Effiziente Algorithmen und Datenstrukturen zur inhaltsbasierten Suche in Audio- und 3D-Moleküldaten

Dissertation

zur

Erlangung des Doktorgrades (Dr. rer. nat.)

der

Mathematisch-Naturwissenschaftlichen Fakultät

der

Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von

Andreas Ribbrock

aus

Issum am Niederrhein

Bonn, 24. September 2006

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn.

Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn http://hss.ulb.uni-bonn.de/diss_online elektronisch publiziert.

1. Referent: Prof. Dr. Michael Clausen
2. Referent: Prof. Dr. Rainer Manthey

Tag der Promotion: 15. Februar 2007

Erscheinungsjahr: 2007

Danksagung

Diese Arbeit ist in der Arbeitsgruppe von Prof. Dr. Clausen im Rahmen des DFG-Projekts „Digitale Bibliotheken“ entstanden. Ohne diese Förderung durch die Deutsche Forschungsgemeinschaft läge diese Arbeit nun nicht vor.
Ich möchte mich bei einer Vielzahl von Personen bedanken, die auf unterschiedliche Art und Weise dazu beigetragen haben, dass diese Arbeit entstanden ist:

- Mein ganz besonderer Dank gilt Herrn Prof. Dr. Michael Clausen, der jederzeit für Diskussionen zur Verfügung stand und dessen Einsatz und Hartnäckigkeit diese Arbeit erst möglich gemacht haben,
- Bedanken möchte ich mich Herrn Prof. Dr. Rainer Manthey dafür, dass er sich bereit erklärt hat das Zweitgutachten anzufertigen,
- Ein ebenfalls grosses Dankeschön geht an alle Kollegen für die gute Zusammenarbeit . Im einzelnen erwähnen möchte ich Rolf Bardeli, Frank Kurth, Axel Mosig, Meinard Müller und Tido Röder.
- Sehr herzlich bedanken möchte ich mich bei meiner Frau Michaela für vieles. Ganz besonders dafür, mich immer wieder anzutreiben, diese Arbeit fertigzustellen,
- Natürlich möchte ich bei meinen Eltern bedanken, die mir das Studium ermöglicht haben,
- Dank auch allen Diplomanden dafür, dass sie es mir als Betreuer so leicht gemacht haben. Vielen Dank auch an die StudHKs David Damm und Cristian Fremery für die sehr gute Zusammenarbeit bei der Entwicklung des „Sync-Player“,
- Another big thank you goes to Prof. Dr. Ian Witten for the warm welcome at the Digital Library Lab at the Waikato University. I would also like to thank John McPherson and Michael Dewsnip for taking care of me during my stay in Hamilton,
- Zu danken habe ich auch Hanno Gehron, Martin Schulze und alle anderen ehemaligen Kollegen der Fa. BusinessCoDe.

Bonn, im September 2006

Zusammenfassung

Gegenstand der vorgelegten Arbeit ist die inhaltsbasierte Suche in Audio- und Molekülдaten. Ausgangspunkt sind große Kollektionen $\mathcal{D}' = (D'_1, \ldots, D'_N)$ von solchen multimedialen Dokumenten. Eine Grundaufgabe des inhaltsbasierten Retrievals besteht darin, ein möglicherweise verrauschtes oder deformiertes Fragment eines Dokuments der Kollektion in $\mathcal{D}'$ zu lokalisieren. Zur effizienten Lösung dieses Retrievalproblems greifen wir das von der AG Clausen entwickelte Konzept des gruppentheoretisch basierten Retrievals auf. Dabei wird die Kollektion $\mathcal{D}'$ zunächst mittels geeigneter Merkmalsextraktoren in eine Kollektion $\mathcal{D} = (D_1, \ldots, D_N)$ überführt, die sich auf die relevanten Aspekte von $\mathcal{D}'$ beschränkt. Die Ausgangsdokumente D'_i sind in vielen Anwendungen Teilmengen einer Menge M', auf der eine Gruppe G operiert, und die Merkmalsextraktoren sind G-Morphismen zwischen M' und M zugeordneten G-Mengen. Eine Anfrage Q' ist selbst eine Teilmenge von M', woraus nach Merkmalsextraktion eine Teilmenge Q von M entsteht. Gesucht sind nun alle Gruppenelemente g und alle Dokumentennummern i, so dass die Anwendung von g auf Q' eine Teilmenge von D'_i liefert. Aus Effizienzgründen wird stattdessen nach allen Paaren (g, i) mit $gQ \subseteq D_i$ gesucht. Aufgrund des Informationsverlustes beim Übergang von $\mathcal{D}'$ nach $\mathcal{D}$ können nun Treffer in der Kollektion $\mathcal{D}$ auftreten, die keine Treffer in der Originalkollektion sind („false positives“). Wegen der Morphismuseigenschaft des Merkmalsextraktors sind aber „false negatives“ prinzipiell nicht möglich. Damit nimmt die Entwicklung geeigneter Merkmalsextraktoren für unterschiedliche Dokumentenarten eine Schlüsselrolle ein.

