Goethe-Universität Frankfurt am Main
Fachbereich Informatik und Mathematik

Diplomarbeit

Entwurf, Implementierung und Bewertung von Methoden zur Realisierung eines personalisierten Internet-Radioangebotes mit Live-Moderationen

Marc Adler

21. Juli 2011

Betreuer:
Prof. Dr. Manfred Schmidt-Schauß
Professur für Künstliche Intelligenz und Softwaretechnologie

Hiermit erkläre ich, dass ich die vorliegende Diplomarbeit ohne fremde Hilfe und nur unter Verwendung der zulässigen Mittel sowie der angegebenen Literatur angefertigt habe.

Frankfurt am Main, 26. Juli 2011

_____
Marc Adler

Inhaltsverzeichnis

1 Zusammenfassung

Diese Diplomarbeit betrachtet die Möglichkeiten zur Entwicklung eines neuartigen, Nutzerprofil-gestützten Radioangebotes. Dieses kombiniert live gesendete Moderationen sowie Informations- und Unterhaltungselemente mit einem für jeden Hörer individuell berechneten Musikprogramm, dessen Titel anhand persönlicher musikalischer Vorlieben des Nutzers ausgewählt werden. Hierbei wird der Informationswert und Produktionsstandard eines redaktionell betreuten Radiosenders mit den Vorteilen einer auf Nutzerprofilen basierenden Musikzusammenstellung kombiniert. Um dies zu erreichen, wurden Eigenarten und Produktionsschritte eines Radiosenders analysiert und aus den gewonnenen Erkenntnissen Möglichkeiten zur Einbindung personalisierter Musikprogramme entwickelt. Es wurde ein Konzept zur Planung und Beschreibung von Radio-Sendungen ohne konkrete Musikfestlegung entworfen und ein dazugehöriges Austauschformat auf XML-Basis entwickelt.

Es konnte gezeigt werden, dass sich die Erstellung der personalisierten Musikprogramme als Rucksackproblem modellieren lässt. Mit Hilfe von Daten eines Anbieters personalisierter Musikdienste und unter Verwendung von Sendeprotokollen eines deutschen Privatradio-Senders wurde die Erstellung eines personalisierten Radioprogramms mit unterschiedlichen Heuristiken simuliert. Dabei wurde festgestellt, dass trotz ähnlicher erwarteter Leistungen, nicht alle Heuristiken gleich gut geeignet sind um die betrachtete Problemstellung zu lösen. Einzelne Heuristiken erzeugten bei der Berechnung des personalisierten Radioprogramms deutlich schlechtere Ergebnisse als bei zufälligen Probleminstanzen vergleichbarer Größe. Außerdem konnte festgestellt werden, dass neben der für das Rucksackproblem üblichen Zielsetzung der Gewinn-Maximierung, für ein Radioprogramm auch die verbleibenden Restkapazitäten in den Rucksäcken von großer Bedeutung sind: diese entsprechen ungefüllter Sendezeit, die der Programmanbieter anderweitig füllen muss. Dabei zeigte sich, dass nicht alle Heuristiken in der Lage waren, neben guten Gegenstandswerten auch eine geringe Restkapazität zu erzeugen, weshalb sie sich nicht zur Erstellung eines personalisierten Musikprogrammes eignen. Im Laufe der Untersuchungen wurde daher der spezialisierte Algorithmus KptreeExp entwickelt, der gute Ergebnisse bei den Gegenstandswerten erzielte und zusätzlich nur sehr wenig Platz in den Rucksäcken ungenutzt lässt.

Ferner konnte nachgewiesen werde, dass eine Verringerung des Berechnungsaufwandes durch Zusammenfassung von Nutzern mit ähnlichem Musikgeschmack nicht möglich ist. Hierfür wurden in Musikprofilen von 17.500 Nutzern nach Übereinstimmungen der gehörten Titel oder gehörten Künstler gesucht. Es stellte sich heraus, dass nur bei etwa 5% der Nutzer deutliche Übereinstimmungen mit anderen Nutzern existieren. Daraus ließ sich schließen, dass eine individuelle Berechnung des Musikprogrammes für jeden Hörer notwendig ist.

2 Einleitung

2.1 Motivation

Mit der weitgehenden Verfügbarkeit schneller Internetzugänge ergeben sich neue Möglichkeiten für die Gestaltung eines Hörfunkprogramms. Derzeitige Hörfunkangebote beschränken sich - aufgrund der technischen Limitierungen der Verbreitung per Funk - darauf jedem Hörer das selbe Programm anzubieten. Eine personalisierte Gestaltung des Programms ist bisher nicht möglich. Dies stellt besonders bei der Musikauswahl ein Problem dar, da verschiedene Hörergruppen sehr unterschiedliche Musikgenres bevorzugen [12]. Um eine möglichst breite Hörerschicht zu erreichen, muss ein Radiosender derzeit eine möglichst massenkompatible Musikauswahl treffen. Die Erstellung einer solchen Auswahl ist dabei mit großem Aufwand verbunden. Radiosender beschäftigen Musikredakteure, die neu erschienene Titel probehören und entscheiden, welche für die gewünschte Zielgruppe geeignet scheinen. Diese Vorauswahl wird durch regelmäßige Hörerbefragungen unterstützt, bei denen zufällig ausgewählten Testpersonen telefonisch Ausschnitte aus Musikstücken zur Bewertung vorgespielt werden. Nur Titel, die bei den befragten Hörern gut ankommen, werden auch in das Programm aufgenommen. Üblicherweise führt ein Radiosender solche Befragungen wöchentlich durch [21]. Offensichtlich wird trotz des großen Aufwandes eine solche allgemeine Musikauswahl nie den genauen Geschmack aller Hörer treffen können. Ferner haben Studien nachgewiesen, dass Hörer dazu neigen, einem einmal gewählten Radiosender so lange treu zu bleiben, bis ein Titel gespielt wird, der ihnen nicht gefällt. Daher versuchen Radioanbieter nicht nur Titel zu spielen die dem Publikum besonders gefallen, sondern vor allem solche Titel zu vermeiden, die den Hörern nicht zusagen („reverse programming“) [4].

Eine Lösung für dieses Problem versuchen Anbieter wie *Pandora*, *Grooveshark* oder *last.fm* im Internet zu finden, indem sie Profile vom Musikgeschmack ihrer Hörer erstellen und damit für jeden Hörer einen personalisierten Musikstream anbieten. Bisher sind allerdings all diese Anbieter reine Musik-Abspiel-Systeme, ohne den zusätzlichen Mehrwert, den ein redaktionell betreutes Radioprogramm bietet: Informationen, Unterhaltung, Interaktion mit dem Hörer - wie zum Beispiel Gewinnspiele oder Talk-Formate - gibt es nicht. In dieser Arbeit soll untersucht werden, ob und wenn ja, mit welchen Methoden, sich eine Verbindung dieser beiden Konzepte realisieren lässt: ein Radioprogramm mit Live-Inhalten und personalisierter Musik. Durch die Verbreitung individueller Streams wäre ein solches Angebot streng genommen kein Radio mehr, sondern würde laut Rundfunkstaatsvertrag unter die *Telemedien* fallen [38]. Aus Gründen der Einfachheit und der inhaltlichen Nähe zu echten Radioprogrammen soll aber in dieser Arbeit auf die rein rechtliche Unterscheidung verzichtet werden und auch ein solches personalisierbares Programm mit redaktionellen Inhalten als Radio bezeichnet werden.

Auch beim Wortanteil des Programmes bestehen unterschiedliche Präferenzen in verschiedenen Nutzergruppen ([41], [39]). Um diesen unterschiedlichen Ansprüchen Rechnung zu tragen, soll ein System entwickelt werden, mit dem

jeder Hörer individuell Einfluss auf das Verhältnis von Musik zu Wortbeiträgen in seinem personalisierten Programm nehmen kann.

2.2 Lösungsansatz

Bei der Realisierung eines Live-Programmes mit personalisierter Musik ergibt sich ein zentrales Problem: jeder Benutzer möchte gerne unterschiedliche Musiktitel mit unterschiedlichen Längen hören, Moderationen, Live-Reportagen und Hörerinteraktionen sollen aber an alle User zur gleichen Zeit ausgestrahlt werden. Ein Lösungsansatz, der diesen beiden konkurrierenden Ansprüchen gerecht wird, lässt sich aus den Arbeitsweisen der Redaktion eines Radiosenders entwickeln:

Bereits heute verwenden diese einen sogenannten Sendungsfahrplan, um den Ablauf einer Programmstunde im Voraus festzulegen. Darin ist die zeitliche Reihenfolge aller Elemente einer Sendung vorgegeben. Ein solcher Sendungsfahrplan soll als Grundlage auch für das personalisierte Radio erstellt werden, mit dem Unterschied, dass im Voraus lediglich Zeitfenster für Musik festgelegt werden, anstatt die Titel genau zu benennen. Eine Software versucht dann aus dem Titelvorrat eines jeden Users die Musikstücke so zu kombinieren, dass sie das Zeitfenster möglichst vollständig ausfüllen. Mit welchen Algorithmen dies am besten gelingt, soll in dieser Arbeit untersucht werden.

Des weiteren soll überprüft werden, ob es nötig ist, die Musikberechnung für jeden User einzeln auszuführen oder ob sich Gruppen von Nutzern mit einem derart ähnlichen Musikgeschmack finden lassen, dass für sie ein gemeinsames Programm berechnet werden kann.

Für beide Teilprobleme soll die Nutzerdatenbank des Anbieters *last.fm* benutzt werden.

2.3 Aufgabenstellung

Diese Arbeit umfasst folgende Teil-Aufgaben:

- Entwickeln einer Software, die in der Lage ist, einen im XML-Format vorliegenden Ablaufplan für eine oder mehrere Sendestunden zu lesen und für eine gegebene Menge von Nutzern möglichst effizient die darin enthaltenen Musikblöcke mit Titeln aus der Menge der jeweiligen Lieblingsstücke zu füllen.

- Gewinnen und Aufbereiten von Musikdaten über *last.fm*.

- Testen und Bewerten verschiedener algorithmischer Ansätze zur Ermittlung geeigneter Titelkombinationen. Dabei soll neben der Effizienz der verwendeten Algorithmen vor allem die Qualität der gefundenen Ergebnisse untersucht werden. Das heißt: Der Algorithmus sollte in den einzelnen Blöcken möglichst geringe Fehlzeiten erzeugen und die entstehenden Fehlzeiten sollten möglichst gleichmäßig über eine Sendestunde verteilt sein - ein Algorithmus der beispielsweise in 5 Blöcken jeweils 5 Sekunden

zu wenig füllt, ist einem Algorithmus, der einmal 25 Sekunden Fehlzeit erzeugt, vorzuziehen.

- Entwickeln von Methoden zur Ermittlung von Nutzern mit gemeinsamem Musikgeschmack. Hierbei soll überprüft werden, ob sich die Berechnung der Titellisten vereinfachen lässt, indem Gruppen von Nutzern gemeinsam bearbeitet werden. Dafür soll in Nutzerdaten von *last.fm* nach Clustern gesucht werden, so dass bei den darin enthaltenen Nutzern eine möglichst große Übereinstimmung in den bevorzugten Titeln vorliegt. Neben verschiedenen Cluster-Algorithmen sollen auch unterschiedliche Kriterien, anhand der sich Nutzer zusammenfassen lassen, geprüft und bewertet werden.

2.4 Aufbau der Arbeit

Diese Arbeit ist in neun Kapitel unterteilt. Nach der Zusammenfassung in Kapitel 1 beschreibt Kapitel 2 die Motivation und den Umfang der Arbeit und führt das Konzept eines personalisierbaren Radios ein. In Kapitel 3 wird eine kurze Einführung in bereits existierende Angebote gegeben. Es werden klassische Radioangebote betrachtet und einige derzeit verwendete Konzepte der Programmgestaltung erläutert. Unterkapitel 3.2 betrachtet die Eigenschaften und Funktionen derzeit angebotener personalisierbarer Radioangebote. Eine genauere Analyse des Anbieters last.fm, dessen Datenbestand für diese Arbeit verwendet wurde, erfolgt in Unterkapitel 3.3. Ein besonderes Augenmerk liegt hierbei auf den von *last.fm* gesammelten Datenbeständen und der angebotenen Programmschnittstelle. Kapitel 4 bietet einen theoretischen Blick auf die Modellierung der Musikzusammenstellung eines personalisierbaren Radios als Rucksackproblem und beschreibt die in dieser Arbeit verwendeten Lösungsalgorithmen. Der Ansatz, das Radioprogramm für eine Gruppe Nutzer mit ähnlichem Geschmack gemeinsam zu berechnen. wird in Kapitel 5 eingeführt. Dort erfolgt auch eine theoretische Betrachtung der verwendeten Algorithmen zur Cluster-Bildung und Distanzbewertung. Kapitel 6 beschreibt ein ausführliches Konzept zur Umsetzung eines personalisierbaren Radios mit Live-Moderationen und variablem Wortanteil. Die experimentelle Implementierung eines Systems zur Sendeplanerzeugung - unter Verwendung der zuvor theoretisch betrachteten Algorithmen und einer mit den Daten von *last.fm* erzeugten Musikdatenbank - wird in Kapitel 7 erläutert. Die Testergebnisse dieser experimentellen Anordnungen werden im Anschluss in Kapitel 8 präsentiert. Die Arbeit schließt mit einer Bewertung der Ergebnisse, einem Fazit und einem Ausblick auf offene Fragen und mögliche weiterführende Arbeiten in Kapitel 9.

2.5 Vokabular

Zur einfacheren Verständlichkeit der Arbeit werden an dieser Stelle einige Bezeichnungen eingeführt, die, falls nicht anders vermerkt, Gültigkeit für die komplette Arbeit besitzen:

- *Fehlbeträge/Fehlzeiten* bezeichnet die Differenz zwischen der geplanten Länge eines Musikblocks und der Summe der Längen der für diesen Block ausgewählten Musiktitel. Diese Differenz muss bei der Ausstrahlung des Programms mit alternativen Inhalten, wie beispielsweise Trailern, gefüllt werden.

- Mit dem Begriff *klassische Radioangebote* sollen in dieser Arbeit alle Angebote bezeichnet werden, die eine redaktionell betreute Zusammenstellung aus Musik und/oder gesprochenem Wort per Funk oder Internet an eine größere Gruppe von Hörern verbreiten, welche alle zeitgleich das selbe Programm zu hören bekommen. Ein Hörer hat keinen direkten Einfluss auf den Ablauf des von ihm empfangenen Programms.

- *Musikblock* bezeichnet einen für Musiktitel reservierten Zeitraum im Programm. Dieser kann unausgefüllt lediglich die Position und Dauer enthalten oder in einem Sendungsplan mit den ausgewählten Titeln gefüllt sein.

- *Sendungsplan* (oder *Sendungsfahrplan*) entsteht aus einem Sendungsschema, in dem alle Musikblöcke durch spezifische Titelangaben ersetzt wurden. Ein solcher Plan wird in dieser Arbeit individuell für jeden Zuhörer erzeugt.

- *Sendungsschema* bezeichnet die geplante Struktur einer Sendung mit allen festen Inhalten. In einem Sendungsschema sind lediglich Musikblöcke als Platzhalter eingeplant. Individuelle Titel sind, solange nicht aus redaktionellen Gründen notwendig, nicht angegeben.

- *Titelbibliothek* bezeichnet die Menge der Musiktitel, die einem Nutzer zugeordnet werden. Diese werden üblicherweise mit Hilfe eines Geschmacksprofils erstellt oder aus der Menge der in der Vergangenheit gehörten Titel gewonnen. Alle Titel der Titelbibliothek eines Nutzers können genutzt werden um den Sendungsplan eines Nutzers zu erzeugen.

3 Radioangebote

Ziel dieser Arbeit ist es, ein personalisierbares Radioprogramm zu erschaffen, das sich hinsichtlich Gestaltung und Format möglichst wenig von den heute üblichen Standards eines professionell betriebenen Programms unterscheidet. Im Gegensatz zu früheren Zeiten sind die meisten Radioangebote heute stark „durchformatiert", das heißt, sie folgen in der Präsentation und Zusammenstellung der Musiktitel einem festen, oftmals durch Marktforschung aufwändig überprüften, Muster. Um möglichst große Teile dieser professionellen Standards auch in das personalisierbare Radio übernehmen zu können, ist es notwendig, erst einmal die heute verbreiteten Eigenschaften von Radioangeboten näher zu betrachten.

3.1 Klassische Radioangebote

3.1.1 Frühe Entwicklung

Erste Grundlagen für die Entwicklung des Radios wurden bereits Ende des 19. Jahrhunderts mit der Erfindung der drahtlosen Telegraphie gelegt. Lange Zeit wurde diese dem Italiener Guglielmo Marconi (* 25. April 1877, † 20. Juli 1973 [44]) zugeschrieben, der seine Erfindung im Jahr 1896 patentierte[1] [18]. Ähnliche Experimente wie Marconi führte in den USA aber auch Nikola Tesla (* 10. Juli 1856, † 7. Januar 1943 [44]) durch, der seinen fortschrittlicheren Entwurf im Jahr 1897 patentierte[2]. Als Marconi die Weiterentwicklungen seiner ursprünglichen Erfindung in den USA patentierte, kam es zu rechtlichen Streitigkeiten zwischen Marconis Firma, der *Marconi Wireless Telegraph co. of America,* und der US Regierung. Im Rahmen dieser Streitigkeiten verwarf der U.S. Supreme Court am 21. Juni 1943 Marconis US-Patente mit der Begründung, dass die patentierten Weiterentwicklungen bereits in Nikola Teslas Patenten enthalten seien[3]. Dies führte dazu, dass einige Autoren Tesla als wahren Erfinder der drahtlosen Telegraphie ansehen [11]. Neben Marconi und Tesla wird auch der Russe Alexander Stepanowitsch Popow (* 16. März 1859, † 13. Januar 1906 [30]) als Erfinder der drahtlosen Telegraphie bezeichnet. Dieser führte seine Experimente zur drahtlosen Übertragung bereits 1895 durch und veröffentlichte seine Erkenntnisse im Januar 1896, allerdings ohne diese zu patentieren. Das von Marconi ein halbes Jahr später in Groß-Britanien patentierte Design entsprach in großen Teilen den von Popow veröffentlichten Entwürfen ([51], [42]). Obwohl diese Technik ursprünglich nur zur drahtlosen Übermittlung von Informationen im Morse-Code gedacht war, lies sie auch zur Übertragung von Sprache nutzen. Dies gelang Reginald Aubrey Fessenden (* 06. Oktober 1866, † 22. Juli 1932 [7]) erstmals am 23. Dezember 1900 über eine Strecke von 1600 Metern, indem er die Sprache als eine Folge von etwa 10.000 Signalen pro Sekunde codierte [6]. Sechs Jahre später, am 24. Dezember 1906 sendete Fessenden eine Ansprache und ein Musikstück von Georg Friedrich Händel, gespielt auf der Violine über

[1] UK Patent No. 12,039

[2] US Patent No. 645,576

[3] *Marconi Wireless Telegraph co. of America v. United States.* 320 U.S. 1. Nos. 369,373. Argued April 9–12, 1943. Decided June 21, 1943

eine Distanz von mehreren hundert Kilometern, welches von zahlreichen Schiffen der US Navy und der United Fruit Company empfangen werden konnte. Diese kurze Zusammenstellung aus Musik und Sprache gilt als erstes Radioprogramm der Geschichte ([7], [14], [6]).

Nach der Entwicklung der technischen Grundlagen dauerte es noch bis zum 22. Dezember 1920, bis mit der Übertragung eines Konzerts, die erste Radiosendung in Deutschland ausgestrahlt wurde [57]. Die Programmverbreitung über Ultra-Kurzwelle (UKW) begann in Deutschland im Jahr 1949 durch den Bayrischen Rundfunk [57]. Bis heute gilt UKW als vorherrschender Standard in der drahtlosen Verbreitung von Radioprogrammen. Versuche, eine digitale Verbreitung von Radioprogrammen (Digital-Audio-Broadcasting, DAB) zu etablieren, waren bisher nicht von Erfolg gekrönt. Heute ist Radio weiterhin ein wichtiges Massenmedium. 97% der Haushalte in der Bundesrepublik Deutschland verfügen über ein Radiogerät [52], im Schnitt hört die deutschsprachige Bevölkerung ab einem Alter von 10 Jahre pro Tag 242 Minuten Radio [13].

3.1.2 Formatierung

Da Radio bisher an alle Hörer die selbe Musik ausliefert, haben sich eine Vielzahl von sogenannten *Formaten* entwickelt. Diese enthalten üblicherweise Vorgaben über die gewünschte Zielgruppe, die Zusammensetzung der Musikauswahl und die inhaltliche Gestaltung des Programms. Auf diese Weise versuchen Radiostationen, ein bestimmtes Publikum möglichst genau anzusprechen und so an den Sender zu binden. Bei einem personalisierbaren Radio entfällt offensichtlich die Format-Komponente der Musikauswahl, die erprobten Formatierungsmethoden für die inhaltliche Gestaltung des Programms sind aber auch für ein personalisierbares Radio von Interesse, um Hörer möglichst lange an das Programm zu binden. Daher soll nachfolgend, basierend auf den Beschreibungen von Frigge, Haas und Zimmer [40] und dem Medialexikon der Burda-Mediengruppe [4], eine kurze Zusammenfassung der Eigenschaften besonders verbreiteter Formate betrachtet werden. Ferner soll erörtert werden, wie Teile der Formate in ein personalisierbares Radio übertragen werden können.

Adult Contemporary (AC) Dieses weit verbreite Format soll vor allem Hörer im Alter von 25 bis 49 Jahren ansprechen. Musikalisch werden vor allem Popmusik und Mainstream-Rock der letzten 3 Jahrzehnte gespielt. Das Programm ist auf eine leichte „Durchhörbarkeit“ ausgelegt, das heißt, es sollen möglichst keine plötzlichen Stilbrüche oder Tempowechsel stattfinden. Die gespielten Titel entsprechen dem breiten Massengeschmack. Üblicherweise folgen drei oder vier Titel aufeinander, Moderationen und Beiträge sind kurz gehalten. Die meisten deutschen Radiostationen nutzen dieses Format [13]. Neben AC existieren eine Reihe von Unterformaten, die sich vor allem in der Musikauswahl unterscheiden:

[4] http://www.medialine.de/deutsch/wissen/medialexikon.php?snr=2485 (Abgerufen am 07.07.11)

- Hot-AC orientiert sich an einer etwas jüngeren Zielgruppe und spielt vor allem Titel der letzten 20 Jahre. Der Anteil an Musik aus den aktuellen Charts ist höher als bei AC.
- Oldie-Based-AC ist der Gegenpart zu Hot-AC und legt seinen musikalischen Schwerpunkt auf ältere Titel. Üblicherweise werden hier vor allem Titel der 1960er bis 1990er Jahre gespielt.
- Soft AC spielt einen höheren Anteil an Balladen als andere AC-Formate.

Contemporary Hit Radio (CHR) Dieses Format soll vor allem junge Hörer im Alter von 14 bis 29 Jahren ansprechen. Im Programm werden vor allem schnelle und derzeit am meisten verkaufte Titel gespielt. Gespielte Titel werden häufig wiederholt. Das Programm selbst wird „schnell gefahren“, dass heißt, Titel werden dicht aneinander geschnitten, oftmals durch treibende Jingles verbunden. Moderationen sind kurz gehalten, bedienen sich oft der Jugendsprache und werden grundsätzlich mit Musik unterlegt. Nachrichten und Informationen werden auf kurze Zusammenfassungen reduziert. Zur Hörerbindung werden neben Gewinnspielen auch zahlreiche Außenaktionen eingesetzt, bei denen Hörer vor Ort an Spielen teilnehmen können.

Middle of the Road (MOR) Kernzielgruppe dieses Formats sind Hörer zwischen 35 und 55 Jahren. Musikalisch werden vor allem ruhige, melodische Popsongs gespielt. Ziel des Programms ist eine ausgewogene Mischung aus Titeln, die üblicherweise zwischen 10 und 30 Jahren alt sind. In diesem Format werden häufig auch längere Wortbeiträge gesendet und politische oder gesellschaftliche Themen näher beleuchtet. Informationen nehmen einen höheren Programmanteil ein als beispielsweise im AC oder CHR-Format. Moderationen erfolgen ruhig und sachlich.

Melodie Melodie ist ein Radioformat, dessen Zielgruppe Hörer über 40 Jahren sind. Das sehr ruhige Programm sendet eine Mischung aus Schlagern und gemäßigten englischen Oldies. Moderationen erfolgen üblicherweise in einem ruhigen Ton.

Bereits diese vier verschiedenen Formatbeschreibungen zeigen, dass ein einzelner Radiosender, trotz individueller Musik, kein vollwertiger Ersatz für alle Radiostationen sein kann. Soll nur eine personalisierte Radiostation betrieben werden, so muss sich ein Anbieter im Moderations- und Beitragsstil auf ein Format festlegen. Wegen des großen Erfolges in der breiten Masse bietet sich hierfür das AC-Format an. Die Integration anderer Eigenarten der verschiedenen Formate ist aber, wenigstens in Teilen, denkbar. So könnte die Wiederholungsfrequenz der Titel und die Gestaltung der Übergänge vom Hörer nach eigenen Wünschen angepasst werden, um so einige Elemente des CHR-Formats realisieren zu können. Ebenso ist es denkbar, das Radioprogramm mit einem hohen

Wortanteil zu produzieren. Auf Wunsch können dann für einige Nutzer einzelne Wortstrecken übersprungen und durch längere Musikblöcke ersetzt werden. So ließe sich das Radio dem MOR-Format annähern. Ein Implementierungs-Vorschlag für einen flexiblen Wortanteil wird in Kapitel 6 vorgestellt.

3.1.3 Vorteile eines Live-Programms

Es stellt sich die Frage nach den Vorteilen eines Live-Programms. Gerade für ein personalisiertes Radio ließen sich viele Probleme umgehen, wenn auch alle redaktionellen Inhalte vorproduziert und zeitversetzt an den Nutzer gesendet würden. Aber auch für ein klassisches Radioprogramm könnte man erwarten, dass am Tagesanfang einmal erstellte und dann über den Tag verteilt ausgestrahlte redaktionelle Programmteile und Moderationen Kosten sparen könnten. Ausreichend Moderationen für einen Tag lassen sich schnell produzieren und danach müssten für den Betrieb des Senders keine Sprecher anwesend sein. Trotzdem sind live erstellte Sendungen weiterhin ein fester Bestandteil vieler Radioprogramme. Ein Grund mag darin liegen, dass so in der laufenden Sendung auf aktuelle Nachrichten reagiert werden kann und durch die stets aktuellen Informationen für die Hörer ein Mehrwert entsteht. Daneben gibt es auch viele Sendeformate, die sich nicht anders realisieren lassen. Dies gilt zum Beispiel für Live-Übertragungen von Sportereignissen, die sich großer Beliebtheit beim Hörer erfreuen. So verfolgten im Jahr 2009 im Durchschnitt deutschlandweit 15,14 Millionen Menschen die samstägliche Übertragung der Fußball-Bundesliga im Radio [54]. Ein solches Format lässt sich aber nur realisieren, wenn ein live im Sender anwesender Sprecher jederzeit auf aktuelle Entwicklungen (wie zum Beispiel Tore in einem Fußballspiel) reagieren und in das laufende Programm eingreifen kann.

Neben direkter Berichterstattung von aktuellen Ereignissen und Veranstaltungen ermöglicht ein Live-Programm zusätzlich die Durchführung von interaktiven Programmteilen, wie zum Beispiel Gewinnspielen oder Call-Ins (Hörer können per Telefon im Studio anrufen und sich zu den gerade behandelten Themen äußern). Derartige Aktionen erhöhen die Bindung eines Hörers an „sein Programm“ und führen außerdem zu einer erhöhten Erinnerung von Werbebotschaften, die im Umfeld solcher Aktionen gesendet werden [16]. Dies ist besonders für private Radioanbieter interessant, welche sich über die Ausstrahlung von Werbung finanzieren.

3.1.4 Radio im Internet

Die meisten großen Radio-Sender sind heutzutage auch im Internet präsent und nutzen das neue Medium als zusätzlichen Verbreitungskanal für ihr Programm (Simulcast), da sich die Nutzung von Internetstreams immer größerer Beliebtheit erfreut. So haben im Jahr 2010 bereits 16,7% der deutschsprachigen Bevölkerung Deutschlands ab 10 Jahre Radio auch mindestens einmal per Internet gehört, 4,4 % besitzen sogar ein sogenanntes IPRadio-Gerät, also ein speziell für den Empfang per Internet konstruiertes Radio, mit dem sich Internetstreams auch

abseits von PC oder Laptop nutzen lassen [13]. Noch stärker wird die Tendenz zum Internet-Radio in der Gruppe der 14 bis 29 jährigen deutlich. Hier hört bereits jeder Sechste regelmäßig (mindestens einmal in der Woche) Radio über seinen PC oder Laptop, 7% nutzen ihr Handy oder Smartphone zum Empfang von Radio [2]. Zusätzlich zu der einfachen Verbreitung ihres regulären Programms nutzen Radiosender das Internet auch um programmbegleitende Informationen anzubieten. So lassen sich in den Internetpräsenzen von Radiosendern beispielsweise zusätzliche Nachrichten, besondere Gewinnspiele, Listen mit gespielten Musiktiteln und ähnliches finden. Oftmals werden auch ausgewählte Inhalte als Podcasts zum Herunterladen angeboten. Damit wird dem Nutzer die Möglichkeit gegeben, sich - in begrenztem Rahmen - ein persönliches Radioprogramm nach seinem Geschmack zusammenzustellen.

