobei S_F das gewünschte Stück F ist, zu welchem die Wiedergabeliste generiert werden soll. Wendet man diese Regel auf das vorherige Beispiel 9.2 an, so resultiert dabei der „sanfteste Pfad“.

Damit erhalten wir eine relativ simple „best effort“ Lösung, welche für die Pfadfindung umgesetzt werden kann.

9.5 Optimierung

Da dieser Algorithmus eine Laufzeit von $O(n * \frac{n-1}{2}) = O(n^2)$ hat, wurden nach Optimierungspotential gesucht.

Dafür wird die zusätzliche Regel in (III)

$$NID(S_c, S_F) \leq NID(S_a, S_F)$$

genauer betrachtet. Diese beinhaltet nämlich, dass nur Musikstücke in Betracht gezogen werden, welche eine kleinere Distanz haben als die Distanz zwischen den gewünschten Stücken. Sind also wieder *A* und *F* die gewünschten Stücke zwischen denen der Pfad gefunden werden soll, sind nur Kandidaten S_k aus der Bibliothek in Betracht zu ziehen, welche die Bedingung erfüllen

$$\mathrm{NID}(S_k, S_F) \leq \mathrm{NID}(S_A, S_F), \tag{IV}$$

wobei S_A das gewünschte Stück A ist.

Damit kann die Anzahl zu vergleichende Musikstücke im Vorhinein reduziert werden. Zur Veranschaulichung eine vereinfachte Grafik, in welcher die Musikstücke in der Bibliothek (schwarz) relativ zu den Distanzen zu A und F angeordnet sind:

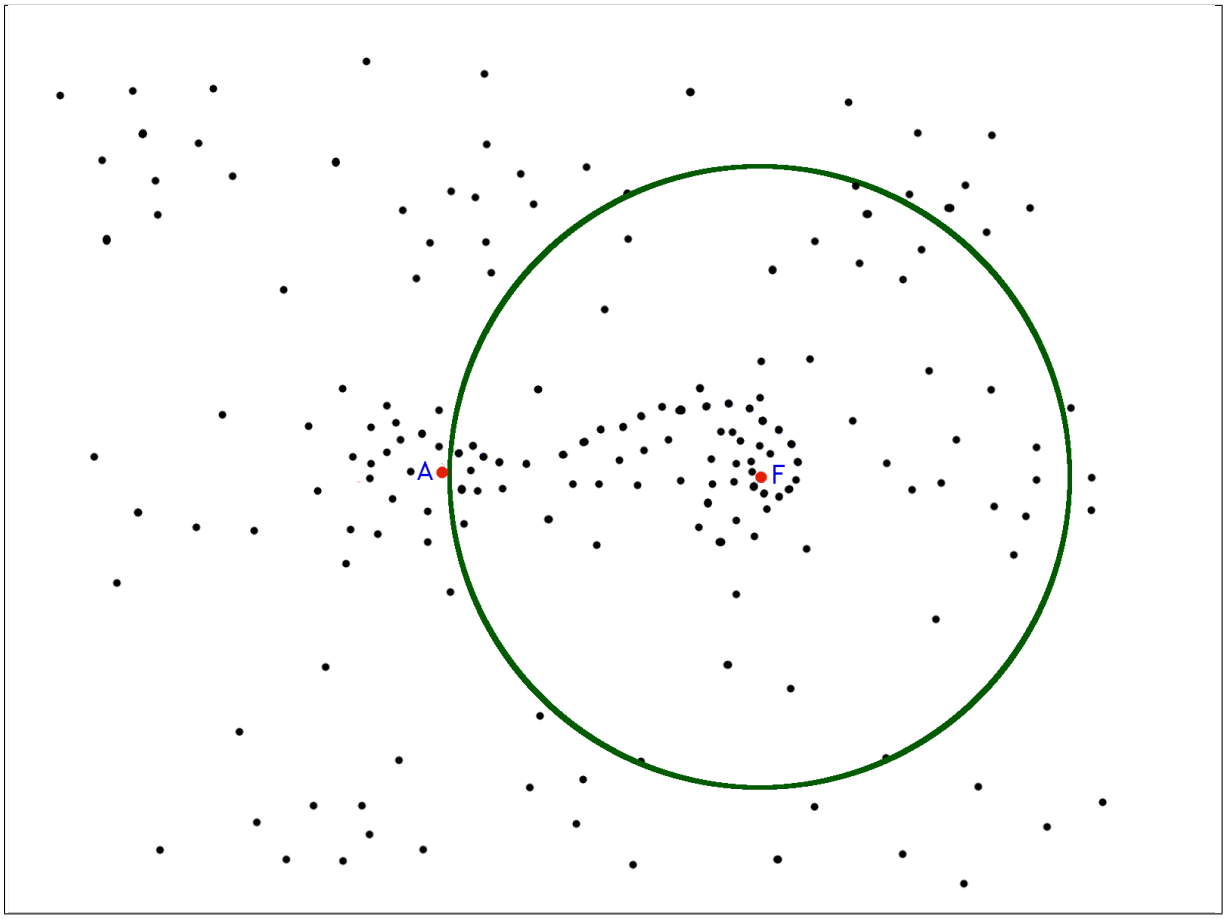


Abbildung 9.3: Vorauswahl der Kandidaten mit Regel (IV)

Betrachtet man sich die Landschaft um F, fällt eine starke Dichte von Musikstücken auf, welche sehr ähnliche Distanzen zu A und F aufweisen.

Wird nun der „sanfteste“ Pfad gesucht, nähert sich die Wiedergabeliste in Kreisen langsam F an, bis schliesslich kein Musikstück mehr vorhanden ist, welches eine kürzere Distanz zu F hat.

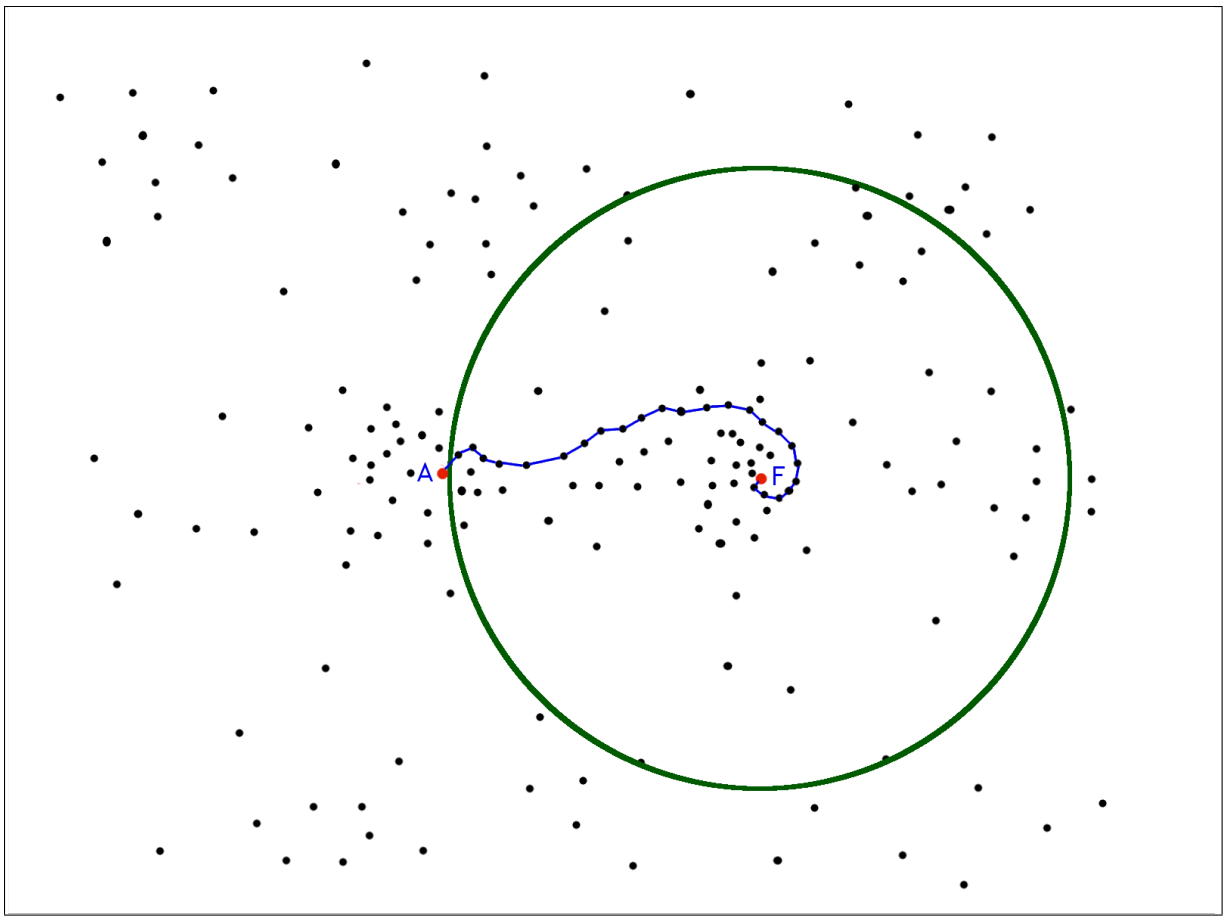


Abbildung 9.4: Wiedergabeliste mit Regel (IV)

Dieses Verhalten ist nicht unbedingt erwünscht, weil es so bei einer sehr grossen Bibliothek mit vielen ähnlichen Musikstücken unter Umständen sehr lange dauern kann, bis ein gewünschtes Stück drankommt. Das Phänomen verstärkt sich dabei zusätzlich mit zunehmender Distanz zwischen zwei gewünschten Stücken.

Wieder ausgehend vom vorherigen Beispiel wird deshalb Regel (IV) zur Vorauswahl der möglichen Kandidaten um die Bedingung

$$\mathrm{NID}(S_k, S_A) \leq \mathrm{NID}(S_A, S_F) \tag{V}$$

erweitert, wodurch folgendes Bild entsteht:

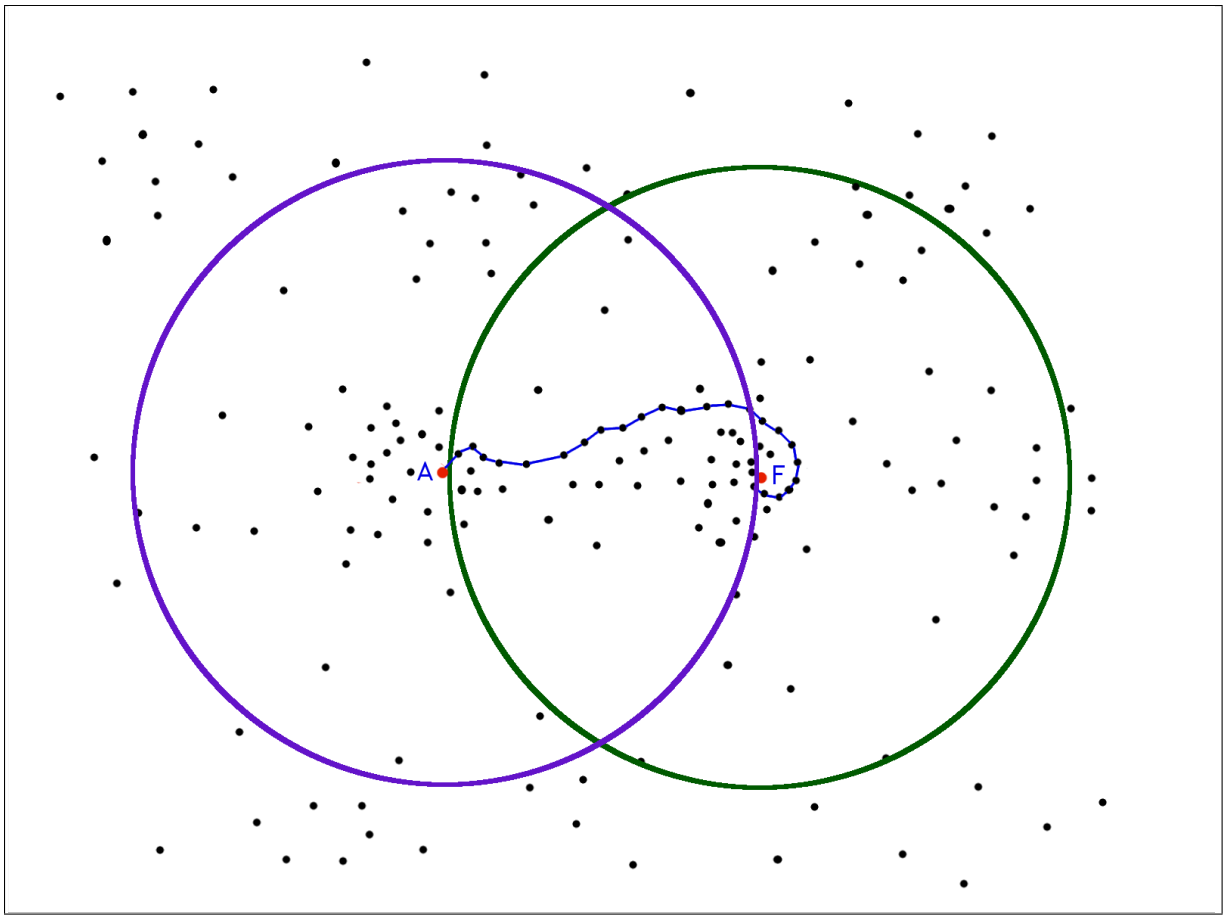


Abbildung 9.5: Vorauswahl der Kandidaten mit Regel (IV) und (V)