Zur effizienten Trefferberechnung werden die Elemente der Dokumente D_i in invertierten Listen abgelegt. Dabei genügt es, pro G-Bahn eine invertierte Listen abzuspeichern, da die Listen zu Elementen einer G-Bahn durch leichte Umrechnung auseinander hervorgehen. Um redundanzfreie Listen zu bekommen sollten alle Stabilisatoren trivial sein. Die Trefferberechnung kann nun durch Schnitt der an Q beteiligten Listen durchgeführt werden.

Im ersten Teil der Arbeit werden auf dieser Basis neue effiziente Suchalgorithmen vorgestellt, die es erlauben Retrievalsysteme zu entwickeln, die mit praxisrelevanten Datenmengen umgehen können. Besonders hervorzuheben ist dabei die Möglichkeit, Suchindexe mit unterschiedlichen Auflösungsstufen zu verwenden, um so die benötigte Zeit für die Bearbeitung einer Suchanfrage zu reduzieren. Es wird eine Übersicht über unterschiedliche Audio-Retrievalsysteme gegeben, die derzeit zum Teil kommerziell eingesetzt werden. Weiterhin werden unterschiedliche Merkmalsextraktoren für CD-Audiodaten vorgestellt, die es z.B. erlauben, einen Suchindex für viele tausend Musikstücke aufzubauen und dabei auch verlustbehaftet kodierte Audiodaten (z.B. MP3) als Anfragen zuzulassen.

Im zweiten Teil der Arbeit spielen Dokumente eine Rolle, die dreidimensionale Ortskoordinaten von Atomen von biologischen Makromolekülen (Proteine, DNA-Sequenzen aus der Protein Data Bank, PDB) beinhalten. Ausgehend von den zuvor vorgestellten Algorithmen wird ein Retrievalsystem zur inhaltsbasierten Suche in dreidimensionalen Molekülдaten entwickelt. Zunächst geht es um die aus Anwendungssicht geeignete Modellierung der Elemente der euklidschen Bewegungsgruppe $SE(3)$. Aufgrund des geringeren Speicheraufwands werden Quaternionen hierbei verwendet. Diese Grup-