Neben den klassischen Radioprogrammen findet sich auch eine große Anzahl von reinen Webradios im Internet. Diese Sender verbreiten ihr Programm ausschließlich über das Internet und können daher die Kosten für terrestrische Sendeanlagen und Frequenzen sparen. Inzwischen gibt es deutlich mehr reine Webradios als klassische Radioangebote - eine Anfrage bei der GEMA[5] ergab, dass dort 1603 deutsche Webradios lizenziert sind (Stand: 10.05.2011). Hinzu kommt noch eine unbekannte Dunkelziffer von Sendern, die keine GEMA-Lizenz für ihr Programm erworben haben. Den reinen Webradios stehen etwa 210 Simulcast-Programme gegenüber [38]. Trotz der großen Zahl von Angeboten erreichen ausschließliche Internetradios nur eine vergleichsweise geringe Hörerzahl. Nur etwa 378.000 Personen, bzw. 0,5% der Bevölkerung ab 14 Jahren, hören täglich einen reinen Webradio-Sender - 913.000 haben schon mindestens einmal einen solchen Sender gehört [2].

3.2 Personalisierbare Musikangebote

Neben den herkömmlichen Radioangeboten existiert im Internet noch eine weitere Gattung von Musikangeboten. Diese sogenannten *personalisierbaren Radios* senden unterschiedliche Musik-Streams an jeden Hörer. Sie enthalten keine redaktionellen Inhalte oder Live-Elemente, bezeichnen sich aber häufig selbst als Radio. Um in dieser Arbeit zwischen solchen Musikdiensten und dem hier entwickelten personalisierbaren Radio mit Live-Moderationen unterscheiden zu können, werden im Folgenden alle Dienste ohne Live-Moderationen als *personalisierbare Musikangebote* bezeichnet.

Üblicherweise nutzen solche Dienste Nutzerfeedback, um ein Geschmacksprofil eines Users zu erstellen und anhand dessen einen personalisierten Musikstream zu erzeugen. Solche Angebote werden zumindest gelegentlich von 5% und mindestens einmal wöchentlich von 2% der Onlinenutzer ab 14 Jahren genutzt. Und auch hier ist eine überdurchschnittliche Nutzung in der Gruppe der jungen User von 14 - 29 Jahren feststellbar [1].

Alle diese Dienste erzeugen für ihren Betrieb Listen von Titeln, die ein Nutzer derzeit wahrscheinlich hören möchte. Dabei fallen die genauen Techniken der

[5] *Gesellschaft für musikalische Aufführungs- und mechanische Vervielfältigungsrechte*, deutsche Verwertungsgesellschaft für Musikrechte

Anbieter meistens unter das Betriebsgeheimnis. Trotzdem bleibt aber festzuhalten, dass bereits eine Vielzahl von Ansätzen existieren, um Listen von „Lieblingstiteln“ für Internetnutzer zu erzeugen. Solche Angebote sind also prinzipiell geeignet, als Basis für ein personalisierbares Radio zu dienen. Die Funktion einiger solcher Angebote soll hier kurz betrachtet werden, um einen Überblick über die heutigen Möglichkeiten von personalisierbaren Internetstreams zu geben.

3.2.1 Entwicklung

Konzepte für personalisierbare Musikangebote wurden im Jahr 2001 sowohl von Conor Hayes und Pádraig Cunningham unter dem Namen *Smart Radio* [23] als auch von David B. Hauver and James C. French unter dem Namen *Flycasting* [22] vorgeschlagen.

Smart Radio basiert auf Usergenerierten Playlists. Ein Nutzer stellt sich aus einer Sammlung von verfügbaren Musiktiteln eine oder mehrere Abspiel-Listen zusammen. Die Titel dieser Liste kann er sich dann als Stream über seinen PC anhören. Alle Titel, die der Nutzer auswählt, erhalten vom System einen bestimmten Punkte-Wert, welcher weiter erhöht wird, wenn eine bestimmte Liste oftmals abgespielt wurde. Zusätzlich hat der Nutzer noch die Möglichkeit jeden einzelnen Titel gesondert mit 0 bis 5 Punkten zu bewerten. Aus den so gewonnenen Bewertungen berechnet der *Smart Radio*-Server Übereinstimmungen mit anderen Nutzern. Hierzu schlagen die Autoren die Verwendung des Pearson Korrelationskoeffizienten vor. Die von den n Nachbarn mit der geringsten Distanz angelegten Playlisten werden dann dem Nutzer als Vorschläge unterbreitet. Diese kann er dann anhören, die Zusammenstellung für sich ändern und die gehörten Titel bewerten.

Flycasting verbessert die Idee eines „Wunsch-Radios“. Für die Autoren dient der Ansatz einer automatisierten Radio-Station, bei der sich Hörer über ein Internet-Formular Titel wünschen können, als Ausgangspunkt. Diese Musiktitel, falls in der Sammlung der Station vorhanden, werden an das Ende der derzeitigen Abspielliste des Radios angehängt und dann zur gegebenen Zeit gesendet. Es stellen also alle Hörer gemeinsam das Programm des Radios zusammen. Als Verbesserung dieses Konzepts werden für *Flycasting* die gewünschten Titel eines jeden Nutzers x in einer Wunsch-History gespeichert. Aus diesen Daten wird eine Künstlerwertung für diesen Nutzer berechnet. Jeder Wunsch eines Titels von Künstler k steigert dessen Wertung. Dabei wird ein exponentieller Dämpfungs-Faktor verwendet, so dass jeder zusätzliche Wunsch eines Musiktitels von Künstler k dessen Wertung weniger stark ansteigen lässt, als der vorhergegangene. Zusätzlich wird der Einfluss des mehrfachen Wünschens des selben Titels stärker gedämpft, als das Wünschen von unterschiedlichen Titeln des selben Künstlers. Ebenso wird die vergangene Zeit seit einem Wunsch berücksichtigt. Wünsche, die weit in der Vergangenheit getätigt wurden, beeinflussen die Wertung weniger stark, als solche, die vor kurzer Zeit geäußert wurden.

Aber nicht nur die vom Nutzer bisher gewünschten Künstler sollen bewertet werden. Mit Hilfe von Collaborative Filtering-Techniken [20] soll ferner eine Vorhersage darüber getroffen werden, wie gut der Nutzer x Künstler bewerten würde, die er bisher noch gar nicht gewünscht hat. Hierzu berechnet *Flycasting* die mittlere quadrierte Distanz (Mean Square Distance, MSD) aller Künstler-Bewertungen zwischen x und allen anderen Nutzern. Die Nutzer, deren MSD unter einem bestimmten Grenzwert Φ liegt - also solche, die viele Künstler ähnlich wie x bewertet haben - werden als Nachbarschaft von x bezeichnet und zur Vorhersage herangezogen. Die erwartete Bewertung des Künstlers k für Nutzer x entsteht dann aus den gewichteten Bewertungen der Nachbarschaft von x:

$$w(x,k) = \frac{\sum_{i \in (NH(x) \cap P(k))} (wt(x,i) \cdot W(i,k))}{\sum_{i \in (NH(x) \cap P(k))} wt(x,i)}$$

$NH(x)$ bezeichnet die Nachbarschaft von x. $P(k)$ ist die Menge aller Nutzer, die sich einen Titel von k gewünscht haben. $wt(x,y) = 1 - MSD(x,y)$ und $W(i,k)$ bezeichnet die Wertung von Nutzer i für Künstler k. Collaborative Filtering ermittelt seine Vorhersagen also unter der Annahme, dass Nutzer, die bei den gemeinsam gewünschten Liedern einen ähnlichen Geschmack haben, diesen auch für Titel teilen, welche bisher nur ein Teil der Nachbarschaft gewünscht hat. *Flycasting* berechnet für alle zuhörenden Nutzer auf diese Weise die Künstlerbewertung und ermittelt dann die Künstler, die im derzeitigen Publikum besonders populär sind. Deren Musik wird dann in das Programm aufgenommen. Damit beschreibt *Flycasting* genau genommen keinen personalisierbaren Musikdienst, da jeder Nutzer das selbe Programm hört. Dieses passt sich aber aber dynamisch an die derzeitige Hörergruppe an. Die hierbei genutzten Techniken lassen sich auch verwenden, um für jeden Nutzer ein eigenes Programm zu erstellen.

3.2.2 Bekannte Vertreter

Bis heute haben sich zahlreiche Dienste entwickelt, die ihren Nutzern mit verschiedensten Ansätzen ein personalisiertes Musikangebot bieten wollen. Beispielhaft sollen hier die unterschiedlichen Herangehensweisen einiger Anbieter kurz vorgestellt werden.

Pandora[6] setzt für sein Angebot auf manuell generierte Informationen zu Musiktiteln. Im Rahmen des *Music Genome Projects* werden musikalische Werke anhand verschiedener Charakteristiken, wie zum Beispiel Instrumentalisierung, Rhythmus, Harmonie oder Art des Gesangs klassifiziert[7]. Insgesamt werden so über 400 unterschiedliche Charakteristiken erfasst, deren Einordnung durch Projektmitarbeiter mit musiktheoretischer Ausbildung erfolgt[8]. Um eine persönliche Radiostation zu erzeugen, muss ein Nutzer einen Künstler oder einen Titel als

[6] http://www.pandora.com
[7] http://www.pandora.com/mgp.shtml (Abgerufen am 20.05.2011)
[8] http://pandora.com/corporate/mgp (Abgerufen am 20.05.2011)

Seed angeben. Pandora beginnt dann, ausgehend von dem Seed, Titel mit ähnlichen Charakteristiken zu spielen. Ein Nutzer kann die Entwicklung der Station beeinflussen, indem er einzelne Titel als besonders gut markiert, so dass dessen Charakteristiken stärker gewichtet werden. Ferner können Titel auch als schlecht markiert werden oder weitere Seeds der Station hinzugefügt werden. Insgesamt wurden seit Januar 2000 von Pandora nach eigenen Angaben über 800.000 Titel analysiert und zum Streaming zur Verfügung gestellt. Aufgrund von Lizenzeinschränkungen ist Pandora bisher nur für Nutzer in den USA verfügbar und erreicht dort 80 Millionen User[9].

Slacker[10] verfolgt ein stark am herkömmlichen Radio orientiertes Prinzip. Neben der Möglichkeit, sich eigene sogenannte *Radiostationen* zu erstellen, welche die eigene Musikauswahl und von *Slacker* als ähnlich erachtete Titel spielen, bietet der Anbieter seinen Nutzern auch eine Reihe von professionellen DJs zusammengestellten *Radiostationen* zu verschiedenen Themen und Genres. Darunter befinden sich auch Stationen die beispielsweise Nachrichten oder personalisierte Sportberichterstattung liefern. Bei ausgewählten Stationen lassen sich zusätzlich voraufgezeichnete Moderationen in das Programm einfügen. Stationen können vom Hörer an den eigenen Geschmack angepasst werden, indem sie das Verhältnis von ausgewählten und von *Slacker* vorgeschlagenen Titeln verändern oder den Anteil an besonders populären Stücken beeinflussen. *Slacker* vertreibt auch Software für Mobiltelefone und spezielle Hardware-Player für das eigene Angebot, mit dem der Dienst unterwegs genutzt werden kann. Nach eigenen Angaben verfügt *Slacker* über eine Auswahl von über 8 Millionen Titeln, aus denen Nutzer wählen können[11]. Aufgrund von Lizenzeinschränkungen ist der Dienst bisher nur für Nutzer in den USA und Kanada verfügbar.

Musicovery[12] versucht eine Musikauswahl nach Stimmungslage zu ermöglichen. Nutzer werden aufgefordert, ihre derzeitige Stimmung in einem zweidimensionalen Raum zu verorten. Dabei unterscheidet diese, vom Anbieter *Moodpad* genannte, Funktion in der horizontalen zwischen „Dark“ und „Positive“, in der vertikalen zwischen „Calm“ und „Energetic“ (ein zweites *Moodpad* mit den Achsen „sehr tanzbar“, „wenig tanzbar“ und „hohes Tempo“, „niedriges Tempo“ ist ebenfalls verfügbar) . Alle Titel in der *Musicovery* Datenbank wurden in diese Koordinatensysteme eingeordnet, so dass nach Auswahl der Stimmung ein Musikstream mit den für diesen Punkt hinterlegten Titeln beginnt. Der Stream kann noch weiter den eigenen Wünschen angepasst werden, indem der Erscheinungszeitraum und die Genres der abgespielten Titel eingeschränkt werden können (Abbildung 1). Kennt *Musicovery* zu einem Titel mehrere Versionen, so werden diese als Alternativen angeboten, die in das Musikprogramm aufgenommen

[9] http://blog.pandora.com/press/pandora-company-overview.html (Abgerufen am 20.05.2011)
[10] http://www.slacker.com
[11] http://www.slacker.com/company/pressreleases/05172011-Slacker-Premium.jsp (Abgerufen am 20.05.2011)
[12] http://www.musicovery.com

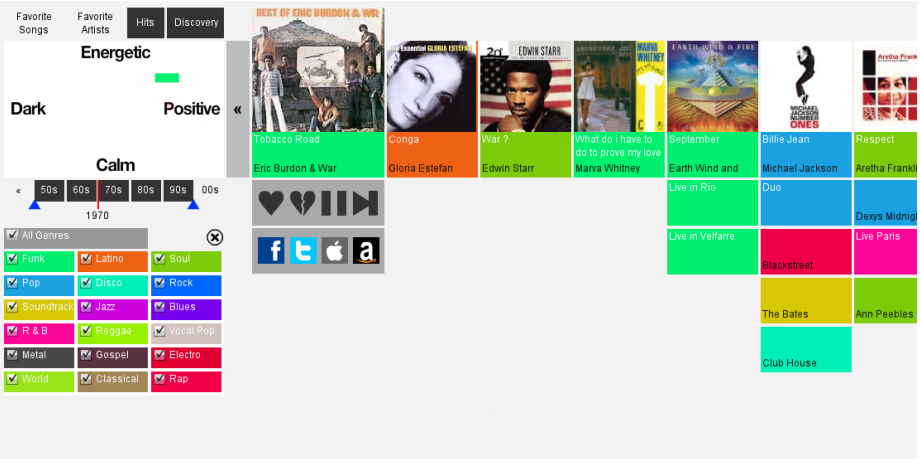

Abbildung 1: Screenshot des *Musicovery*-Nutzungsinterfaces (Abgerufen am 23.05.2011) mit dem *Moodpad* am linken Rand, der Genre- und Jahresauswahl darunter und dem derzeit gewählten Musikprogramm rechts daneben. Genres sind farblich markiert, für die Titel „September“, „Billie Jean“ und „Respect“ kennt *Musicovery* alternative Versionen, die ausgewählt werden könnten.

werden können. Für registrierte Nutzer bietet *Musicovery* zusätzlich die Möglichkeit, Lieblingstitel und -künstler zu speichern und diese auf Wunsch in das gewählte Musikprogramm mit einzufügen. Ebenso können ungeliebte Titel und Künstler vom zukünftigen Abspielen ausgeschlossen werden.

Grooveshark[13] nimmt eine gewisse Sonderstellung ein, da das Angebot nicht nur Titelinformationen, sondern auch die Titel selbst aus der Masse der Nutzer bezieht. Bei *Grooveshark* kann jeder Nutzer Musik aus seiner persönlichen Sammlung auf den Server laden. Diese Musik lässt sich in Abspiellisten verwalten und von unterwegs abrufen. Zusätzlich können Nutzer in den Musiksammlungen anderer Nutzer stöbern und so auch fremde Musik in ihre Abspiellisten aufnehmen. Auch Funktionen eines sozialen Netzwerks sind in *Grooveshark* integriert, so können andere Nutzer zu „Freunden“ erklärt werden. *Grooveshark* informiert dann über gehörte Titel der Freunde und ermöglicht es Freunden einen Titel zu empfehlen. Ähnlich wie bei anderen Anbietern verfügt auch *Grooveshark* über eine Vorschlag-Funktion, mit der einem Nutzer Titel aus der Sammlung anderer Nutzer mit ähnlichem Geschmack vorgeschlagen werden können. Eine Radio-Funktion nutzt die erzeugten Vorschläge um eine gegebene Abspielliste kontinuierlich mit ähnlicher Musik fortzuschreiben. Auf Wunsch kann die so erzeugte verlängerte Abspielliste wiederum gespeichert und später als Basis für ein neues Radio genutzt werden.

[13] http://www.grooveshark.com

3.2.3 Vorteile eines personalisierten Musikdienstes für Anbieter

Der Betrieb eines personalisierten Musikdienstes verspricht auch für den Anbieter einige zusätzliche Vorteile. So können die gewonnenen Hörerdaten gesammelt und ausgewertet werden, um Rückschlüsse über gerade aktuelle musikalische Themen zu ziehen. So ist es beispielsweise denkbar, dass ein Werbekunde seine Werbespots mit dem derzeit aktuellen Musikgeschmack des Publikums unterlegt und damit eine höhere Aufmerksamkeit für sein Produkt erreicht. Ebenso ließen sich Werbezeiten für eine neue Musikgruppe gezielt im passenden musikalischen Umfeld schalten. Auch Plattenfirmen könnten sich für die gewonnenen Hörerdaten interessieren, um frühzeitig Informationen über neue musikalische Trends zu erhalten und über Wiederveröffentlichungen zu entscheiden. So könnte das häufige Abspielen eines älteren Titels eine Plattenfirma veranlassen, diesen Titel erneut in die Läden zu bringen oder eine neue Version zu veröffentlichen. Kombiniert man die musikalischen Daten mit Informationen zur Geolokalisation, ließe sich sogar ein Land oder Gebiet ermitteln, in dem ein bestimmter Geschmack herrscht. Mit diesem Wissen könnte eine Band die Konzerte ihrer Tour so planen, dass sie ihre Auftritte vor allem in Gegenden mit einer hohen Fandichte stattfinden lässt. Das Risiko eines wenig besuchten Konzertes könnte damit gemindert werden. Auch eine aktive Vermarktung von neuen Titeln ist im Rahmen eines personalisierten Musikdienstes denkbar. So könnte der Betreiber mit einer Plattenfirma Verträge schließen, um einen neuen Titel dieser Firma bevorzugt in das Programm aufzunehmen. Irritationen beim Zuhörer ließen sich dabei vermeiden, indem der neue Titel nur an solche Hörer gesendet wird, deren Musikgeschmack zu den Charakteristiken dieses Titels passt.

3.3 last.fm

Da für diese Arbeit Daten vom Anbieter *last.fm*[14] verwendet werden, soll dieser im Folgenden Abschnitt noch einmal ausführlicher vorgestellt werden. *last.fm* bezeichnet sich selber als „Empfehlungsservice für Musik“[15], umfasst aber auch Funktionen eines sozialen Netzwerkes, wie zum Beispiel die Möglichkeit andere Nutzer als Freunde zu markieren und Meinungen in Diskussionsgruppen auszutauschen. *last.fm* führt für seine Nutzer eine Liste mit Titeln, die diese sich auf der Webseite des Anbieters angehört haben oder die vom Musik-Abspielprogramm des Nutzers an *last.fm* gesendet wurden. Damit entsteht für aktive Nutzer ein ausführliches Musikprofil, das von *last.fm* zum Vorschlagen ähnlicher Titel genutzt wird. Genauere Informationen, anhand welcher Kriterien ähnliche Titel ermittelt werden, werden von *last.fm* nicht veröffentlicht. Im Jahr 2007 wurde *last.fm* vom amerikanischen Medienkonzern CBS aufgekauft[16] und wird seitdem auch in das Radioangebot von CBS integriert. So werden von CBS terrestrische Radiostationen betrieben, deren Musikauswahl sich auf Daten von

[14] http://www.last.fm

[15] http://www.lastfm.de/about (Abgerufen am 21.04.2011)

[16] http://www.guardian.co.uk/media/2007/may/30/citynews.musicnews (Abgerufen am 08.07.2011)

last.fm stützt[17]. Webstreams von CBS-Radiostationen sind aus Lizenzgründen außerhalb der USA nicht zu empfangen, stattdessen werden Nutzer an *last.fm* verwiesen.

3.3.1 Datenbasis

Die von *last.fm* gesammelten und verwendeten Daten sind weitgehend nutzergeneriert. Plugins, die *last.fm* für viele Abspielprogramme für Musik, aber auch für zahlreiche mobile Geräte wie Smartphones oder MP3-Player anbietet, speichern Informationen über die vom Nutzer gehörten Titel und übermitteln diese an *last.fm*. Dazu werden die in gängigen Dateiformaten enthaltenen Titelinformationen (zum Beispiel die ID3-Tags in MP3s) ausgelesen, die üblicherweise neben Interpret und Titel des Musikstückes auch Informationen wie etwa das dazugehörige Album oder das Erscheinungsjahr enthalten. Wird auf diese Weise ein Titel oder Künstler übermittelt, der bisher noch nicht in der Datenbank von *last.fm* enthalten ist, wird automatisch eine Webseite für diesen Künstler oder Titel angelegt. Diese Seite kann, vergleichbar mit Wikipedia, von allen Nutzern bearbeitet und mit weiteren Informationen zu Künstler oder Titel angereichert werden. Zusätzlich können alle Titel, Künstler oder Alben mit Tags[18] versehen werden. Jeder übermittelte Titel wird auch mit dem übermittelnden Nutzer verknüpft, so dass ein Profil des persönlichen Musikgeschmacks entsteht und sich oftmals zusammen gehörte - also mutmaßlich ähnliche - Lieder erkennen lassen. *Last.fm* gibt an, dass bisher über 43 Milliarden Titelübermittlungen (von *last.fm* "scrobbels" genannt) bei ihnen eingegangen sind[19]. Die Verwendung nutzergenerierter Daten ermöglicht es *last.fm* ohne redaktionellen Aufwand eine sehr große Titel-Datenbank aufzubauen, enthält aber auch ein höheres Fehlerpotential. So finden sich in den *last.fm* Daten viele Titel und Künstler mehrfach in unterschiedlichen Schreibweisen. Besonders bei Kollaborationen mehrerer Künstler gibt es oftmals keine allgemeingültige Schreibweise, was Reihenfolge und Trennzeichen angeht. Beispielsweise finden sich für den Titel „Apologize," der vom amerikanischen Hip-Hop-Produzenten *Timbaland* und der Rock-Gruppe *OneRepublic* gemeinsam aufgenommen wurde, mindestens 35 verschiedene Künstlerschreibweisen: „Timbaland Feat. One Republic", „Timbaland ft One Republic", „Timbaland [ft. OneRepublic]" um nur einige zu nennen. Auch die Reihenfolge der Künstlernennungen wird von den Usern oftmals willkürlich oder nach persönlichen Vorlieben gewählt. So finden sich auch „One Republic Feat Timbaland" und „OneRepublic (f. Timbaland)" in der Datenbank. Andere Nutzer wiederum geben nur den Haupt-Künstler an und fügen die weiteren Beteiligten als Zusatz an den Titelnamen an. Also: „Timbaland - Apologize (feat. OneRepublic)". Ähnliche Probleme ergeben sich bei Künstlern mit fremdsprachigen Namen oder Namen mit Sonderzeichen. Der deutsch-britische Komponist *Georg Friederich Händel* befindet sich mit mindestens 16 verschiedenen Schreibweisen in den Daten von

[17] *Last.fm Discover* - ausgestrahlt in Chicago, Detroit, Los Angeles, New York und San Francisco

[18] Frei wählbare Schlagworte

[19] http://www.lastfm.de/about (Abgerufen am 26.05.2011)

last.fm. Beispielsweise als „Georg Frederic Handel“, „Georg Friedrich Haendel“, „Georg Frederich Händel“ oder „Georg Friederich Hndel“. Ein weiteres Problem stellen Zusatz-Informationen dar, die Nutzer an den Künstlernamen anhängen, da es für ihr persönliches Ablagesystem angemessen erscheint. Zum Beispiel: „George Frideric Handel (1685-1759)“. Um diesen Problemen zumindest teilweise entgegen zu wirken, wurden für diese Arbeit die Daten bereinigt, exotische Schreibweisen entfernt und versucht, ähnliche Schreibarten zusammenzufassen. Siehe dazu auch 7.3.2.

3.3.2 API

last.fm bietet über eine API einen weitreichenden Zugriff auf ihre Daten. So können Informationen über Künstler, Musiktitel, Tags, Benutzer und Events mit einfachen Methoden abgerufen und von Anwendungen externer, unabhängiger Entwickler verwendet werden. Verfügt ein Programm über gültige Zugangsdaten eines *last.fm*-Nutzers, ist es sogar möglich, über die API Daten zu verändern. So können Programme Titel in die persönliche Datenbank eines Nutzers einfügen oder Nachrichten an andere Nutzer schicken.

Die Kommunikation zwischen Anwendungen und *last.fm* entspricht dem REST (Representational State Transfer)-Prinzip [15] und erfolgt über HTTP. Die Verwendung eines REST-artigen Protokolls bedeutet insbesondere, dass die Kommunikation zustandlos erfolgen muss, das heißt jede Anfrage muss alle Informationen enthalten, die der Server zu ihrer Bearbeitung benötigt. *last.fm* löst die Kommunikation direkt über ihre HTTP Server. Die API Dokumentation spezifiziert eine Reihe von, mit variablen Parametern versehenen, URLs, bei deren Aufruf eine XML-Datei mit dem Ergebnis der Anfrage zurück geliefert wird. Für diese Arbeit werden 4 API-Methoden genutzt:

- *artist.gettoptags* - liefert eine Liste der für einen Künstler am meisten vergebenen Tags und einen Count-Wert, der angibt wie oft das jeweilige Tag im Verhältnis zum häufigsten Tag vergeben wurde
- *library.gettracks* - liefert eine Liste mit allen von einem Nutzer „gescrobbelten“ Titeln, dem jeweiligen Künstler, der Länge und der Häufigkeit mit der ein Titel von diesem Nutzer an *last.fm* übermittelt wurde
- *track.getcorrection* - prüft, ob es zu einem Titel und Künstler alternative Schreibweisen gibt und liefert die von *last.fm* als korrekt angesehene zurück
- *track.gettoptags* - analog zu artist.gettoptags

4 Problemmodellierung

4.1 Mögliche Modelle

Um ein personalisiertes Radio zu realisieren, gilt es, eine Reihe von Musikblöcken mit fest vorgegebener Größe möglichst effizient mit den zur Verfügung stehenden Titeln auszufüllen. Hierbei hat jeder Titel für einen Hörer einen bestimmten Wert, d.h. ihm sagen einzelne Titel mehr oder weniger stark zu. Ohne weitere Modifikationen lässt sich diese Aufgabe als ein *Rucksack-Problem (KP)* modellieren. Eine anschauliche Betrachtung beschreibt das Rucksack-Problem (engl. auch knapsack problem) als die Situation eines fiktiven Wanderers. Dieser hat n Gegenstände zur Auswahl, die ihm bei seiner Wanderung von Nutzen sein könnten. Jeder Gegenstand hat ein Gewicht w_j und einen speziellen Nutzen p_j für den Wanderer. Insgesamt kann der Wanderer nur ein maximales Gewicht c in seinen Rucksack packen. Die Aufgabe lautet nun, die Teilmenge der Gegenstände auszuwählen, die den maximalen Nutzen versprechen, ohne die Kapazitätsbegrenzung zu verletzen. Formal gesehen gilt es das folgende Problem zu lösen:

$$
\begin{aligned}
maximiere\ z &= \sum_{j=1}^{n} p_j x_j \\
mit \sum_{j=1}^{n} w_j x_j &\leq c \\
x_j &\in \{0,1\}, \quad j = 1, \ldots, n
\end{aligned}
$$

Das Ziel ist, für jeden Gegenstand j zu entscheiden, ob er zur Lösung gehört ($x_j = 1$) oder nicht ($x_j = 0$). Ein Vektor mit einer Belegung, die zum maximal möglichen Nutzen z^* führt, wird mit $x^* = \{x_1^*, \ldots, x_n^*\}$ bezeichnet. x^* ist also eine (oder die) optimale Lösung für das Problem [29].