Die Schnittmenge $\mathbb{A} \bigcap \mathbb{F}$ der beiden Auswahlkreise bilden somit die Menge aller möglichen Kandidaten für die Generierung der Wiedergabeliste. Dadurch ändert sich der Pfad folgendermassen:

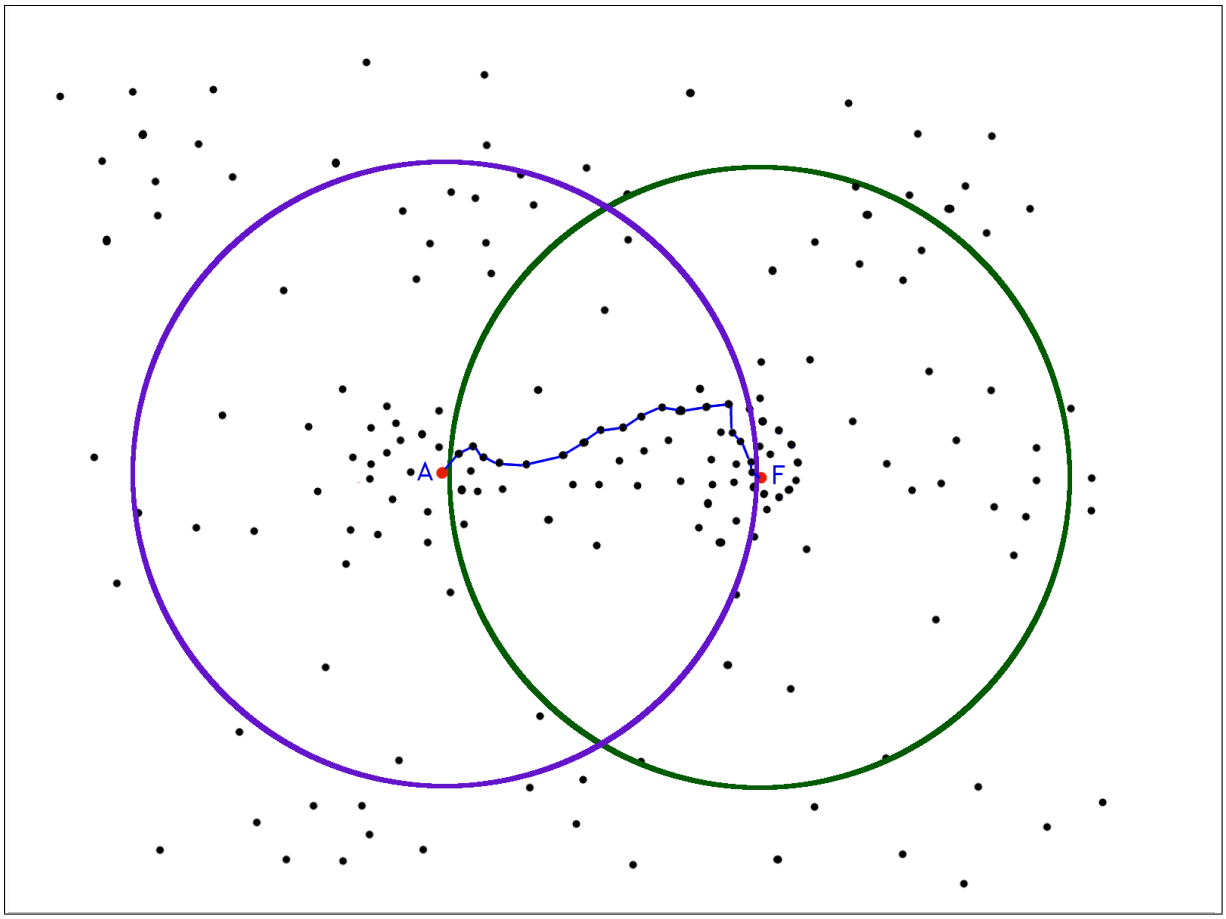


Abbildung 9.6: Neuer Pfad durch Regel (IV) und (V)

Eine weitere Aussage der Regel (III) besagt, dass ein Musikstück nur dann als Kandidat für das nächste Stück in der Wiedergabeliste in Betracht gezogen wird, falls es eine kleinere Distanz zum zweiten gewünschten Stück F aufweist, als das momentane. Wird also ein Kandidat S_p der Wiedergabeliste hinzugefügt, kann die Menge der möglichen Kandidaten um diejenigen S_k reduziert werden, welche eine grössere Distanz zu F haben als S_p und somit die Bedingung

$$\mathrm{NID}(S_k, S_F) \leq \mathrm{NID}(S_p, S_F) \tag{VI}$$

nicht erfüllen. Betrachten wir nochmals dasselbe Beispiel, wobei jedoch der Auswahlradius nach Regel (VI) angepasst wird, was hier an zwei Orten exemplarisch durch die grünen Kreise illustriert wird:

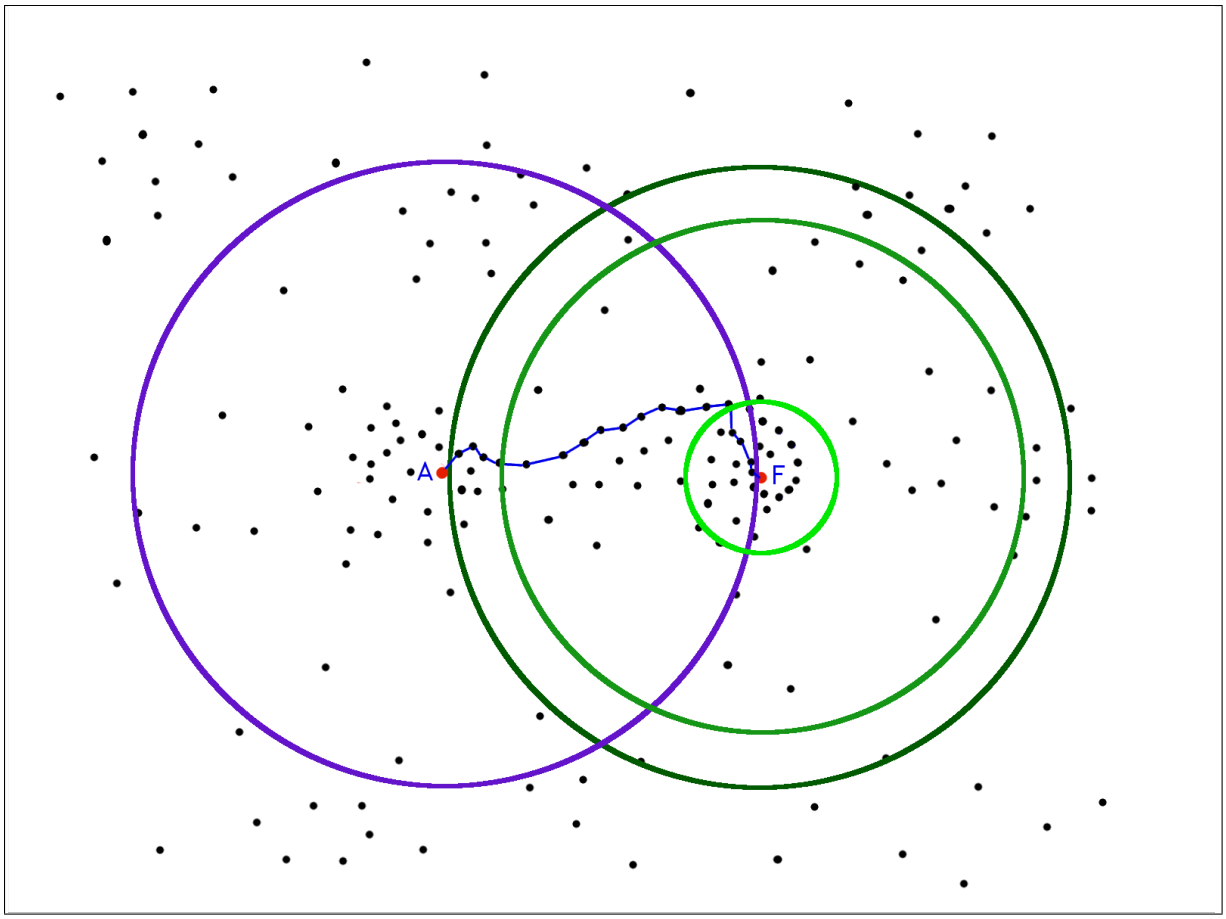
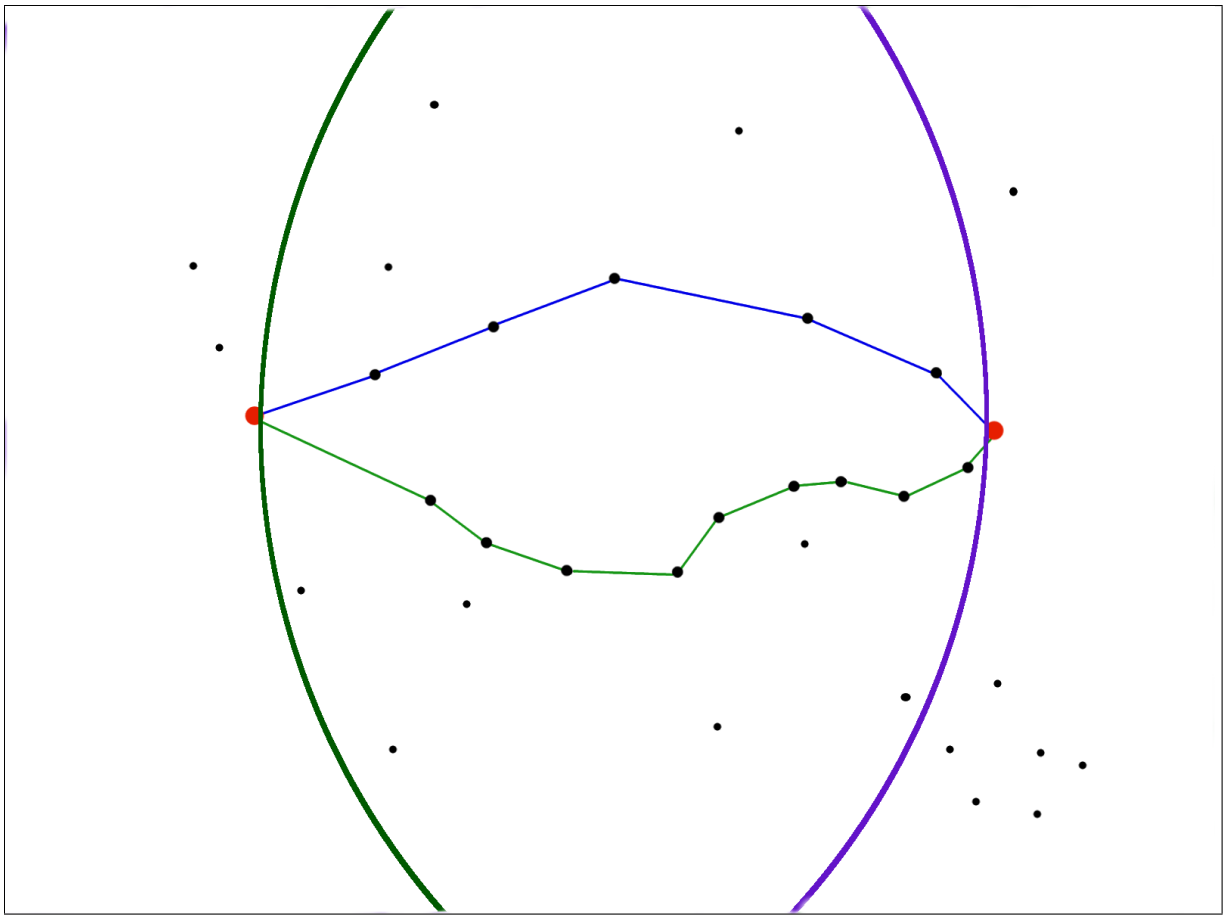


Abbildung 9.7: Anwedung der Regel (VI) nach jedem Schritt

Dadurch kann die Anzahl der zu vergleichenden Kandidaten mit jedem Schritt noch weiter reduziert werden.

9.6 Probleme und Verbesserungspotential

Genauer betrachtet ist die Wiedergabeliste, welche mit Regel (III) generiert wird, nur ein möglicher Kandidat für den „sanftesten“ Pfad. Betrachtet man in folgender Grafik den von Regel (III) gewählten Pfad (blau), sieht man, dass es einen besseren Pfad (grün) gibt, welcher über den ganzen Pfad gesehen „sanftere“ Übergänge hat.