pe stellt an die extrahierten Merkmale besonderen Anforderungen. Um bis auf pathologische Fälle stets triviale Stabilisatoren zu garantieren, werden jeweils drei benachbarte Atome zu einem Merkmal zusammengefasst. Bei der Indexerstellung werden alle möglichen Kombinationen bzgl. eines maximalen euklidschen Abstands zwischen zwei Atomen δ betrachtet. Bei einer Anfrage hingegen, ist die minimale Anzahl von Merkmalen zu bestimmen, so dass dennoch alle Atome der Anfrage in mindestens einem Merkmal enthalten sind. Für allgemeine Graphen führt dies auf ein NP-vollständiges Überdeckungsproblem. Im Rahmen dieser Arbeit konnte jedoch gezeigt werden, dass für die speziellen Graphen basierend auf Moleküldaten ein Algorithmus existiert, der in Polynomialzeit eine fast-optimale Überdeckung berechnet. Dieses Ergebnis kann allgemein für die inhaltsbasierte Suche in attributierten 3D-Punktdaten genutzt werden.
Anstelle der Suche auf atomarer Ebene bietet sich im Fall von Proteinen an, die Moleküle der jeweiligen Aminosäuren als Merkmale zu benutzen. Wie sich zeigt, führt dies zu einer deutlichen Reduktion des Indexierungsaufwands. Der höhere Semantikgehalt dieser Merkmale spiegelt sich auch in der Aussagekraft der Treffer wider. Außerdem kann die Sequenz einer angefragten Konstellation von Aminosäuren dazu benutzt werden, der inhaltsbasierten Suche bereits bestehende Sequenzalignmentalgorithmen vorzuschalten. Die Suchalgorithmen sind derart erweitert worden, dass die in den invertierten Listen abgespeicherten Elemente nur dann zur Suche herangezogen werden, wenn die jeweilige Dokumentennummer durch den vorgeschalteten Alignmentalgorithmus berechnet worden ist. Dieses Prinzip lässt sich auch auf die Verwendung anderer Metadaten übertragen.
Die theoretischen Ergebnisse wurden in einer generischen C++ Bibliothek implementiert und über die Java-RMI Technologie in ein bestehendes Visualisierungstool für 3D-Moleküldaten integriert. Die Verwendung der Java-RMI-Technologie ermöglicht eine Client-Server Architektur und die Bereitstellung eines entsprechenden Suchdienstes über das Internet.

Stichworte: Inhaltsbasierte Suche, Indexierung, Suchalgorithmen, Audiosignalverarbeitung, 3D-Moleküldaten, Protein Data Bank (PDB), Gruppenaktionen

Inhaltsverzeichnis

Kapitel 1

Einleitung

Suchmaschinen wie z.B. „Google“ haben maßgeblich zum Erfolg des Internets beigetragen. Die gigantische Menge an Information, die ständigen Änderungen unterliegt, erfordert zur Befriedigung des Informationsbedarfs zahlreicher Nutzer die maschinelle Erstellung eines Katalogs, der nach Stichworten, Phrasen etc. durchsucht werden kann. Dabei spielt die Art, in der die Information vorliegt, eine entscheidende Rolle. Die klassischen Suchmaschinen im Internet bauen lediglich auf die Erfassung und Auswertung von *textuellen* Informationen. Neben klassischen Textdokumenten gehören hierzu auch sog. *Metadaten* von Bild-, Video- oder Audiodateien, die den Inhalt der Dateien mehr oder weniger gut *beschreiben*.

Im Gegensatz dazu werden unter dem Begriff der „inhaltsbasierten Suche“ (bzw. des „Content Based Retrieval“) Ansätze zusammengefasst, welche bei der Suche in einem Datenbestand sich eben gerade nicht (nur) auf ggf. fehlerhafte textuelle Informationen verlassen, sondern die sich auf den *Inhalt* eines *Dokuments* stützen. In der Regel sind inhaltsbasierte Retrievalsysteme auf einen bestimmten Typ von Dokument, z.B. CD-Audiodaten, Bilder, 3D-Moleküldaten usw. oder sogar auf nur eine bestimmte Gruppe von Dokumenten (z.B. nur CD-Audiodaten zu klassischer Musik) eines Typs beschränkt. Im Gegensatz zur Suche in textuellen Dokumenten erwarten die meisten inhaltsbasierten Retrievalsysteme als Anfrage ein Dokument (z.B. ein Fragment eines Audiostücks oder ein Teil eines Moleküls), welches vom Typ her den indexierten Dokumenten entspricht. Allgemein wird diese Art der Anfrage als „Query by Example“ (QbE) bezeichnet.

In der Arbeitsgruppe Clausen an der Universität Bonn wurde ein grundlegendes Konzept zur inhaltsbasierten Suche basierend auf einem abstrakten Ansatz zur Konstellationssuche entwickelt. Dieser beruht auf dem Konzept der *Operation einer Gruppe G auf einer Menge* und der Speicherung der indexierten Information in *G-invertierten Listen*. Dabei soll der Zusatz „G-“ verdeutlichen, dass auf den Elementen in den Listen eines klassischen invertierten Suchindexes ein Element einer Gruppe G angewendet wird. Dies wird als *Justierung* der G-invertierten Liste bezeichnet. Da die dazu verwendeten Gruppenelemente aus der Anfrage Q hervorgehen, wird auch häufig von der Justierung einer Liste bzgl. einer Anfrage gesprochen.