Für das Rucksack-Problem existieren eine Reihe von verwandten Problemen. Zum Beispiel das *Rucksack-Problem mit Restriktionen*, bei dem nicht jeder Gegenstand in jeden Rucksack gepackt werden kann oder das *d-dimensionale Rucksack-Problem*, bei dem nicht nur eine Gewichtsbeschränkung eingehalten werden muss, sondern d verschiedene Restriktionen beachtet werden müssen. Eine Übersicht über die verschiedenen Variationen des Rucksack-Problems findet sich unter anderen in den Büchern von Martello und Toth [36] oder Kellerer et al. [29]. Neben dem klassischen Rucksack-Problem sind noch zwei Varianten für die hier betrachtete Anwendung besonders interessant: Beim *multiplen Rucksack-Problem (MKP)* müssen die Gegenstände auf m verschiedene Rucksäcke mit unterschiedlichen Kapazitäten c_i verteilt werden. Hier entspricht nun x_{ij} dem Zustand, dass Gegenstand j in Rucksack i gepackt wird. Offensichtlich kann kein Gegenstand in mehrere Rucksäcke gleichzeitig gepackt werden, daher

ist eine zusätzliche Bedingung nötig (4.3)

$$maximiere\, z = \sum_{i=1}^{m}\sum_{j=1}^{n} p_j x_{ij} \tag{4.1}$$

$$mit \sum_{j=1}^{n} w_j x_{ij} \leq c_i, \quad i = 1, \ldots, m \tag{4.2}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1, \quad j = 1, \ldots, n \tag{4.3}$$

$$x_j \in \{0, 1\}, \quad i = 1, \ldots, m,\ j = 1, \ldots, n \tag{4.4}$$

Haben alle Gegenstände das gleiche Verhältnis zwischen Nutzen und Gewicht, so spricht man vom *Subset Sum Problem (SSP)*. Da hier kein Gegenstand profitabler ist als ein anderer (relativ zu seinem Gewicht), lässt sich der Gesamtnutzen nur maximieren, indem man die Kapazität möglichst gut ausnutzt.

$$\begin{aligned} maximiere\, z &= \sum_{j=1}^{n} w_j x_j \\ mit \sum_{j=1}^{n} w_j x_j &\leq c \\ x_j &\in \{0, 1\}, \quad j = 1, \ldots, n \end{aligned}$$

Eine andere Interpretation dieses Problems sieht die Aufgabe darin, aus einer Menge von Zahlen die Untermenge zu ermitteln, deren Summe möglichst nah an einem Wert c liegt. Daher der Name Subset Sum Problem. Analog zu MKP lässt sich auch ein *multiples Subset Sum Problem (MSSP)* definieren.

Das Rucksack-Problem gehört zur Komplexitätsklasse der NP-Schweren Probleme [17]. Daraus ergibt sich, dass eine exakte Lösung für große Datenmengen eine hohe Laufzeit zur Folge hätte. Daher sollen in dieser Arbeit Heuristiken getestet werden, die das Problem nur approximativ lösen. Um diese Algorithmen bewerten zu können, ist es nötig, ein Maß für ihre Genauigkeit zu finden, also zu ermitteln, wie weit die betrachtete Methode maximal von der optimalen Lösung abweicht. Sei z^A der vom Algorithmus A erreichte Profit, so ist A ein ε-Approximations-Algorithmus, wenn für alle Probleminstanzen I gilt

$$\frac{z^*(I) - z^A(I)}{z^*(I)} \leq \varepsilon$$

4.2 Modellierung als Knapsack

Zurück zum Problem des personalisierten Radios. Dieses lässt sich direkt mit KP modellieren. Dabei entsprechen die zu füllenden Musikblöcke dem leeren Rucksack und ihre Länge der verfügbare Kapazität. Die Musiktitel entsprechen den zur Verfügung stehenden Gegenständen und ihre Spieldauer dem Gewicht.

Auch der Score eines Titels lässt sich fast direkt mit dem Nutzen eines Gegenstandes gleichsetzen. Hier gilt es lediglich zu beachten, dass bisher der Score nicht von der Länge eines Liedes abhängt. Das heißt, mit einem kurzen Lied lassen sich deutlich mehr Score-Punkte pro Sekunde erreichen als mit einem gleich gut bewerteten langen Lied. In der Praxis sollen aber nicht möglichst viele kurze Lieder in einen Musikblock gefüllt werden, sondern alle Titel unabhängig von ihrer Länge in die Auswahl mit einfließen. Daher wird ein *relativer Score* verwendet, der sich aus dem eigentlichen Score multipliziert mit der Länge eines Titels zusammensetzt. Das heißt aber auch, dass eine Korrelation zwischen Gewicht und Profit der zu packenden Gegenstände vorliegt. Experimente haben gezeigt, das korrelierte Probleminstanzen zu den schwereren Rucksack-Problemen gehören ([37], [49]) und damit eine höhere Laufzeit als für unkorrelierte Daten zu erwarten ist.

4.2.1 Bewertung

Die Modellierung als KP ist eine intuitive Modellierung. Jeder Musikblock wird vom System nacheinander betrachtet und ausgefüllt. Das Ziel liegt dabei auf der Optimierung des Titel-Scores, also auf dem Abspielen möglichst beliebter Titel. Nachteile dieser einfachen Betrachtung finden sich vor allem in der Verteilung der Titel. So besteht die Möglichkeit, dass KP einen Musiktitel verwendet, der in einem anderen Block besser passen würde und dort dann nicht mehr zur Verfügung steht. Da KP beim Bearbeiten eines Musikblockes versucht den Gesamtscore des Blocks zu maximieren, besteht ferner die Gefahr, dass alle Lieblingslieder eines Nutzers auf die zuerst bearbeiteten Musikblöcke verteilt werden. So können Häufungen von besonders beliebten und weniger beliebten Titeln entstehen, anstatt einer ausgewogenen Mischung aus absoluten Lieblingstiteln und Titeln, die der Nutzer weniger häufig hört.

4.2.2 Algorithmen

Der GREEDY Algorithmus basiert auf dem intuitiven Ansatz aus den verfügbaren Gegenständen solange die lohnendsten (also solche, die möglichst viel Profit pro Gewichtseinheit versprechen) in den Rucksack zu packen, bis die Kapazitätsgrenze erreicht ist. Als Algorithmus bedeutet dies, alle Gegenstände anhand des Verhältnisses $\frac{p_j}{w_j}$ (*Effizienz* eines Gegenstandes) absteigend zu sortieren und dann, dieser Sortierung folgend, nacheinander in den Rucksack zu packen, solange die Summe der Gewichte die verfügbare Kapazität nicht übersteigt.

Sofern die Gegenstände bereits sortiert sind, muss GREEDY jeden maximal einmal betrachten. Daher kommt der Algorithmus mit einer Laufzeit von $O(n)$ aus. Da die Gegenstände üblicherweise nicht sortiert vorliegen, kommt noch ein einmaliger Aufwand von $O(n \log n)$ für die Sortierung hinzu. Der guten Laufzeit und einfachen Implementierung steht eine ausgesprochen schlechte approximative Genauigkeit ε gegenüber. Es lässt sich zeigen, dass sie beliebig nah gegen 1 gehen kann; allerdings lassen sich auch ohne Verschlechterung der theoretischen Laufzeit Gegenmaßnahmen implementieren, die ε auf $\frac{1}{2}$ begrenzen [36].

Für zufällige Probleminstanzen ohne Korrelation zwischen Gewicht und Nutzen ist ferner auch der durchschnittliche Fehler des verbesserten GREEDY bekannt. Calvin und Leung ([8]) haben 2003 gezeigt, dass dieser bei $\frac{1}{\sqrt{n}}$ liegt. GREEDY ist also ein sehr einfacher, schnell zu implementierender Algorithmus, der für viele Anwendungen ausreichend gute Ergebnisse liefert. Allerdings sind in der hier betrachteten Anwendung Gewicht und Nutzen (bzw. Länge und relativer Score) nicht unkorreliert, weswegen höhere durchschnittliche Fehler möglich sind. Trotzdem ist GREEDY als Vertreter der einfachen Heuristiken gut als Vergleichsobjekt geeignet, um die erzielten Ergebnisse der anderen Algorithmen einordnen zu können.

Neben GREEDY soll noch eine weitere Heuristik implementiert und getestet werden. Diese wurde von Kellerer et al. [29] vorgestellt und von den Autoren nicht benannt. Im Interesse der Leserlichkeit soll dieser Algorithmus im Weiteren mit KPTREE bezeichnet werden. Die Autoren haben gezeigt, dass er über eine approximative Genauigkeit von $\varepsilon = \frac{1}{3}$ und eine Worst-Case Laufzeit von $O(n \log n)$ verfügt. Im Gegensatz zu GREEDY verwendet KPTREE zwei Listen (P, E), die jeweils alle verfügbaren Gegenstände enthalten. P ist absteigend nach den Gegenstandswerten sortiert, E absteigend nach der Effizienz der Gegenstände. Aus der mit E definierten Reihenfolge wird zusätzlich ein Baum erstellt. Für jeden Gegenstand wird ein Blatt angelegt und mit dem Gewicht des Gegenstandes versehen. Nun werden sukzessive immer zwei benachbarte Knoten an einen Elternknoten gehängt, welcher die Summe der Gewichte der Kinder als Wert bekommt. Die Eltern werden daraufhin wieder zu zweit zusammengefügt. Dieser Vorgang wird fortgesetzt, bis nur noch ein Knoten übrig bleibt - die Wurzel, deren Wert die Summe aller Gewichte ist. Aus dem Baum lässt sich somit an jedem Knoten ablesen, welche Kapazität nötig ist, um alle unter dem Knoten liegenden Blätter in den Rucksack zu packen. Da die Blätter sortiert sind enthält jeweils das linke Kind die effizientere Hälfte der Gegenstände. Mit einer gegebenen Restkapazität $c' \leq c$ wird nun solange jeweils das linke Kind eine Knotens gewählt, bis dessen Wert $w \leq c'$ ist. Ist ein solcher Knoten gefunden, können alle Blätter dieses Teilbaums in die Lösung übernommen werden. Daraufhin wird mit der verbleibenden Kapazität $c' - w$ im rechten Teilbaum des Knotens weiter gesucht, ob sich noch weitere Blätter in die Lösung einfügen lassen. Diese Baumsuche wird n mal wiederholt, wobei der profitreichste Gegenstand in der Lösung jeweils vorab festgelegt wird. Dazu wird in jedem Durchlauf das erste Element aus der Liste P gewählt, aus dem Baum entfernt (dabei müssen alle Gewichte auf dem Weg zur Wurzel angepasst werden) und in die Lösung übernommen. Die verbleibende Kapazität wird dann mit Hilfe der zuvor beschriebenen Baumsuche aufgefüllt. Für diese Arbeit wird der Algorithmus KPTREE mit einer eigenen Erweiterung versehen, die es ermöglicht, nicht passende Teilbäume frühzeitig zu erkennen und zu meiden. Diese Erweiterung ist in den Implementierungsdetails unter 7.5.2 erläutert.

Es existieren noch zahlreiche weitere Heuristiken für KP, deren Eignung in weiterführenden Arbeiten untersucht werden könnte (siehe z.B. [53], [26], [25] oder auch [46]). Allerdings ist zu beachten, dass beim hier betrachteten Problem

komplexere Heuristiken nicht zwangsweise zu guten Laufzeiten führen. So hat zum Beispiel der heuristische Algorithmus CKPP eine Laufzeit von $O(n^{\lceil \frac{1}{\varepsilon} \rceil - 2})$ mit $\varepsilon < \frac{1}{3}$ [9]. Bereits für $\varepsilon = \frac{1}{4}$ ergibt sich somit eine quadratische Laufzeit. Bei den für die Experimente veranschlagten Größenordnungen von etwa 5000 Titeln pro User und einer Länge von etwa 10 Minuten bzw. 600 Sekunden pro Musikblock ($n = 5000$, $c = 600$) lässt sich das Problem mit einem Algorithmus, der mit einer Laufzeit von $O(nc)$ eine optimale Lösung findet, möglicherweise schneller lösen (abhängig von der gewählten Implementierung und den damit hinzukommenden Konstanten). Nachfolgend soll ein Algorithmus vorgestellt werden, der die Laufzeit von $O(nc)$ noch unterbietet.

Neben den zuvor vorgestellten Heuristiken soll noch ein Algorithmus getestet werden, der das Problem optimal löst. Auch hier gibt es eine große Auswahl unterschiedlichster algorithmischer Ansätze mit verschiedenen Stärken und Schwächen. Im Folgenden soll der Algorithmus COMBO kurz betrachtet werden. Dieser wurde 1999 von Martello, Pisinger und Toth vorgestellt [34] und schnitt in einem von Pisinger im Jahr 2005 veröffentlichten experimentellen Vergleichen zwischen fünf weit verbreiteten exakten Algorithmen für KP deutlich besser ab, als die anderen Testkandidaten. Lediglich der etwas simplere MINKNAP Algorithmus konnte bei einigen wenigen Probleminstanzen mit den von COMBO erzielten Ergebnissen mithalten [49]. Wie der Name bereits andeutet, handelt es sich bei COMBO um eine Kombination aus mehreren Techniken, die je nach Art des Problems eingesetzt werden sollen. Seine Basis bildet MINKNAP, ebenfalls ein Algorithmus, der optimale Lösungen für KP liefert und der das Prinzip der dynamischen Programmierung ([5]) verwendet. Dabei nutzt er aus, dass oftmals nur eine kleine Menge aller Variablen für die Optimalität der Lösung entscheidend ist (Gegenstände mit einem sehr gutem Verhältnis zwischen Profit und Gewicht sind sehr wahrscheinlich in der optimalen Lösung, umgekehrt ist es unwahrscheinlich, dass ein sehr schwerer Gegenstand mit wenig Profit gepackt wird) [3]. Hieraus ergibt sich, dass ein optimaler Algorithmus in der Theorie nur den sogenannten Kern des Problems betrachten muss. Nehmen wir an, dass alle Gegenstände absteigend nach ihrem relativem Profit sortiert sind $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \ldots \geq \frac{p_n}{w_n}$, so ist der Kern als das Intervall $[a, b]$ definiert, mit

$$a := min\{j | x_j^* = 0\}, \qquad b := max\{j | x_j^* = 1\}$$

Um den Kern genau zu bestimmen, muss also die optimale Lösung bekannt sein. Dies ist in praktischen Anwendungen aus offensichtlichen Gründen nicht der Fall, daher verwendet MINKNAP einen geschätzten Kern, der bei Bedarf um weitere Einträge ergänzt wird. Selbst für unsortierte Eingaben lässt sich ein solcher Kern in $O(n)$ berechnen [36]. Aufbauend auf diesen Ansatz entwickelte Pisinger einen modifizierten Quicksort-Algorithmus (Quicksort: [24]) zum Finden des Kerns [45]. Auf den so gefundenen Kern wird nun mit Methoden der dynamischen Programmierung eine *teilweise optimale* Lösung errechnet. Durch Abschätzung einer oberen Grenze kann dann entschieden werden, ob eine Variable außerhalb des Kerns das Potential hat, die bisher berechnete Lösung zu verbessern. Durch diese Techniken gelingt es MINKNAP in vielen

Fällen die Anzahl der betrachteten Zustände zu begrenzen und damit die Laufzeit und den benötigten Speicher klein zu halten. Trotzdem gilt natürlich die NP-Härte von KP unverändert, so dass auch MINKNAP nicht besser als eine pseudo-polynomiale Worst-Case Laufzeit sein kann. Insgesamt gelingt es die Laufzeit auf $O(n + min\{2^{t-s+1}, c(t-s+1)\})$ zu beschränken. Hierbei bezeichnet s den Index des ersten Elementes des geschätzten Kerns, t den Index des letzten Elementes (Für umfassendere Informationen siehe [47]).

COMBO nutzt MINKNAP zur Lösung des Problems, fügt aber einige weitere Verbesserungs-Schritte für besonders schwer zu lösende Probleme hinzu. Die Schwierigkeit eines Problems wird anhand der im Laufe des MINKNAP Algorithmus benötigten Zustände ermittelt. Überschreiten diese zuvor experimentell ermittelte Grenzwerte $M_1 < M_2 < M_3$, so wird versucht, mit rechentechnisch immer kostspieligeren Methoden das Problem zu reduzieren. Dabei wird angenommen, dass sich diese aufwändigen Operationen lohnen, wenn das Ausgangsproblem schwer genug ist. Gleichzeitig können dabei engere Grenzwerte für MINKNAP ermittelt werden, anhand derer entschieden werden kann, für welche Variablen sich die Betrachtung lohnt. Für eine komplette Aufzählung aller von COMBO verwendeten Methoden und detailliertere Informationen siehe Martello, Pisinger und Toth [34].

4.3 Modellierung als Multi-Knapsack

Analog zur Modellierung als Reihe von Rucksack-Problemen lässt sich das personaliserbare Radio mit Moderationen auch als ein Multi-Knapsack-Problem auffassen, indem eine Reihe von Musikblöcken gemeinsam bearbeitet wird. Wie zuvor, wird auch hier der *relative Score* als Gewicht verwendet.

4.3.1 Bewertung

Die Bewertung als MKP betrachtet eine Reihe von Musikblöcken gemeinsam. Für jeden Titel wird dabei der am besten passende Block gefunden. Dies verringert die Wahrscheinlichkeit, dass alle Lieblingstitel in den zuerst bearbeiteten Blöcken verbleiben. Allerdings hängt dies stark von der Vorgehensweise des Algorithmus ab, die Verwendung von MKP garantiert keine gleichmäßige Verteilung der Musiktitel, sondern lediglich eine Optimierung des Gesamtscores für alle betrachteten Musikblöcke.

4.3.2 Algorithmen

Auch für Multi-Knapsack gibt es zahlreiche Lösungsansätze. Martello und Toth [33] haben eine Reihe von unterschiedlichen Algorithmen zum Finden und Verbessern von Lösungen für MKP vorgestellt. Viele der vorgestellten Algorithmen lassen sich untereinander kombinieren, womit unterschiedliche Laufzeiten und Ergebnis-Qualitäten erreicht werden. Da für die betrachtete Radioanwendung nicht nur eine Maximierung des Profits erwünscht ist, sondern auch andere Kriterien in die Bewertung einfließen sollen (siehe Kapitel 4.5), bietet sich dieser

```
1  K = Liste mit Rucksackkapazitäten
2  G = Liste von Gegenständen            # Tuppel (gewicht, wert)
3  R = Rückgabe-Liste
4
5  sortiere K aufsteigend
6  sortiere G nach (wert/gewicht)
7
8  for i in 0,...,länge(K)-1:
9      R[i] = Greedy(K[i], G)
10     G = G - r[i]
11
12 return R
```

Code 1: Pseudocode für Algorithmus MK1

„modulare“ Ansatz sehr gut an, um die Auswirkung einzelner Algorithmusteile zu ergründen. Daher werden hier diese Algorithmen implementiert und getestet, obwohl inzwischen neuere und effizientere Heuristiken bekannt sind (Beispielsweise: [10] oder [28]).

Martello und Toth unterteilen die vorgestellten Algorithmen in zwei Gruppen. Algorithmen zum Finden einer ersten zulässigen Lösung (MK1, MK2 und MK3) und Algorithmen zur Verbesserung der ersten Lösung (I1, I2). Alle fünf sollen im Folgenden kurz beschrieben werden. Im Weiteren gehen wir davon aus, dass die einzelnen Rucksäcke nach steigenden Kapazitäten geordnet sind: $c_1 \leq c_2 \leq \ldots \leq c_m$. Die verfügbaren Gegenstände werden absteigend nach ihrem relativen Profit $\frac{p_i}{w_i}$ geordnet. Der Rechenaufwand für die Sortierung der Gegenstände und Rucksäcke ist also noch zusätzlich zu den angegebenen Laufzeitabschätzungen notwendig.

Zum Finden einer Initiallösung verwendet MK1, beginnend vom kleinsten Rucksack, eine GREEDY-Heuristik, wie sie in Kapitel 4.2.2 beschrieben ist. Dabei werden nach jedem gefüllten Rucksack die verpackten Gegenstände aus der Menge der verfügbaren entfernt, so dass jeder nur einmal verwendet werden kann.

MK2 fügt die Elemente direkt in die Rucksäcke ein. Sofern der Platz es erlaubt, wird das erste Element in den erste Rucksack, das zweite Element in den zweiten usw. gepackt. Ist der letzte Rucksack erreicht, so beginnt der Algorithmus wieder beim Ersten. Passt ein Element nicht in einen Rucksack, so wird dieser übersprungen und MK2 versucht es im nächsten unterzubringen. Reicht die Kapazität von keinem der Rucksäcke so wird dieses Element verworfen. Im schlimmsten Fall muss also bei MK1 und MK2 jedes der n Elemente m mal betrachtet werden, womit sich eine Worst-Case Laufzeit von $O(nm)$ ergibt.

MK3 benutzt wiederum GREEDY, um die Rucksäcke zu füllen. Im Gegensatz zu MK1 ist es nun aber im ersten Schritt erlaubt, dass ein Gegenstand in mehr als einen Rucksack gepackt wird. In einem zweiten Durchlauf werden noch einmal alle Gegenstände $j = 1, \ldots, n$ betrachtet und solche, die sich in mehr als einem Rucksack befinden, ermittelt. Sei j' ein solcher Gegenstand, $A = \{i \mid \textit{Rucksack } i \textit{ enthält } j'\}$ die Menge der Rucksäcke, in denen j' enthalten ist und $\overline{c_i}$ die aktuelle Restkapazität für Rucksack i. Für jedes $i \in A$ wird erneut ein GREEDY mit einer neuen Kapazitätsgrenze $w_{j'} + \overline{c_i}$ und den Gegenständen $j' + 1, \ldots, n$ durchgeführt und der aus der gefundenen Lösung resultierende

```
K = Liste mit Rucksackkapazitäten
G = Liste von Gegenständen            # Tuppel (gewicht, wert)
R = Rückgabe-Liste

sortiere G nach (wert/gewicht)

for i in 0,...,länge(G)-1:
    start = j
    reset = 0

    while j != start or reset == 0: # reset wird 1 gesetzt,
                                    # wenn j den letzten Rucksack
                                    # erreicht hat und wieder
                                    # bei 1 anfängt

        j = j + 1

        if j >= länge(K):           # Letzter Rucksack erreicht,
            j = 0                   # beginne von Vorne
            reset = 1

        if G[i]["gewicht"] < K[j]
            R[j].append(G[i])
            K[j] = K[j] - G[i]["gewicht"]
            break

return R
```

Code 2: Pseudocode für Algorithmus Mk2

```
K := Liste mit Rucksackkapazitäten
G := Liste von Gegenständen           # Tuppel (gewicht, wert)
R := Rückgabe-Liste

sortiere G nach (wert/gewicht)

for i in 0,...,länge(K)-1:
    R[i] := Greedy(K[i], G)
    K[i] := K[i] - Summe aller Gewichte in R[i]

for i in 0,...,länge(G)-1:
    A := Liste von Rucksäcken, in denen G[i] enthalten ist

    min["wert"] = +inf

    if länge(A) > 1:
        for a in A:
                            # Führe Greedy mit allen Gegenständen an
                            # Position > i+1 durch
            tmp[a] := Greedy(K[a] + G[i]["gewicht"], G[i+1:länge(G)])

            if summe aller Werte in tmp[a] < min["wert"]:
                min["wert"] := summe aller Werte in tmp[a]
                min["id"] := a

        setze R[a] = (R[a] - G[i]) + tmp[a] für alle a in (A - min["id"])

return R
```

Code 3: Pseudocode für Algorithmus Mk3

Profit errechnet. Der Rucksack, der durch das Entfernen von j' den größten Profit-Verlust hätte, bleibt unberührt, aus allen anderen wird j' entfernt und stattdessen die neu ermittelte Füllung verwendet. Dadurch, dass GREEDY jeweils nur Gegenstände $j'+1, \ldots, n$ berücksichtigt, ist sichergestellt, dass alle neu gepackten Gegenstände im weiteren Verlauf noch untersucht und eventuell vorhandene Mehrfachnutzungen aufgelöst werden. Am Ende ist jeder Gegenstand nur einmal in den Rucksäcken enthalten. Im schlimmsten Fall muss allerdings pro Gegenstand m mal GREEDY ausgeführt werden, wodurch sich eine Worst-Case-Laufzeit von $O(nm^2)$ ergibt.

Zum Verbessern einer gefundenen gültigen Lösung schlagen Martello und Toth zwei Algorithmen vor. I1 betrachtet alle Paare von Gegenständen, die in unterschiedliche Rucksäcke sortiert wurden und prüft, ob ein Austauschen dieser beiden Gegenstände genug Platz schaffen würde, um einen bisher noch nicht untergebrachten Gegenstand noch zu verpacken. Die theoretische Worst-Case-Laufzeit von I1 ist $O(n^2)$.

I2 prüft für jeden Gegenstand, der derzeit in den Rucksäcken ist, ob sich durch dessen Entfernen und ein Durchführen von GREEDY auf den bisher nicht genutzten Gegenständen der Gesamtprofit erhöhen ließe. Falls ja, wird der Gegenstand entfernt und die von GREEDY gefundenen hinzugefügt. Sind alle Gegenstände überprüft und wurde ein Austausch vorgenommen, beginnt der Prozess mit der neuen Rucksackbelegung von vorne, bis eine vorher festgelegte Anzahl an Iterationen erreicht wurde oder kein Austausch mehr vorgenommen werden kann. Martello und Toth schlagen vor, insgesamt maximal n komplette Schleifendurchläufe zu tätigen, so dass sich eine Laufzeitkomplexität von $O(n^3)$ ergibt. Experimente haben gezeigt, dass sich die besten Ergebnisse erreichen lassen, wenn zuerst I1 und danach I2 angewendet wird. Für genauere Beschreibungen der Algorithmen und experimentelle Ergebnisse zur Laufzeit und Ergebnisqualität siehe Martello und Toth [33].

Um MKP exakt zu lösen wird der Algorithmus MULKNAP verwendet, der von David Pisinger vorgestellt wurde [48]. Er basiert in Teilen auf MTM von Silvano Martello und Paolo Toth [35], hat sich aber in Experimenten von Pisinger als effizienter erwiesen ([48], [29]). Er nutzt eine Kombination aus Branch-and-Bound-Techniken und dynamischer Programmierung [31]. MULKNAP bedient sich dabei zweier zentraler Techniken um die Anzahl der nötigen Schritte zu reduzieren. Zuerst wird für die einzelnen Rucksäcke mit Kapazitäten $c_1, \ldots, c_m$ jeweils ein Subset-Sum-Problem gelöst und damit getestet, welche Kapazität $c'_1, \ldots, c'_m$ mit den zur Verfügung stehenden Gegenständen überhaupt maximal ausgefüllt werden kann (Anm.: für das Lösen der SSP wird der Algorithmus DECOMP verwendet, welcher in 4.4.2 näher betrachtet wird). Offensichtlich kann keine Lösung für MKP in einen Rucksack i Gegenstände mit einem größeren Gesamtgewicht als c'_i packen, da ansonsten diese Gegenstandskombination auch eine bessere Lösung für SSP wäre. Es kann also für weitere Schritte mit den reduzierten Kapazitäten gearbeitet werden. Um eine obere Grenze zu ermitteln wird mit den neuen Kapazitäten nun ein vereinfachtes Problem gelöst. Hierzu wird statt mehrerer Rucksäcke nur ein Rucksack mit der Gesamtkapazität $c = \sum_{i=1}^{m} c'_i$ betrachtet. Es sei x' die Lösung für dieses vereinfachte Problem. Ist

sie besser als bisher gefundene Lösungen, so wird versucht, durch Lösen von SSP die in x' gefundenen Gegenstände auf die m Rucksäcke zu verteilen. Eventuell frei bleibende Kapazitäten werden mit einer Greedy-Heuristik aufgefüllt. Ist die so gefundene Lösung immer noch besser als die bisher bekannten, so wird x' als neue beste Lösung gemerkt. In einem letzten Schritt werden die betrachteten Variablen reduziert und der Algorithmus rekursiv aufgerufen. Es sei noch einmal darauf hingewiesen, dass es sich hierbei nur um eine grobe Beschreibung der verwendeten Techniken handelt. Für eine ausführliche Betrachtung siehe Pisinger [48].

4.4 Modellierung als Subset-Sum

4.4.1 Bewertung

Da es für ein angenehm zu hörendes Radioprogramm von großer Wichtigkeit ist, dass die einzelnen Musikblöcke möglichst wenig Fehlbetrag aufweisen, soll hier auch untersucht werden, in wie weit sich das Berechnen der Blöcke als SSP betrachten lässt. Da keine Score-Informationen in den SSP-Algorithmen benutzt werden, steht zu erwarten, dass sich für den User eine insgesamt weniger passende Musikauswahl ergibt. Allerdings ist ein SSP-Algorithmus, im Gegensatz zu den zuvor vorgestellten Algorithmen, auch nicht dem konkurrierende Ziel einen hohen Gesamtscore erreichen zu müssen ausgesetzt, das sich möglicherweise nur mit einem größeren Fehlbetrag in der Kapazitätsausnutzung erreichen lässt. Daher kann man erwarten, dass SSP die fehlende Genauigkeit beim Treffen des User-Geschmacks durch eine besser ausgefüllte Musikzusammenstellung ausgleicht.