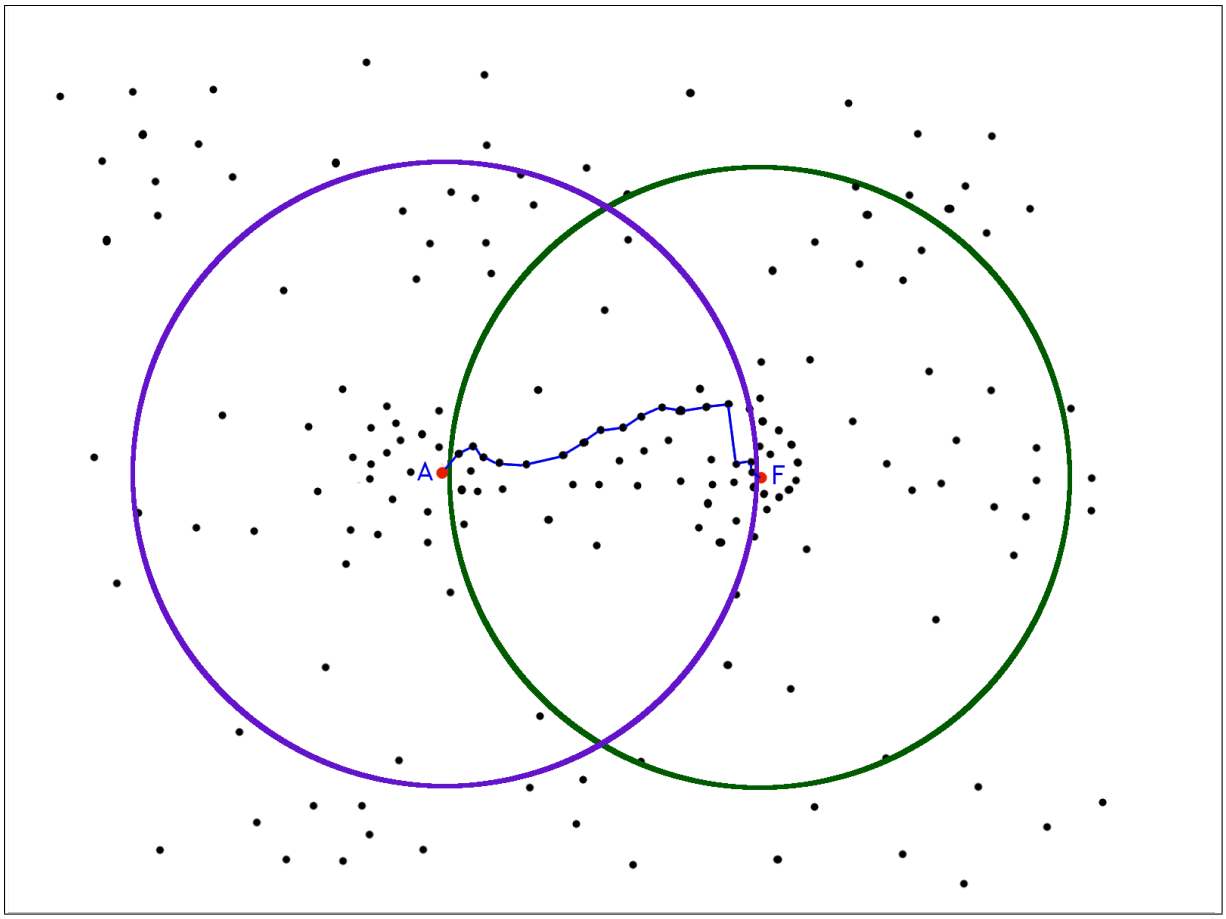


Daraus kann man schliessen, dass um den „sanftesten“ Pfad zu finden, alle Pfade welche die Teilbedingung aus Regel (III)

$$NID(S_c, S_F) \leq NID(S_a, S_F)$$

erfüllen, gegeneinander verglichen werden müssen, um mit Sicherheit sagen zu können, welcher der „sanfteste“ Pfad ist. Dieses Problem in einer für eine praktische Applikation annehmbaren Laufzeit zu lösen, ist sehr schwierig und konnte hier nicht weiter verfolgt werden.

Ein Kompromiss muss auch eingegangen werden, wenn die Regel (V) angewandt wird. Wie an folgender Grafik illustriert wird, kann es bei ungünstigen Landschaften zu unschönen Sprüngen kommen, wenn der Auswahlradius erreicht wird:



Dieses Phänomen ist vor allem bei kleinen Bibliotheken präsent. Daher sollte beachtet werden, dass diese Regel nur bei grossen Bibliotheken eingesetzt wird. Da diese Optimierung bei kleinen Bibliotheken kaum spürbare Effekte auf die Laufzeit hat, kann sie ohne Probleme weggelassen werden.

Kapitel 10

Anforderungsspezifikation

In diesem Kapitel werden wir auf die zentralen Anforderungen an Mixtape eingehen und die zwei wichtigsten Use Cases beschreiben.

10.1 Einsatzszenarien

10.1.1 UC1: Fest im Restaurant

Im Restaurant *Fancy Pants* wird ein Fest mit Apéro, Abendessen und anschliessender Party für die *Bowlingfreunde Schweiz* organisiert. Die Restaurantbetreiber wollen nun eine passende Wiedergabeliste für den Abend zusammenstellen. Sie erstellen in der Mixtape Applikation eine neue Wiedergabeliste *Bowling Fest*. Für die ersten zwei Stunden während dem Apéro wählen sie eine Mischung aus Jazz, Funk und Blues. Damit sich die Gäste während dem darauf folgenden Essen gut unterhalten können, setzen sie für die nächsten zwei Stunden langsame Hintergrundmusik mit sanften Tönen als Kriterium. Um die Party danach in Schwung zu bringen, entscheiden sie sich für den Rest des Abends für eine Mischung aus Rock, Pop und Disco Musik. Nachdem die gewünschten Einstellungen gemacht wurden, generiert die Mixtape Applikation eine Wiedergabeliste und zeigt diese visuell an. Die Restaurantbetreiber schauen sich die Wiedergabeliste durch und bestätigen diese.

10.1.2 UC2: Radio Wunschkonzert

Beim Radiosender *JRadio* können die Hörer jeden Freitagabend für 20 Musikstücke abstimmen, welche am selben Abend ausgestrahlt werden sollen. Diese 20 Musikstücke werden vom Moderator in der Mixtape Applikation als gesetzte Musikstücke markiert. Da die Sendung vier Stunden dauert, wird diese Zeit als Wiedergabelistendauer festgelegt. Nun lässt der Moderator die Wiedergabeliste für den Abend generieren. Die Mixtape Applikation füllt die Wiedergabeliste mit den 20 festgesetzten Musikstücke mit weiteren dazu passenden Musikstücken aus der Musiksammlung der Radiostation auf, um die vier Stunden Sendezeit auszufüllen.

10.1.3 UC3: Neue Musik in eine bestehende Wiedergabeliste passend einfügen

Alan Smithee hat sich gerade 3 neue Alben gekauft, zwei Metal und ein Funk Album. Nun möchte er die Alben passend in seine bestehende Wiedergabeliste integrieren. Dafür wählt er die Alben in der Mixtape Applikation aus und fügt sie der bestehenden Wiedergabeliste hinzu. Da die Wiedergabeliste bereits mehrere Stunden Musik enthält und er die neu gekauften Alben nicht erst nach drei Stunden hören möchte, gibt er diesen eine höhere Priorität. So werden diese möglichst am Anfang der Wiedergabeliste eingereiht. Nach Bestätigung der Eingaben werden die hinzugefügten Alben automatisch sinnvoll in die Wiedergabeliste integriert.

10.2 Funktionale Anforderungen

Mixtape soll es ermöglichen, mit wenigen Klicks eine Wiedergabeliste mit definierten Ähnlichkeitskriterien zu generieren. Ausserdem sollen nachträglich weitere Musikstücke in die Wiedergabeliste eingefügt werden können, welche ebenfalls das Pathfinding durchlaufen, um so optimal in die Wiedergabeliste eingepasst zu werden.

10.2.1 Anforderungen an den Core

Das Core-Projekt soll als eigenständige Library verwendet werden können und unabhängig sein von einem allfälligen Graphical User Interface. Die Struktur des Core-Projekts ist so aufgebaut, dass es ohne grossen Aufwand mit weiteren Algorithmen zur Extraktion von Musikeigenschaften ergänzt werden kann. Aufgrund des hohen Ressourcenbedarfs, ist eine skalierbare und hochparallelisierte Architektur von zentraler Bedeutung. Weitere Details zum Parallelisierungs-Konzept wird im entsprechenden Abschnitt 10.3.1 beschrieben.

10.2.2 Anforderungen an die WebApp

Während im Core keinerlei Berechtigungskonzepte umgesetzt wurden, wird in der WebApp zwischen Administrator und Standard-Anwender unterschieden. Letzterer hat lediglich die Möglichkeit, Musikwünsche zu platzieren, welche durch den Pathfinding-Prozess an geeigneter Position in die Wiedergabeliste eingefügt werden. Dem Administrator stehen nach der Anmeldung zusätzlich folgende Funktionen zur Verfügung:

- Erstellen von Wiedergabelisten.
- Abrufen von Statistiken zum Zustand von Mixtape.
- Das Einlesen des Musik-Verzeichnisses auf dem Server veranlassen, um nach neu hinzugefügten Musikstücken zu suchen und analysieren zu lassen.
- Die Sortierung der Musikstücke in der generierten Wiedergabeliste ändern sowie Musikstücke aus derselben löschen.

Die WebApp wurde so umgesetzt, dass möglichst keine spezifischen Kenntnisse zu MIR vorausgesetzt werden und dass damit die Applikation von einem breiteren Publikum genutzt werden kann.

10.2.3 Use Case 1: Anwender wünscht ein Musikstück

Use Case	UC1 - Anwender wünscht ein Musikstück
Beschreibung	Der Anwender sucht nach einem gewünschten Musikstück und bestätigt dieses als Wunsch.
Beteiligte Akteure	Standard-Anwender
Preconditions	Mixtape muss über analysierte Musikstücke verfügen. Eine Wiedergabeliste muss vom Administrator generiert worden sein.
Standardablauf	1. Der Anwender gibt im Suchformular der WebApp einen Suchbegriff zu einem gewünschten Musiktitel, Interpreten oder Album ein. 2. Das System sucht nach passenden Treffern in der Datenbank und stellt die Ergebnisse in einer Liste dar. 3. Der Anwender klickt auf das gewünschte Musikstück. 4. Das System führt mit dem gewünschten Musikstück und der aktuellen Wiedergabeliste ein Pathfinding durch und ergänzt entsprechend die Wiedergabeliste. 5. Der Anwender erhält eine Bestätigung, dass der Musikwunsch hinzugefügt wurde und nun die Wiedergabeliste in Kürze entsprechend erweitert wird.
Alternative Ablaufschritte	1. a Es wurde noch keine Wiedergabeliste initialisiert. i. Die Suche steht nicht zur Verfügung und dem Anwender wird eine entsprechende Meldung angezeigt.

10.2.4 Use Case 2: Administrator erstellt eine neue Wiedergabeliste

Use Case	UC2 - Administrator erstellt eine neue Wiedergabeliste
Beschreibung	Der Administrator möchte eine neue Wiedergabeliste generieren und gibt dazu Ähnlichkeitskriterien an.
Beteiligte Akteure	Administrator
Preconditions	Mixtape muss über analysierte Musikstücke verfügen.
Standardablauf	Alle nachfolgenden Schritte sind optional. Wo keine Einstellungen durch den Administrator gemacht werden, werden sinnvolle Standardwerte angenommen. 1. Der Administrator kann eins bis mehrere Musikstücke hinzufügen, welche als Ausgangslage für die Generierung der Wiedergabeliste genutzt werden sollen. 2. Der Administrator kann die Dauer der zu generierenden Wiedergabeliste definieren. Er kann die Länge entweder über Anzahl Minuten oder Anzahl Musikstücke angeben. Diese Länge betrifft nur die initiale Wiedergabeliste und beschränkt danach nicht die Möglichkeiten beim Hinzufügen von Musikwünschen. 3. Der Administrator kann zu jeder einzelnen der derzeit vier Hauptkategorien von Musikeigenschaften die Ähnlichkeitsbedingung festlegen. 4. Der Administrator übermittelt die Eingaben ans System. 5. Das System führt mit den gemachten Einstellungen ein Pathfinding aus und erstellt entsprechend die Wiedergabeliste. 6. Der Anwender erhält eine Bestätigung, dass die Wiedergabeliste derzeit generiert wird und in Kürze entsprechend angezeigt wird.
Alternative Ablaufschritte	1. a Es wurde noch keine Wiedergabeliste initialisiert. i. Die Suche steht nicht zur Verfügung und dem Anwender wird eine entsprechende Meldung angezeigt.

10.3 Nichtfunktionale Anforderungen

10.3.1 Parallelisierung

Die Prozesse in Mixtape sind in Bezug auf Prozessor-Leistung und Arbeitsspeicher sehr ressourcenintensiv. Daher wurde die Architektur so aufgebaut, dass Multithreading möglichst gut ausgenutzt werden kann. Wir haben die Parallelisierung nicht auf die Musikstücke als Ganzes, sondern vielmehr auf die Analyse innerhalb eines einzelnen Musikstücks aufgeteilt. Nachfolgend wird der Aufbau erläutert.

Sobald beim Einlesen eines Musikstücks genügend Daten für ein Window vorhanden sind, wird bereits mit der Extraktion der Musikeigenschaften begonnen. Dabei werden diese Windows an parallel ablaufende Extraktionsprozesse übergeben. Dies hat zur Folge, dass die Windows theoretisch in beliebiger Reihenfolge verarbeitet werden können, von so vielen Threads, wie eben verfügbar sind. Gleichzeitig mit dem Einlese- und dem Extraktionsprozess beginnt auch bereits das Postprocessing. Beim Postprocessing kommt pro Feature genau ein Thread zum Einsatz, da hier die verarbeiteten Windows wieder in korrekter Reihenfolge benötigt werden. Das bedeutet, dass das Postprocessing zwar schon von Anfang an läuft, aber solange blockiert, bis es jeweils das in der Reihenfolge nachfolgende Window aus dem Extraktionsprozess erhält.

10.3.2 Migrierbarkeit

Mixtape basiert auf Java und ist dadurch auf jeder Java-fähigen Umgebung mit Java EE-Unterstützung ausführbar. Dies bringt grosse Vorteile mit sich, gerade wenn die Applikation sehr ressourcenlastig ist und somit dafür prädestiniert ist, in einer Cloud-Umgebung ausgeführt zu werden.

Kapitel 11

Software Architektur

In diesem Abschnitt wird der Aufbau der Software Architektur beschrieben. Dabei wird im Abschnitt 11.1 auf architektonische Ziele und Einschränkungen eingegangen. Im darauf folgenden Abschnitt 11.2 werden das Design Modell und die wichtigsten Komponenten darin vorgestellt. Abschnitt 11.3 beschreibt schliesslich die Persistenzschicht.