Ziel dieser Arbeit ist es zu zeigen, dass dieser abstrakte Ansatz es erlaubt, inhaltsbasierte Retrievalsysteme für verschiedenste Arten von Dokumenten zu entwickeln und dabei auf ein und dieselbe Bibliothek von Suchalgorithmen und Datenstrukturen zurückzugreifen. Durch die deutliche Trennung von Dokumenteninhalt (modelliert durch Merkmale (engl. „features“)) und Gruppenelementen, können die Suchalgorithmen unabhängig von dem jeweiligen Dokumententyp implementiert werden, da diese prinzipiell lediglich auf den gespeicherten Gruppenelementen arbeiten.

Dazu werden in dieser Arbeit generische Algorithmen und Datenstrukturen vorgestellt, die unter-

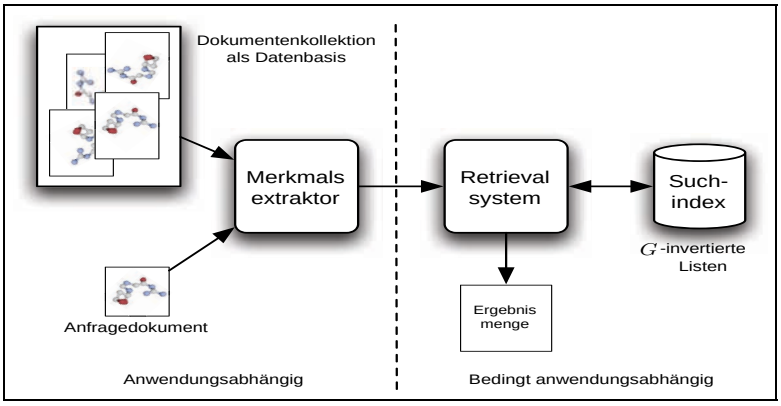


Abbildung 1.1: Die Abbildung beschreibt den grundsätzlichen Aufbau, wie er Retrievalsystemen zugrunde liegt, die in dieser Arbeit beschrieben und entwickelt werden. Die Rolle des Merkmalextraktors als Adapter zwischen den Dokumenten und dem eigentlichen Retrievalsystem ist deutlich zu erkennen. Wenn keine anwendungsspezifischen Arten von Fehlertoleranz benötigt werden, ist der rechte Teil der Abbildung prinzipiell ohne Bezug zur Anwendung, wenn man von der Realisierung der jeweiligen Gruppe G absieht.

schiedlichen Anforderungen aus der Praxis genügen. Je nach Art der betrachteten Gruppe und Datenmenge bieten sich unterschiedliche Algorithmen an. Allgemein kann gesagt werden, dass die Wahl eines Algorithmus von Faktoren wie Datenmenge, Länge der Anfrage, Komplexität der Gruppenoperationen und der geforderten Fehlertoleranz abhängt.
Um die oben erwähnte Basistechnologie mit den unterschiedlichsten Arten von Dokumenten verwenden zu können, ist ein wichtiger Bestandteil eines jeden Retrievalsystems, eine Art *Adapter*, der die jeweiligen Dokumente eines Typs in eine Darstellung überführt, die die Verwendung der Basistechnologie ermöglicht. Die Entwicklung dieser sog. *Merkmalsextraktoren* stellt häufig die größte Schwierigkeit bei der Umsetzung eines Retrievalsystems dar. Zum einen sollen diese lediglich die „wichtigsten“ Eigenschaften eines Dokuments extrahieren, um auf diese Weise die Datenmenge, die im Suchindex abgelegt werden muss, zu minimieren, da dies einen direkten Einfluss auf die Antwortzeit des Retrievalsystems hat. Welche Eigenschaften dies sind, hängt entscheidend von der jeweiligen Anwendung ab. Beschreiben die extrahierten *Merkmale* ein Dokument nicht eindeutig, steigt die Gefahr von sog. „False-Positive“-Treffern. Umgekehrt kann es sein, dass Merkmale, die ein Dokument zu genau beschreiben, für die inhaltsbasierte Suche nicht geeignet sind, da die Merkmale zu einer nur leicht veränderten Anfrage bereits völlig unterschiedlich sind. In einem solchen Fall bliebe die Trefferliste meist leer. Neben den vorgestellten Fehlertoleranzmechanismen wird meist ein gewisses Maß an Unschärfe bereits durch die Merkmale modelliert. Wie unterschiedlich Merkmalsextraktoren sein können, zeigen die beiden Anwendungen, die in diese Arbeit diskutiert werden.