4.4.2 Algorithmen

Da KP eine Verallgemeinerung von SSP darstellt ist offensichtlich, dass auch die Algorithmen für KP in der Lage sind SSP zu lösen. Allerdings verfügen bei SSP alle Gegenstände über die gleiche Effizienz, welche von allen hier vorgestellten KP-Algorithmen als Sortier-Kriterium verwendet wird, um die Auswahl der erfolgversprechenden Gegenstände einzugrenzen. Diese Möglichkeit der Problemreduktion ist für SSP nicht gegeben.

SSP ist also ein Sonderfall, der vielen allgemeinen KP-Algorithmen die Arbeit erschwert, daher ist es angebracht auch noch einige spezialisierte SSP-Algorithmen zu testen. Auch für SSP existiert eine große Auswahl an Heuristiken mit unterschiedlichen Stärken (bekannte Heuristiken wurden unter anderem vorgeschlagen von Ibarra und Kim [26] Martello und Toth [36] oder Ghosh und Chakravarti [19]). Im Folgenden soll der approximative Algorithmus RGLI [50] betrachtet und auf seine Tauglichkeit für die untersuchte Radioanwendung getestet werden. RGLI verfolgt einen randomisierten Ansatz mit Laufzeit-Komplexität $O(n \log n)$. Wie gewohnt soll nun eine kurze Übersicht über das Vorgehen des Algorithmus gegeben werden. Für detaillierte Beschreibungen sei auf den Original-Artikel des Autors verwiesen, Details zur Implementierung finden sich in Kapitel 7.

RGLI arbeitet in zwei Phasen. Zuerst wird eine zufällige Lösung generiert. Hierzu wird aus allen verfügbaren Gegenständen zufällig einer gezogen und getestet, ob dieser in die Lösung aufgenommen werden kann, ohne die verfügbare Restkapazität zu übersteigen. Falls ja, so wird er in die Lösung eingefügt, falls nein, so wird er verworfen und in diesem Schritt nicht weiter betrachtet („Ziehen ohne zurücklegen"). Dies wird solange fortgesetzt, bis kein Gegenstand mehr verfügbar ist. Somit ist eine gültige und *maximale* Lösung erreicht, das heißt, sie überschreitet nicht die Kapazitätsgrenze und es ist nicht möglich einen weiteren Gegenstand hinzuzufügen, ohne vorher einen anderen zu entfernen. In der zweiten Phase wird versucht die Lösung zu verbessern. Dazu wird für jeden Gegenstand j überprüft, ob sich der Profit erhöhen lassen würde, wenn j aus der Lösung entfernt und stattdessen ein derzeit nicht in der Lösung enthaltener Gegenstand eingefügt werden würde. Finden sich für j mehrere mögliche Ersetzungskandidaten, so wird der mit dem größten Gewicht gewählt.

Die Phasen 1 und 2 werden für eine konstante Anzahl k Durchläufe wiederholt und dabei die beste gefundene Lösung gemerkt. Experimentelle Ergebnisse des Autors zeigen, dass ~ 50 eine geeignete Größe für k ist, die im Mittel gute Lösungen produziert.

Wie auch schon bei Knapsack und Multi-Knapsack soll auch für Subset-Sum ein exakter Algorithmus getestet werden. Dieser wurde von Pisinger [48] als Teil des MULKNAP Algorithmus vorgestellt und verfügt über eine Laufzeit von $O(min\{nc,\ nv,\ 2^b + 2^{n-b}\})$. Dabei ist b das sogenannte *Break-Item*, also der Gegenstand, mit dem die verfügbare Kapazität zum ersten mal verletzt wird $b = min\{j : \sum_{i=1}^{j} w_i > c\}$, $w_1 \leq w_2 \leq \ldots \leq w_n$ und v ist das Restgewicht, definiert durch $v = \sum_{j=1}^{n} w_j - w' < c$ mit $w' = \sum_{j=1}^{b-1} w_j$. Dieser Algorithmus hat in der ursprünglichen Publikation vom Autor keinen Namen bekommen, wurde aber in späteren Veröffentlichungen erneut aufgegriffen und dort als DECOMP-Algorithmus bezeichnet [29]. Wie auch bei den vorigen Algorithmen soll an dieser Stelle nur eine grobe Übersicht über die Funktionsweise gegeben werden. Für eine detaillierte Betrachtung und Laufzeitanalyse sei auf die ursprüngliche Beschreibung in [48] verwiesen. DECOMP nutzt das Prinzip der dynamischen Programmierung und besteht aus zwei rekursiv darstellbaren Funktionen, die ausgehend vom Break-Item, die betrachteten Gegenstände schrittweise in beide Richtungen erweitern:

$$\begin{aligned} &maximiere\ f_t(\tilde{c}) = \sum_{j=b}^{t} w_j x_j \\ &mit \sum_{j=b}^{t} w_j x_j \leq \tilde{c} \\ &\quad x_j \in \{0,1\},\ j = b, \ldots, t \end{aligned}$$

$$
\begin{aligned}
maximiere\ g_s(\tilde{c}) &= \sum_{j=1}^{s-1} w_j + \sum_{j=s}^{b-1} w_j x_j \\
mit \sum_{j=1}^{s-1} w_j + \sum_{j=s}^{b-1} w_j x_j &\leq \tilde{c} \\
x_j &\in \{0,1\},\ j = s, \ldots, b-1
\end{aligned}
$$

Diese beiden Funktionen lassen sich rekursiv darstellen als

$$
f_t(\tilde{c}) = \begin{cases} f_{t-1}(\tilde{c}) & für\ \tilde{c} = 0, \ldots, w_t - 1 \\ max\{f_{t-1}(\tilde{c}),\ f_{t-1}(\tilde{c} - w_t) + w_t & für\ \tilde{c} = w_t, \ldots, c \end{cases}
$$

$$
g_s(\tilde{c}) = \begin{cases} g_{s+1}(\tilde{c}) & für\ \tilde{c} = c - w_s + 1, \ldots, c \\ max\{g_{s+1}(\tilde{c}),\ g_{s+1}(\tilde{c} - w_s) + w_s & für\ \tilde{c} = 0, \ldots, c - w_s \end{cases}
$$

Ausgangswerte sind

$$
\begin{aligned}
f_{b-1}(\tilde{c}) &= 0\ für\ \tilde{c} = 0, \ldots, c \\
g_b(\tilde{c}) &= \begin{cases} 0 & für\ \tilde{c} = 0, \ldots, c,\ \tilde{c} \neq w' \\ w' & für\ \tilde{c} = w' \end{cases} \\
mit\ w' &= \sum_{j=1}^{b-1} w_j
\end{aligned}
$$

DECOMP beginnt also mit einer Initiallösung, die aus allen Gegenständen bis zum Break-Item besteht, f_t fügt der Menge der betrachteten Gegenstände einen bisher noch nicht berücksichtigten hinzu, g_s entfernt einen aus den bisher festgehaltenen Gegenständen in der Initiallösung. Der Algorithmus startet mit $(s,t) = (b, b-1)$ und erniedrigt dann s und erhöht t schrittweise. Die beiden Teile werden nach jedem Schritt vereinigt und der Wert der Lösung ermittelt. Ein Ergebnis ist erreicht, wenn die erzielte Lösung den Wert c hat - also die Kapazität komplett genutzt wurde - oder $(s,t) = (1,n)$ erreicht ist.

4.5 Qualitäts-Maßstab

Die Ergebnisse aller vorgestellten Algorithmen sollen anhand von vier Kriterien bewertet werden:

- Scorewert - welchen Scorewert erreichte der Algorithmus im Durchschnitt, d.h. wie gut trifft die vom Algorithmus erstellte Musikauswahl den Geschmack des Nutzers?
- Summe von Fehlzeiten - wie gut gelingt es dem Algorithmus die vorgegebenen Musikblöcke auszufüllen? Wieviele Sekunden bleiben im Durchschnitt unausgefüllt?

- Maximale Fehlzeit - wie lang ist die maximale unausgefüllte Zeitspanne in einem Musikblock?
- Verteilung der Fehlzeiten - wie gleichmäßig sind Fehlzeiten über die einzelnen Musikblöcke verteilt? Als Maßzahl hierfür wird der Variationskoeffizient der einzelnen Fehlzeiten berechnet.

5 Clustering

Auch mit der Verwendung von Heuristiken ist die Berechnung von personalisierten Radioprogrammen aufwändig. Es wäre also wünschenswert, wenn eine solche Berechnung nicht für jeden Nutzer einzeln, sondern möglichst für eine Gruppe von Nutzern gemeinsam erfolgen könnte. Dies bedeutet natürlich, dass alle Nutzer einer Gruppe das selbe Radioprogramm zugespielt bekommen würden. Um das Gefühl eines personalisierbaren Radios, auf das der Hörer direkten Einfluss hat, zu erhalten, sollte eine solche gemeinsam berechnete Gruppe eine möglichst große Übereinstimmung im Musikgeschmack aufweisen. Die Vermutung, dass es solche Gruppen gibt, stützt sich auf die Tatsache, dass sich auch für das bisherige, klassische Radio genug Hörer finden, deren Musikgeschmack so gut getroffen wird, dass sie das Programm dauerhaft hören. Auch die Existenz des sogenannten „Mainstreams“[20] und die Tatsache, dass in Diskotheken, Musik- und Nachtclubs große Gruppen von Menschen die selbe Musik gemeinsam genießen können, deutet darauf hin, dass sich für bestimmte Gruppen ein gemeinsamer Geschmack finden lässt. Ob dies wirklich so ist, soll anhand der vorliegenden *last.fm*-Daten überprüft werden. In ihnen soll mit verschiedenen Methoden nach Gruppen von Usern gesucht werden, die eine hohe Übereinstimmung der gehörten Titel haben. Existieren solchen Gruppen, so könnte für sie mit einer einmaligen Berechnung das Musikprogramm gestaltet werden und dabei trotzdem für die einzelnen Nutzer das Gefühl erhalten bleiben, das gehörte Programm würde exklusiv auf ihren persönlichen Geschmack zugeschnitten.

Selbst wenn sich keine Gruppen von solch hoher Qualität finden, dass ein Programm dauerhaft für eine ganze Gruppe berechnet werden kann, so kann die Suche doch hilfreich sein. So könnten für Usercluster berechnete Programme als Überbrückungs-Lösungen bereitgehalten werden, um in Zeiten mit hoher Last bereits bekannten Nutzern zuerst das Programm „ihres Clusters“ zu zuspielen, bis das System das persönliche Programm berechnet hat. Auf diese Weise lassen sich Wartezeiten vermeiden, die ansonsten möglicherweise zu Frustration bei Benutzern führen könnten.

5.1 Verfahren

5.1.1 SLR

SLR [58] ist ein Algorithmus, der ursprünglich für Handels-Transaktionsanalysen vorgeschlagen wurde. Aus einer Reihe von Transaktionen, zum Beispiel den von Kunden getätigten Einkäufen, soll ermittelt werden, welche Produkte häufig zusammen gekauft werden. Ähnliche Einkäufe sollen dazu zu Clustern zusammen gefasst werden. Bei einer solchen *Market-Basket Analyse* ergeben sich besondere Herausforderungen, die auch für das Clustering von Usern anhand ihrer bevorzugten Musik zutreffen. So steht üblicherweise eine sehr große Auswahl von Produkten zur Verfügung, von denen ein Kunde nur eine sehr kleine Teilmenge erwirbt. „*Market-Basket Analyse* ist also ein Clustering-Problem mit sehr hoher

[20] in diesem Fall: Musik, die den Geschmack der Masse trifft

Dimensionalität, kleinen Subsets und vielen Ausreißern in der Datenbasis“ [55]. Diese Situation trifft auch auf das hier betrachtete Radioproblem zu - es liegt eine sehr große Anzahl von Titeln, Künstlern oder Tags vor, aus denen jeder User nur eine kleine Teilmenge nutzt. Es liegt also nahe, ein *Market-Basket* Verfahren zu testen, auch wenn diese Verfahren üblicherweise keine Gewichtung der einzelnen Produkte erlauben.

SLR („Small/Large-Ratio“) ist die Verfeinerung eines Ansatzes von Ke Wang, Chu Xu und Bing Liu [56]. Deren Algorithmus ordnet die Transaktionen einzelnen Clustern zu und unterteilt dann alle Gegenstände in einem Cluster in *groß* und *klein*. Als *groß* werden dabei alle Gegenstände gezählt, die in einem zuvor festgelegten Anteil β oder mehr der Transaktionen dieses Clusters enthalten sind. Alle anderen bezeichnen Wang et al. als *klein*. Sei G_i die Menge aller *großen* Gegenstände in Cluster i, K_i analog die Menge aller *kleinen* und n die Anzahl der Cluster. So berechnen sich die Kosten anhand der Anzahl der als *klein* markierten Gegenstände innerhalb der Cluster und der Streuung der als *groß* markierten Gegenstände:

$$intra = \left| \bigcup_{i=1}^{n} K_i \right|$$

$$inter = \sum_{i=1}^{n} |G_i| - \left| \bigcup_{i=1}^{n} G_i \right|$$

intra steigt also, wenn eine große Anzahl von unterschiedlichen Gegenständen *klein* ist, *inter* steigt, wenn der selbe Gegenstand in unterschiedlichen Clustern *groß* ist. Die Gesamtkosten ergeben sich als

$$kosten = w \cdot intra + inter$$

Mit dem Parameter w lässt sich eine Gewichtung vornehmen, um intra oder inter stärker einfließen zu lassen. Der Algorithmus von Wang et al. berechnet für jede Transaktion die Kosten, die sich ergeben würden, wenn diese in einen Cluster eingefügt werden würde und entscheidet sich dann für den Cluster, bei dem die geringsten Kosten anfallen. Hierbei wird auch geprüft, ob die billigste Variante das Anlegen und Einfügen in einen neuen Cluster ist. In einer zweiten Phase wird versucht die gefundene Lösung durch Austausch zwischen den Clustern zu verbessern.

SLR erweitert dieses Konzept durch das Einfügen von mittleren Gegenständen, welche weder *groß* noch *klein* sind. Hierzu ist eine zweite Grenze $\alpha : \alpha < \beta$ nötig. Gegenstände gelten nun als *klein*, wenn ihr Anteil an allen Transaktionen des Clusters weniger als α beträgt. Auch die *Refinement*-Phase (also das Verbessern der gefunden Lösung) wurde für SRL geändert. Statt einer neuen Kostenberechnung für alle Cluster und Transaktionen und anschließenden Umsortierung wird in SRL eine zweite Größe, die Small-Large-Ratio verwenden. Sei $K_i(t)$ bzw. $G_i(t)$ die Menge der *kleinen* bzw. *großen* Gegenstände in Cluster i,

die in Transaktion t vorkommen:

$$srl(t) = \frac{|K_i(t)|}{|G_i(t)|},\ i : \mathit{Cluster\ der\ t\ enthält}$$

Alle Transaktionen deren Small-Large-Ratio kleiner als ein Grenzwert γ ist werden aus ihren Clustern entfernt; leere Cluster werden gelöscht. Die entfernten Transaktionen werden nun in den Cluster eingefügt, für den sich die beste Small/Large-Ratio ergeben würde, sofern diese über dem Grenzwert γ liegt. Wurden alle Transaktionen betrachtet, wird die *Refinement*-Phase wiederholt, bis keine Transaktionen mehr verschoben werden können. Möglicherweise bleiben dabei Transaktionen übrig, die in keinem Cluster untergebracht werden konnten. Diese sind als Ausreißer zu betrachten.

Als besondere Eigenschaften von SLR gilt es zu vermerken:

- Die Anzahl der Cluster muss nicht vorgegeben werden, sondern wird von SLR anhand der Daten ermittelt
- SLR sortiert Ausreißer eigenständig aus
- SLR kann keine Gewichtung einzelner Gegenstände berücksichtigen

5.1.2 Clarans

Clarans (Clustering Large Applications based on RANdomized Search) [43] ist eine von Raymond T. Ng und Jiawei Han vorgeschlagene Verbesserung der Cluster-Algorithmen Pam (Partitioning around medoids) [27] und Clara (Clustering LARge Applications) [27]. Alle drei nutzen das Konzept der *Medoids*, also einzelne zentrale Elemente um die ein Cluster gebildet wird. In einem ersten Schritt werden k verschiedene Medoids beliebig gewählt. Jeder Medoid repräsentiert einen Cluster, die restlichen Elemente werden jeweils dem Cluster zugeordnet, zu dessen Medoid die Distanz am geringsten ist. Diese Initiallösung wird in einem zweiten Schritt verbessert, indem ein Medoid ausgewählt und zu einem normalen Element degradiert wird. Ein anderes Element wird daraufhin als neuer Medoid ernannt. Über eine Kostenfunktion wird getestet, ob die nun gefundene Lösung besser ist als die vorherige. Als Kosten für den Tausch wird die Änderung der Summe der Distanz zum jeweiligen Medoid über alle Elemente angesehen. Sei E die Menge aller Elemente, $M \subset E$ die Menge aller Medoids, $e_i \in E/M$ ein Cluster-Element, $d(e_i, e_j)$ die Distanz zwischen zwei Elementen, $m_{opt1,i} \in M$ ein Medoid mit $min_{m \in M} d(e_i, m)$ vor dem Tausch und $m_{opt2,i} \in M$ ein Medoid mit $min_{m \in M} d(e_i, m)$ nach dem Tausch, so berechnen sich die Tauschkosten als:

$$cost = \sum_{e_i \in E/M} d(e_i, m_{i2}) - d(e_i, m_{i1})$$

Ist $cost < 0$ so ist der Tausch von Vorteil, ist $cost \geq 0$ so wird der Tausch verworfen. Entscheidend ist also, ob sich die Summe aller Distanzen zu ihrem jeweiligen Medoid mit dem Tausch verringert hat oder nicht.

Diese gemeinsame Ausgangsbasis teilen alle drei erwähnten Algorithmen, der Unterschied liegt in der Art und Weise, wie neue Medoid-Kombinationen ausprobiert werden. PAM berechnet *cost* für alle möglichen Paare aus $e_i \in E/M$ und $m_i \in M$ und führt dann den Tausch zwischen dem Element und dem Medoid aus, der die niedrigsten Kosten besitzt. Dies wird solange wiederholt, bis kein Tausch mit negativen Kosten mehr gefunden werden kann. CLARA zieht zu Beginn eine zufällige Stichprobe aus allen Elementen (die Autoren schlagen eine Probengröße von $20 + 2k$ vor) und berechnet mit Hilfe von PAM die passenden Medoids. Die restlichen Elemente werden nun jeweils dem nächsten Medoid zugeteilt und ergeben so eine Lösung. Dieser Vorgang wird mehrfach wiederholt und die beste Lösung schließlich zurück gegeben.

CLARANS hat sich in Experimenten als die effizienteste der drei Varianten herausgestellt [43]. Er beginnt ebenfalls mit einer beliebigen Auswahl von k Medoids. Aus diesen k Medoids wird einer zufällig ausgewählt und die Kosten für den Tausch mit einem zufälligen Cluster-Element berechnet. Dies wird solange wiederholt, bis ein Tauschpaar mit negativen Kosten gefunden wurde. Mit diesem wird der Tausch durchgeführt und ausgehend von der neuen Lösung wieder randomisiert nach einem Tauschpaar gesucht. Sollten für eine Lösung in $maxNeighbor$ zufälligen Zügen kein Tauschpaar mit negativen Kosten gefunden werden, so wird die derzeitige Lösung gemerkt und der Prozess beginnt mit der Bestimmung von k neuen Medoids von vorne. Insgesamt werden so $numLocal$ verschiedene Lösungen generiert, von denen die Lösung mit den insgesamt geringsten Distanzen zwischen allen Elementen und ihren Medoids als finale Lösung zurück gegeben wird. Die beiden Parameter $maxNeighbor$ und $numLocal$ müssen vom Benutzer je nach gewünschter Laufzeit und Ergebnisqualität gewählt werden. Ebenso muss der Nutzer die Zahl der gewünschten Cluster k wählen.

Als besondere Eigenschaften von CLARANS gilt es zu vermerken:

- CLARANS kann über die Distanzfunktion Gewichtung einzelner Titel berücksichtigen
- Die Anzahl der Cluster muss zuvor festgelegt werden
- CLARANS kann keine Ausreißer aussortieren
- CLARANS arbeitet randomisiert, Ergebnisse sind schwer reproduzierbar

5.1.3 MAXDIST

Basierend auf den in 5.1.2 vorgestellten Medoid-Ansatz wird speziell für diese Arbeit noch ein weiterer Ansatz entwickelt. Dieser einfache Algorithmus wird im weiteren mit MAXDIST bezeichnet und basiert auf der Idee, statt der Anzahl von Clustern k, einen maximalen Abstand (bzw. eine Mindestähnlichkeit) zwischen Medoid und Clusterelement vorzugeben. Ist ein Element zu weit von allen anderen Medoids entfernt, so wird es selber zu einem neuen Medoid erklärt. Dieser Ansatz ist hilfreich, da für die hier betrachtete Radio-Anwendung eine

```
1  Füge erstes Element in "medoids" ein
2
3  for e in elemente:
4
5      for m in medoids:
6
7          berechne dist = d(e,m)
8
9          wenn dist < maxDist:
10
11             füge e in clusterElemente ein
12             break            # Brich Schleife ab und
13                              # fahre mit nächstem e aus
14                              # elemente fort
15
16     # dist war für alle medoids größer als maxDist
17     # e wird neuer Medoid
18     füge e in medoids ein
19
20 for c in clusterElemente:
21
22     berechne Medoid m_min mit minimaler Distanz d(c,m)
23
24     füge c dem von m_min repräsentierten Cluster hinzu
```

Code 4: Pseudocode für Algorithmus MAXDIST

Mindest-Übereinstimmung innerhalb eines Clusters benötigt wird, um genug gemeinsame Titel zum Ausfüllen des Sendeplans zu erhalten. Zusätzlich lässt sich eine Mindest-Ähnlichkeit ohne vorherige Analyse der Daten leichter festlegen als die nötige Zahl der Cluster.

Aus diesen Überlegungen ist MAXDIST entstanden, dessen Vorgehensweise dem Pseudo-Code 4 entnommen werden kann.

Kurz zusammengefasst testet Maxdist zuerst, ob ein Element nah genug an mindestens einem Medoid ist. Ist dies nicht der Fall, wird das Element selbst zu einem Medoid ernannt. Liegt ein Element hingegen nahe genug an einem Medoid, so wird es in eine Warteschlange eingefügt. Nachdem alle Elemente auf diese Weise bearbeitet wurden, werden die Elemente der Schlange jeweils dem Medoid mit der geringsten Distanz zugeordnet.

Als besondere Eigenschaften von MAXDIST gilt es zu vermerken:

- MAXDIST kann über die Distanzfunktion die Gewichtung einzelner Titel berücksichtigen
- MAXDIST garantiert eine Mindest-Übereinstimmung zwischen einem Element und seinem Medoid (es gilt zu beachten: zwischen zwei Elementen des selben Clusters ist keine Übereinstimmung sichergestellt)
- Die Anzahl der nötigen Cluster wird dynamisch bestimmt
- Ausreißer werden zuverlässig in jeweils eigene Cluster gepackt

5.2 Cluster-Kriterien

Neben unterschiedlichen Algorithmen entscheidet auch die Wahl des Cluster-Kriteriums, also der Dateneigenschaft anhand der Cluster gebildet werden, über die Ergebnisqualität. Daher soll auch untersucht werden, welche Eigenschaft der

gesammelten Datensätze am besten geeignet ist Cluster zu bilden, für die Sendungsabläufe gemeinsam berechnet werden können. Im Folgenden werden 3 Kriterien betrachtet: Anzahl und Wertung gemeinsamer Titel zwischen zwei Usern, Anzahl und Wertung gemeinsamer Künstler und Tags von gehörten Titeln.

5.2.1 Anzahl gemeinsamer Titel

Das erste untersuchte Cluster-Kriterium betrachtet den Anteil von Musikstücken, die zwei Nutzer gemeinsam in der Liste ihrer gehörten Titel haben. Für die Distanz-Funktion wird die Anzahl der gemeinsamen Titel von der Zahl der Musikstücke in der größeren der beiden Listen abgezogen. Dieser Wert wird zusätzlich normiert, indem er durch die Zahl der Musikstücke in der größeren der beiden Listen geteilt wird:

$$dist_1(u_1, u_2) = \frac{max(|T_{u_1}|, |T_{u_2}|) - |(T_{u_1} \setminus T_{u_2}) \cup (T_{u_2} \setminus T_{u_1})|}{max(|T_{u_1}|, |T_{u_2}|)}$$

Dabei ist T_x die Menge der Titel, die der Nutzer x gehört hat. Das Zusammenfassen von Nutzern, die viele Titel gemeinsam gehört haben, soll sicherstellen, dass in Sendeplänen, die für einen solchen Cluster erzeugt werden, möglichst wenig Titel enthalten sind, die einzelne Mitglieder des Clusters nicht kennen. Allerdings berücksichtigt dieses Kriterium nicht, dass Nutzer Titel unterschiedlich gerne mögen. So ist es denkbar, dass zwei Nutzer eine hohe Ähnlichkeit bescheinigt bekommen, obwohl einer der beiden einen Titel t als sein absolutes Lieblingslied ansieht und es entsprechend oft hört, der andere hingegen das selbe Stück einmal gehört und es als für ihn unpassend verworfen hat.

5.2.2 Scoreunterschied der gemeinsamen Titel

Ähnlich wie das zuvor vorgestellte Cluster-Kriterium, basiert auch das Zweite auf den gemeinsam gehörten Titeln der Nutzer. In diesem Fall wird aber nicht nur die Existenz eines gemeinsamen Titels betrachtet, sondern auch die unterschiedlichen Score-Werte, die einem Titel und Nutzer zugeordnet sind, berücksichtigt. Die Distanzberechnung für CLARANS und MAXDIST erfolgt als

$$t_2(u_1, u_2) = \sum_{i \in T_{u_1} \cap T_{u_2}} 1 - \sqrt{|s_{i,u_1} - s_{i,u_2}|}$$

$$dist_2(u_1, u_2) = \frac{max(|T_{u_1}|, |T_{u_2}|) - t_2(u_1, u_2)}{max(|T_{u_1}|, |T_{u_2}|)}$$

mit T_{u_x} der Menge von Titeln des Nutzers x und s_{i,u_x} dem Scorewert des Titels i für User x. Der Wert von $t_2(u_1, u_2)$ kann damit zwischen 0 und $max(|T_{u_1}|, |T_{u_2}|)$ liegen. Für jeden Titel, den zwei Nutzer gemeinsam in der Menge ihrer Lieblingstitel haben, wird der Unterschied der jeweiligen Score-Bewertungen ermittelt. Da diese Unterschiede sehr kleine Zahlen ausmachen können, hat es sich als zweckmäßig erwiesen, sie durch eine Wurzelfunktion hervorzuheben. Die Unterschiede werden für alle Titel aufsummiert. Damit der Distanzwert für Nutzer

mit unterschiedlich großen Titelbibliotheken vergleichbar bleibt und damit die Anzahl der gemeinsamen Titel zu der Anzahl der nicht gemeinsamen Titel in Relation gesetzt werden kann, wird $t_2(u_1, u_2)$ in der zweiten Formel mit der Größe der Titelbibliothek auf einen Wert im Intervall $[0, 1]$ normiert. Dabei entspricht ein Wert von 0 zwei Nutzern mit identischer Titelbibliothek, eine Distanz von 1 bedeutet dass diese beiden Nutzer nichts gemeinsam haben.

Da SLR keine Gewichtung von Titeln unterstützt, sondern direkt auf den Titel-Mengen arbeitet, gibt es für das Verhalten von SLR keinen Unterschied zu 5.2.1.

5.2.3 Scoreunterschied gemeinsamer Interpreten

Die Clusterberechnung lässt sich vereinfachen, indem nicht mehr einzelne Titel, sondern die Interpreten betrachtet werden, da sich deutlich weniger Interpreten als Titel in der Datenbank finden. Wie zuvor soll auch hier der Score-Unterschied mit berücksichtigt werden. Die Berechnung des Distanzwertes erfolgt, indem zuerst die Summe der Score-Werte aller Titel eines Künstlers gebildet wird und diese dann mit der Summe der Scorewerte des selben Künstlers für einen anderen User verglichen wird:

$$t_3(u_1, u_2) = \sum_{k \in A_{u_1} \cap A_{u_2}} 1 - \sqrt{\left| \sum_{i \in T_{u_1,k}} s_{i,u_1} - \sum_{i \in T_{u_2,k}} s_{i,u_2} \right|}$$

mit A_{u_x} Menge der Künstler, die Nutzer x bisher gehört hat und $T_{u_x,k}$ der Menge aller Titel von Künstler k die Nutzer x in seiner Titelbibliothek hat. Wie zuvor wird auch dieser Wert noch normiert:

$$dist_3(u_1, u_2) = \frac{max(|T_{u_1}|, |T_{u_2}|) - t_3(u_1, u_2)}{max(|T_{u_1}|, |T_{u_2}|)}$$

Der Scoreunterschied gemeinsamer Interpreten von zwei Nutzern ist ein weniger strenges Cluster-Kriterium als der Scoreunterschied von gemeinsamen Titeln. Voraussichtlich werden damit größere Cluster entstehen. Allerdings spielen einzelne Interpreten nicht zwangsläufig immer die selbe Musikrichtung. Es ist also denkbar, dass ein Nutzer einige Titel eines Künstlers sehr gerne mag und andere des selben Künstlers ihm nicht gefallen. Diese Information geht beim Clustern nach Interpreten möglicherweise verloren.