11.1 Architektonische Ziele und Einschränkungen

11.1.1 Ziele

Webserver

Die Bedienung von Mixtape geschieht über das Webinterface. Aus diesem Grund ist es von zentraler Bedeutung, dass dieses stabil läuft. In der Entwicklungsphase dieser Bachelorarbeit wurde festgestellt, dass Tomcat 7 als Webserver stabiler und zuverlässiger läuft als der zu Beginn eingesetzte Jetty Webserver in der Version 9.0. In der Konfiguration sind nach wie vor beide Webserver vorhanden und können auch entsprechend eingesetzt werden. Es wird jedoch empfohlen, sich auf Tomcat zu beschränken.

Datenbanken

Bei der Implementation der Persistenz-Schicht wurde insbesondere auf die Einhaltung des JPA Standards geachtet, um die Austauschbarkeit der Objekt-relationalen Mapper, der Persistence Provider und der zugrundeliegenden Datenbank zu ermöglichen. Insbesondere wurde auf den Einsatz von NativeQueries verzichtet.

Fehlerbehandlung

Die Prozesse in Mixtape sind vor allem aus zwei Gründen fehleranfällig. Einerseits bietet das hohe Mass an Parallelisierung grosses Potential für Fehler und Lock-Problematiken. Andererseits besteht durch die hohe Ressourcenlast die Gefahr, nicht genügend Ressourcen für die Prozesse zur Verfügung zu haben. Diese beiden Fehlerquellen treten jedoch erst zur Laufzeit auf. Bis auf wenige Ausnahmen findet in den Low-Level-Komponenten des Cores keine Fehlerbehandlung statt, sondern Fehler werden in die höheren Schichten in die Services weitergereicht, um dort geeignet behandelt zu werden.

11.1.2 Einschränkungen

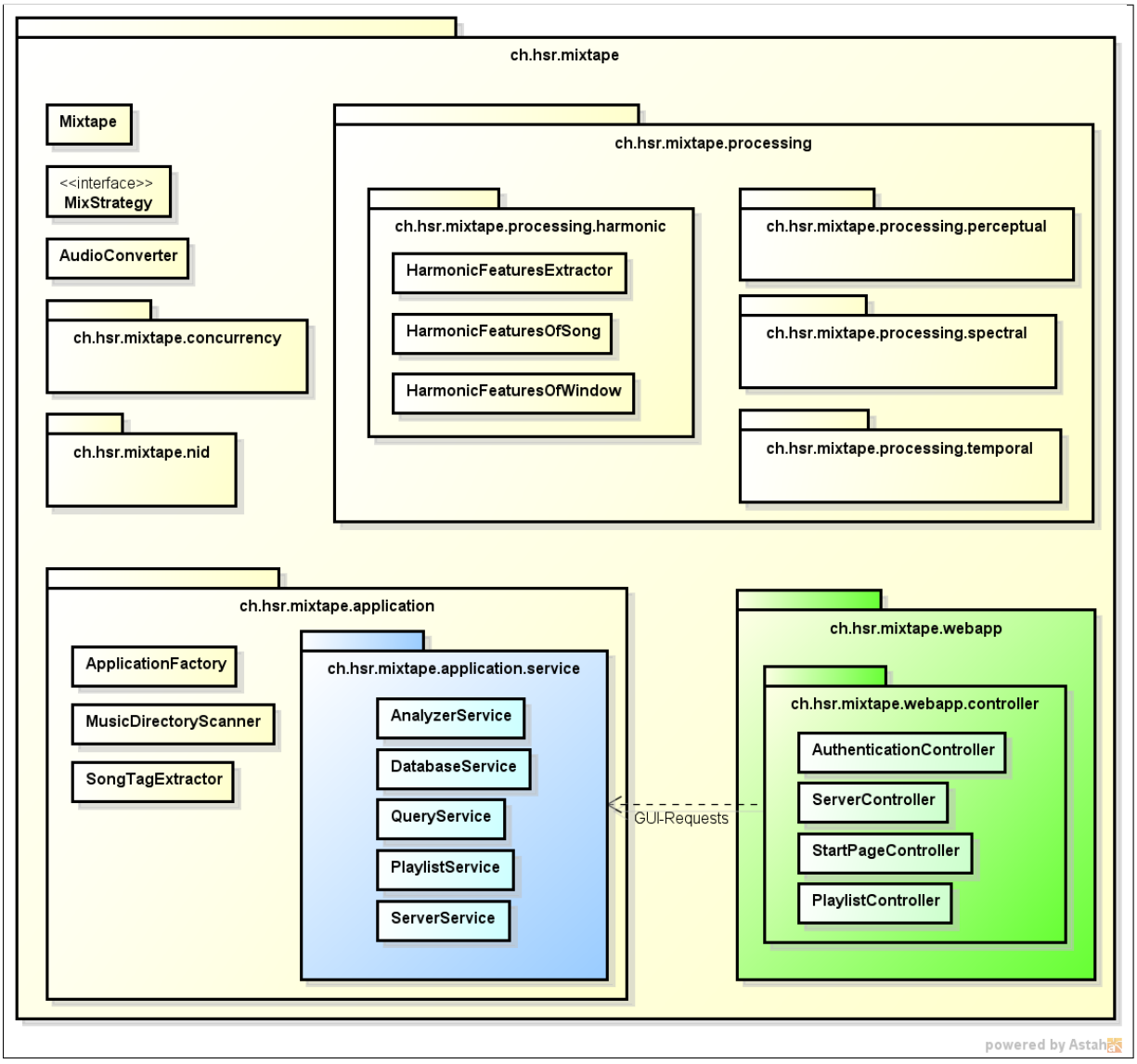


Abbildung 11.1: Design Modell mit den wichtigsten Komponenten.

11.2 Logische Architektur

Mixtape ist in zwei Projekte aufgeteilt. Es besteht aus einem Core-Projekt und einem WebApp-Projekt (in Abbildung 11.1 grün hervorgehoben). Vordergründig gibt es dafür zwei Gründe. Zum einen wird dadurch sichergestellt, dass der Core als unabhängige Softwarebibliothek genutzt werden kann, ohne den Ballast einer grafischen Benutzeroberfläche mitzuschleppen. Zum anderen bietet der Core durch diese Aufspaltung ein klar definiertes Interface gegenüber Dritthersteller-Software (in Abbildung 11.1 blau dargestellt). Auch bleibt dadurch die graphische Benutzeroberfläche austauschbar. Die Datenhaltung geschieht im Core, wodurch Mixtape gleichzeitig aus mehreren Benutzeroberflächen angesprochen werden kann.

11.2.1 Aufbau des Core Projekts

Beim Aufbau des Cores wurde darauf geachtet, dass die einzelnen Komponenten möglichst voneinander unabhängig sind. So können neue Funktionen, wie zum Beispiel neue Algorithmen zur Eigenschaftsextraktion oder ein neuer Pathfinding-Algorithmus, einzeln hinzugefügt werden, ohne dabei in anderen Komponenten tiefgreifende Änderungen hervorzurufen.

Package ch.hsr.mixtape.processing

Im *processing*-Package befinden sich alle Funktionen, welche sich mit der Extraktion der Musikeigenschaften befassen. In Mixtape sind die Musikeigenschaften in vier Oberkategorien aufgeteilt, welche sich auch in der Package-Unterteilung innerhalb vom *processing*-Package wiederfindet. Jedes dieser vier Subpackages enthält einerseits Klassen, welche einem gemeinsamen Namensschema folgen, damit deren Funktion innerhalb von Mixtape einheitlich ersichtlich ist. Dies wird in Abbildung 11.1 am Beispiel der harmonischen Eigenschaftskategorie gezeigt.

Package ch.hsr.mixtape.application

Dieses Package bietet eine Laufzeitumgebung mit entsprechenden Schnittstellen zur Kommunikation mit Graphical User Interface oder Dritthersteller-Software.

Package ch.hsr.mixtape.nid

In diesem Package finden sich die Funktionen zur Berechnung der Informationsdistanzen.

Weitere wichtige Klassen und Packages

Klasse ch.hsr.mixtape.application.MusicDirectoryScanner Diese Klasse ist dafür zuständig, das Audioverzeichnis nach neu hinzugefügten Musikstücken zu scannen und falls nötig in der Datenbank zu erfassen. Der MusicDirectoryScanner übergibt nach dem Scannen die neu gefundenen Musikstücke dem AnalyzerService.

Klasse ch.hsr.mixtape.application.SongTagExtractor Mit dieser Klasse bietet Mixtape die Möglichkeit, einige Informationen wie Titel, Interpret, Album und Dauer aus den ID3-Tags der Musikstücken zu extrahieren.

Klasse ch.hsr.mixtape.AudioConverter Die verwendeten Musik-Datenformate sind heute sehr unterschiedlich (MP3, AAC, OGG, etc.). Mixtape enthält bereits zwei Service Provider Interfaces für MP3 und OGG. Für alle anderen Formate wird versucht, eine Transcodierung ins OGG-Format vorzunehmen, um dies Musikstücke von einer Eigenschaftsextraktion nicht auszuschliessen. Der AudioConverter basiert auf JAVE (Java Audio Video Encoder), einer Java-basierten Programmbibliothek, die ihrerseits auf die frei verfügbare Programmbibliothek Eine frei verfügbare Programmbibliothek zur Codierung und Decodierung diverser Video- und Audio-Formate. Je nach kompillierter Version werden auch Funktionen wie Streaming und Recording unterstützt. via JNI zugreift.

Klasse ch.hsr.mixtape.Mixtape Diese Klasse ist verantwortlich für die paralleliserte Ausführung des Extraktionsprozesses. Sie erhält zu analysierende Daten vom AnalyzerService und gibt diese am Ende auch wieder an diesen zur Persistierung zurück.

Interface ch.hsr.mixtape.MixStrategy Eine *MixStrategy* ist die Beschreibung eines Pathfinding-Algorithmus. Das Pathfinding-Verfahren wurde so entworfen, dass es mit verschiedenen *MixStrategies* genutzt werden kann.

11.2.2 Aufbau des WebApp Projekts

Für die Umsetzung der WebApp wurde das Spring Framework und das darauf aufbauende Spring Security Framework eingesetzt. Diese beiden Frameworks verfolgen den Ansatz „Convention over Configuration“ und vieles wird über XML-Files konfiguriert. Da die gesamte Programmlogik von Mixtape über das Core-Projekt gesteuert wird, kümmert sich die WebApp auch nur um die Verarbeitung von Web-Requests und deren Weiterleitung in den Core. Dementsprechend besteht der serverseitige Programmteil der WebApp aus wenigen Controller- und Hilfsklassen. Der clientseitige Teil der WebApp wurde weitgehendst mit modernen HTML5 und CSS3 Mitteln sowie mit der JavaScript-Bibliothek jQuery erstellt. Es wurde besonderen Wert darauf gelegt, die Kommunikation zwischen Client und Server so weit wie möglich asynchron mittels AJAX umzusetzen.

11.3 Persistenz

Wie zu Beginn des Abschnitts 11.2 erwähnt wurde, wird die Datenbank vom Core verwaltet. Die Architektur wurde so umgesetzt, dass weder die Low-Level-Klassen im Core, noch die WebApp direkten Zugriff auf die Datenbank benötigen. Der Zugriff auf die Datenbank wird durch die Klassen und Services im *Application-Package* (ch.hsr.mixtape.application.*) gesteuert. Mixtape wird mit Apache Derby 10.9 ausgeliefert und als OR-Mapper und Persistence Provider wird EclipseLink 2.4.0 eingesetzt. Mehr zur Begründung, warum Apache Derby eingesetzt wird, kann im Anhang B nachgelesen werden.

11.3.1 Datenbank Modell

Die Abbildung 11.2 zeigt diejenigen Datenstrukturen, welche in die Datenbank persistiert werden.

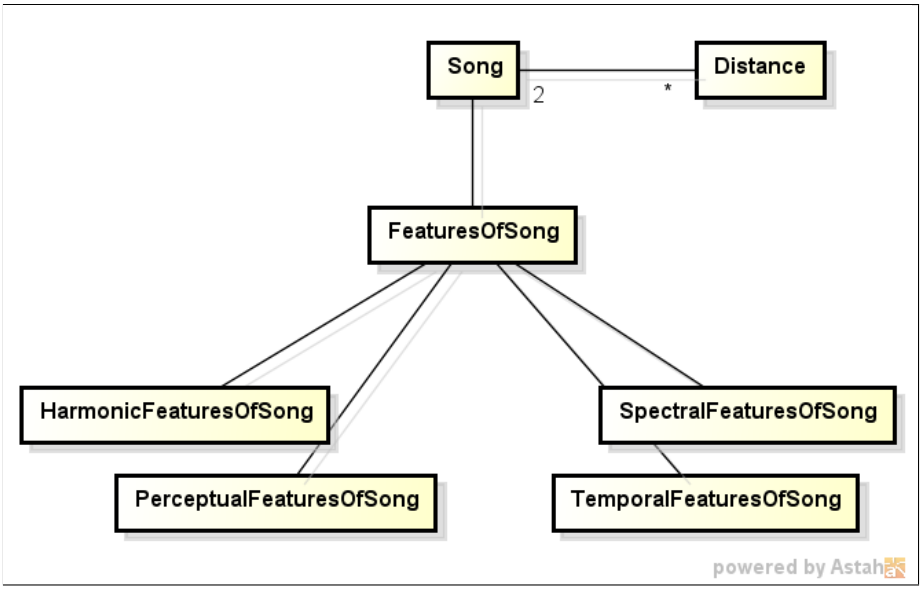


Abbildung 11.2: Datenbankmodell

11.4 Resultate

Sowohl für die Extraktion der Grundfrequenz sowie auch für die Bestimmung des Rhythmus konnten Verfahren kombiniert und um neue Ansätze erweitert werden. Da viele Eigenschaften auf der Grundfrequenz basieren, ist die Genauigkeit bei ihrer Bestimmung von essentieller Bedeutung. Für die Analyse der Ähnlichkeit von Musikeigenschaften konnte ein neuer Ansatz für die Annäherung an die Kolmogorov-Komplexität umgesetzt werden. Sie ist der Grundstein für die NID, welche von uns als Metrik für die Ähnlichkeit eingesetzt wird.