Retrievalalgorithmen

Im Grunde ist bei der jeder Suchanfrage Q bestehend aus n Elementarobjekten $q_1, \ldots, q_n$ und k zugelassenen Fehlstellen die Frage zu beantworten, ob es in den bzgl. Q *justierten* G-invertierten Listen $G_1, \ldots, G_n$ Einträge gibt, die in mindestens $(n - k)$ vielen dieser Listen vorkommen. Das einfachste Suchverfahren besteht darin, eine Liste mit den Elementen aller der n G-invertierten Listen

zu erstellen, diese zu sortieren und danach sequentiell zu durchlaufen, wobei die Häufigkeiten gleicher aufeinanderfolgender Elemente gezählt wird.
Das von Clausen et al. z.B in [CK04a] vorgestellte Suchverfahren basiert hingegen auf der Berechnung von Listenschnitten mittels dynamischer Programmierung. Jedoch hat sich bei großen Datenmengen dies nicht immer als die beste Wahl herausgestellt. Bei der Fehlstellensuche wird aufwändig eine temporäre Liste von Trefferkandidaten verwaltet, die in späteren Schritten des Algorithmus schnell wieder auf die Zahl der endgültigen Treffer reduziert wird. Dabei sollte diese temporäre Kandidatenliste zu jedem Zeitpunkt in den Hauptspeicher passen. Bei sog. *Fuzzyanfragen* werden an einer Position mehrere Alternativen in der Anfrage erlaubt. Dies wird in [CK04a] durch die Vereinigung von G-invertierten Listen erreicht – ggf. auch ein sehr rechenaufwändiger Schritt.
Aus diesem Grund werden in dieser Arbeit eine Vielzahl von neu entwickelten Suchalgorithmen vorgestellt, wobei sich zwei Arten unterscheiden lassen. Ist die Gruppe G *angeordnet*, so ist zu jedem Zeitpunkt sichergestellt, dass die Elemente einer G-invertierten Liste bzgl. einer Relation „$<$“ angeordnet sind. Diese Information kann bei der Suche genutzt werden, um z.B. Einträge in den G-invertierten Listen erst gar nicht betrachten zu müssen. Operiert hingegen ein Element einer nicht angeordneten Gruppe auf den sortiert vorliegenden Elementen einer G-invertierten Liste, so ist deren Sortierung danach in der Regel nicht mehr gegeben. Die nachträgliche Sortierung der G-invertierten Liste ist bei großen Datenmengen keine Alternative, da häufig alle zu betrachtenden G-invertierten Listen nicht in den Hauptspeicher passen und somit nach einer Sortierung erst wieder auf den Hintergrundspeicher geschrieben werden müssen.
Durch die Verwendung von dynamischen Datenstrukturen wird allen unterschiedlichen Elementen der zu betrachtenden G-invertiertelen Listen ein Zähler zugewiesen, der die jeweilige Häufigkeit während der Suche erfasst. Ist diese größer oder gleich dem Schwellwert $n - k$, gehört ein solches Element zur Treffermenge. Nachteilig an den dazu verwendeten Baumdatenstrukturen ist, dass diese häufig reorganisiert werden müssen (z.B. bei Knotenüberläufen in R*-Bäumen), was die Suche ggf. deutlich verlangsamen kann.
Da die Dokumente der Dokumentenkollektion den Raum der Trefferkandidaten einschränken, können Hyperwürfeln verwendet werden, wobei *jedem* möglichen Treffer eine Zelle in dem Würfel zugeordnet wird, in der die Häufigkeit gezählt wird. Schon bei nur zwei Dimensionen kann jedoch der Hauptspeicher dazu nicht mehr ausreichen. Aus diesem Grund wird ein hierarchisches Verfahren vorgestellt, wo zunächst grob Bereiche in dem Hyperwürfel gesucht werden, in denen sich überhaupt ein Trefferkandidat befinden kann. Diese Teile des Würfels werden dann rekursiv auf einem höheren Detailgrad näher untersucht. Zusammen mit speziell an die Auflösungsstufen der verwendeten Würfel vorberechneten und vergröberten G-invertierten Listen kann auf Kosten des Speicherplatzes auf dem Hintergrundspeicher eine Reduktion der Anzahl zu berechnender Gruppenoperationen erreicht werden. Die Arbeit widmet sich zudem der effizienten Beantwortung von Fuzzy-Anfragen, *ohne* die vorherige Vereinigung zweier oder mehrerer G-invertierter Listen zu berechnen.