5.2.4 Bevorzugte Genres/Tags

last.fm bietet seinen Nutzern die Möglichkeit Künstler und Titel mit sogenannten *Tags* zu versehen, also frei wählbaren Schlagwörtern. Die häufigsten Begriffe werden von *last.fm* dann in einer sogenannten Tag-Cloud angezeigt, in der Begriffe, abhängig von der Häufigkeit mit der sie vergeben wurden, in unterschiedlichen Größen und Farbtönen dargestellt werden. Die Roh-Daten für diese Tag-Cloud lassen sich auch über die API von *last.fm* abrufen. Dort erhält man eine

Liste mit den vergebenen Begriffen und dazu einen Score-Wert zwischen 0 und 100, der die relative Häufigkeit angibt. Da diese Tags besondere Charakteristiken eines Künstlers oder Titels wiedergeben, lassen sich darüber möglicherweise auch Rückschlüsse auf den Geschmack eines Nutzers ziehen. Es könnte also erfolgversprechend sein, Nutzer anhand der Tags ihrer Titel zusammenzufassen. Dazu werden für jeden Titel, den ein User gehört hat, die dazugehörigen Tags ausgewertet und nach Tag- und Titelscore gewichtet:

$$w(u,g) = \sum_{i \in T_u} s_{i,u} \times r_{i,g}$$

$$t_4(u_1,u_2) = 1 - \sqrt{\left| \sum_{g \in G} [w(u_1,g) - w(u_2,g)] \right|}$$

$$dist_4(u_1,u_2) = \frac{max(|T_{u_1}|, |T_{u_2}|) - t_4(u_1,u_2)}{max(|T_{u_1}|, |T_{u_2}|)}$$

Hierbei ist $r_{i,g}$ der Score-Wert des Tags g für den Titel i und G die Menge aller Tags.

Eine Gefahr bei diesem Cluster-Kriterium besteht darin, dass zwar Nutzer mit ähnlichem Musikgeschmack zusammengefasst werden, der für eine solche Gruppe erzeugte Sendungsplan aber trotzdem viele Titel enthält, den einzelne Nutzer nicht kennen.

6 Konzept

Aus den bisher vorgestellten Anforderungen und Möglichkeiten lässt sich nun ein Konzept für eine internetbasierte, personalisierbare Radiostation erstellen.

Eine Sendung wird von der Redaktion ohne feste Musiktitel geplant. Mit Hilfe einer Datenbank von Lieblingstiteln der Hörer kann das Programm dann personalisiert werden, indem die passenden Musikstücke für den Hörer eingefügt werden. Eine noch weitergehende Personalisierung ist möglich, wenn der Nutzer den Wort-Anteil des Programmes beeinflussen kann. Studien haben gezeigt, dass sich der gewünschte Anteil von Wortbeiträgen in unterschiedlichen Hörergruppen stark unterscheidet. Vor allem jüngere Nutzer erwarten von ihrem Radioprogramm viel Musik und wenig Wort, mit zunehmendem Alter verschieben sich diese Erwartungen hin zu einem höheren Anteil an Berichten und Informationen ([41], [39]). Diesen unterschiedlichen Wünschen lässt sich Rechnung tragen, indem geplante Beiträge mit einer Gewichtung versehen werden. Jeder Nutzer wählt seine individuelle *Wortstufe*. Beim Erzeugen des personalisierten Sendeplans kann das System dann alle Beiträge, deren Wichtigkeit geringer ist als die Wortstufe des Nutzers, übergehen und die dafür reservierte Zeit den umliegenden Musikblöcken zurechnen. Im XML-Formatentwurf für Sendungsschemata (siehe 7.4.1) sind für diesen Zweck Gruppen-Tags vorgesehen, mit denen sich zusammengehörige Elemente inhaltlich gruppieren lassen. Diese Gruppen, nicht aber einzelne Elemente, können mit einem Wichtigkeits-Wert versehen werden. Dies ist sinnvoll, da üblicherweise mehr als ein Element in einem Wortbeitrag zusammengehört - beispielsweise Anmoderation, die eigentliche Reportage und Abmoderation. Möchte ein Nutzer die Reportage nicht hören, so müssen auch die dazugehörigen Elemente „Anmoderation“ und „Abmoderation“ übersprungen werden.

Für die Realisierung eines solchen Programmes muss von der Planung der Sendung bis zur eigentlichen Verbreitung über den Internetstream eine weitgehende Digitalisierung des Funkhauses vorgenommen worden und die verwendeten Systeme gut miteinander integriert sein. Auf dem Markt gibt es heute bereits eine Reihe von Broadcast-Solutions, deren Ziel es ist, alle Arbeitsschritte in einer Redaktion mit einem Programmpaket abzudecken[21]. Die Funktionen solcher Pakete können von Schnittsoftware über die Einbindung von Nachrichtenagenturen, Planung von Sendungen, Verwaltung der Musikdatenbanken bis hin zur komplett automatisierten Durchführung von Sendungen reichen. Für die Realisierung eines komplett personalisierten Radios müsste eine entsprechende Software an mehreren Schlüsselpositionen des Senders zum Einsatz kommen und derart erweitert sein, dass unterschiedliche Programme an die Hörer ausgestrahlt werden können.

Um einen umfassenden Eindruck von dem geplanten Konzept zu erhalten, sollen die Workflows der drei am stärksten beteiligten Nutzergruppen des Systems betrachtet werden: die für die Vorbereitung der Sendung zuständige Redaktion, die für die Durchführung zuständigen Moderatoren und die das Angebot

[21] Beispielsweise die *d'accord radio.suite* oder *DABiS800*

nutzenden Hörer.

Vor Beginn einer jeden Sendung muss diese von einem verantwortlichen Redakteur geplant werden. Die zuvor von der Redaktion festgelegten Themen und Inhalte der Sendung müssen über die Laufzeit der Sendung verteilt werden. Für diese Aufgabe sollte ein Redaktionssystem zur Verfügung stehen, mit dem feste Sendungselemente, die an alle Hörer ausgestrahlt werden sollen, wie beispielsweise Moderationen oder Beiträge als Bausteine in der Sendestunde angeordnet werden können. Um den Hörern die Möglichkeit zu geben, auch einen individuellen Wortanteil festzulegen, müssen in diesem Schritt auch alle Bausteine mit einer Wichtigkeit versehen werden. Eine Möglichkeit mit Vorlagen zu arbeiten, die die grobe Struktur einer Sendung vorgeben, könnte diese Arbeit zusätzlich vereinfachen. Sind alle Elemente der Sendung untergebracht, kann das Redaktionssystem automatisch den Rest der Zeit mit Musikblöcken füllen und das erstellte Sendungsschema in ein standardisiertes Format übertragen. Zum Beispiel in das für diese Arbeit konzipierte XML-Format (siehe Kapitel 7.4.1). Für die Ausstrahlung der Sendung wird das erstellte Schema eingelesen und für alle zugeschalteten Nutzer mit Musik gefüllt.

Für den Moderator einer Sendung ändert sich wenig gegenüber seiner bisherigen Arbeit. Es kann davon ausgegangen werden, dass er in einem der, heute weit verbreiteten, *Selbstfahrerstudios* arbeitet (falls nicht, so erfolgt die Betrachtung analog für den zuständigen Techniker). Das heißt, es gibt keinen Techniker, der aus der Regie die Sendung abwickelt. Stattdessen verfügt der Moderator direkt im Studio über ein Mischpult und verschiedene Abspielgeräte. Er ist für das Starten von Titeln und Einpegeln der Musik selbst verantwortlich. In einem digitalisierten Studio ist das Mischpult direkt mit einem Rechner verbunden und steuert das darauf laufende Playout-System. So kann beispielsweise eine Musikdatei direkt gestartet werden, sobald der zugehörige Kanal am Mischpult geöffnet wird. Im personalisierbaren Radio wird das Öffnen eines Kanals nicht ein bestimmtes Musikstück starten, sondern stattdessen den Playoutrechner anweisen die vorberechneten, individuellen Musikblöcke für jeden Nutzer abzuspielen. Selbstverständlich kann ein Moderator eines solchen Senders nicht mehr auf individuelle Titel eingehen, er kann aber, abgesehen davon, seinem gewohnten Arbeitsablauf folgen. Auch das Zuspielen von Beiträgen, Jingles und sonstigen Tonelementen, die an alle Nutzer gleichzeitig gesendet werden sollen, könnten direkt über das Mischpult und den Senderechner gestartet erfolgen.

Für einen Nutzer sollten die Einstiegshürden möglichst gering sein. Das heißt auch ein Nutzer, der nicht bereit ist, sich genauer mit dem System zu beschäftigen, sollte möglichst nach einigen wenigen Klicks bereits ein Programm zugespielt bekommen. Bisherige Anbieter von personalisierten Musikstreams (siehe Kapitel 3.2.2) lösen dies, indem sie bereits nach der Eingabe eines Lieblingstitels oder Lieblingskünstlers beginnen Musik abzuspielen. Der Nutzer bekommt dann während dem laufenden Stream die Möglichkeit, die vom Dienst vorgeschlagen Titel zu bewerten und damit die Auswahl des nächsten Titels zu beeinflussen. Damit kann sich das Programm an den Geschmack anpassen, während dem User bereits Musik zugespielt wird. Für ein echtes personalisierbares Radio ist dies nur bedingt möglich, da das Musikprogramm für eine gewisse Zeit vorberechnet

wird und jede Anpassung eine neue Berechnung erfordern würde. Es muss also vor der eigentlichen Berechnung bereits mehr über den Geschmack des Nutzers bekannt sein. Eine Lösung für dieses Problem stellt die verzögerte Zuschaltung in das Programm da: hierzu wird für einen neuen User nach Eingabe eines Titels, wie auch bei anderen Diensten, zuerst ein einfacher Musikstream gespielt, dessen Entwicklung der Nutzer durch Feedback (beispielsweise durch Vergabe von 0 bis 5 Punkten für den derzeit gespielten Titel) direkt beeinflussen kann. Ist auf diese Weise der Musikgeschmack des Hörers ausreichend genau definiert, berechnet das System den persönlichen Sendungsfahrplan für den Nutzer und schaltet dann auf das Programm mit Moderationen um. Selbstverständlich kann auch weiterhin Feedback zu der gespielten Musik gegeben werden. Diese wirkt sich dann allerdings erst bei der nächsten Neuberechnung aus, es muss also durch die erste Geschmacksbestimmung sichergestellt sein, dass die Musikauswahl nicht zu stark am Nutzergeschmack vorbei geht. Damit ein Hörer seinen Geschmack nicht jedesmal aufs Neue angeben muss, wird auf dem Server eine Liste aller bewerteten Titel gespeichert und diese für die zukünftige Musikauswahl verwendet.

Die Ausstrahlung des Programmes kann entweder als ein serverseitig gemischtes Signal unter Verwendung herkömmlicher Streaming-Technologien erfolgen oder über einen beim Nutzer installierten Client, der das finale Signal mischt. Die heute verbreiteten personalisierbaren Musikangebote wie *last.fm* oder *Pandora* übertragen Musikstücke einzeln an ihre Hörer und unterscheiden sich damit auf der Serverseite nicht besonders von beliebigen anderen Downloadangeboten. Der zum Angebot gehörende Server ermittelt den nächsten Titel und sendet diesen als einfache Datei an den Player. Ein reguläres Radioprogramm hingegen erstellt während der Produktion eine Signalmischung verschiedener Audioquellen und sendet die finale Summe an seine Hörer. Dadurch ist es möglich, Titel ineinander übergehen zu lassen ("fading"), Moderationen und Beiträge mit Hintergrundmusik zu unterlegen, Gespräche mit mehreren beteiligten Personen zu realisieren oder Ansagen erst im Intro eines Titels enden zu lassen ("ramp talking"). Um diese Möglichkeiten auch im personalisierten Radio zur Verfügung zu stellen, ist es nötig, für jeden zugeschalteten Hörer einen eigenen Signalmix zu erzeugen. Dies könnte, mit entsprechendem Ressourcen-Verbrauch, serverseitig erfolgen. Alternativ könnte die Mischung auch ein spezieller Client übernehmen, indem das Signal in zwei Komponenten aufgeteilt wird. Ein vorgemischter globaler Stream enthält alle Signale, die an jeden Hörer gleichzeitig gesendet werden sollen (beispielsweise ein mit Hintergrundmusik unterlegtes Studiogespräch mit mehreren Gästen und darauf folgender Moderation). Die für jeden Hörer individuellen Musiktitel lädt der Client als einzelne Dateien. Auf Kommando vom Server mischt der Client zwei (oder mehrere) Komponenten und realisiert auf diese Weise ein Fading oder Ramp-Talking. Da diese Vorgehensweise allerdings zeitkritisch ist setzt es ein funktionierendes Echtzeit-Protokoll zwischen Server und Client voraus, damit die Musikblöcke aller Hörer gleichzeitig starten und auch mit Beginn der nächsten Live-Moderation enden. Eine solche Client-seitige Mischung spart Ressourcen beim Anbieter, setzt aber auch kompliziertere Protokolle voraus. Außerdem sollte beachtet werden, dass ein solcher Client für jedes

Zielgerät angepasst werden müsste, ein serverseitig gemischter Stream könnte ohne weiteren Entwicklungsaufwand auf einer Vielzahl von mobilen und nicht mobilen Geräten empfangen werden.

7 Implementierung

7.1 Verwendete Techniken

Die Implementierung aller Algorithmen erfolgt in Python Version 2.6.5. Python wurde wegen der großen Auswahl an vorgefertigten Modulen und der einfachen Zusammenarbeit mit verschiedenen Datenbanksystemen gewählt. Außerdem verfügt Python über eine gute Unterstützung des XML-Formates, das von der *last.fm* API verwendet wird und für die Ein- und Ausgabe der Sendepläne benutzt werden soll. Allerdings bietet Python keinen Zugriff auf verschiedene hardware-nahe Funktionen wie Pointer oder einen direkten Speicherzugriff. Benötigt ein Algorithmus derartige Funktionen, so werden sie in Python unter Verwendung komplexerer Datentypen simuliert.

Zum Speichern der Titel- und Nutzerdatenbank kommt ein MySQL Datenbankserver der Version 5.1.41 zum Einsatz. Als Datenbank-Engine wird InnoDB anstatt des MySQL-Standards MyISAM verwendet, da InnoDB eine Unterstützung von Transactions und Key-Constraints bietet, welche bei der Sicherstellung konsistenter Datensätze helfen.

Für die Implementierung einiger Algorithmen wird zusätzlich eine BerkeleyDB in der Version 4.8 zur Zwischenspeicherung von Daten genutzt.

7.2 Anforderungen/Umfang

Für diese Arbeit wird ein Test-Framework zur Generierung von personalisierten Sendungsfahrplänen mit verschiedenen Algorithmen erstellt. Das Framework ist in der Lage aus vorgegebenen Sendungsschemata für eine Menge von Nutzern personalisierte Sendungsfahrpläne im XML-Format zu erstellen, bei denen Musikblöcke mit Titeln aus der Menge der gehörten Musikstücke eines Nutzers gefüllt werden. Diese Sendungsfahrpläne sind derart formatiert, dass sie zur Produktion eines personalisierbaren Radios mit Live-Moderationen durch ein entsprechendes Sendesystem verwendet werden könnten. Zur Erstellung der Sendungsfahrpläne wird eine Datenbank mit Musiktiteln und Titel-Nutzerrelationen verwendet, welche auf Basis der Daten von *last.fm* erstellt wurde. Das Framework soll ferner in der Lage sein, die erzeugten Sendungsfahrpläne automatisiert auszuwerten und Informationen zu denen in Kapitel 4.5 definierten Qualitäts-Kriterien zur Bewertung der verwendeten Algorithmen zu erzeugen.

Ein zweites Framework verwendet die gesammelten Nutzer-Daten zur Erzeugung von Nutzergruppen mit ähnlichem Musikgeschmack. Auch hier sollen verschiedene Algorithmen erprobt und Daten über die erzeugten Ergebnisse gesammelt werden.

7.3 Datenbasis

Die Nutzer und Titel-Daten für diese Arbeit stammen von *last.fm* und wurden über den von *last.fm* angebotenen API-Zugang gewonnen. Im Zeitraum von

Tabelle	Felder	Indizes[1]	Constraints
artist	artistUrl, artistName, artistMbid	*artistUrl	-
tagrelationenArtist	tagID, tagZiel, score	tagZiel, *<tagID-tagZiel>	tagID - tags.tagID, tagZiel - artist.artistUrl
tagrelationenTitel	tagID, tagZiel, score	tagZiel, *<tagID-tagZiel>	tagID - tags.tagID, tagZiel - titel.titelUrl
tags	tagID, tagName	*tagID, tagName	-
titel	titelUrl, titelName, artistUrl, laenge	*titelUrl, artistUrl	artitsUrl - artist.artistUrl
user	userName, wortstufe	*userName	-
userTitel	userName, titelUrl, playcount, score	userName, titelUrl, *<userName-titelUrl>	userName - user.UserName, titelUrl - titel.titelUrl

[1]Mit Sternchen (*) versehene Einträge bezeichnen den Primärschlüssel, spitze Klammern symbolisieren kombinierte Indizes über mehr als eine Spalte

Tabelle 1: Datenbankstruktur (Der SQL-Code zur Erzeugung der Datenbank findet sich in Anhang B)

Februar bis März 2011 wurden auf diese Weise insgesamt etwa 65 Millionen Datensätze abgerufen und in eine MySQL Datenbank eingefügt. Die genaue Struktur der Datenbank wird in Kapitel 7.3.1 erläutert. Insgesamt wurden Daten für 17609 Benutzer abgerufen. Um möglichst wenig personenbezogene Daten zu speichern, wurden während dieses Vorgangs alle Benutzer anonymisiert und die eigentlichen Benutzernamen durch zufällig erzeugte Pseudonyme ersetzt. Für jeden Benutzer wurden die am häufigsten gehörten Titel (bis zu 5000 pro Nutzer) gespeichert. Zusätzlich wurden Informationen zu Titeln und Künstlern sowie von den Nutzern vergebene Tags in die Datenbank eingefügt. Details zum Umfang und zur Verarbeitung der Daten finden sich in Kapitel 7.3.2.

7.3.1 Struktur der Datenbank

Die gesammelten Daten wurden in sieben MySQL Tabellen gespeichert. Die speziell für dieses Forschungssprojekt gewählte Tabellenstruktur ist vor allem auf effizientes Auslesen der Daten ausgelegt. Für ein Produktiv-System mit sich ändernden Datensätzen sollte eine andere, besser geeignete Struktur gewählt werden. Folgende sieben Tabellen speichern die gesammelten Daten:

1. artist - enthält alle Interpreten: Eindeutig identifizierbar über die *artistUrl*.
2. tagrelationenArtist - stellt eine Verbindung zwischen einem Künstler und den diesem Künstler zugewiesenen Tags her. Ein Eintrag besteht aus *tagID*, *tagZiel* (eine *ArtistUrl*) und einem *score*-Wert, der angibt, wie oft ein Tag zu einem Künstler vergeben wurde.
3. tagrelationenTitel - analog zu 2.

Tabelle	Datensätze vor Bereinigung	Datensätze nach Bereinigung
artist	683.907	173.962
tagrelationenArtist	5.447.364	3.872.373
tagrelationenTitel	13.914.456	12.483.857
tags	1.137.255	1.016.016
titel	6.559.311	2.056.806
user	17.609	14.807
userTitel	39.952.931	33.647.208

Tabelle 2: Datensatzanzahl vor und nach der Datenbereinigung

4. tags - alle vergebenen Tags, kombiniert mit einer eindeutigen, numerischen *tagID*.

5. titel - alle Titel, eindeutig identifiziert über eine *titelUrl*. Ein Feld *artistUrl* stellt eine Verbindung zum Eintrag des Interpreten in der Tabelle *artist* her. Das Feld *laenge* enthält die Laufzeit des jeweiligen Titels.

6. user - enthält alle User mit ihrem jeweiligen Benutzernamen. Zusätzlich existiert das Feld *wortstufe*, dessen Daten nicht von *last.fm* stammen, sondern für Experimente zufällig ausgefüllt wurde.

7. userTitel - enthält die Informationen über die gehörten Titel aller User. Die Felder *userName* und *titelUrl* stellen eine Verbindung zur *user*- bzw. *titel*-Tabelle her. *playcount* und *score* geben an, wie oft ein Titel von diesem User insgesamt bzw. wie oft er prozentual gehört wurde.

Eine komplette Übersicht über alle verwendeten Tabellen, ihre Felder, sowie die definierten Indizes und Constraints findet sich in Tabelle 1.

7.3.2 Vorhandene Daten

Die von *last.fm* gesammelten Daten sind nutzergeneriert und daher nicht zuverlässig. Viele Titel und Künstler existieren in unterschiedlichen Schreibweisen mehrfach in der Datenbank. Um trotzdem möglichst gute Ergebnisse zu erreichen, wurden die Daten bereinigt. *last.fm* bietet über seine API eine Korrekturfunktion an, die für einige falsche oder strittige Schreibweisen und Tippfehler eine allgemein anerkannte Schreibart zurück liefert. Mit dieser Funktion wurden alle Titel und Künstler überprüft und die so entdeckten Variationen zu einer Schreibweise zusammengefasst.

Weiterhin enthält die Datenbank einige Titel, die nur von jeweils einem User gehört wurden - entweder weil sie der Phantasie dieses Users entsprungen sind, weil die Schreibweise derart ungewöhnlich ist, dass sie nicht mit anderen Variationen zusammengefasst werden konnte oder weil sie so exotisch sind, dass kein anderer Nutzer in der Stichprobe diesen Geschmack teilt. Solche Titel wird

eine Internet-Radiostation mit großer Wahrscheinlichkeit nicht auf ihren Servern zum Abspielen vorhalten. Daher scheint es sicher, dass solche Titel aus der Datenbank entfernt werden können. Dies gilt analog für exotische Künstler. Zusätzlich wurden auch noch alle Titel mit einer Laufzeit von unter 30 Sekunden entfernt und alle Nutzer mit weniger als 500 Titeln gelöscht.

Tabelle 7.3.2 enthält eine Übersicht über die Anzahl aller Datensätze vor und nach der Bereinigung.

7.4 Architektur

Der komplette Testaufbau besteht aus 4 Haupt-Komponenten:

Ein Eingabeparser liest den im XML vorliegenden Sendeplan. In einem Produktiv-Umfeld könnte dieser von einer Redaktion mit einem dafür ausgelegten Planungssystem erstellt und in das passende XML-Format exportiert werden. Eine nähere Betrachtung des verwendeten Formats findet sich in Kapitel 7.4.1. Der Eingabeparser validiert die übergebene XML-Datei zuerst und erzeugt dann für die interne Darstellung eine Baumstruktur, bei der jedes XML-Element einem Knoten entspricht[22].

Aus diesem Baum werden im nächsten Schritt alle Musikblöcke extrahiert. Dabei werden direkt aufeinander folgende Musikblöcke zusammengelegt und optionale Wortbeiträge (siehe Kapitel 6), sofern möglich, einem angrenzenden Block hinzugerechnet. Die Menge aller Musikblocklängen wird in einer Liste zusammengefasst und an den Rucksack-Problemlöser (*KPLöser*) übergeben. Die Algorithmen zur Lösung des Rucksack-Problems sind dabei in eigenständige Python-Module verpackt, welche über eine einheitliche Schnittstelle angesprochen und sich dynamisch zur Laufzeit laden lassen. Damit kann dasselbe Nutzer-Set mit verschiedenen Algorithmen bearbeitet und die Ergebnisse verglichen werden.

Der KPLöser erhält neben der Liste mit Längenangaben auch die Menge der Lieblingstitel eines Nutzers mit deren jeweiligen Längen und Score-Werten. Soll nicht nur für einen Nutzer alleine, sondern für einen ganzen Cluster eine Lösung ermittelt werden, so wird an dieser Stelle eine gemeinsame Liste mit der Schnittmenge der Lieblingstitel der User im Cluster übergeben. Mit diesen Informationen löst der KPLöser das Problem mit einem der in Kapitel 4 vorgestellten Algorithmen und gibt für jeden Block eine Liste mit den ausgewählten Titeln zurück.

Die Einträge dieser Liste werden in XML-Elemente verpackt, welche an Stelle der ursprünglichen Musikblöcke in den XML-Baum eingesetzt werden. Abschließend wird aus dem Baum wieder ein XML-Dokument erzeugt und dieses ausgegeben.

Dieses Dokument könnte in einer Produktiv-Umgebung von der für die Sende-Abwicklung zuständigen Server-Software verarbeitet und damit der Ablauf des personalisierten Programmes gesteuert werden.

[22] Die Validierung und Verarbeitung der XML-Dateien erfolgt mit Hilfe der Open Source Bibliothek lxml

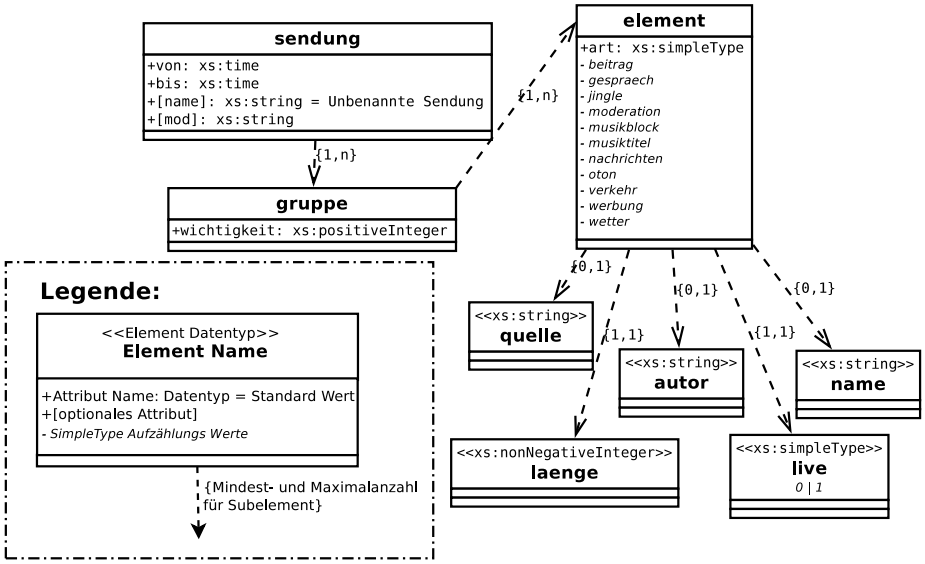

Abbildung 2: Grafische Darstellung des XML-Schemas der Ein- und Ausgabe-Dateien

7.4.1 Eingabe- und Ausgabeformat

Die Ein- und Ausgabe der Sendeschemata bzw. Sendepläne erfolgt über XML-Dateien, deren Format durch ein Dokumenten-Schema fest definiert ist. Als Schema-Beschreibungs-Sprache kommt das vom World Wide Web Consortium veröffentlichte W3C XML Schema (XSD)[23] zur Anwendung. Die Formatdefinition umfasst 8 Element-Typen und sechs Attribute, deren Abhängigkeiten in Abbildung 2 aufgezeigt werden. In jeder Sendeschema-Datei können beliebig viele Sendungen definiert sein. Der geplante Zeitraum für eine Sendung ist in den Attributen *von* und *bis* angegeben. Diese Attribute dienen nur zur besseren Übersichtlichkeit und werden bei der Erzeugung der Sendepläne nicht berücksichtigt. Das Gleiche gilt für die optionalen Attribute *name* für den Sendungsnamen und *mod* für den Namen des zuständigen Moderators. Jede Sendung besteht aus mindestens einer, meistens aber mehreren Gruppen. Eine solche Gruppe bezeichnet eine zusammenhängende Untereinheit im Programm, wie zum Beispiel Nachrichten, Wettervorhersage und Verkehrsinformationen oder einen Beitrag und seine An- und Abmoderation davor bzw. danach. Jede Gruppe muss ein Attribut *wichtigkeit* besitzen. Hiermit lässt sich der Wort-Anteil eines Programmes für jeden Nutzer individuell regeln: der Sendeplangenerator überspringt - je nach Usereinstellung - Gruppen mit einer zu geringen Wichtigkeit und ersetzt sie durch Musik. Die Untereinheiten einer Gruppe werden als Element bezeichnet. Diese symbolisieren die einzelnen Teile einer Sendung,

[23] W3C Recommendation 28 Oktober 2004 : http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/ und http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/

wie zum Beispiel eine Moderation, einen Beitrag oder einen Musiktitel. Um die einzelnen Arten zu unterscheiden, muss für jedes Element das Attribut *art* angegeben und mit einem von 11 möglichen Werten versehen sein. Alle Werte können Abbildung 2 entnommen werden. Alle weiteren Eigenschaften eines solchen Programm-Elementes sind durch maximal 5 weitere Unterelemente definiert:

- *laenge* (positive Zahl): Gibt die geplante Länge des Elements in Sekunden an.
- *live* (1|0): Zeigt an, ob es sich um ein Live-Element handelt (beispielsweise eine Moderation oder ein nicht aufgezeichnetes Gespräch). Dies ist für die hier durchgeführten Experimente nicht interessant, aber eine wichtige Information für ein mögliches Produktiv-System. Da Live-Elemente möglicherweise länger oder kürzer als die ursprünglich geplante Länge dauern, sollte ein Produktiv-System auf eventuell notwendige Änderungen des Sendeplans vorbereitet sein.
- *name* (string, optional): Ein Bezeichner für das Element. Dient der Übersichtlichkeit, wird in dieser Arbeit nicht ausgewertet.
- *autor* (string, optional): Der für das Element verantwortliche Autor oder Künstler. Dient der Übersichtlichkeit, wird in dieser Arbeit nicht ausgewertet.
- *quelle* (string, optional): Angabe von welcher Quelle (beispielsweise ein anderer Server, ein externes Gerät wie ein CD-Player und ähnliches) ein Element kommt. Diese Information ist wichtig für ein Produktiv-System, wird in dieser Arbeit aber nicht ausgewertet.

```
w = 0                       # Länge aller Titel in der bisher gefundenen Teil-Lösung
ret = list()                # Rückgabe Liste

for index, item in enumerate(self.titelbib):    # iteriere über alle Titel
    if "used" in item:                          # Um Länge von self.titelbib
                                                # nicht zu verändern, werden
                                                # genutzte Titel nur markiert
                                                # und nicht gelöscht
        if item["used"] == True:
            continue

    if w + item['laenge'] < blocksize:          # Füge Titel in die Lösung ein,
                                                # falls seine Länge plus die
                                                # bisher verbrauchte Länge (w)
                                                # die Größe des betrachteten
                                                # Blocks nicht übersteigt
        w += item['laenge']
        ret.append(item)
        self.titelbib[index]["used"] = 1
        self.usedCounter += 1

        if w == blocksize:                      # Brich die Schleife vorzeitig
                                                # ab, wenn Block komplett
                                                # ausgefüllt ist
            break

return ret
```

Code 5: Python-Implementierung von GREEDY

7.5 Knapsack Algorithmen

7.5.1 GREEDY

Für das in dieser Arbeit betrachtete Problem wird der GREEDY-Algorithmus für jeden zu füllenden Musikblock aufgerufen. Dabei entfernt GREEDY die verbrauchten Musiktitel nicht, sondern markiert sie lediglich als „used" und überspringt sie beim nächsten Aufruf. Offensichtlich hat diese Vorgehensweise negative Auswirkungen auf die Anzahl der nötigen Arbeitsschritte während der Schleifendurchläufe, spart aber das aufwändige Entfernen von Listenelementen[24]. Zusätzlich eröffnet dieses Vorgehen eine einfache Möglichkeit, einzelne Titel nach einer Zeit wieder freizugeben, um in einer finalen Anwendung beliebte Stücke mehrfach in das Programm einbauen zu können.