Um auf Basis der Distanzen Wiedergabelisten zu generieren, haben wir den Begriff des „sanften“ Pfads definiert. Dieser beschreibt eine „best effort“ Methode, um zwischen zwei gewünschten Musikstücken sinnvoll andere Musikstücke zu platzieren.

Die entwickelte Applikation ist so aufgebaut, dass sie mit der Anzahl verfügbaren CPUs skaliert, um möglichst effizient die Rechenleistung auszunutzen. Dies ist für eine praktisch orientierte Anwendung in diesem Bereich essentiell, da die lange Dauer der Musikanalyse den praktischen Einsatz sonst fast verunmöglicht. Zusätzlich konnte durch Optimierungen trotz hohem Parallelisierungsgrad ein relativ konstanter *Memory Footprint* erreicht werden, was die Usability erhöht.

11.5 Future Work

Leider war es aufgrund des Umfangs der Arbeit nicht möglich, eine gute Testumgebung aufzubauen, um die Resultate unserer Verbesserungsversuche zu überprüfen. Da es sehr schwierig ist, durch reines hören der Wiedergabeliste, eine begründete Aussage über die Qualität des Resultats zu machen, wäre es denkbar, mit einem Synthesizer ein Set von künstlich erzeugter Musik zu erstellen. Mit einem solchen Set können Aussagen über die Qualität der Features und der berechneten Distanz gemacht werden. Dies weil das gewünschte Resultat im Voraus bekannt ist.

Weiter wäre es denkbar, die Generierung der Wiedergabelisten genauer zu betrachten. Der vorgeschlagene Algorithmus in 9.4 für die Generierung von Wiedergabelisten ist, wie bereits erwähnt, ein „best effort“ Algorithmus, welcher auf eine akzeptable Laufzeit ausgerichtet und einfach umzusetzen ist. Bei der Annäherung an den „sanftesten“ Pfad, wie er in 9.2 beschrieben wurde, steckt jedoch noch viel Optimierungspotential.

Die Architektur wurde entsprechend so gestaltet, dass dieser Algorithmus einfach ausgewechselt werden kann.

Kapitel 12

Persönliche Berichte

12.1 Curdin Barandun

Der Beginn dieser Arbeit war eine ziemliche Herausforderung, da unser Vorwissen im Bereich Musiktheorie, Signalanalyse und Klassifizierungsverfahren sehr durchschnittlich war. Die Möglichkeit, in einem Bereich mit noch soviel Potential eine Arbeit zu machen, war aber sehr motivierend für uns und hat uns angetrieben. Die Möglichkeit, eigene Verfahren zu entwickeln und sich mit dem Kern von Problemen auseinanderzusetzen, war eine sehr gute Erfahrung und war etwas vom Interessantesten, das ich in diesem Studium mache konnte. Wir waren doch etwas zu ambitioniert oder haben die Zeit unterschätzt, die nötig war, um ein gewisses Grundverständnis aufzubauen. So war es uns leider nicht mehr möglich unsere Verfahren genau zu analysieren und das Feintuning vorzunehmen. Alles in Allem konnten wir zumindest teilweise neue Wege beschreiten, was sehr aufregend war.

12.2 Stefan Derungs

Die Arbeit an dieser Bachelorarbeit hat mir sehr viel Spass gemacht. Einerseits konnte ich aus dem Vollen schöpfen und mich mal über längere Zeit in ein Thema einarbeiten und damit beschäftigen. Dabei konnte ich auf vieles, was ich im Studium gelernt hatte, zurückgreifen. Aufgrund der Grösse des Forschungsgebiets bestand immer die Gefahr, dass man sich zu sehr in Details verrennt, bei Problemen stecken bleibt und dadurch kostbare Zeit verliert. So wurde gegen Schluss der Arbeit der Zeitdruck immer spürbarer. Trotzdem bin ich davon überzeugt, dass wir ein gutes Endergebnis erzielt haben, dies nicht zuletzt auch wegen der guten Zusammenarbeit im Team, wo man sich immer gut ergänzt hat.

12.3 Gino Paulaitis

Die Arbeit an Mixtape war sehr spannend. Während dieser Zeit konnte ich einige neue Dinge kennenlernen und ausprobieren, von denen ich sicherlich noch profitieren werde. So wird mich wohl die Signalanalyse noch einige Zeit beschäftigen. Dementsprechend froh bin ich darüber, dass wir diese Möglichkeit erhalten haben. Natürlich gab es auch mehr als genügend Momente, in denen die Motivation auf ein Minimum sank. Besonders wenn man realisierte, was man sich noch alles vorgenommen hat, während einem die Zeit davon rast.

Und doch bin ich froh, diese Erfahrungen gesammelt zu haben, was nicht zuletzt an meinem Team zu verdanken ist. Ich bin der Meinung, zusammen haben wir eine ziemlich gute Arbeit geleistet.

Anhang A

System-zentriertes vs. Benutzer-zentriertes MIR

In MIR werden zwei unterschiedliche Formen von Verfahren unterschieden: System-basierte und Benutzer-zentrierte [49]. Ersteres Verfahren setzt ausschliesslich auf die Auswertung von aus Musikstücken algorithmisch extrahierten Daten, während das zweite Verfahren den Benutzer, dessen Umfeld, seine Erfahrungen und weitere Eigenschaften mit einbezieht.

Der Grossteil der heute verfügbaren MIR verfolgt einen System-basierten Ansatz. Dieser Ansatz hat jedoch seine Schwächen. Betrachten wir einmal das Beispiel der Musik-Genres. Die Einteilung von Musikstücken in Genres und Sub-Genres ist eine Aufgabe, welcher Anbieter von Musik-Downloads wohl täglich begegnen. Diese Aufgabe ist aber selbst für einen geübten Musik-Kenner manchmal alles andere als trivial. Insbesondere heute, wo Musikstücke immer mehr durch *multikulturelle* Einflüsse geprägt werden - sei es nun ein Rap, der durch einen Einspieler mit indischer Musik ergänzt wird, oder ein Pop Musikstück, das mit einem Concerto von Vivaldi gepaart wird. In welche Genres sollen diese *gemischten* Musikstücke nun eingeteilt werden? Sie werden nun sagen „in beide!“, aber was für Sie eine einfache Aufgabe erscheint, ist eine Herausforderung für die Technik.

Ein Ziel der Einteilung in Genres ist der Wunsch, einem Benutzer ähnliche Inhalte präsentieren zu können. Stellen Sie sich vor, Sie hören ein Musikstück, das Ihnen gefällt, und Sie möchten nun weitere Musikstücke finden, die ähnlich sind. Hier ist das Problem, dass nicht jeder Benutzer dieselben „Ähnlichkeitskriterien“ hat.

Ein Beispiel: Sie sind absolut begeistert von der mitteleuropäischen klassischen Musik. Dementsprechend gut kennen Sie sich aus - für Sie ist Beethoven nicht nur ein Hund[1], Sie kennen jede von Liszt's Klaviersonaten und jedes von Vivaldi's Concertos. Nun werden Sie der Behauptung sicher zustimmen, dass die Werke von Vivaldi und Liszt sich in kaum einer Weise ähnlich sind und auch nicht ähnlich klingen. Würde Ihnen ein MIR-System nun auf ein Liszt-Konzert ein Konzert von Vivaldi vorschlagen, würden Ihnen vermutlich die Haare zu Berg stehen. Andererseits ist Ihnen die indische Klassik weitgehendst unbekannt. Dementsprechend werden Sie den Unterschied zwischen den Stilrichtungen Dhamar und Kajari kaum bemerken und dementsprechend toleranter auf Vorschläge aus einem MIR-System sein.

Am persönlichen Beispiel möchten wir damit aufzeigen, dass nicht jeder Musik auf die gleiche Art und Weise einstuft und somit ein rein system-basiertes Verfahren, welches den

[1] „Ein Hund namens Beethoven“, ein Film von 1991.

Benutzer vollkommen ausser Acht lässt, bemerkenswerte Mängel aufweist.

Schedl und Flexer[49] weisen in ihrem Aufsatz auf diese Schwächen hin und präsentieren Ansätze, wie der Benutzer zukünftig in einem MIR-System modelliert werden könnte.

Anhang B

Persistenz

Die Datenmengen, die Mixtape verarbeiten muss, sind gross und ebenso sind die Aufwände für die Rechenprozesse. Eine Musiksammlung ist selten statisch, sondern ändert sich im Verlauf der Zeit. Um beim Hinzufügen neuer Musikstücke nicht die gesamte Musiksammlung erneut analysieren zu müssen, scheint es daher sinnvoll, ein gewisses Volumen an berechneten Daten zu speichern. Konkret sind das die Eigenschaftsvektoren der einzelnen Musikstücke sowie die Distanzen zwischen allen Musikstücken. Nachfolgend werden zwei Persistenz-Möglichkeiten beschrieben und deren Eignung für dieses Projekt untersucht. Die zwei Hauptkategorien sind die Datei-basierte und die Datenbanksystem-basierte Persistenz.

B.1 Datei-basierte Persistenz

Ein Vorteil dieses Ansatzes liegt in der individuell gestaltbaren Dateistruktur für die Ablage der Daten. Im Gegensatz zu vielen eingebetteten Datenbanksystemen [DBS], wie zum Beispiel SQLite 3, können hier die Daten beliebig auf verschiedene Dateien aufgeteilt werden. Allerdings kann es zu Problemen bei der Portierbarkeit der Applikation kommen, da man sich als Entwickler selbst um die korrekte Behandlung von Zeichencodierungen und Zeichensatztabellen[1] kümmern muss. Des Weiteren müssen Create, Read, Update, Delete-Operationen auf Dateiebene selber implementiert werden, was vermutlich weniger effizient realisiert wird, als in DBS. Ausserdem können die gespeicherten Daten aufgrund von fehlenden Index-Strukturen nur sequentiell gelesen werden.

B.2 Datenbanksystem-basierte Persistenz

Um eine Applikation zu erstellen, die möglichst einfach zu installieren und zu nutzen ist, sollen nur eingebettete Datenbanksysteme untersucht werden, die keinerlei Installation durch den Endbenutzer erfordern. Die Vorteile eines DBS gegenüber einer reinen Datei-basierten Persistenz sind die bereits vorhandenen Create, Read, Update, Delete-Operationen sowie vorhandene Indizes. Auch muss man sich nicht um allfällige Dateisystem-Unterschiede kümmern. Wie bereits erwähnt, können beim reinen Datei-basierten Ansatz Daten nur sequen-

[1] *engl.* Codepages

tiell aus den Dateien gelesen werden. Klare Nachteile sind der zusätzliche Datenbank-Layer und die zusätzliche Abhängigkeit von Drittsoftware.

In den folgenden Unterabschnitten wurden einige DBS und deren Eignung für diese Arbeit untersucht.

B.2.1 SQLite 3

SQLite 3 ist eine Software-Bibliothek, welche ein eigenständiges[2], serverloses, konfigurationsfreies, transaktionsbasiertes relationales SQL DBS implementiert[3]. Die Software ist gemeinfrei und kann kostenlos genutzt werden. Das DBS ist sehr leichtgewichtig, einfach zu administrieren und erlaubt die Anbindung an Java und verschiedenen anderen Programmiersprachen. Es können mehrere simultan lesende und einen einzigen schreibenden Zugriff stattfinden. Ausserdem können mehrere Prozesse gleichzeitig auf die Datenbank zugreifen. Dedizierte Datenbank-Berechtigungen werden nicht unterstützt, was bedeutet, dass ein Benutzer immer vollen Zugriff auf die komplette Datenbank hat. SQLite 3 ist Atomicity, Consistency, Isolation and Durability-konform, d.h. die Datenbank wird durch Programmabstürze nicht beschädigt, und bietet eine integrierte Konsistenz-Überprüfungsfunktion. Eine online Backup/Restore-Fähigkeit fehlt allerdings. Grundsätzlich wird nur eine einzelne Datenbank pro Instanz unterstützt und es bietet keine Schema-Unterstützung. SQLite 3 verfügt nur über einen Teil der SQL92 Sprachspezifikation, ist schwach typisiert und unterstützt nur Basisdatentypen. Weil die gesamte Datenbank in einer einzigen Datei gespeichert wird, müssen gewisse Dinge berücksichtigt werden. Auf der SQLite 3 Webseite wird darauf hingewiesen, dass die Datenbank eine maximale Grösse von 2 Terabytes unterstützt (bei einer Seitengrösse von 1024 Bytes). Diese Zahl kann allerdings nicht für sich alleine betrachtet werden, da man auch den Dateigrössengrenzen des Dateisystems unterliegt. Für NTFS liegt diese standardmässig bei 16 Terabytes [15], bei EXT3 hingegen nur bei 1 Terabyte [11]). Gerade bei grösseren Musik-Bibliotheken könnte diese Einschränkung zum Problem werden.