Audioidenti kation

Eine wichtige Anwendung im Bereich der inhaltsbasierten Suche in Audiodaten ist die sog. *Audioidenti kation*. Dabei geht es um die Zuordnung eines Audiosignalfragmentes zu dem Signal, in dem es enthalten ist. Auf diese Weise können dem unbekannten Signal entsprechende Metadaten zugeordnet werden. Einige der Systeme sind bereits kommerziell verfügbar und werden meist von Mobilfunkanbietern genutzt, um dem Kunden zu ermöglichen, unbekannte Musik z.B. aus dem Radio einen Namen zu geben und ggf. den Titel oder das Album zum Kauf anzubieten. Eine weitere

Anwendung verfolgt z.B. die Firma MusicDNS [mus06a], die hochwertige Metadaten (Titel, Album, Künstler,...) anbietet. Durch die Audioidentifikation können unbekannte oder falsch benannte Musikstücke aus einer großen Sammlung automatisch um Metainformationen ergänzt werden.
Das Problem der Audioidentifikation ist von mehreren Forschungsgruppen betrachtet worden, wobei sich die gewählten Ansätze sehr unterscheiden. Über Clustering-Verfahren, Hidden Markov Modelle, riesigen Hashtabllen bis hin zum gruppenbasierten Ansatz der AG Clausen. Dabei verfolgt das System der Firma Shazam [Wan03] noch am ehesten einen vergleichbaren Ansatz wie das von Clausen et al. vorgestellte allgemeine Verfahren zur inhaltsbasierten Suche. Jedoch ist der Ansatz von Shazam eher näher dem bekannten „Geometric Hashing“ [WR97] anzusiedeln.
Bei der detaillierten Betrachtung der einzelnen Audioidentifikationssysteme wird deutlich werden, dass der Speicherplatzbedarf des Suchindexes basierend auf G-invertierten Listen meist deutlich geringer ausfällt als bei Systemen, die im wesentlichen Hashing einsetzen. Mit den in dieser Arbeit vorgestellten neuen Suchalgorithmen und dem ebenfalls im Rahmen dieser Arbeit entwickelten Frameworks [Rib06] zum Verteilen von Suchindexen auf verschiedenen Rechnern in einem Netzwerk, ließe sich durchaus ein Audioidentifikationssystem erstellen, das den Datenmengen von 10 Millionen und mehr Musiktiteln gewachsen wäre. Gerade im Bezug auf die kommerzielle Verwendung ist die Merkmalsextraktion im Falle von Audiosignalen besonders interessant. Die Betrachtung der gewählten Merkmale in den unterschiedlichen Systemen zeigt die Bandbreite an Lösungen. Allerdings sind bei vielen Systemen die gewählten Merkmale sehr eng mit der jeweils gewählten Indexdatenstruktur und den Retrievalalgorithmen verbunden.
Anhand der Audioidentifikation werden die unterschiedlichen Anforderungen an einen Merkmalsextraktor diskutiert, wie sie bei nahezu allen Anwendungsfällen unabhängig vom betrachteten Dokumententyp auftreten. So sollen die aus einem Audiosignal von einer CD extrahierten Merkmale eine gewisse *Robustheit* aufweisen, so dass z.B. aus dem gleichen, aber MP3-komprimierten Signal, nahezu die gleichen Merkmale extrahiert werden. Ausserdem wird an diesem Anwendungsbeispiel sehr leicht deutlich, warum ein Merkmalsextraktor möglichst *invariant* gegenüber Fragmentbildung sein soll. Ist die Anfrage ein Fragment von 10 Sekunden Länge aus der Mitte eines Audiosignals, so dürfen die extrahierten Merkmale zu keiner Zeit von den Signalteilen abhängen, die in dem Originalsignal vor bzw. hinter dem Fragment vorhanden sind. Das Beispiel der Audioidentifikation zu betrachten bietet sich auch deswegen an, da die verwendete Gruppe und deren Operation sehr einfach sind und man daher sehr gut die Funktionsweise von Suchalgorithmen verdeutlichen kann.