7.5.2 KPTREE

Die Implementierung von KPTREE ist sehr aufwändig und weicht in mehreren Punkten von den Vorgaben von Kellerer et al. [29] ab. Dies geschieht, zum einen um den in Kapitel 7.1 geschilderten Begrenzungen von Python Rechnung zu tragen und um einige eigene Verbesserungen umsetzen zu können. Um den benötigten binären Baum zu realisieren, wird eine Knoten-Klasse angelegt. Deren Objekte werden durch Referenzen untereinander verbunden. Damit sowohl

[24] Das Ändern von Listen ist eine aufwändige Operation in Python. Für einen Test wurden 100 mal zwei Listen mit 300.000 zufälligen Einträgen (nummerische Werte zwischen 0 und 500.000) angelegt. Bei einer wurden nacheinander alle Einträge mit einem „x" als gelöscht markiert, bei der anderen wurden alle Einträge tatsächlich aus der Liste entfernt. Dabei ergab sich auf dem Testsystem eine durchschnittliche Laufzeit von 685,16 Sekunden für das schrittweise Entfernen aller Elemente, das einzelne Markieren aller Einträge benötigte dagegen nur 0,27 Sekunden.

```
class TreeNode:

    def setItem(self, value, itemId):          # Objekt repräsentiert ein Blatt und
                                               # erhält einen Wert und eine Referenz
                                               # auf den zugehörigen Gegenstand
        self.value["self"] = value
        self.itemId = itemId
        self.updateValueToParent()             # Das Objekt gibt den geänderten Knoten-
                                               # wert selbstständig an den Elternknoten
                                               # weiter
        return True

    def updateValueToParent(self):             # Triggert Funktion im Eltern-Knoten,
                                               # damit dieser seinen Wert aktualisiert
        if self.parent is not None:
            self.parent.updateValueFromChild(self.parentPos)

    def updateValueFromChild(self, pos):   # Funktion wird von Kind-Knoten ausgelöst,
                                           # wenn sich deren Wert geändert hat

        self.value[pos] = self.children[pos].getValue() # ruft den aktualisierten
                                                        # Wert des Kindes ab
        self.minValue[pos] = self.children[pos].getMinValue() # Jeder Knoten
                                                              # speichert die jeweilige
                                                              #  Minimalgewichte der
                                                              # beiden darunterliegenden
                                                              # Teilbäume
        self.updateValueToParent()             # Löst Wert-Aktualisierung im Eltern-
                                               # knoten aus

    def getMinValue(self):                     # Liefert das minimale Gewicht der
                                               # beiden nachfolgenden Teilbäume zurück
        minValue = min(self.minValue)
        if minValue == float("inf"):
            return self.value["self"]
        else:
            return minValue

    def getValue(self):                        # Gibt den eigenen Wert zurück
        return sum(self.value.values())
```

Code 6: KPTREE - Ausschnitt aus der Knotenklasse für den selbstorganisierenden Baum

eine schnelle Suche von der Wurzel aus, als auch ein effizientes Weitergeben geänderter Gewichte von den Blättern ausgehend möglich ist, erhält jedes Knotenobjekt die Referenzen seiner (bis zu) zwei Kinder und die Referenz auf seinen Elternknoten. Die Klasse ist so angelegt, dass alle Gewichtsänderungen an einem Blatt automatisch an den Elternknoten gemeldet werden. Dieser wiederum aktualisiert sein eigenes Gewicht und gibt diese Änderung in Richtung Wurzel weiter. Somit stellt der Baum automatisch sicher, dass das Gewicht eines inneren Knotens immer der Summe der Gewichte seiner Kinder entspricht.

Eine Schwäche in dem Original-Entwurf für KPTREE ist, dass beim Durchlauf des Baumes von der Wurzel aus nicht erkannt werden kann, ob sich im gerade betrachteten Teilbaum überhaupt Gewichte finden, die noch in den Rucksack gepackt werden könnten. Um diesem Problem zu begegnen, speichert in der hier verwendeten Implementierung jeder Knoten zusätzlich das Minimalgewicht aller Blätter im Teilbaum. Dieses kann einfach, wie auch die Summe der Gewichte, von den Blättern ausgehend in Richtung Wurzel durchgereicht und in jedem Knoten auf dem Weg gemerkt werden. Mit dieser Erweiterung kann KPTREE Teilbäume mit unpassenden Gegenständen frühzeitig entdecken und diese meiden. Code 6 verdeutlicht den Aufbau des Suchbaumes.

KPTREE durchläuft den aufgebauten Baum mehrfach. In jedem Durchlauf wird in absteigender Reihenfolge der profitabelste Gegenstand fixiert und die

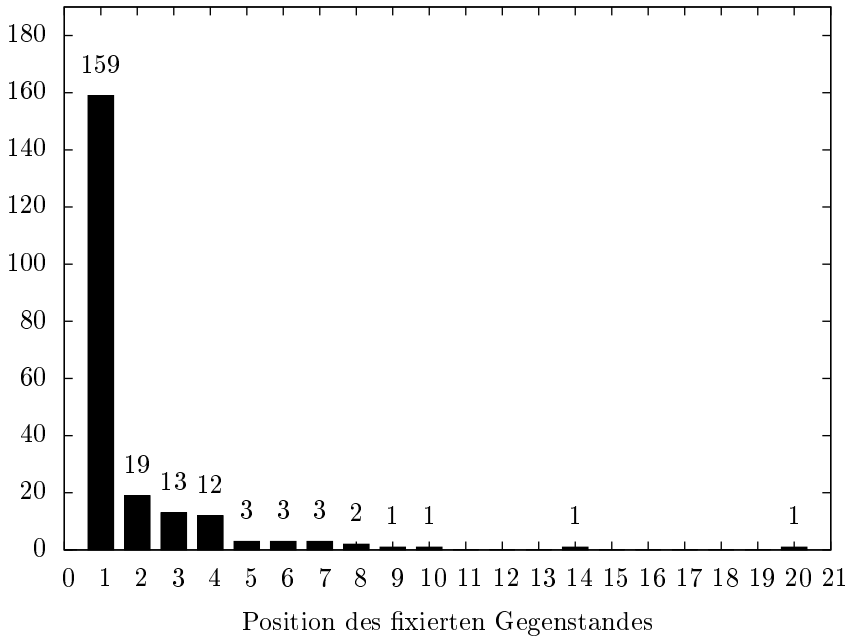

Abbildung 3: Verteilung der Position des besten fixierten Gegenstandes

verbleibende Kapazität per Baumsuche gefüllt. Im nächsten Durchlauf werden jeweils alle zuvor fixierten Gegenstände nicht mehr betrachtet. Aus allen Durchläufen ermittelt KPTREE das beste Ergebnis und gibt dieses zurück. Um den von Kellerer et al. bewiesenen maximalen Fehler von $\varepsilon = \frac{1}{3}$ zu erreichen, muss dieser Vorgang für jeden der n Gegenstände erfolgen. Experimente haben allerdings gezeigt, dass KPTREE nur sehr selten weniger profitable Gegenstände als „besten Gegenstand“ auswählt. Die Laufzeit des Algorithmus lässt sich also deutlich verbessern, wenn der Algorithmus nur die k profitabelsten Gegenstände jeweils als „bester Gegenstand“ fixiert und damit auch nur k Baumdurchläufe nötig sind. In einem Vorabexperiment wurde KPTREE ohne Beschränkung für 218 Musikblöcke ausgeführt. Dabei wurde aufgezeichnet, an welcher Position der fixierte Gegenstand gefunden wurde, der das beste Ergebnis geliefert hat. Dabei zeigte sich, dass in über 99% der Fälle das beste Ergebnis mit der Fixierung einer der ersten 10 Gegenstände erreicht wird (siehe Abbildung 3). Aus diesem Grund erscheint es sinnvoll, die Anzahl der Schleifendurchläufe auf $k = 10$ zu begrenzen und so die Laufzeit zu reduzieren.

7.5.3 KPTREEEXT

Die Ergebnisse für den Algorithmus KPTREE offenbaren eine deutliche Schwäche. Hat ein Nutzer einige besonders lange Titel in seiner Titelbibliothek, so

kommt es vor, dass KPTREE einen solchen Titel erst sehr spät von der Kandidaten-Liste streicht und dadurch einen sehr großen Fehlbetrag erzeugt. In einigen Fällen blieben über 1000 Sekunden ungefüllt. Eine genauere Analyse dieses Problems findet sich bei den experimentellen Ergebnissen in Sektion 8. Um dieses Problem zu beseitigen, wird KPTREE um eine Korrektur-Routine erweitert. Der Algorithmus überprüft, ob die Gesamtlänge der Titel in seinem gefundenen Ergebnis um mehr als den Faktor g von der maximal verfügbaren Länge abweicht. Ist dies der Fall so startet eine neue Baumsuche mit der verbleibenden Restkapazität. Diese beschränkt sich lediglich auf solche Blätter, die sich weiter rechts im Baum befinden als das am weitesten rechts befindliche Blatt der bisherigen Lösung. Als geeigneten Wert für g wurde $0,85$ experimentell ermittelt.

7.5.4 MK1, MK2, MK3, I1 und I2

Die Implementierung der fünf von Martello und Toth vorgeschlagenen Algorithmen erfolgt sehr nah an den Originalvorgaben. Lediglich die Begrenzung durch die in Python verfügbaren Datenstrukturen muss umgangen werden. Hierzu sind an einigen Stellen zusätzliche Sortier- und Kopiervorgänge nötig, wodurch die theoretisch mögliche beste Laufzeitkomplexität nicht erreicht wird.

Nach einigen ersten Experimenten zeigte sich, dass bei MK3 mit den gegebenen Daten bei einer hohen Anzahl von Musikblöcken immer wieder vereinzelten Blöcken kein einziger Titel zugewiesen wurde. Der Grund hierfür liegt in der Art und Weise, mit der MK3 seine Lösung korrigiert. Wird ein Gegenstand in mehr als einen Rucksack verpackt, so versucht MK3 aus allen nachfolgenden Gegenständen einen passenden Ersatz zu finden. Dieses Verfahren verringert den Suchaufwand, führt aber auch dazu, dass möglicherweise passende Gegenstände übersprungen werden. Ein einfaches Beispiel verdeutlicht das Problem:

Es seien zwei Rucksäcke mit Kapazität 2 gegeben. Diese sollen mit den Folgenden Gegenständen gefüllt werden.

	Gewicht	Profit
a	0,9	1
b	2	1,9
c	1,1	0,5

MK3 geht nun wie folgt vor: Zuerst wird jeder Rucksack unabhängig per Greedy-Algorithmus gefüllt. Anhand des Verhältnisses Gewicht zu Profit landen in beiden Rucksäcken die Gegenstände $\{a, c\}$. MK3 prüft nun (erneut sortiert nach dem Verhältnis Gewicht zu Profit) ob Gegenstände doppelt gepackt wurden und, falls ja, wie sich das Entfernen dieser Gegenstände auswirken würde:

1. MK3 stellt fest: a wurde mehrfach gepackt, prüfe für Rucksack 1 und 2 welcher Verlust entsteht, wenn a entfernt und die verbleibende Restkapazität mit Gegenständen aus $\{b,c\}$ gefüllt werden würde. Für beide Rucksäcke entstünde ein Profit-Verlust von 1. Der Algorithmus entscheidet oBdA in solch einem Fall den ersten Rucksack zu bevorzugen und entfernt a aus Rucksack 2.

2. b wurde nicht mehrfach gepackt und wird von MK3 übersprungen.

3. c wurde mehrfach gepackt. Da MK3 den Greedy-Algorithmus nur auf die nachfolgenden Gegenstände anwendet, wird Gegenstand b nicht berücksichtigt und weder für Rucksack 1 noch für Rucksack 2 ein Ersatz gefunden. Wie zuvor entsteht bei beiden Rucksäcken ein identischer Verlust und der Algorithmus entfernt c aus Rucksack 2. Obwohl mit b ein passender Gegenstand verbleibt, gibt MK3 einen leeren Rucksack 2 als Lösung zurück.

Um diesem Verhalten entgegenzuwirken, wird MK3 erweitert. Nach Abschluss der Berechnungen wird auf evtl. entstehende leere Rucksäcke erneut ein einfacher Greedy mit den verbleibenden nicht benutzten Gegenständen angewendet, um diese zu füllen.

Um die Laufzeit von MK3 zu verringern, ist ferner festgelegt worden, dass bei einem Greedy-Durchgang ein Gegenstand nicht mehr als 10 mal verpackt werden darf. Dies verschlechtert zwar die erwartete Ergebnisqualität, verhindert aber, dass besonders profitable Gegenstände sich in einem Großteil der Rucksäcke finden und in entsprechend vielen Schleifendurchläufen durch andere ersetzt werden müssen.

Die Algorithmen I1 und I2 dienen der Verbesserung gefundener Lösungen und können in Kombination mit MK1, MK2 und MK3 verwendet werden. Zusätzlich ist es auch möglich I1 und I2 nacheinander zu verwenden und so bessere Lösungen zu erhalten. Neben den einfachen Algorithmen kommen so noch 12 zusätzliche Kombinationen hinzu. Diese sollen nicht alle untersucht werden. Stattdessen wurden alle Kombinationen in kleinen Experimenten mit 100 Benutzern und einem Sendungstyp getestet und lediglich die vielversprechendsten Kombinationen einem kompletten Test unterzogen. Bei den Vorabtests ergaben sich die in den Tabellen 4 und 5 aufgeführten Veränderungen für den durchschnittlichen Fehlbetrag und den durchschnittlichen Score-Wert gegenüber einem Durchlauf der Basisalgorithmen ohne Verbesserung. Bei der Betrachtung des durchschnittlichen Scorewerts liegen die Verbesserungen von I1 und I2 für die Basisalgorithmen MK1 und MK2 sehr dicht beieinander. Aufgrund der geringeren Komplexität von I1 wäre dieser vorzuziehen. Anders sieht das Ergebnis aus, wenn auch der durchschnittliche Fehlbetrag betrachtet wird. Hier gelingt es I2 deutlich besser, Lücken im Programm zu schließen und die Musikblöcke weitgehend zu füllen. I2 erzielt die Verbesserung des Gesamtscores durch bessere Ausnutzung des verfügbaren Platzes, I1 erhöht den Gesamtscore ohne die Ausnutzung deutlich zu verbessern. Durch diese schlechtere Ausnutzung und die ähnliche Verbesserung beim Score ist I1 in allen Fällen die schlechtere Wahl.

	I1	I2	I1+I2	I2+I1
Mk1	+0,026 (+0,21%)	-0,934 (-7,41%)	-1,0297 (-8,17%)	-0,9556 (-7,59%)
Mk2	-0,2246 (-1,79%)	-1,0732 (-8,55%)	-1,2625 (-10,06%)	-1,0975 (-8,74%)
Mk3	-0,936 (-3,72%)	-17,4039 (-69,08%)	-17,3524 (-68,87%)	-17,4039 (-69,08%)

Tabelle 4: Veränderung der Ergebnisse für den durchschnittlichen Fehlbetrag bei Anwendung der Verbesserungsalgorithmen I1 und I2 (Absolute Veränderung und veränderung prozentual zum Ergebnis ohne Verbesserungsalgorithmus)

	I1	I2	I1+I2	I2+I1
Mk1	+0,0425 (+7,92%)	+0,0444 (+8,28%)	+0,0428 (+7,98%)	+0,0443 (+8,26%)
Mk2	+0,0503 (+8,94%)	+0,052 (+9,24%)	+0,0506 (+8,99%)	+0,0519 (+9,22%)
Mk3	+0,0189 (+8,14%)	+0,0391 (+16,83%)	+0,0369 (15,89%)	+0,0391 (+16,83%)

Tabelle 5: Veränderung der Ergebnisse für den durchschnittlichen Fehlbetrag und den durchschnittlichen Scorewert bei Anwendung der Verbesserungsalgorithmen I1 und I2 (Absolute Veränderung und Veränderung prozentual zum Ergebnis ohne Verbesserungsalgorithmus)

Allerdings fällt auf, dass I2 den Variationskoeffizienten des Ergebnisses in allen Fällen erhöht, die Fehlbeträge sind also weniger gleichmäßig verteilt, als ohne Verbesserung. I1 hingegen sorgt für eine leicht gleichmäßigere Verteilung. Aus diesem Grund soll auch in einem Versuch I1 in Kombination mit Mk2 näher betrachtet werden.

Besonders auffällig sind die Veränderungen für den Algorithmus Mk3. Sein Ergebnis wird durch die Anwendung von I2 dramatisch verbessert. In Kombination mit I2 liefert Mk3 durchweg die besten Ergebnisse im Vergleich zu den anderen getesteten Kombinationen. Scheinbar ist I2 in der Lage die Schwierigkeiten, die Mk3 mit den hier gestellten Problem hat (siehe 8), auszugleichen. Anhand dieser Ergebnisse werden die Kombinationen Mk1+I2, Mk2+I1, Mk2+I2 und Mk3+I2 noch einmal näher betrachtet. Die Kombination von I1 und I2 bringt nur geringfügige Verbesserungen. Daher wird nur die Kombination Mk2+I1+I2 ausführlicher analysiert.

7.5.5 Rgli

Zur Implementierung von Rgli wird folgende Vorgehensweise gewählt: Für jeden Durchlauf des Algorithmus wird eine Kopie der Liste mit den verfügbaren Titeln angelegt und deren Einträge randomisiert gemischt. Nun wird, solange die Liste nicht leer ist, das jeweils erste Element entfernt und überprüft, ob es noch in den Musikblock X eingefügt werden kann ohne die Längenbegrenzung zu überschreiten. Falls nein, wird es in eine zweite Liste U eingefügt, die alle nicht verwendeten Titel enthält. Für den zweiten Schritt, das Prüfen und Austauschen von Elementen, sortiert Rgli U absteigend nach den Titellängen sortiert. Eine solche Sortierung ist zwar nicht unbedingt nötig, vereinfacht aber

das Finden eines passenden Tauschkandidatens. Soll für ein Element $j \in X$ mit Länge w_j geprüft werden, ob es einen Titel zum Tauschen gibt, so ist dieser der erste, der beim Durchlaufen von U die Bedingung $w_x \leq w_j + c'$ erfüllt. c' ist die derzeit noch verfügbare Restkapazität im Block X. Sobald ein solcher Titel gefunden ist, kann die Suche abgebrochen und die beiden Elemente können ausgetauscht werden. Hierbei wird j so in U eingefügt, dass die Sortierung erhalten bleibt. Die Suche in U kann ebenfalls abgebrochen werden, sobald ein Element x mit $w_x \leq j$ gefunden wird. In diesem Fall gibt es kein Element in U, das die Qualität der Lösung beim Austauschen verbessern würde.

7.6 Cluster Algorithmen

7.6.1 SLR

Das Bilden von Clustern mit SLR wird realisiert, indem ein User als Transaktionen und die sich aus dem genutzten Clusterkriterium ergebende Menge (z.B. gehörte Titel, gehörte Künstler, verwendete Tags) als Gegenstände angesehen wird. Die Implementierung von SLR erfolgte mit einer Mischung aus den von Wang et al. [56] und den von Yun et al. [58] vorgeschlagenen Methoden. Ein besonderes Augenmerk liegt dabei auf der Berechnung der Kosten für das Verändern einzelner Cluster und der Berechnung der Small/Large-Ratio (S/L-Ratio), da diese beiden Aufgaben sehr oft durchgeführt werden müssen. Ein mehrfaches Scannen kompletter Cluster soll vermieden werden. Ebenso muss ein schneller Zugriff auf die einzelnen Titel in einem Cluster gewährleistet werden. Für die Verwaltung der Cluster-Daten wird auf ein *Berkeley DB* Datenbankmanagementsystem (DBMS)[25] zurückgegriffen. *Berkeley DB* verwaltet Schlüssel-/Daten-Paare und lässt sich über eine Schnittstelle direkt aus Python ansprechen[26]. Die Notwendigkeit für einen eigenen Server und das Verwenden einer entsprechenden Query-Sprache entfällt. Zusätzlich erlaubt *Berkeley DB* einen weitgehenden Einfluss auf die Art der Datenverwaltung. So lassen sich unterschiedliche Datenstrukturen vorgeben (z.B.: B-Bäume, Hashtabellen, Warteschlangen) und die Funktionen zum Vergleich zwischen den Datenbankeinträgen manipulieren. Zum Speichern der einzelnen Cluster werden jeweils zwei *Berkeley DB* Datenbanken verwendet. Für jeden Cluster speichert eine Hashtabelle alle enthaltenen Gegenstände als Schlüssel und die Anzahl der Transaktionen in diesem Cluster, in denen der Gegenstand vorkommt, als Wert. Somit kann auf die Häufigkeit eines Gegenstandes effizient zugegriffen werden. Da es auch notwendig ist, alle Gegenstände mit einer gewissen Häufigkeit zu ermitteln, wird zusätzlich ein B-Baum verwendet, der die Häufigkeit als Schlüssel erhält und als Wert eine Referenz auf den Gegenstand. Damit beide Datenbanken zueinander konsistent bleiben, werden sie mit vom DBMS angebotenen Methoden verknüpft (*assoziiert*).

Zur schnellen Berechnung der Gesamtkosten werden zusätzlich zwei globale

[25] http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html (Abgerufen am 03.04.2011)

[26] http://www.jcea.es/programacion/pybsddb_doc/4.7.3/ (Abgerufen am 03.04.2011)

Hashtabellen - *smallTable* und *largeTable* - angelegt. Diese erhalten die Gegenstände als Schlüssel und speichern als Wert die Anzahl der Cluster, in denen dieser Gegenstand *klein* bzw. *groß* ist. Für die Kostenberechnung wird nun für den jeweiligen Cluster überprüft welche Gegenstände ihren Status verändern, wenn eine weitere Transaktion eingefügt werden würde. Da sich die Grenzwerte für *kleine* bzw. *große* Gegenstände mit dem Hinzufügen einer Transaktion um maximal $\alpha < 1$ bzw. $\beta < 1$ verändern[27] und nur solche Gegenstände in Frage kommen, deren Anzahl zwischen altem und neuem Grenzwert liegt, ist zu erwarten, dass dies nur auf einen geringen Anteil der Gegenstände zutrifft. Diese lassen sich mit Hilfe des Cluster-B-Baums ermitteln. Mit Hilfe der *smallTable* und *largeTable* wird dann überprüft, ob diese veränderten Gegenstände noch in anderen Clustern *klein* bzw. *groß* sind. Nur wenn dies nicht der Fall ist, verändern sich $|\bigcup_{i=1}^{n} K_i|$ oder $|\bigcup_{i=1}^{n} G_i|$. Auch die Veränderung von $\sum_{i=1}^{n} |G_i|$ lässt sich direkt ermitteln, wenn bekannt ist, wieviele Gegenstände aus der Menge der großen Gegenstände im Cluster entfernt werden.

Zur Berechnung der S/L-Ratio muss für eine Reihe von Gegenständen ermittelt werden, ob diese in einem Cluster *klein* oder *groß* sind. Dies lässt sich einfach über eine Abfrage der Hash-Tabelle realisieren. Aus der dort hinterlegten Anzahl von Transaktionen, in denen ein Gegenstand vorkommt, lässt sich direkt feststellen, ob dieser zu einer der beiden Gruppen gehört.

7.6.2 CLARANS

Im Gegensatz zu SLR benötigt CLARANS weniger Daten, die im Speicher vorgehalten werden müssen. Daher lässt bei der Implementierung von CLARANS auf die Verwendung einer externen Datenbank verzichten. Um die zeitintensive Distanz-Berechnung zwischen zwei Cluster-Elementen nicht mehrfach wiederholen zu müssen wird zu jedem Element eine Liste mit k Einträgen angelegt, in der die Distanzen zu den k Medoids gespeichert wird. Beim Ersetzen eines Medoids müssen damit insgesamt n Distanzberechnungen durchgeführt werden. Für $n - k$ Elemente ist eine solche Berechnung in jedem Fall nötig, die Distanzberechnung zwischen zwei Medoids wird, wenn auch nicht zwingend nötig, durchgeführt, um beim „degradieren“ eines Medoids zu einem einfachen Element direkt alle Distanzen zu den verbleibenden Medoids zur Verfügung zu haben. Das Zwischenspeichern aller Distanzen zu den Medoids erspart Berechnungen für alle Elemente, die einem Medoid zugerechnet werden, der ersetzt werden soll. Statt erneut die Distanzen zu allen unveränderten Medoids zu ermitteln, ist es nur nötig, die Distanz zum neuen Medoid zu berechnen. Das Minimum aller Distanzen gibt direkt die neue Clusterzugehörigkeit an.

[27] zur Erinnerung: α, β - Anteil an Transaktionen in denen ein Gegenstand enthalten sein muss, um als nicht mehr *klein* bzw. als *groß* zu gelten

Der Ablauf des Algorithmus erfolgt in 3 Phasen. Zuerst wird eine Liste mit k Usern übergeben und ihre Distanzliste mit den Distanzwerten untereinander gefüllt. Diese sind die initialen Medoids. Im zweiten Schritt werden die restlichen zu clusternden User schrittweise eingefügt und jeweils die Distanzen zu allen Medoids berechnet. Die resultierende Clusterzugehörigkeit wird in einem Python Dictionary vermerkt. Zuletzt wird die Lösung verbessert, indem einzelne Medoids randomisiert nach den in Kapitel 5.1.2 beschriebenen Vorschriften ausgetauscht werden.