B.2.2 Apache Derby

Apache Derby ist ein quelloffenes relationales DBS, das unter der Apache License 2.0 veröffentlicht wird und kostenlos genutzt werden kann [2][19]. Das DBS kann nur mit Java oder JVM-basierten Sprachen verwendet werden. Gegenüber SQLite 3 hat dieses System einige Vorteile. Einerseits teilt es die Daten in der Datenbank auf mehrere Dateien auf dem Dateisystem auf, wodurch die Limitierungen der Dateisystemgrenzen irrelevant werden. Des Weiteren unterstützt es die komplette SQL92 und einen Grossteil der SQL99 Sprachspezifikation. Auch unterstützt es Datenbank-Berechtigungen, Datenbank-Schemas und die Verwendung von mehreren Datenbanken, sowie eine breite Unterstützung von Datentypen und Fremdschlüsseln. Nebst einer Konsistenz-Überprüfungsfunktion hat es auch eine online Backup/Restore-Fähigkeit. Die Administration soll aber genauso einfach sein wie bei SQLite 3 und ist ebenfalls Atomicity, Consistency, Isolation and Durability-konform. Apache Derby weist zwei klare Nachteile gegenüber SQLite 3 auf. Zum Einen hat es einen höheren Bedarf an Arbeitsspeicher, wodurch es weniger leichtgewichtig ist. Zum Anderen erlaubt es im Embedded-Mode nur einem Prozess den Zugriff auf die Datenbank.

[2] *engl.* self-contained.
[3] Frei übersetzt aus [18].

B.2.3 Microsoft SQL Server Compact

Microsoft SQL Server Compact [SQL CE] ist ein relationales DBS, welches kostenlos von Microsoft zur Verfügung gestellt wird. Es bietet eine native 64 Bit Unterstützung und ist für Bereiche, wo Datenmengen nicht allzu gross sind (die Datenbankgrösse ist auf 4 Gigabytes limitiert). Ebenso wie SQLite 3 speichert es die Datenbank in einer einzelnen Datei auf dem Dateisystem. In diese Arbeit ist diese Datenbank allerdings nicht nutzbar, da keine native Java-Unterstützung von Microsoft geboten wird und man nur mit Umweg via ODBC und Drittanbieter-Software von Java aus zugreifen kann. Mit dem Microsoft JDBC Driver 4.0 für SQL Server kann nämlich nicht auf SQL CE zugegriffen werden [14].

B.2.4 MySQL

MySQL ist ein weltweit weit verbreitetes Client/Server basiertes DBS. Nebst der traditionellen Client/Server basierten Implementation, existiert inzwischen auch eine eingebettete Variante. Gemeinsam mit SQLite 3 und Apache Derby hat es, dass die Datenbank nicht separat installiert werden muss und vom Benutzer verborgen bleibt. Es unterscheidet sich jedoch grundsätzlich, da ein separater Server-Prozess für die Datenbank gestartet wird. Dadurch ist die Datenbank sicherlich weniger leichtgewichtig als die beiden zuvor genannten DBS, jedoch bietet es eine höhere Leistung und eine umfangreichere Funktionalität.

B.3 Fazit

Von den untersuchten DBS wurde Apache Derby als geeignete Variante gewählt. Es ist zwar nicht so leichtgewichtig wie SQLite 3. Jedoch sind Vorteile wie die ausgedehnte Implementation der SQL Sprachspezifikationen, die Möglichkeit Fremdschlüssel einzusetzen sowie die Speicherung der Daten in mehreren Dateien stärker zu gewichten. Sollte zukünftig die Geschwindigkeit der Applikation aufgrund schlechter Datenbank-Leistungen in Mitleidenschaft gezogen werden, wäre die eingebettete Variante von MySQL sicherlich eine ernstzunehmende Alternative.

Anhang C

Parallelisierung

Die in diesem Projekt verwendeten Algorithmen sind durchaus rechenaufwändig. Dies stellt ein grosses Problem dar, besonders für eine Applikation, die praktischen Nutzen aufweisen soll. Aus diesem Grund war es uns äussert wichtig, dass möglichst viel parallelisiert ablaufen kann, um zumindest auf modernen Rechnern, oder gar auf einem Cluster, entsprechend gut zu skalieren. In diesem Abschnitt sollen nun unsere Ansätze und die damit verbundenen Probleme etwas genauer betrachtet werden.

C.1 Ablauf

Der grundsätzliche Ablauf der Verarbeitung eines Musikstücks sieht wie folgt aus:

1. Das Musikstück wird geöffnet und die einzelnen Samples werden geladen.
2. Die Samples werden zu Windows zusammengefasst und an die Feature Extraktoren übergeben. Hierbei ist zu beachten, dass die Grösse der Windows von Extraktor zu Extraktor variieren.
3. Die Windows werden von den Extraktoren verarbeitet.
4. Wurden alle Windows verarbeitet, wird durch jeden der Extraktoren ein Postprocessing durchgeführt.
5. Das Resultat des Postprocessings wird nun verwendet, um die Distanz zu anderen Musikstücken zu berechnen.

Dieser grobe Ablauf wiederspiegelt sich auch im Softwaredesign, welches bereits in Abschnitt 11 vorgestellt wurde.

C.2 Ansatzpunkte

Anhand dieser Übersicht können bereits einige mögliche Ansatzpunkte für die Parallelisierung identifiziert werden.

- Mehrere Musikstücke können gleichzeitig analysiert werden.
- Die Windows pro Musikstück können gleichzeitig verarbeitet werden.
- Innerhalb der Verarbeitung eines Windows können beispielsweise Arrays gleichzeitig bearbeitet werden.
- Die Distanzen zwischen den Musikstücken können gleichzeitig berechnet werden.

Hierbei haben wir uns für die Parallelisierung der Window-Verarbeitung und der Distanzberechnung entschieden. Die Parallelisierung der Window-Verarbeitung macht Sinn, da wohl schneller Daten geladen und in Windows zusammengefasst werden können, als diese dann verarbeitet werden. Zum Zeitpunkt der Distanzberechnung sind alle nötigen Daten bereits im Arbeitsspeicher vorhanden, womit einer einfachen Parallelisierung nichts im Wege steht. Hingegen kommt die gleichzeitige Verarbeitung von mehreren Musikstücken nicht zum Zuge, da davon auszugehen ist, dass der Zugriff auf den Massenspeicher einen Flaschenhals bildet. So würden zwei Musikstücke nicht doppelt so schnell geladen werden, nur weil beide gleichzeitig geladen werden. Im Gegenteil, es kann sogar davon ausgegangen werden, dass das gleichzeitige Laden mehrerer Dateien einen negativen Einfluss auf die Einlesegeschwindigkeit hat, da beispielweise eine Festplatte ihren Lesekopf ständig zwischen den beiden Speicherorten hin und her bewegen würde. Auch die Parallelisierung innerhalb der Window-Verarbeitung wurde nicht umgesetzt. Dies basiert auf der Annahme, dass die Datenmengen pro Window für eine sinnvolle Parallelisierung zu klein sind.

C.3 Probleme und Lösungen

Erste Ansätze brachten vor allem Speicherprobleme mit sich. Durch die Parallelisierung der Window-Verarbeitung konnte zwar eine sehr hohe Prozessorauslastung erreicht werden, gleichzeitig stieg aber der Arbeitsspeicherverbrauch ins Unermessliche, woraufhin selbstverständlich auch die Prozessorauslastung zusammenbrach und sogar in Programmabstürzen endete.

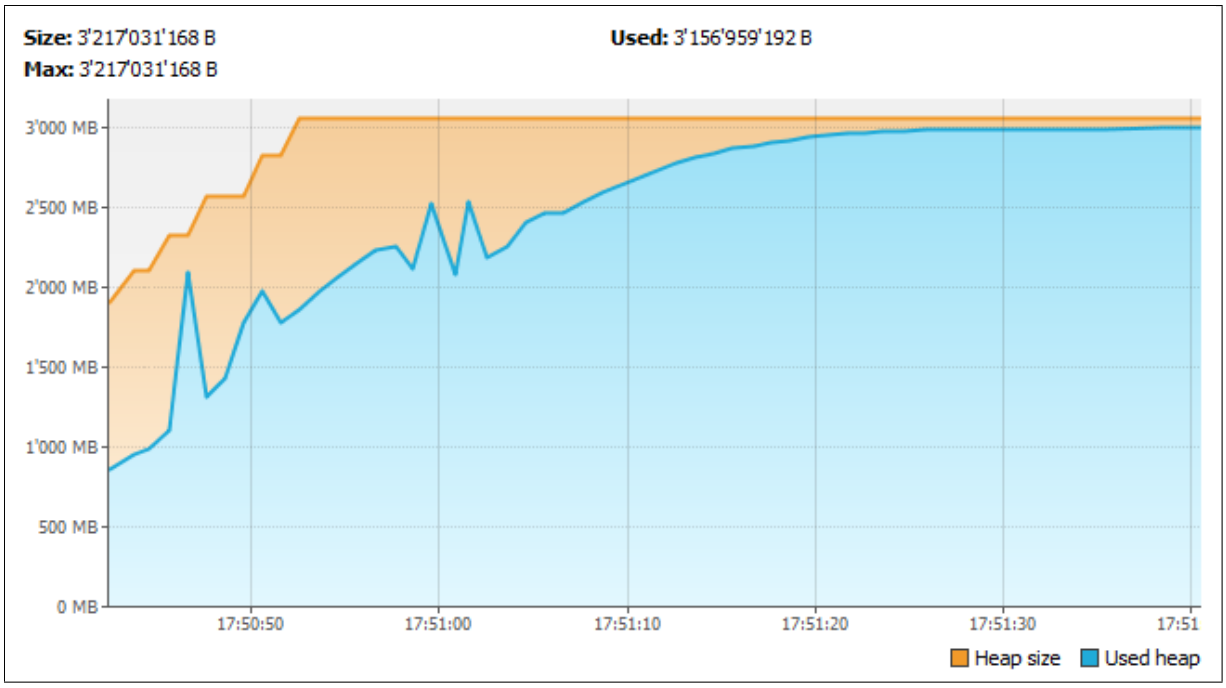


Abbildung C.1: Überlastung des Arbeitsspeichers

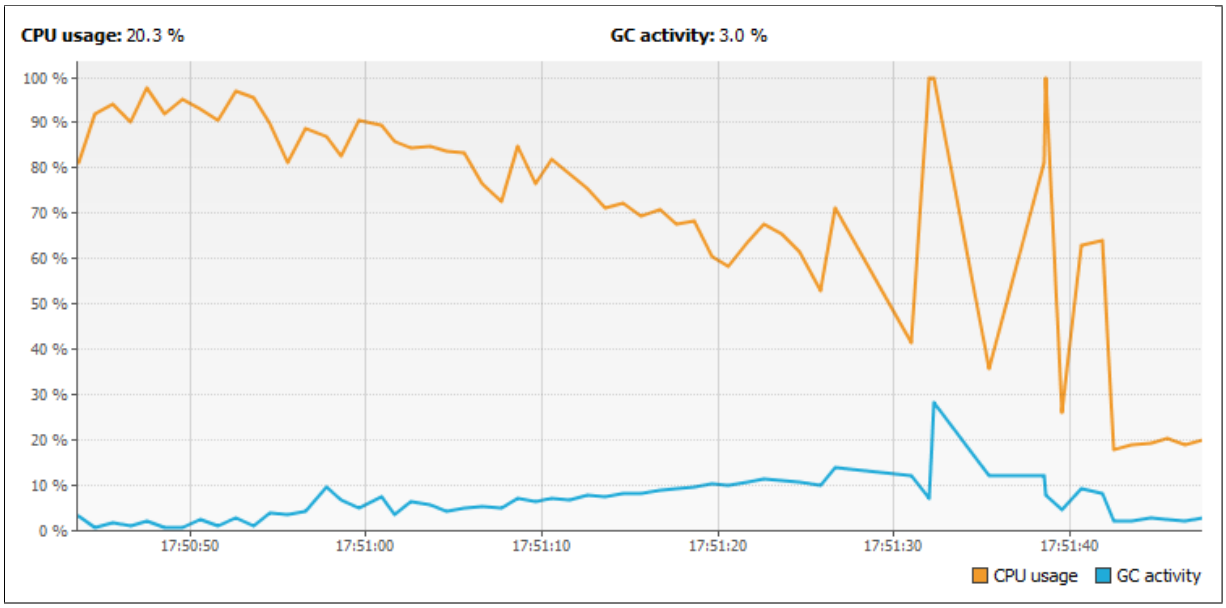


Abbildung C.2: Negativer Einfluss auf die Prozessorauslastung

Nach einigen Analysen konnte das Problem genauer spezifiziert werden. Das Laden der Daten ist um einiges schneller als die nachfolgende Verarbeitung der Windows. Nun wurden die Tasks zur Verarbeitung der Windows sogleich an einen Threadpool weitergeben. Dessen Warteschlange füllte sich nun mit diesen Tasks, welche die Rohdaten des Musikstückes beinhaltenen, was in einer gigantischen Menge von noch unverarbeiteten Daten im Arbeitsspei-

cher endete. Dieses Problem konnte gelöst werden, indem ein spezifischer Threadpool erstellt wurde. Diesem wurde eine angepasste Taskwarteschlange mit beschränkter Grösse übergeben, die die Eigenschaft besitzt, nicht nur Anfragen zu blockieren, falls keine Tasks zur Verfügung stehen, sondern auch die Übergabe von neuen Tasks zu blockieren, falls die Warteschlange voll ist.