7.6.3 Maxdist

Da Maxdist speziell für diese Anwendung entwickelt wurde, gibt es keine Abweichungen von der in 5.1.3 beschriebenen Algorithmus-Idee.

```
1 Die an dieser Stelle in der Arbeit befindlichen Informationen unterliegen den Geschä
      ftsgeheimnissen der Rheinland-Pfälzische Rundfunk GmbH & Co KG und dürfen
      leider nicht in digitaler Form verbreitet werden.
```

Ausschnitt des Sendelogfiles vom 04.01.2011, wie es vom RPR1-Senderechner erstellt wurde. Der Ausschnitt zeigt alle von der Festplatte ausgespielten Sound-Dateien am Beginn der 7-Uhr Stunde mit Nachrichten, Verkehrshinweisen, Show-Opening und Musikstücken. In der zweiten Spalte findet sich der genaue Start-Zeitpunkt eines jeden Elements, in der dritten seine Länge. Die letzte Spalte bezeichnet die Art des Elements, beispielsweise J für Jingle oder M für Musik. Lücken im Zeitcode zeigen Sequenzen während denen keine Datei abgespielt wurde (zum Beispiel während trockener Moderationen, das heißt Moderationen, die nicht von Musiktiteln oder einem Klangbett unterlegt sind).

```
1 Die an dieser Stelle in der Arbeit befindlichen Informationen unterliegen den Geschä
      ftsgeheimnissen der Rheinland-Pfälzische Rundfunk GmbH & Co KG und dürfen
      leider nicht in digitaler Form verbreitet werden.
```

Aus dem Sendeprotokoll erstelltes XML-Sendungsschema für den Beginn der 7-Uhr-Stunde. Aufeinander folgende Musiktitel wurden zu Musikblöcken zusammengefasst, welche für das personalisierte Radio mit individueller Musik gefüllt werden sollen (Das komplette verwendete Sendungsschema für die Morning-Show findet sich in Anhang C).

8 Experimentelle Ergebnisse

8.1 Testaufbau

Die zuvor beschriebenen Methoden und Algorithmen sollen nun auf ihre Tauglichkeit hin untersucht werden, indem aus den Sendungsschemata beispielhafter Sendungen ein kompletter Sendefahrplan für die Nutzer berechnet wird. Die Ergebnisse werden auf Qualität der Musikauswahl sowie Länge und Verteilung der Fehlbeträge hin untersucht. Um ein möglichst realistisches Test-Szenario zu erreichen, werden die betrachteten Sendungsschemata aus echten Sendeprotokollen des Senders RPR1. erstellt. RPR1. (betrieben von der *Rheinland-Pfälzische Rundfunk GmbH & Co KG*) ist ein kommerzielles Privatradio in Rheinland-Pfalz. Der Sender wurde im April 1986 gegründet und war nach einigen Pilotprojekten in München und Ludwigshafen das erste private Radioprogramm in Deutschland, welches flächendeckend in einem Bundesland sendete[32]. Heute ist RPR1. der erfolgreichste private Radiosender in Rheinland-Pfalz und erreicht im Schnitt etwa 286.000 Hörer pro Stunde beziehungsweise 1,6 Millionen Hörer pro Tag [13]. Für diese Arbeit stellte RPR1. zwei Tagesprotokolle (vom 04.01. und 05.01.2011) zur Verfügung. Diese Protokolle enthalten eine sekundengenaue Mitschrift aller Musikstücke, Beiträge, Verpackungselemente und sonstiger Audiodateien, die über den Senderechner ausgestrahlt wurden. Zur Durchführung der Experimente werden die Logfiles analysiert und in das

in Kapitel 7.4.1 beschriebene XML-Format umgewandelt. Dabei werden alle Musiktitel durch Platzhalter ersetzt und aufeinander folgende Titel zu einem Musikblock zusammengefasst. Sofern ihre Länge nicht 30 Sekunden überstieg werden Verpackungselemente wie Trailer, Jingles oder Dropins, die sich zwischen den Titeln befinden, ebenfalls zu den Musikblöcken hinzugezählt. Die so entstehenden, von spezifischen Musiktiteln bereinigten, Sendungsschemata können nun mit personalisierter Musik befüllt werden und bieten eine realistische Verteilung von Musikblöcken und Inhalten über den Verlauf eines Tages. Hierbei gilt es zu beachten, das RPR1. ein Sender des AC Formats ist und sich die Musikverteilung bei anderen Sendern mit anderen Formaten deutlich unterscheiden kann. Da AC aber das in Deutschland am weitesten verbreitete Format ist [13], erscheint es zulässig, für die entwickelte Radioanwendung dieses Format zu Grunde zu legen.

Um die Algorithmentauglichkeit in verschiedenen Situationen noch genauer bewerten zu können wird jedes Experiment mit vier verschiedenen Sendungsschemata durchgeführt. Jeder Algorithmus musse einen Sendungsplan aus dem Schema für die kompletten 24 Stunden des 04.01. berechnen. Zusätzlich soll dann für den 05.01. jeweils 2 Stunden lange Ausschnitte aus der Mitte einzelner Sendungen getrennt berechnet werden. Diese Sendungen unterscheiden sich in Gestaltung und Musikverteilung deutlich:

- Das *Nachtprogramm* (0 Uhr bis 5 Uhr - gewählter Ausschnitt von 2 Uhr bis 4 Uhr) wird von RPR1. automatisiert gefahren. Abgesehen von Unterbrechungen durch Nachrichten jeweils zur vollen Stunde und Verkehrsinformationen zur halben Stunde besteht es weitgehend aus Musik. Im gewählten Ausschnitt des Nachtprogramms finden sich 4 Musikblöcke mit einer durchschnittlichen Länge von 1714 Sekunden.

- Die *Morgen-Show* (5 Uhr bis 10 Uhr - gewählter Ausschnitt von 6 Uhr bis 8 Uhr) ist ein deutlicher Kontrast zum Nachtprogramm. Sie besteht aus einer Vielzahl von Inhalten, Berichten und Moderationen und nur wenigen, kurzen Musikblöcken. Im Laufe des gewählten Ausschnittes der Morgen-Show gibt es 15 Musikblöcke mit einer durchschnittliche Länge von 264 Sekunden.

- Das *Nachmittagsprogramm* (14 Uhr bis 18 Uhr - gewählter Ausschnitt von 15 Uhr bis 17 Uhr) ist ein Mittelweg zwischen Nachtprogramm und Morgen-Show. Eine Stunde enthält 4 bis 5 Moderationen, dazu einmal Nachrichten zur vollen Stunde und Verkehr zur halben. Im gewählten Ausschnitt finden sich 14 Musikblöcke mit einer durchschnittlichen Länge von 381 Sekunden

Am komplett betrachteten 04.01. sind insgesamt 154 Musikblöcke mit einer durchschnittlichen Länge von 434 Sekunden gesendet worden. Der Test aller Algorithmen findet auf einem 64bit System mit 2048 MB Arbeitsspeicher und einem AMD 64 3000+ Prozessor statt.

```
1 Die an dieser Stelle in der Arbeit befindlichen Informationen unterliegen den Geschä
      ftsgeheimnissen der Rheinland-Pfälzische Rundfunk GmbH & Co KG und dürfen
      leider nicht in digitaler Form verbreitet werden.
```

Ausschnitt aus einem per MK2 erstellten Sendungsfahrplan für einen Beispielnutzer für den Beginn der 7-Uhr Stunde. Alle Musikblöcke wurden mit personalisierter Musik dieses Nutzers gefüllt und auftretende Lücken durch einen Jingle geschlossen. Ein finales System könnte an dieser Stelle noch zufällig aus mehreren möglichen Verpackungselementen, Trailern oder ähnlichem wählen.

8.2 Ergebnisse der Heuristiken

Jeder der in Kapitel 7.5 aufgeführte Algorithmen hat Sendungspläne für dieselben 1000 Benutzer und für alle vier in Kapitel 8.1 vorgestellten Testfälle berechnet. Die von den Algorithmen erzeugten Sendungsfahrpläne wurden daraufhin, mit Blick auf die erzeugten Fehlbeträge in den Musikblöcken und den erreichten Scorewert der gewählten Titel, ausgewertet. Die gefundenen Ergebnisse werden im Folgenden dargestellt und bewertet. In Tabelle 6 finden sich die durchschnittlichen Fehlbeträge pro Nutzer. Hierzu wird die Differenz zwischen geplanter Länge eines Musikblockes und der Gesamtlänge der vom Algorithmus für diesen Block gewählten Titel für alle Musikblöcke und Nutzer aufsummiert und durch die Anzahl der Nutzer geteilt. Eine Aufstellung desselben Wertes geteilt durch die Anzahl der insgesamt bearbeiteten Musikblöcke findet sich in Tabelle 7. Je geringer der erzeugte Fehlbetrag ausfällt, um so weniger Zeit muss im Programm mit Trailern und sonstigen Füllelementen überbrückt werden. Neben der durchschnittlichen Fülllänge ist auch der maximale Fehlbetrag von Interesse. Die höchsten von einem Algorithmus erzeugten Werte sind in Tabelle 8 aufgeführt. Ein zu hoher maximaler Fehlbetrag ist nicht wünschenswert, da ein zu hoher Anteil an Füllmaterial in einem einzelnen Musikblock die Erwartungen der betroffenen Nutzer enttäuschen könnte. Tabelle 9 zeigt den durchschnittlichen Variationskoeffizient[28] der Fehlbeträge in den erzeugten Sendungsfahrplänen. Dieser gibt ein Maß für die Gleichmäßigkeit der Verteilung der Fehlbeträge an. Eine gleichmäßige Verteilung der nötigen Füllelemente ist wünschenswert, da Häufungen vom Hörer als störend empfunden werden könnten. Die durchschnittliche Summe der Scorewerte der ausgewählten Titel für einen Musikblock ist in Tabelle 11 aufgeführt. Die Maximierung dieser Werte ist das Ziel der einzelnen Rucksack-Algorithmen.

[28] definiert als die empirische Standardabweichung geteilt durch Mittelwert der Reihe: $VarK(x_1, x_2, \ldots, x_n) = \frac{\sqrt{Var(x_1, x_2, \ldots, x_n)}}{\frac{\sum_{i=1}^{n} x_i}{n}}$

	Nacht	Morning-Show	Nachmittag	kompletter Tag
Ext-Greedy	50,45	168,02	173,41	2018,28
Kptree	162,61	166,71	199,84	1446,56
KptreeExt	121,54	143,77	169,07	1273,73
Mk1	50,04	187,9	176,33	2506,82
Mk2	49,64	188,35	175,78	3282,71
Mk3	59,43	271,22	352,72	4697,32
Mk1+I2	47,00	164,91	156,52	2323,33
Mk2+I1	49,65	183,6	172,99	3279,51
Mk2+I2	46,96	164,66	158,14	2951,57
Mk2+I1+I2	46,96	160,66	154,81	2993,9
Mk3+I2	20,74	111,28	105,56	2262,86
Rgli	0	0,34	0,31	0,12
Combo	*23,51*	*116,38*	*115,12*	*752,43*
Mulknap	*24,61*	*113,24*	*110,2*	*768,33*
Decomp	*0*	*0,32*	*0,31*	*0,11*

Die Algorithmen Combo, Mulknap und Decomp sind keine Heuristiken, sondern exakte Algorithmen für KP, MPK bzw. SSP

Tabelle 6: **Summe Fehlbetrag pro Nutzer:** Durchschnittliche Fehlzeiten pro Nutzer, die der Algorithmus in den Musikblöcken erzeugt hat (in Sekunden)

	Nacht	Morning-Show	Nachmittag	kompletter Tag
Ext-Greedy	12,61	11,20	12,38	13,10
Kptree	40,65	11,11	14,27	9,39
KptreeExt	30,38	9,58	12,07	8,27
Mk1	12,51	12,52	12,59	16,27
Mk2	12,41	12,55	12,55	21,31
Mk3	14,85	18,08	25,19	30,50
Mk1+I2	11,75	10,99	11,17	15,08
Mk2+I1	12,41	12,23	12,35	21,29
Mk2+I2	11,74	10,97	11,29	19,16
Mk2+I1+I2	11,74	10,71	11,05	19,44
Mk3+I2	5,18	7,41	7,53	14,69
Rgli	0	0,03	0,03	0,01
Combo	*7,06*	*7,75*	*8,22*	*4,88*
Mulknap	*7,39*	7,54	7,87	4,98
Decomp	*0*	*0,03*	*0,03*	*0,01*

Die Algorithmen Combo, Mulknap und Decomp sind keine Heuristiken, sondern exakte Algorithmen für KP, MPK bzw. SSP

Tabelle 7: **Durchschnittlicher Fehler:** Durchschnittliche Fehlzeiten pro Musikblock, die der Algorithmus erzeugt hat (in Sekunden)

	Nacht	Morning-Show	Nachmittag	kompletter Tag
Ext-Greedy	107	90	118	183
Kptree	648	206	185	1030
KptreeExt	474	81	94	337
Mk1	61	63	85	205
Mk2	102	62	102	197
Mk3	157	165	190	362
Mk1+I2	46	57	85	205
Mk2+I1	102	62	102	197
Mk2+I2	61	60	90	197
Mk2+I1+I2	61	60	90	197
Mk3+I2	119	131	189	223
Rgli	0	11	12	2
Combo	*39*	*49*	*48*	*127*
Mulknap	*37*	*48*	51	*123*
Decomp	*0*	*8*	*10*	*2*

Die Algorithmen Combo, Mulknap und Decomp sind keine Heuristiken, sondern exakte Algorithmen für KP, MPK bzw. SSP

Tabelle 8: **Maximale Fehlzeit:** Die längste Fehlzeit, die in einem Musikblock erzeugt wurde (in Sekunden)

	Nacht	Morning-Show	Nachmittag	kompletter Tag
Ext-Greedy	0,7007	0,7459	0,7272	0,8538
Kptree	0,9935	1,0509	0,9869	1,4318
KptreeExt	0,9807	0,9616	0,9383	1,331
Mk1	0,6789	0,7089	0,7198	0,8362
Mk2	0,7067	0,7178	0,7276	0,8225
Mk3	0,7824	0,9356	0,8905	1,0529
Mk1+I2	0,6931	0,7533	0,7594	0,8529
Mk2+I1	0,7067	0,7318	0,734	0,8226
Mk2+I2	0,7253	0,7662	0,7678	0,8625
Mk2+I1+I2	0,7253	0,778	0,7772	0,8618
Mk3+I2	0,7083	0,9875	0,9948	1,2955
Rgli	0	0,0004	0,0004	0,0003
Combo	*0,7144*	*0,9083*	*0,9167*	*1,1942*
Mulknap	0,7258	*0,9102*	*0,9277*	*1,1948*
Decomp	*0*	*0,0003*	*0,0004*	*0,0003*

Die Algorithmen Combo, Mulknap und Decomp sind keine Heuristiken, sondern exakte Algorithmen für KP, MPK bzw. SSP

Tabelle 9: **Verteilung des Fehlbetrags:** Variationskoeffizient aus den Fehlzeiten der Musikblöcke zur Analyse der Gleichmäßigkeit der Verteilung

8.2.1 Fehlzeit

Ext-Greedy Unter allen implementierten Heuristiken ist Ext-Greedy die einfachste. Trotzdem sind die in den Ergebnissen des Algorithmus entstandenen Fehlzeiten in einzelnen Fällen nicht deutlich schlechter als die Ergebnisse der komplizierteren Verfahren, in Einzelfällen sogar besser. Besonders bei der Erzeugung des Sendeplans für den ganzen Tag schneidet Ext-Greedy gut ab und erzeugt weniger Fehlzeiten als die meisten anderen Algorithmen. Die Ergebnisse für die kürzeren Sendestrecken liegen im Mittelfeld. Auch bei der Verteilung der Fehlzeiten sind die gefundenen Ergebnisse mittelmäßig.

Kptree Die große Stärke von Kptree liegt in der Bearbeitung von langen Sendestrecken. Die verursachten Fehlzeiten für den kompletten Tag liegen deutlich unter denen der anderen Heuristiken, die versuchen, den Score-Wert zu optimieren. Lediglich Rgli, der die Scorewerte der Titel gar nicht betrachtet, die verbesserte Version KptreeExt und die zum Vergleich betrachteten optimalen Algorithmen erreichen noch geringere Fehlzeiten. Der Abstand zwischen Kptree und der nächst schlechteren Heuristik Ext-Greedy beträgt 571,72 Sekunden (etwa 28%). Im Kontrast dazu sind die durchschnittlichen Fehlbeträge bei Berechnungen der kürzeren Sendestrecken deutlich schlechter als die von allen anderen Algorithmen erreichten. Am stärksten ausgeprägt ist der Unter-

schied bei der Berechnung des Nachtprogramms: hier erzeugt KPTREE 2,7 mal größere Fehlzeiten als der zweitschlechteste Algorithmus. Die zweite Schwäche von KPTREE liegt in der Verteilung der Fehlbeträge. Diese ist weniger ausgeglichen als bei den anderen untersuchten Heuristiken. Besonders die längsten Fehlzeiten, die der Algorithmus erzeugt, sind in einzelnen Fällen inakzeptabel groß. Der Grund hierfür liegt allerdings in seiner Vorgehensweise und der Beschränkung der Suchdurchläufe, die für diese Arbeit vorgenommen wurde (siehe Kapitel 7.5.2). Das Problem tritt dann auf, wenn ein Nutzer einen besonders langen Titel t_l mit einer hohen Wertung in seiner Titelbibliothek hat. Da KPTREE in seiner Suche einen Teilbaum auswählt, werden nur zusammenhängende Blätter betrachtet. Symbolisiert nun eines dieser Blätter den besonders langen Titel t_l wird der Teilbaum solange verkleinert, bis die Titelauswahl die Längenbegrenzung erfüllt. Je nach Position von t_l steigt die Suche sehr tief in den Baum hinab, bis ein Teilbaum ohne t_l erreicht wird. Ist dieser gefunden, so schrumpft die Gesamtlänge der ausgewählten Titel in einem Schritt um einen großen Betrag und kann dabei deutlich unter die eigentlich gewünschte Gesamtlänge fallen. Obwohl noch eine große Kapazität übrig ist, übernimmt Kptree zunächst nur alle Blätter des linken Teilbaumes in die Lösung und setzt die Suche im rechten Teilbaum fort. Da t_l spät entfernt wurde, ist auch dieser nur noch recht klein und enthält nicht genug Blätter, um die übrige Kapazität auszufüllen. Eine Verbesserung wurde mit KPTREEEXT (Kapitel 7.5) umgesetzt und getestet. Mögliche weitere Lösungsansätze für dieses Problem werden in Kapitel 9.2 vorgeschlagen.

KPTREEEXT Die Schwächen von KPTREE können mit KPTREEEXT gemindert werden. Die maximal erzeugten Fehlbeträge sinken bei der Berechnung der Morning-Show und des Nachmittagsprogramms deutlich und liegen nur noch wenig über den Werten der anderen Heuristiken. Auch die durchschnittlichen Fehlbeträge liegen für diese beiden Sendungstypen nun im Mittelfeld. Noch besser schneidet der Algorithmus bei der Berechnung eines kompletten Tages ab. Dort erzeugt er den geringsten durchschnittlichen Fehlbetrag unter allen Heuristiken. Das Nachtprogramm stellt, trotz leicht verbesserter Ergebnisse, aber weiterhin ein Problem dar. Hier liegen die durchschnittlichen und maximalen Fehlbeträge deutlich über denen, die durch andere Heuristiken erzeugt werden. Dieses schlechte Abschneiden lässt sich teilweise durch die langen Musikblöcke im Nachtprogramm erklären. Da der Nachbesserungs-Schritt von KPTREEEXT nur angewandt wird, wenn der Fehlbetrag mehr als 15% der Blocklänge ausmacht, werden im Nachtprogramm sehr große Fehlzeiten toleriert. Soll KPTREEEXT zur Berechnung langer Musikblöcke verwendet werden, muss daher der Schwellwert entsprechend gesenkt werden. Auch die Verteilung der einzelnen Fehlzeiten hat sich gegenüber dem nicht erweiterten KPTREE verbessert. Trotzdem erzeugt KPTREEEXT weiterhin sehr ungleich verteilte Fehlzeiten im Vergleich zu anderen Heuristiken.

MK1, MK2 und MK3 Im Vergleich zwischen MK1, MK2 und MK3 ist besonders das schlechte Abschneiden des Algorithmus MK3 auffällig. In den

	Durchschnittlicher Fehlbetrag	Durchschnittlicher Scorerwert
MK1	14,16 / 14,09	49,99 / 50,13
MK2	13,94 / 13,84	50,05 / 49,93
MK3	6,03 / 6,23	50,46 / 50,66

Tabelle 10: **Vergleichsexperiment MK1, MK2 und MK3 mit randomisierten Daten (Ergebnisse zweier unabhängiger Durchläufe):** Durchgeführt für jeweils 500 Nutzer mit dem Sendungsschema *nachmittag* und mit randomisierten Werten zwischen 0 und 300 für die Titellänge und 0 und 100 für den Titelscore

Experimenten von Martello und Toth [33] war dieser MK2 und MK1 deutlich überlegen und auch eigene Experimente mit den hier verwendeten Implementierungen bestätigten die von Martello und Toth festgestellten Ergebnisse, sobald anstatt der Originaldaten randomisierte Werte für Titelscore und Titellänge verwendet werden (siehe Tabelle 10). Mit diesen Daten liefern MK1 und MK2 deutlich höhere Fehlbeträge und erreichen geringere Scorewerte als der komplexere MK3 Algorithmus. Scheinbar unterscheidet sich das Problem der Titelzuordnung jedoch von den herkömmlichen Rucksackproblemen und fordert andere Qualitäten von den Algorithmen. Möglicherweise liegt dies in der Korrelation zwischen Titellänge und dem für die Berechnung verwendeten *relativen Score* begründet, die in der Musikdatenbank dadurch entsteht, dass der Gesamtscore auf die Titellänge bezogen wird (siehe auch 4.2).

Insgesamt ähneln sich die Ergebnisqualitäten von MK1 und MK2 in Hinsicht auf die erzeugten Fehlbeträge. Lediglich bei der Berechnung des kompletten Tages liefert MK1 die besseren Ergebnisse. Die Verwendung des Verbesserungsalgorithmus I1 in Kombination mit MK2 bringt kaum Verbesserung der erzeugten Fehlzeiten. Anders dagegen die Verwendung von I2: hier verbessern sich die Ergebnisse durchgehend. Besonders MK3 kann von der zusätzlichen Anwendung von I2 profitieren und liefert dann die geringsten Fehlbeträge aller Scorebeachtenden Heuristiken. Einzige Ausnahme sind KPTREE und KPTREEEXT, welche im Falle des kompletten Tages besser abschneiden. Allerdings erzeugt MK3 auch die schlechtesten maximalen Fehlbeträge und die ungleichmäßigste Verteilung unter den MK-Algorithmen. I2 kann den ersten Wert verbessern, trotzdem bleiben die maximalen Fehlzeiten schlechter als die von MK1 und MK2. Die Gleichmäßigkeit der Verteilung der Fehlzeiten hingegen verschlechtert sich unter der Verwendung von I2 in den meisten Fällen. Einzige Ausnahme ist die Kombination MK3+I2 bei der Berechnung des Nacht-Schemas. Hier verbessert sich die Verteilung leicht. Die Anwendung von I1 hat keinerlei Auswirkung auf die maximalen Fehlbeträge. Insgesamt hat die Anwendung von I2 die deutlicheren Auswirkungen auf die Fehlzeiten, führt aber auch zu einer ungleichmäßigeren Verteilung.

RGLI Wie erwartet, liefert RGLI die geringsten Fehlzeiten. Trotz der unterschiedlichen Titelbibliotheken der einzelnen Nutzer entstehen in keinem Fall

	Nacht	Morning-Show	Nachmittag	kompletter Tag
Ext-Greedy	0,6892	0,52	0,5627	0,2782
Kptree	0,7766	0,6136	0,6674	0,3043
KptreeExt	0,7517	0,5886	0,6523	0,2992
Mk1	0,6845	0,4578	0,5361	0,2519
Mk2	0,6806	0,4291	0,5109	0,2429
Mk3	0,4038	0,2297	0,2323	0,1965
Mk1+I2	0,6846	0,4589	0,5367	0,2521
Mk2+I1	0,6806	0,427	0,5097	0,243
Mk2+I2	0,6903	0,4339	0,5178	0,2433
Mk2+I1+I2	0,6903	0,4319	0,5165	0,2466
Mk3+I2	0,4095	0,2481	0,254	0,2063
Rgli	0,1023	0,0630	0,0721	0,0761
Combo	*0,7864*	*0,6403*	*0,6703*	*0,3087*
Mulknap	*0,7901*	*0,6428*	*0,6767*	*0,3124*
Decomp	*0,1137*	*0,0731*	*0,0885*	*0,1076*

Die Algorithmen Combo, Mulknap und Decomp sind keine Heuristiken, sondern exakte Algorithmen für KP, MPK bzw. SSP

Tabelle 11: **Scorewert:** Durchschnittlicher Scorewert pro Nutzer und Musikblock

eine Lücke von mehr als 12 Sekunden, die mit Trailern gefüllt werden muss. Im Durchschnitt entstehen sogar Fehlzeiten von unter einer Zehntel Sekunde pro bearbeitetem Musikblock.

8.2.2 Scorewert

Neben den erzeugten Fehlzeiten ist die Qualität der ausgewählten Musik das zweite wichtige Kriterium für die Eignung eines Algorithmus. Wie gut die Titelauswahl eines Algorithmus den Nutzern gefällt, geben die Score-Wert der Titels an. Ein hoher durchschnittlicher Scorewert bedeutet also, dass die gewählten Titel gut zum Geschmack des Hörers passen. Die von den Algorithmen erzielten durchschnittlichen Scorewerte finden sich in Tabelle 11. Unter den getesteten Heuristiken zeichnet sich Kptree durch besonders hohe erreichte Werte aus. In allen getesteten Anwendungsszenarien liegen die Ergebnisse deutlich über denen anderer Heuristiken und sehr nahe an den Ergebnissen der exakten Algorithmen. Die Erweiterung KptreeExt liefert etwas schlechtere Scorewerte als der Ursprungsalgorithmus. Dies lässt sich allerdings dadurch erklären, dass Kptree-Ext die zu hohen Fehlzeiten durch wenig beliebte Titel zu füllen versucht und damit den Durchschnitt eines Musikblocks verschlechtert. Etwas überraschend liefere auch der sehr einfache Ext-Greedy gute Ergebnisse und übertrifft in vielen Fällen die erreichten Scorewerte der Algorithmen der MK-Familie. Unter

	Cluster-Anzahl	Größter Cluster	Übereinstimmung (Ø)	Übereinstimmung (Max)
CLARANS (k=300)	300	335 User	0.0137	0.3786
CLARANS (k=400)	400	300 User	0.0174	0.3786
CLARANS (k=500)	500	245 User	0.0211	0.3673
MAXDIST (maxDist=0,5)	Keine Cluster gefunden			
MAXDIST (maxDist=0,7)	3652	7 User	0.2163	0.3457
MAXDIST (maxDist=0,9)	2810	71 User	0.1375	0.3457

Durchschnittliche und maximale Übereinstimmung, jeweils paarweise berechnet (bezogen auf die Werte der betrachteten Distanzfunktion) zwischen allen Mitgliedern eines Clusters
Der Algorithmus SLR unterstützt keine Gegenstandsgewichtung und ist daher hier nicht aufgeführt

Tabelle 12: Clustering unter Berücksichtigung des Titel-Scores

Letztgenannten überrascht erneut das schlechte Abschneiden von MK3, dessen Ergebnisqualität deutlich unter denen von MK1 und MK2 liegt. Auch die Kombination von MK3 und I2 bringt, im Gegensatz zu den zuvor betrachteten Fehlbeträgen, keine deutliche Verbesserung. Generell liefern die Verbesserungsalgorithmen I1 und I2 nur sehr leichte Erhöhungen des durchschnittlichen Scorewertes. Ohne Berücksichtigung der Titelqualitäten erreicht RGLI Ergebnisse mit Scorewerten, die zwischen einem Viertel (bei der Berechnung des kompletten Tages) und einem Zehntel (bei Berechnung der Morning-Show) der jeweiligen, durch die exakten Algorithmen bestimmten, Werte liegen.

8.3 Clustering Ergebnisse

Um zu ermitteln, ob sich Nutzer zu Gruppen zusammenfassen lassen, deren Musikgeschmack derart ähnlich ist, dass Sendungsfahrpläne für sie gemeinsam berechnet werden können, wird versucht, in den *last.fm* Daten mit verschiedenen Methoden nach Clustern zu suchen (siehe Kapitel 5). Erste Experimente werteten sowohl die Anzahl der übereinstimmenden Titel, als auch die Distanz zwischen den Score-Werten dieser Titel (Tabelle 12). Da dieser Ansatz nur sehr kleine Cluster bzw. Cluster mit sehr geringen Übereinstimmungen zwischen den Nutzern erzeugt wurde die Distanzfunktion schrittweise allgemeiner gestaltet. Zunächst wurden anstatt den Score-Wert-Distanzen zwischen Titeln nur noch die Distanzen zwischen Interpreten betrachtet (Tabelle 13) und zuletzt nur noch die reine Anzahl gemeinsamer Titel ohne Berücksichtigung von Scorewerten (Tabelle 14). Aber auch diese allgemeineren Distanzfunktionen bringen nur geringfügige Verbesserungen in der Anzahl der Cluster und der Verteilung der Nutzer.