Trotz dieser Anpassungen war der Speicherverbrauch noch immer viel zu hoch. Dies lag nun daran, dass die Resultate der Window-Verarbeitung, bis zur Übergabe als Liste ans Postprocessing, im Speicher behalten werden mussten. Doch auch dieses Problem konnte gelöst werden. Hierfür musste das Postprocessing dahingehend angepasst werden, dass es bereits mit der ersten Window-Verarbeitung startet, sobald diese Daten vorliegen. Als Argument wird nun aber keine Liste, sondern eine Eigenimplementation eines Iterators übergeben. Dieser Iterator besitzt die Eigenschaft, bei Zugriffen jeweils solange zu blockieren, bis das nächste Element zur Verfügung steht. Hinzu kommt die Eigenschaft eines Iterators, dass dasselbe Objekt nicht zweimal abgefragt werden kann. Durch diese Lösung kann im Postprocessing immernoch auf eine sortierte Aneinanderreihung aller Resultate der Window-Verarbeitung zugegriffen werden und trotzdem können bereits verarbeitete Resultate wieder durch den Garbage Collector eingesammelt werden. Ein weiterer Vorteil ergibt sich durch die erhöhte Parallelisierung des Postprocessings, wobei hierfür ein weiterer Threadpool eingerichtet werden musste, da durch die blockierenden Iteratoren sonst die Prozessorauslastung wiederum zusammenbrechen würde.

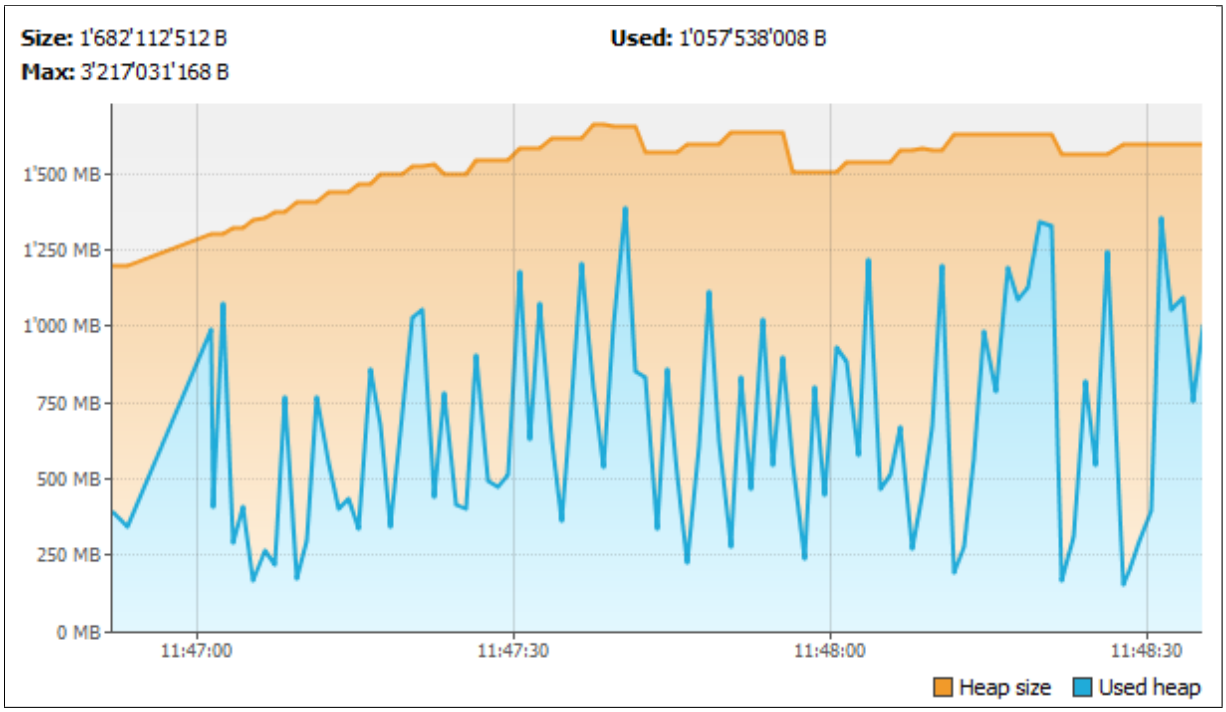


Abbildung C.3: Auslastung des Arbeitsspeichers nach Problemlösung

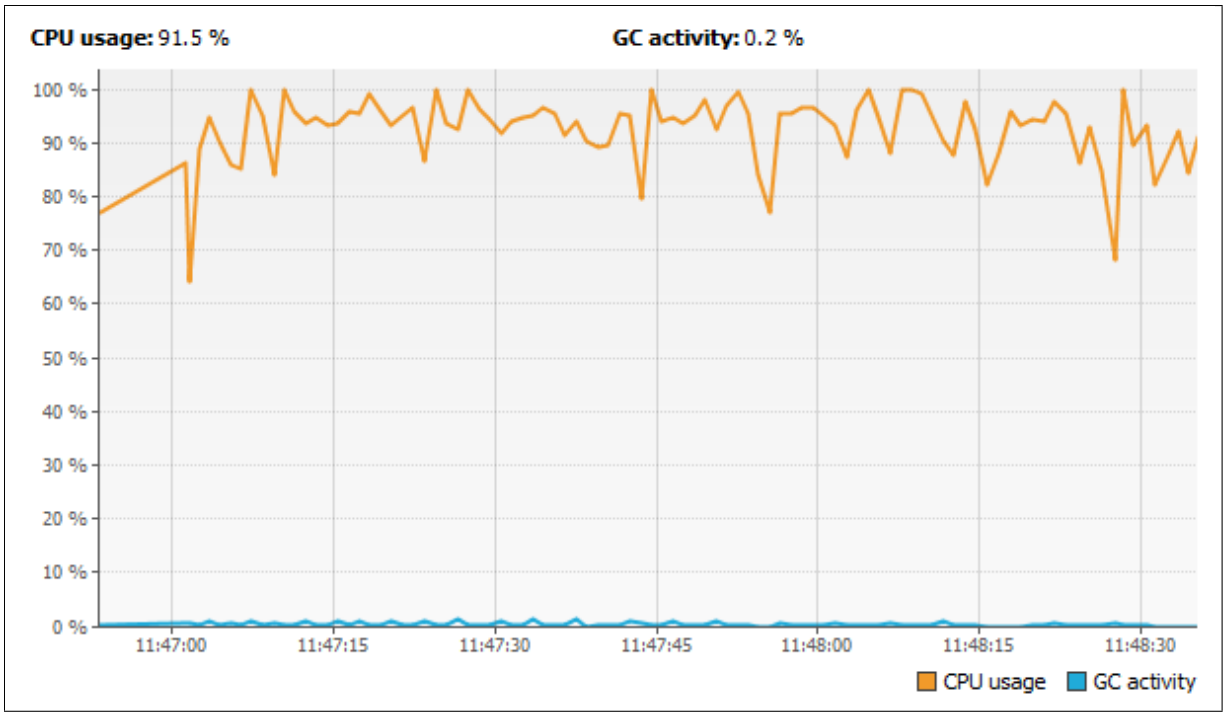


Abbildung C.4: Prozessorauslastung nach Problemlösung

Literaturverzeichnis

[1] Ace-xml. http://jmir.sourceforge.net/ACE_XML.html. Letzter Aufruf 1.3.2013.

[2] Apache derby. http://db.apache.org/derby/. Letzter Aufruf 12.03.2013.

[3] aubio. http://aubio.org/. Letzter Aufruf 25.4.2013.

[4] ELKI - Environment for Developing KDD-Applications Supported by Index-Structures. http://elki.dbs.ifi.lmu.de/. Letzter Aufruf 01.03.2013.

[5] jMIR. http://jmir.sourceforge.net. Letzter Aufruf 1.3.2013.

[6] JUNG - Java Universal Network/Graph Framework. http://jung.sourceforge.net/. Letzter Aufruf 12.06.2013.

[7] KNIME - Konstanz Information Miner. http://www.knime.org/. Letzter Aufruf 12.06.2013.

[8] Marsyas - Ein frei verfügbares Framework zur Verarbeitung von Audiosignalen. http://marsyas.info. Letzter Aufruf 01.03.2013.

[9] NEMA - Networked Environment for Music Analysis. http://www.music-ir.org/?q=nema/overview. Letzter Aufruf 01.03.2013.

[10] Gtzan genre collection. http://marsyasweb.appspot.com/download/data_sets/. Letzter Aufruf 26.03.2013.

[11] Linux ext3 faq. http://batleth.sapienti-sat.org/projects/FAQs/ext3-faq.html. Letzter Aufruf 12.03.2013.

[12] Lua. http://www.lua.org/about.html. Letzter Aufruf 5.3.2013.

[13] Magnatune. http://magnatune.com/. Letzter Aufruf 26.03.2013.

[14] Microsoft jdbc driver 4.0 für sql server. http://www.microsoft.com/de-de/download/details.aspx?id=11774. Letzter Aufruf 12.03.2013.

[15] Microsoft technet: Working with file systems. http://technet.microsoft.com/de-de/library/bb457112%28en-us%29.aspx. Letzter Aufruf 12.03.2013.

[16] The oxford dictionary of music, oxford music online. http://www.oxfordmusiconline.com/. Letzter Aufruf 26.3.2013, Anmeldung erforderlich.

[17] Rwc music database. http://staff.aist.go.jp/m.goto/RWC-MDB/. Letzter Aufruf 26.03.2013.

[18] Sqlite 3. http://www.sqlite.org/. Letzter Aufruf 12.03.2013.

[19] Sqlite: Sqlite versus derby. http://www.sqlite.org/cvstrac/wiki?p=SqliteVersusDerby. Letzter Aufruf 12.03.2013.

[20] Weka 3. http://www.cs.waikato.ac.nz/ml/weka/. Letzter Aufruf 4.3.2013.

[21] Weka arff. http://www.cs.waikato.ac.nz/~ml/weka/arff.html. Letzter Aufruf 1.3.2013.

[22] Arlindo L. Oliveira und Mario A. T. Figueiredo Artur J. Ferreira. On the suitability of suffix arrays for lempel-ziv data compression. http://www.lx.it.pt/~mtf/Ferreira-Oliveira-Figueiredo-2009.pdf, 2009. Letzter Aufruf 13.06.2013.

[23] E. und Liming Chen Boyang Gao, Dellandrea. Music sparse decomposition onto a midi dictionary of musical words and its application to music mood classification. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6269798, 2012. Letzter Aufruf 13.06.2013.

[24] Elmar Bozetti. *Einführung in musikalisches Verstehen und Gestalten.* Diesterweg Moritz, Frankfurt, 1988.

[25] Judith C. Brown. Determination of the meter of musical scores by autocorrelation. http://academics.wellesley.edu/Physics/brown/pubs/meterACv94P1953-P1957.pdf, 1993. Letzter Aufruf 1.4.2013.

[26] Juan Pablo Bello Correa. Towards the automated analysis of simple polyphonic music: A knowledge-based approach. https://qmro.qmul.ac.uk/xmlui/bitstream/handle/123456789/3803/BELLO%20CORREATowards2003.pdf, 2003. Letzter Aufruf 1.4.2013.

[27] Mark DeLoura. The engine survey: General results. http://www.satori.org/2009/03/the-engine-survey-general-results/. Letzter Aufruf 5.3.2013.

[28] Edsger W. Dijkstra. Edsger w. dijkstra: A note on two problems in connexion with graphs. http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf, 1959. Letzter Aufruf 13.06.2013.

[29] Simon Dixon. Automatic extraction of tempo and beat from expressive performances. http://www.cs.washington.edu/education/courses/cse590m/08wi/Dixon%20-%20Automatic%20Extraction%20of%20Tempo%20and%20Beat%20from%20Expressive%20Performances.pdf, 2001. Letzter Aufruf 26.3.2013.

[30] Gunnar Eisenberg. *Identifikation und Klassifikation von Musikinstrumentenklängen in monophoner und polyphoner Musik.* Cuvillier Verlag, Göttingen, 1. auflage edition, 2008.

[31] M. Longari und E. Pollastri G. Agostini. Musical instrument timbres classification with spectral features. http://delivery.acm.org/10.1145/1290000/1283262/p5-agostini.pdf, 2002. Letzter Aufruf 1.4.2013.

[32] P. Gács. *On the symmetriy of algorithmic information.* Soviet Mathematics Doklady, 1974. Letzter Aufruf 13.06.2013.

[33] M. Li P. M. B. Vitányi und W. Zurek H. Bennett, P. Gács. Information distance. http://www.cs.bu.edu/faculty/gacs/papers/info-distance.pdf, 1998. Letzter Aufruf 13.06.2013.

[34] Luis Gustavo Martins Luis Paulo Reis João Lobato Oliveira, Fabien Gouyon. Ibt: A real-time tempo and beat tracking system. http://ismir2010.ismir.net/proceedings/ismir2010-50.pdf. Letzter Aufruf 3.4.2013.

[35] Igor Karpov. Hidden markov classification for musical genrres. http://www.cs.utexas.edu/~ikarpov/Rice/comp540/finalreport.pdf, 2002. Letzter Aufruf 13.06.2013.

[36] Anssi Klapuri. Sound onset detection by applying psychoacoustic knowledge. http://www.cs.tut.fi/sgn/arg/music/icassp99.pdf, 1999. Letzter Aufruf 1.4.2013.

[37] Beth Logan. Mel frequency cepstral coefficients for music modelling. http://apotheca.hpl.hp.com/ftp/pub/compaq/CRL/publications/logan/musicir_paper.pdf, 2000. Letzter Aufruf 1.4.2013.

[38] Manuel Alfonseca und Alfonso Ortega Manuel Cebri Án. Common pitfalls using the normalized compression distance: What to watch out for in a compressor. http://www.ims.cuhk.edu.hk/~cis/2005.4/01.pdf, 2005. Letzter Aufruf 13.06.2013.