Der Versuch, Nutzer anhand der Tags der von ihnen gehörten Titel in Cluster aufzuteilen, führte zu keinen Ergebnissen, da die benötigte Rechenzeit unter den gegebenen Bedingungen zu hoch war.

	Cluster-Anzahl	Größter Cluster	Übereinstimmung (Ø)	Übereinstimmung (Max)
CLARANS (k=300)	300	411 User	0.0398	0.4151
CLARANS (k=400)	400	315 User	0.0424	0.425
CLARANS (k=500)	500	287 User	0.048	0.4143
MAXDIST (maxDist=0,5)	Keine Cluster gefunden			
MAXDIST (maxDist=0,7)	3147	21 User	0.2627	0.4089
MAXDIST (maxDist=0,9)	2614	113 User	0.2601	0.3958

Durchschnittliche und maximale Übereinstimmung, jeweils paarweise berechnet (bezogen auf die Werte der betrachteten Distanzfunktion) zwischen allen Mitgliedern eines Clusters
Der Algorithmus SLR unterstützt keine Gegenstandsgewichtung und ist daher hier nicht aufgeführt

Tabelle 13: Clustering unter Berücksichtigung des Scoreunterschieds gemeinsamer Interpreten

	Cluster-Anzahl	Größter Cluster	Übereinstimmung (Ø)	Übereinstimmung (Max)
CLARANS (k=300)	300	374 User	0.0422	0.4341
CLARANS (k=400)	400	286 User	0.0468	0.432
CLARANS (k=500)	500	217 User	0.0531	0.4289
MAXDIST (maxDist=0,5)	Keine Cluster gefunden			
MAXDIST (maxDist=0,7)	3147	18 User	0.2863	0.4135
MAXDIST (maxDist=0,9)	2614	89 User	0.231	0.4277
SLR (small=0,4, large=0,8)	Keine Cluster gefunden			
SLR (small=0,1, large=0,6)	Keine Cluster gefunden			

Durchschnittliche und maximale Übereinstimmung, jeweils paarweise berechnet (bezogen auf die Werte der betrachteten Distanzfunktion) zwischen allen Mitgliedern eines Clusters

Tabelle 14: Clustering unter Berücksichtigung der gemeinsamen Titel

9 Fazit und Ausblick

9.1 Bewertung der experimentellen Ergebnisse

Unter den betrachteten Heuristiken erscheint, unter Beachtung aller Kriterien, der für diese Arbeit speziell modifizierte Algorithmus KptreeExt am besten geeignet. Seine Score-Ergebnisse gehören, im Vergleich zu den anderen getesteten Heuristiken, in allen Anwendungsszenarien zu den Besten. Lediglich Kptree liefert einen noch höheren durchschnittlichen Score-Wert. Da KptreeExt die Ergebnisse von Kptree aber nur um weitere Titel erweitert, kann davon ausgegangen werden, dass KptreeExt lediglich mehr Musik einplant als Kptree und dabei auch auf weniger beliebte Titel zurückgreift. Die Anzahl der gespielten sehr beliebten Titel bleibt zwischen Kptree und KptreeExt aber unverändert. Im Vergleich zu anderen Heuristiken liefert KptreeExt einem Hörer den höchsten Anteil an seinem Geschmack entsprechender Musik. Auch die durchschnittliche Fehlzeit, die KptreeExt erzeugt, liegt in der Spitzengruppe der getesteten Algorithmen. Lediglich die Kombination Mk3+I2 erzeugt noch geringere Fehlzeiten, allerdings auch bei einem deutlich geringeren Score-Wert. Die einzige deutliche Schwäche in den Ergebnissen von KptreeExt liegt in der Verteilung der Fehlzeiten. Diese werden von den meisten Heuristiken gleichmäßiger verteilt. Kptree und KptreeExt sind allerdings trotz der guten theoretischen Laufzeit von $O(n \log n)$ auch die aufwändigsten der getesteten Heuristiken. Ihr Overhead erhöht, durch die doppelte Vorhaltung der Daten und den benötigten binären Baum, den Ressourcenverbrauch.

Ein allgemeines Problem aller betrachteten Algorithmen sind die produzierten maximalen Fehlbeträge. Selbst die mit Combo oder Mulknap gefundenen optimalen Lösungen weisen in Einzelfällen Lücken von über 2 Minuten in den erzeugten Musikblöcken auf. Selbst wenn diese Zeit mit gleichmäßig über einen Musikblock verteilten Trailern gefüllt werden kann, stört dies den Hörgenuss. Heutige klassische Radioprogramme enthalten durchschnittlich etwa 4 Sekunden Eigenwerbung zwischen zwei Titeln [40]. Es ist unklar wie Hörer auf eine deutliche Erhöhung dieser Zeit reagieren.

Der Ansatz des Clusterings lässt sich mit den hier betrachteten Methoden nicht realisieren, da sich die Titelbibliotheken zweier Nutzer nur in sehr wenigen Fällen überschneiden. Mit keiner Methode war es möglich, zwei Nutzer zu finden, deren Lieblingstitel sich zu mehr als 43,2% decken. Selbst wenn nur eine 10-prozentige Deckung zwischen Nutzern erwartet wird, bildeten sich nur kleine Cluster. Es lässt sich also schlussfolgern, dass der Musikgeschmack einer großen Menge Nutzer zu stark divergiert, um größeren Gruppen mit einem gemeinsamen Programm das Gefühl zu geben, es bestünde nur aus ihren Lieblingstiteln. Dies ist insofern überraschend, da derzeitige Radioangebote mit ihrer Musikauswahl eine breite Masse ansprechen. Offensichtlich sind die Hörer dieser Programme bereit, Kompromisse einzugehen, hören aber, wenn sie ihre eigene Auswahl treffen, eine deutlich unterschiedliche, individuellere Musikzusammenstellung.

9.2 Weiterführende Aufgaben

Die Ergebnisse dieser Arbeit zeigen, mit welchen Methoden die Sendungsfahrpläne eines personalisierten Radios erzeugt werden können. Es bleiben aber noch eine Reihe weiterführender Aufgaben, die im Rahmen dieser Arbeit nicht betrachtet wurden.

- Die *Durchformatierung* heutiger Radioprogramme bezieht sich nicht ausschließlich auf die Auswahl der gespielten Lieder, sondern enthält auch Vorgaben, wie diese Lieder kombiniert werden. So können beispielsweise Formate Vorschriften enthalten, dass pro Stunde nur ein rockiger Titel aus den 80ern gespielt wird, oder dass zu Beginn einer Sendestunde, nach den Nachrichten, ein besonders beliebtes, schnelles Musikstück gespielt wird [40]. Da sich solche formatierten Radiostationen am Markt als besonders erfolgreich erweisen [13], sollte geprüft werden, wie sich diese Formatierungsregeln auch in die Erzeugung eines personalisierten Programmes übernehmen lassen.

- Bei der Erzeugung der Sendungsfahrpläne wurde ein vereinfachtes Modell zu Grunde gelegt, in dem jeder Titel nur einmal verwendet werden durfte, selbst wenn der Sendeplan für einen kompletten Tag generiert wird. In üblichen Radioprogrammen können Titel mehrfach pro Tag gespielt werden. Dabei hängt die genaue Häufigkeit von ihrer Beliebtheit ab [21]. Ein Algorithmus, der dieses berücksichtigt, kann entwickelt werden.

- Die hier betrachtete Methode geht von der idealisierten Voraussetzung aus, dass keine unvorhergesehenen Ereignisse den vorher festgelegten Ablaufplan stören. Während einer Livesendung kann es aber nötig sein, auf aktuelle Ereignisse zu reagieren, in das laufende Programm einzugreifen und zusätzliche Moderationen einzufügen oder bestehende zu verlängern oder zu kürzen. Mit den hier betrachteten Mitteln lässt sich dies zwar realisieren, daraus resultierende Änderungen im Ablauf müssen vom Moderator aber manuell korrigiert werden. Beispielsweise, indem er nachfolgende Moderationen kürzt, um verlorene Zeit wieder auszugleichen oder indem er laufende Musikblöcke für alle Nutzer früher unterbricht. Es ist aber auch denkbar, dass das System selbst während einer laufenden Sendung erkennt, dass eine Abweichung vom Sendeplan erfolgt ist und dann versucht, diese zu korrigieren, indem einzelne Musikblöcke mit den neuen Längen erneut berechnet werden. Wie gut sich solche Verfahren in das bestehende System integrieren lassen, muss getestet werden.

- Für den KPTREE-Algorithmus kann untersucht werden, inwieweit mögliche andere Verbesserungen die Ergebnisqualität beeinflussen und das Problem des sehr hohen maximalen Fehlbetrags mindern. Neben der in Kapitel 7.5 vorgestellten und in KPTREEEXT umgesetzten Verbesserung sind noch andere Ansätze denkbar. Mögliche Verbesserungen sind: besonders lange Titel bereits vor Erzeugung des Baumes aus der Liste der möglichen Titel zu streichen, die automatische Erweiterung der Anzahl

der maximal getesteten „besten Gegenstände“, falls der resultierende Fehlbetrag einen bestimmten Schwellwert überschreitet oder die Kombination mit einer anderen Heuristik zur Ausfüllung großer Fehlbeträge.

- Wie bereits in Kapitel 6 erläutert, erfordert der Betrieb eines personalisierten Radios eine digitale Redaktions- und Produktionsumgebung im Funkhaus des Senders. Eine solche Umgebung mit der Möglichkeit des personalisierten Radios muss neu entwickelt oder in bestehende Systeme integriert werden.
- Das in Kapitel 6 entworfene Ausstrahlungssystem muss erprobt und entwickelt werden. Hierbei sollte auch genauer untersucht werden, ob eine client- oder eine serverseitige Signalmischung erfolgversprechender ist.

9.3 Fazit

Ein Radioprogramm mit Live-Inhalten und personalisierbarer Musik ist ein realisierbares Konzept, das sowohl den Anbietern als auch den Hörern Vorteile verspricht. Die große Verbreitung des Internets und die steigende Verfügbarkeit mobiler Internetzugänge eröffnet dem klassischen Radio neue Chancen und Möglichkeiten. Durch die Erstellung von musikalischen Geschmacksprofilen der Hörer ist es möglich, einen Radiosender an den individuellen Musikgeschmack der einzelnen Hörer anzupassen und damit die Hörerkreise unterschiedlicher Radioformate gemeinsam anzusprechen. Das Angebot eines personalisierten Radios ermöglicht es, mit dem produktionstechnischen Aufwand eines Radioprogramms eine deutlich größere Zielgruppe als bisher zu erreichen und dabei die Erwartungen und Wünsche der Hörer besser als bisher zu erfüllen. Die Notwendigkeit, eine gemeinsame musikalische Basis für die Zielgruppe zu finden, entfällt dabei.

Durch die Einführung eines personalisierten Radios lässt sich ferner das zweite Unterscheidungsmerkmal verschiedener Zielgruppen, das gewünschte Verhältnis zwischen Wort und Musik, beseitigen, indem wahlweise einzelne Wortblöcke nicht an alle Hörer gesendet sondern für einen Teil durch Musik ersetzt werden. Unterschiedliche Ansprüche an Art und Präsentation der gesendeten Inhalte können auf diese Weise allerdings nicht erfüllt werden.

Das Kernstück der technischen Umsetzung eines personalisierbaren Radios ist die Erzeugung der individuellen Sendepläne. Dies gelingt mit den meisten der getesteten Heuristiken in hinreichender Qualität, es hat sich aber gezeigt, dass einzelne Algorithmen bei diesem speziellen Problem deutlich schlechter abschneiden als die Ergebnisse in der Literatur nahe legen. Am besten geeignet erscheint der speziell für diese Arbeit angepasste Algorithmus KptreeExt. Dieser stellt einen sehr guten Kompromiss zwischen erreichten Scorewerten und erzeugten Fehlzeiten dar. Bei geringen Fehlzeiten erreichte KptreeExt die zweitbesten Scorewerte, erzeugte also ein Musikprogramm, das den Hörergeschmack sehr gut trifft und dabei wenig Füllung durch Trailer oder ähnliches benötigt. Da die gespielte Musik für viele Hörer ein entscheidendes Kriterium ist erscheint die Verwendung eines SSP-Algorithmus als ungeeignet. Eine Verbesserung brächte möglicherweise die Einführung eines Mindes-Scorewertes für

Titel, die dem SSP-Algorithmen zur Verfügung gestellt werden. Dies muss aber noch weiter erforscht werden.

Für ein personalisierbares Radio muss für jeden Hörer das Musikprogramm individuell berechnet werden. Dies ergab eine Analyse der Nutzerdaten von *last.fm*. Hierzu wurde eine zufällige Stichprobe von 17.500 Nutzern untersucht und in der Menge ihrer gehörten Musiktitel nach Übereinstimmungen gesucht. Dabei zeigte sich, dass ein Zusammenfassen von Nutzern mit sehr ähnlichem Musikgeschmack nicht möglich ist, da nur sehr wenige Nutzer eine ausreichende Menge an gehörten Titeln teilen. Selbst der Versuch, Nutzer bereits dann zusammenzufassen, wenn sich nur 10% ihrer gehörten Titel überschneiden, führte nicht zu einer ausreichenden Menge an Nutzer-Clustern.

Bis zur Realisierung eines Radioangebotes mit den in dieser Arbeit entworfenen Eigenschaften, gilt es allerdings noch eine Reihe von Problemen zu lösen. Diese betreffen vor allem die Entwicklung einer technischen Infrastruktur, die in der Lage ist, bei der Produktion eines solchen Programmes zu assistieren und das Programm zuverlässig an die Nutzer zu verbreiten.

Die weitere Erforschung dieses Konzeptes erscheint trotzdem wünschenswert und notwendig, um dem Medium Radio auch in Zukunft seine Konkurrenzfähigkeit zu sichern. Da durch das Internet und persönliche MP3-Sammlungen heutzutage jedem Konsumenten eine bisher nie dagewesene Vielfalt an Musik zur Verfügung steht, aus der er jederzeit ein, seinem Geschmack und seiner Stimmung entsprechendes, Musikprogramm zusammenstellen kann, läuft das Radio Gefahr, als Lieferant von aktueller und abwechslungsreicher Musik an Bedeutung zu verlieren. Lediglich seine Funktion als schneller Lieferant von Unterhaltung und Informationen, die ohne volle Aufmerksamkeit konsumiert werden können, erfüllt das Radio noch weitgehend unangefochten. Um auch im Musikbereich konkurrenzfähig zu bleiben, sollte das Radio in der Lage sein, die Wünsche jedes individuellen Hörers ähnlich gut zu erfüllen, wie es die Auswahl aus der persönlichen Musiksammlung kann. Ein solches Angebot, angereichert mit Unterhaltung und Informationen und weniger aufwändig als die händische Zusammenstellung aus der eigenen Musiksammlung, bietet damit die Vereinigung des „besten aus zwei Welten“ und somit einen echten Mehrwert gegenüber den heute verfügbaren Angeboten. Ein Weg dies zu realisieren wurde in dieser Arbeit aufgezeigt.

Literatur

[1] ARD/ZDF-Medienkommission and Projektgruppe ARD/ZDF-Multimedia. ARD/ZDF-Onlinestudie 2009. http://www.ard-zdf-onlinestudie.de and Media Perspektiven 07, 2009.

[2] ARD/ZDF-Medienkommission and Projektgruppe ARD/ZDF-Multimedia. ARD/ZDF-Onlinestudie 2010. http://www.ard-zdf-onlinestudie.de and Media Perspektiven 07-08, 2010.

[3] Egon Balas and Eitan Zemel. An algorithm for large zero-one knapsack problems. *Operations Research*, 28(5):pp. 1130–1154, 1980.

[4] David Baskerville and Tim Baskerville. *Music Business Handbook and Career Guide*. MUSIC BUSINESS HANDBOOK AND CAREER GUIDE. Sage Publications, 2009.

[5] Richard Bellman. *Dynamic programming*. Princeton University Press, 1957.

[6] John S. Belrose. Reginald aubrey fessenden and the birth of wireless telephony. *Antennas and Propagation Magazine, IEEE*, 44(2):38 – 47, Apr 2002.

[7] Ira Brodsky. How reginald fessenden put wireless on the right technological footing. *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, 2008.

[8] James M. Calvin and Joseph Y. T. Leung. Average-case analysis of a greedy algorithm for the 0/1 knapsack problem. *Operations Research Letters*, 31(3):202 – 210, 2003.

[9] Alberto Caprara, Hans Kellerer, Ulrich Pferschy, and David Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:2000, 1998.

[10] Chandra Chekuri and Sanjeev Khanna. A polynomial time approximation scheme for the multiple knapsack problem. *SIAM Journal on Computing*, 35(3):713–728, 2005.

[11] Lewis Coe. *Wireless radio: a brief history*. McFarland & Co Inc Pub, 2006.

[12] Andreas Egger and Thomas Windgasse. Radionutzung und MNT 2.0. *Media Perspektiven*, (05), 2007.

[13] Arbeitsgemeinschaft Media-Analyse e.V. Mediaanalyse 2010 Radio II, 10 2010.

[14] H.M.T. Fessenden. *Fessenden, builder of tomorrows*. Telecommunications Series. Arno Press, 1974.

[15] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[16] Dr. Parge & Partner Institute for International Market Research. Radiowirkung Plus, 2008. Studie im Auftrag und unter Mitwirkung der ARD Sales & Services und der Bayerischen Rundfunkwerbung GmbH.

[17] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[18] G.R.M. Garratt, Institution of Electrical Engineers, and Science Museum (Great Britain). *The early history of radio: from Faraday to Marconi*. History of technology series. Institution of Electrical Engineers in association with the Science Museum, 1994.

[19] Diptesh Ghosh and Nilotpal Chakravarti. A competitive local search heuristic for the subset sum problem. *Computers & Operations Research*, 26(3):271 – 279, 1999.

[20] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35:61–70, December 1992.

[21] Klaus Goldhammer. *Musikquoten im europäischen Radiomarkt*. Praxisforum Medienmanagement. Fischer, 2005.

[22] David B. Hauver and James C. French. Flycasting: Using collaborative filtering to generate a playlist for online radio. In *Proceedings of the First International Conference on WEB Delivering of Music (WEDELMUSIC'01)*, Washington, DC, USA, 2001. IEEE Computer Society.

[23] Conor Hayes and Padraig Cunningham. Smart radio - building music radio on the fly. *Knowledge-Based Systems*, 14(3-4):197 – 201, 2001.

[24] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[25] T. Ibaraki. Enumerative approaches to combinatorial optimization - part ii. *Ann. Oper. Res.*, 10:3–342, January 1988.

[26] Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *J. ACM*, 22:463–468, October 1975.

[27] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 1990.

[28] Hans Kellerer. A polynomial time approximation scheme for the multiple knapsack problem. In Dorit Hochbaum, Klaus Jansen, Jose Rolim, and Alistair Sinclair, editors, *Randomization, Approximation, and Combinatorial Optimization. Algorithms and Techniques*, volume 1671 of *Lecture Notes in Computer Science*, pages 51–62. Springer Berlin / Heidelberg, 1999.

[29] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack Problems.* Springer, 2004.

[30] Cicily Kodiyan. *Eminent Russian scientists.* Konarak Publishers, 1992.

[31] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3), 07 1960.

[32] Joachim-Felix Leonhard. *Medienwissenschaft.* Handbücher zur Sprach- und Kommunikationswissenschaft / Handbooks of Linguistics and Communication Science Series. Walter de Gruyter, 2001.

[33] S. Martello and P. Toth. Heuristic algorithms for the multiple knapsack problem. *Computing*, 27:93–112, 1981.

[34] Silvano Martello, David Pisinger, and Paolo Toth. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science*, 45(3):pp. 414–424, 1999.

[35] Silvano Martello and Paolo Toth. A bound and bound algorithm for the zero-one multiple knapsack problem. *Discrete Applied Mathematics*, 3(4):275 – 288, 1981.

[36] Silvano Martello and Paolo Toth. *Knapsack problems: algorithms and computer implementations.* Wiley-Interscience series in discrete mathematics and optimization. J. Wiley & Sons, 1990.

[37] Silvano Martello and Paolo Toth. Upper bounds and algorithms for hard 0-1 knapsack problems. *Operations Research*, 45(5):pp. 768–778, 1997.

[38] Dirk Martens and Thomas Windgasse. Nutzungsveraenderung und zukunftsperspektiven von webradio. *Media Perspektiven*, 3, 2010.

[39] Medienpädagogischer Forschungsverbund Südwest. JIM 2006. Jugend, Information, (Multi-)Media. 2006.

[40] Uwe Frigge Michael H. Haas and Gert Zimmer. *Radio-Management: ein Handbuch für Radio-Journalisten.* Reihe Praktischer Journalismus. Ölschläger, 1991.

[41] Dieter K. Müller. Radio - der Tagesbegleiter mit Zukunft. *Media Perspektiven*, 1, 2007.

[42] Paul J. Nahin. *The science of radio: with MATLAB and Electronics Workbench demonstrations.* AIP Press, 2001.

[43] Raymond T. Ng and Jiawei Han. Clarans: A method for clustering objects for spatial data mining. *IEEE Trans. on Knowl. and Data Eng.*, 14:1003–1016, September 2002.

[44] Tim O'Shei. *Marconi and Tesla: Pioneers of Radio Communication*. Inventors Who Changed the World. Enslow Pub Inc, 2008.

[45] David Pisinger. An expanding-core algorithm for the exact 0-1 knapsack problem. *European Journal of Operational Research*, 87:175–187, 1993.

[46] David Pisinger. *Algorithms for Knapsack Problems*. PhD thesis, University of Copenhagen, 1995.

[47] David Pisinger. A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, 45(5):pp. 758–767, 1997.

[48] David Pisinger. An exact algorithm for large multiple knapsack problems. *European Journal of Operational Research*, 114:528–541, 1999.

[49] David Pisinger. Where are the hard knapsack problems? *Comput. Oper. Res.*, 32:2271–2284, September 2005.

[50] Bartosz Przydatek. A fast approximation algorithm for the subset-sum problem. 1999.

[51] M. Radovsky and G. Yankovsky. *Alexander Popov Inventor of Radio*. Men of Russian Science. International Law & Taxation Publishers, 2001.

[52] Helmut Reitze and Christa-Maria Ridder, editors. *Massenkommunikation VII: Eine Langzeitstudie zur Mediennutzung und Medienbewertung 1964-2005*. Nomos, 2006.

[53] Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *J. ACM*, 22:115–124, January 1975.

[54] Karolin Schmidt and Lothar Mai. Die Fussball-Bundesliga im Radio. *Media Perspektiven*, 2, 2010.

[55] Alexander Strehl and Joydeep Ghosh. A scalable approach to balanced, high-dimensional clustering of market-baskets. In *Proceedings of the 7th International Conference on High Performance Computing*, HiPC '00, pages 525–536, London, UK, 2000. Springer-Verlag.

[56] Ke Wang, Chu Xu, and Bing Liu. Clustering transactions using large items. In *Proceedings of the eighth international conference on Information and knowledge management*, CIKM '99, pages 483–490, New York, NY, USA, 1999. ACM.

[57] Thomas Windgasse. Webradio: Potenziale eines neuen Verbreitungswegs für Hörfunkprogramme. *Media Perspektiven*, 3, 2009.

[58] Ching-Huang Yun, Kun-Ta Chuang, and Ming-Syan Chen. An efficient clustering algorithm for market basket data based on small large ratios. *Computer Software and Applications Conference, Annual International*, 0:505, 2001.

Anhang

A DVD-Rom

Auf der beiliegenden DVD-Rom befinden sich die entwickelten Python-Programme zum Erzeugen und Testen der Sendepläne, sowie eine Kopie der verwendeten Daten von *last.fm.*

B SQL Datenbank Struktur

```
1  --
2  -- Table structure for table `artist`
3  --
4
5  CREATE TABLE `artist` (
6     `artistUrl` varchar(255) NOT NULL,
7     `artistName` varchar(255) NOT NULL,
8     `artistMbid` varchar(255) NOT NULL,
9     PRIMARY KEY (`artistUrl`)
10 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
11
12
13 -- --------------------------------------------------------
14
15 --
16 -- Table structure for table `tagrelationenArtist`
17 --
18
19 CREATE TABLE `tagrelationenArtist` (
20    `tagID` int(11) NOT NULL,
21    `tagZiel` varchar(255) NOT NULL,
22    `score` smallint(6) NOT NULL,
23    PRIMARY KEY (`tagID`,`tagZiel`),
24    KEY `tagZiel` (`tagZiel`)
25 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
26
27 -- --------------------------------------------------------
28
29 --
30 -- Table structure for table `tagrelationenTitel`
31 --
32
33 CREATE TABLE `tagrelationenTitel` (
34    `tagID` int(11) NOT NULL,
35    `tagZiel` varchar(255) NOT NULL,
36    `score` smallint(6) NOT NULL,
37    PRIMARY KEY (`tagID`,`tagZiel`),
38    KEY `tagZiel` (`tagZiel`)
39 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
40
41 -- --------------------------------------------------------
42
43 --
44 -- Table structure for table `tags`
45 --
46
47 CREATE TABLE `tags` (
48    `tagID` int(11) NOT NULL AUTO_INCREMENT,
49    `tagName` varchar(255) NOT NULL,
50    PRIMARY KEY (`tagID`),
51    UNIQUE KEY `tagName` (`tagName`)
52 ) ENGINE=InnoDB  DEFAULT CHARSET=utf8;
53
54 -- --------------------------------------------------------
55
56 --
57 -- Table structure for table `titel`
58 --
59
60 CREATE TABLE `titel` (
61    `titelUrl` varchar(255) NOT NULL,
62    `titelName` varchar(255) NOT NULL,
63    `artistUrl` varchar(255) NOT NULL,
64    `laenge` mediumint(9) NOT NULL,
65    PRIMARY KEY (`titelUrl`),
66    KEY `artistUrl` (`artistUrl`)
67 ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
68
```

```
-- --------------------------------------------------------

--
-- Table structure for table `user`
--

CREATE TABLE `user` (
  `userName` varchar(255) NOT NULL,
  `wortstufe` smallint(6) NOT NULL,
  PRIMARY KEY (`userName`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- --------------------------------------------------------

--
-- Table structure for table `userTitel`
--

CREATE TABLE `userTitel` (
  `userName` varchar(255) NOT NULL,
  `titelUrl` varchar(255) NOT NULL,
  `playcount` int(11) NOT NULL,
  `score` float NOT NULL,
  UNIQUE KEY `userTitel` (`userName`,`titelUrl`),
  KEY `userName` (`userName`),
  KEY `titelUrl` (`titelUrl`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

--
-- Constraints for dumped tables
--

--
-- Constraints for table `tagrelationenArtist`
--
ALTER TABLE `tagrelationenArtist`
  ADD CONSTRAINT `tagrelationenArtist_ibfk_1` FOREIGN KEY (`tagID`) REFERENCES `tags` (`tagID`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `tagrelationenArtist_ibfk_2` FOREIGN KEY (`tagZiel`) REFERENCES `artist` (`artistUrl`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `tagrelationenTitel`
--
ALTER TABLE `tagrelationenTitel`
  ADD CONSTRAINT `tagrelationenTitel_ibfk_1` FOREIGN KEY (`tagID`) REFERENCES `tags` (`tagID`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `tagrelationenTitel_ibfk_2` FOREIGN KEY (`tagZiel`) REFERENCES `titel` (`titelUrl`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `titel`
--
ALTER TABLE `titel`
  ADD CONSTRAINT `titel_ibfk_1` FOREIGN KEY (`artistUrl`) REFERENCES `artist` (`artistUrl`) ON DELETE CASCADE ON UPDATE CASCADE;

--
-- Constraints for table `userTitel`
--
ALTER TABLE `userTitel`
  ADD CONSTRAINT `userTitel_ibfk_1` FOREIGN KEY (`titelUrl`) REFERENCES `titel` (`titelUrl`) ON DELETE CASCADE ON UPDATE CASCADE,
  ADD CONSTRAINT `userTitel_ibfk_2` FOREIGN KEY (`userName`) REFERENCES `user` (`userName`) ON DELETE CASCADE ON UPDATE CASCADE;
```

C Sendungsschema (Morning-Show)

```
1  Die an dieser Stelle in der Arbeit befindlichen Informationen unterliegen den Geschä
       ftsgeheimnissen der Rheinland-Pfälzische Rundfunk GmbH & Co KG und dürfen
       leider nicht in digitaler Form verbreitet werden.
```