[39] Noelia Alcaraz Meseguer. Speech analysis for automatic speech recognition. http://ntnu.diva-portal.org/smash/get/diva2:347957/FULLTEXT01, 2009. Letzter Aufruf 13.06.2013.

[40] Benoit Meudic. Automatic meter extraction from midi files. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.5.7461&rep=rep1&type=pdf, 2002. Letzter Aufruf 1.4.2013.

[41] Yang Hong und Kenny Kao Michael Haggblade. Music genre classification. http://cs229.stanford.edu/proj2011/HaggbladeHongKao-MusicGenreClassification.pdf, 2011. Letzter Aufruf 13.06.2013.

[42] Xin Li Bin Ma und Paul M. B. Vitányi Ming Li, Xin Chen. Similarity metric. http://arxiv.org/pdf/cs/0111054v3, 2004. Letzter Aufruf 13.06.2013.

[43] Aaron Master und Craig Sapp Patricio de la Cuadra. Efficient pitch detection techniques for interactive music. https://ccrma.stanford.edu/~pdelac/research/MyPublishedPapers/icmc_2001-pitch_best.pdf, 2001. Letzter Aufruf 13.06.2013.

[44] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. http://www.cs.cornell.edu/Courses/cs4758/2012sp/materials/hmm_paper_rabiner.pdf, 1989. Letzter Aufruf 13.06.2013.

[45] Paul M. B. Vitányi und Ronald de Wolf Rudi Cilibrasi. Algorithmic clustering of music. http://arxiv.org/PS_cache/cs/pdf/0303/0303025v1.pdf, 2003. Letzter Aufruf 13.06.2013.

[46] R. Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. http://www.cs.gsu.edu/~wkim/index_files/papers/sibson.pdf, 1973. Letzter Aufruf 13.06.2013.

[47] George Tzanetakis. Musical genre classification of audio signals. http://jimi.ithaca.edu/~dturnbull/research/MusicGenre/docs/tsap02gtzan.pdf, 2002. Letzter Aufruf 26.03.2013.

[48] A. Galembo und A. Askenfelt. Measuring inharmonicity through pitch extraction. http://www.speech.kth.se/prod/publications/files/qpsr/1994/1994_35_1_135-144.pdf, 1994. Letzter Aufruf 13.06.2013.

[49] Markus Schedl und Arthur Flexer. Putting the user in the center of music information retrieval. http://ismir2012.ismir.net/event/papers/385-ismir-2012.pdf, 2012. Letzter Aufruf 26.02.2013.

[50] Yanni Panagakis und Constantine Kotropoulos. Music structure analysis by ridge regression of beat-synchronous audio features. http://ismir2012.ismir.net/event/papers/271-ismir-2012.pdf, 2012. Letzter Aufruf 26.3.2013.

[51] Antonio Pertusa und José M. Iñesta. Multiple fundamental frequency estimation using gaussian smoothness. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4517557, 2008. Letzter Aufruf 13.06.2013.

[52] S. Jothilakshmi und N. Kathiresan. Automatic music genre classification for indian music. http://www.ipcsit.com/vol41/010-ICSCA2012-S020.pdf, 2012. Letzter Aufruf 1.4.2013.

[53] Rudi Cilibrasi und Paul M. B. Vitányi. Clustering by compression. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.3082&rep=rep1&type=pdf, 2005. Letzter Aufruf 13.06.2013.

[54] Fabien Gouyon und Perfecto Herrera. Determination of the meter of musical audio signals: Seeking recurrences in beat segment descriptors. http://mtg.upf.edu/node/311, 2003. Letzter Aufruf 1.4.2013.

[55] Juha Kärkkäinen und Peter Sanders. Simple linear work suffix array construction. http://www.cs.cmu.edu/~guyb/realworld/papersS04/KaSa03.pdf, 2003. Letzter Aufruf 13.06.2013.

[56] Ming Li und Ronan Sleep. Genre classification via an lz78-based string kernel. http://ismir2005.ismir.net/proceedings/1116.pdf, 2005. Letzter Aufruf 13.06.2013.

[57] David L. Wessel. Timbre space as a musical structure. http://cnmat.berkeley.edu/system/files/attachments/timbre-space.pdf, 1979. Letzter Aufruf 1.4.2013.

Sitzungsprotokolle

Sitzungsprotokoll

Bachelor-Arbeit	Music Classification
Datum	21. Februar 2013
Teilnehmer	Prof. Dr. Josef Joller Curdin Barandun Stefan Derungs Gino Paulaitis

Traktanden:

- Kick-Off Meeting zur Bachelor-Arbeit (BA)

Ergebnisse:

- Allgemeines:
  - Der fixe regelmässig stattfindende Sitzungs-Termin muss ab der 2. Semesterwoche noch festgelegt werden.
  - Nach jeder Sitzung ist ein Protokoll zu erstellen, welches am Ende den Projektplan ergeben soll.
- Zu Beginn wird die BA in zwei Phasen eingeteilt – einer ersten Recherche-Phase und einer darauffolgenden Phase, wo die Recherche-Ergebnisse vertieft analysiert werden.
- Erste Phase – Die Recherche-Phase:
  - Dient dazu, einen Überblick über das Thema Musik-Klassifizierung zu gewinnen. Dabei soll nach typischen Ansätzen Ausschau gehalten werden (Kolmogorov Komplexität? Neuronale Netze? Grammatiken? …?).
  - Aus ihr soll folgendes hervorgehen:
    - Was wird derzeit im Bereich Musik-Klassifizierung gemacht? Wie wird es gemacht? Wer macht es?
    - Gibt es spezielle Algorithmen/Verfahren, die besonders häufig eingesetzt werden?
    - Welches sind die Fragen, die in den gefundenen Dokumenten offengelassen werden?
  - Nach dieser Phase wird der Stand der gefundenen Verfahren „eingefroren“, d.h. dieser Stand bildet die Ausgangslage (ToDo-Liste), wie weiter für die BA vorgegangen werden soll.
  - Diese Phase soll in der zweiten, spätestens aber in der dritten Semesterwoche abgeschlossen sein.
- Zweite Phase – Analyse und Vertiefung:
  - In dieser Phase sollen die Ergebnisse der ersten Phase analysiert und idealerweise anhand von Beispielen getestet werden.
- Erstes Arbeits-Kapitel:
  - Die Ergebnisse der Recherche-Phase werden hier festgehalten. Dabei werden die gefundenen Klassifizierungs-Verfahren kurz beschrieben.
  - Ebenfalls soll einleitend festgehalten werden, wofür die Recherche-Phase dient und was das Ziel der BA ist.
- Idee, wie das zu entwickelnde Tool am Ende aussehen könnte:
  - Das Tool arbeitet mit mehreren verschiedenen Algorithmen.
  - Schön wäre zum Beispiel ein Ansatz, wo man die verwendeten Algorithmen nach Bedarf beliebig austauschen kann, wobei der Benutzer aber davon nichts merkt (eine Art Plug-In Architektur).
- Das Thema Musik-Generierung ist kein Bestandteil dieser Arbeit.

Sitzungsprotokoll

Bachelor-Arbeit	Music Classification
Datum	1. März 2013
Teilnehmer	Prof. Dr. Josef Joller Curdin Barandun Stefan Derungs Gino Paulaitis

Traktanden:

- Abnahme/Ergänzungen Sitzungsprotokoll vom 21.02.2013
- Ergebnisse der Recherche-Phase vorlegen
- Weiteres Vorgehen besprechen

Ergebnisse:

- Sitzungsprotokoll vom 21.02.2013 ist Ok.
- Als Nächstes soll:
    - die Recherche-Phase fertig dokumentiert werden
    - nach verschiedenen MIR-Tools gesucht werden. Diese sollten dann auf Verwendbarkeit für die BA untersucht werden. Für diesen Task wird provisorisch 1 Woche Bearbeitungszeit festgelegt (kann bei Bedarf aber auf 2-3 Wochen verlängert werden).
- Allgemeines: grundsätzlich müssen nicht wöchentlich Meetings stattfinden, sondern nur nach Bedarf oder bei vorzeitigem Abschluss einer Phase.
- Besprechungsnotizen:
    - evtl. Kompressions- & Vektor-Ansatz vergleichen
    - Feature Extraction – Problematik: man muss sich auf ein paar Features festlegen. ➔ mit verschiedenen Feature-Kombinationen testen, wo die Ergebnisse optimal sind
    - falls in einem Paper ein interessanter Ansatz drin steht, den wir weiter verfolgen möchten, die Autoren evtl. anschreiben und um weitere Infos bitten.
    - Meta-Informationen: Indexe bilden und evtl. ans Datei-Ende (nach dem EOF) anhängen?

Sitzungsprotokoll

Bachelor-Arbeit Music Classification

Datum 7. März 2013

Teilnehmer Prof. Dr. Josef Joller
Curdin Barandun
Stefan Derungs
Gino Paulaitis

Traktanden:

- Abnahme/Ergänzungen Sitzungsprotokoll vom 01.03.2013
- Ergebnisse der Phase 2 vorlegen:
    - gefundene/analysierte Tools
    - Ideen für die Umsetzung in der BA
- Weiteres Vorgehen besprechen

Ergebnisse:

- Protokoll vom 01.03.2013 ok.
- Bezüglich Ergebnisse der Phase 2:
    - In den gefundenen Tools:
        - Welche Algorithmen werden verwendet
        - Was können wir besser machen?
            - Geschwindigkeit
            - Präzision
            - Funktionalität ausbauen (z.B. wo sind Schwächen eines Algorithmus)
            - Komplexität (z.B. andere Masse / Algorithmus „aufblähen“ oder “schlanker“ machen)
- Idee einer Playlist-generierenden Software auf Basis von Musik-Wünschen: man soll Musikstücke angeben können und die Software versucht dann möglichst „weiche“/“schöne“ Übergänge zwischen den Liedern aufzubauen, indem es andere Lieder dazwischen schaltet. Genauere Beschreibung der Idee siehe Abschnitt „Ideen“ im Arbeits-Entwurf.
    - Idee ist ok.
    - Vorgehen:
        - jAudio analysieren
        - Welche Features verwenden und mit Tests prüfen?
        - Clustering-Verfahren analysieren.
        - Nutzer wollen wissen, warum ein Entscheid entsprechend gefällt wurde.
        - Nutzer sollen in die Playlist eingreifen können, falls sie nicht einverstanden sind mit den Vorschlägen ➔ „user assisted“
        - Zeitrahmen: ca. 2 Wochen

Sitzungsprotokoll

Bachelor-Arbeit Music Classification

Datum 25. April 2013

Teilnehmer Prof. Dr. Josef Joller
Curdin Barandun
Stefan Derungs
Gino Paulaitis

Traktanden:

- Bisherige Erkenntnisse und Ergebnisse vorlegen
- Weiteres Vorgehen besprechen

Ergebnisse:

- Allgemeines:
  - Präsentation der BA findet in der Regel 1 Woche nach Abgabe statt.
  - Empfehlung zum Vorgehen:
    - Eine Vermutung aufstellen: was wollen wir beweisen?
    - Einen Ansatz verfolgen.
      Hat jemand schon etwas gemacht? Falls ja, können wir es verbessern? Falls ja, Beweis!
    - Resultate auswerten: Vermutung bestätigt? Beweis? Begründungen? Gibt es Unterschiede zur anfänglichen Vermutung? Wenn ja, wieso?
  - Für Tests besteht die Möglichkeit bei IBM Zugang zu einem Grossrechner zu erhalten.
- Bisherige Erkenntnisse und Ergebnisse:
  - Derzeit 1-2 Wochen im Rückstand, halten wir aber für aufholbar.
  - Curdin:
    - hat sich mit Harmonie-Findung beschäftig; jAudio dabei nicht optimal
    - hierarchisches Clustering getestet
    - Redundanzfindung & Kompression für den Feature-Vergleich: herkömmliche Algorithmen können nicht verwendet werden. Es ist schwer, Redundanzen in Double-Werten zu finden; daher das Ziel, Doubles auf Integers abzubilden. Für die Redundanzfindung kommt anschliessend ein Suffix-Array zum Einsatz.
  - Stefan:
    - Hat sich mit Rhythmen-/Tempo-Erkennung beschäftigt
    - Aubio getestet => gute Ergebnisse erzielt => von C nach Java umgeschrieben
    - Versucht Ergebnisse/Genauigkeit zu verbessern durch paralleler Auswertung mit verschiedenen Onset Detection Functions.
    - Problem: wie Ergebnisse nun sinnvoll zusammenführen => verschiedene Ansätze getestet.
  - Gino:
    - hat sich mit der Analyse/Erkennung von Grundtönen beschäftigt
    - Ziel und Idee: mehrere Grundtöne erkennen; Daten ähnlich gut extrahieren können, wie sie bei MIDIs vorliegen
    - Problem: 1 Lied bzw. ein Window enthält nicht nur 1 Grundton, sondern immer mehrere (Polyphonie)
    - hat noch eine Idee/Ansatz um mehrere Grundtöne zu erkennen; sonst mit nur 1 Grundton arbeiten.
- Feedback des Betreuers:

    - Allgemein sehr positives Feedback. Findet die Aufteilung der Arbeit auf 3 Teilgebiete gut, sodass jeder Student etwas bearbeiten kann.
    - Wichtiger Hinweis: alles gut mit Dokumentation und Beweisen absichern (im Hinblick darauf, falls die Arbeit publiziert werden sollte)
    - Evtl. Bedenken wegen allgemeinem Zeitrahmen/Zeitdruck, hält den Rückstand aber auch für aufholbar.
- Weiteres Vorgehen
    - Die 3 Teil-Prototypen der Studenten zu einem einzigen Prototyp vereinen.
    - Tests durchführen, wie Features nun kombiniert werden können und wie sich dadurch die Ergebnisse verändern.