Suche in dreidimensionalen Moleküldaten

Nachdem zuvor von Audiodokumenten gesprochen worden ist, bildet die Entwicklung eines inhaltsbasierten Retrievalsystems zur Suche in biologischen Molekülen einen zweiten Schwerpunkt dieser Arbeit. Wurde durch die Gruppe $G = (\mathbb{Z}, +)$ im Fall der Audioidentifikation lediglich die Verschiebung eines Signalfragmentes um ganzzahlige Werte entlang der Zeitachse modelliert, liegt der Suche in Moleküldaten die Gruppe der euklidschen Bewegungen im $\mathbb{R}^3$ zugrunde. Während der Suche sind die am häufigsten zu berechnenden Operationen die Verknüpfung zweier Gruppenelemente und die Invertierung eines einzelnen Gruppenelements. War dies bei der Audioidentifikation noch die ganzzahlige Addition bzw. Subtraktion, ist es sinnvoll, die Elemente der Gruppe $SE(3)$ möglichst so zu realisieren, dass wenig Speicherplatz benötigt wird und die Operationen dennoch effizient berechnet werden können. Aus diesem Grund werden in dieser Arbeit Elemente der $SE(3)$ durch Quaternionen realisiert.
Neben den aufwändiger zu berechnenden Gruppenoperationen ist die Wahl der Merkmale von ent-

scheidender Bedeutung. Einzelne Atome kommen nicht in Frage, da dadurch die Orientierung des Moleküls im Raum unberücksichtigt bliebe. Der Übergang zu Paaren von benachbarten Atomen löst nur auf den ersten Blick dieses Problem, da nicht alle Stabilisatoren trivial sind. Um dieses zu erreichen, werden *Tripel* von benachbarten Atomen als *Elementar-* oder *Indexobjekte* herangezogen. Wie bereits erwähnt, spielt die Anzahl der Merkmale in einer Anfrage eine entscheidende Rolle, wenn es um die Antwortzeit eines Retrievalsystems geht. Somit drängt sich die Frage auf, mit welcher *minimalen* Anzahl von Merkmalen ein gegebenes Molekülfragment bei einer Anfrage vollständig beschrieben werden kann. Dies führt zu dem NP-vollständigen Problem der Berechnung einer Überdeckung aller Knoten eines ungerichteten Graphens mit einer minimalen Anzahl von Pfaden der Länge zwei [BCR06]. Für den speziellen Fall der Moleküle, kann jedoch ein Algorithmus angegeben werden, der in Polynomialzeit eine optimale Lösung berechnet und so Anfragen minimaler Länge ermöglicht. Da die Grundlage für die Berechnung des ungerichteten Graphen ein Abstandsmaß δ ist, sind die hier gemachten Aussagen auch für andere Dokumentenarten interessant, deren Inhalt ebenfalls Punktmengen im dreidimensionalen Raum darstellen und auf denen ein ähnlicher Abstandsbegriff definiert werden kann.
Die hierbei gewonnenen Erkenntnisse lassen sich direkt nutzen, um ein Retrievalsystem zu entwickeln, welches nicht auf atomarer, sondern auf der semantisch höheren Ebene der Aminosäuren arbeitet. Anstelle einzelne Atome zu Tripeln zusammen zu fassen, werden die Atome, welche in einem Protein zu jeweils einem Aminosäuremolekül gehören, als Indexobjekte aufgefasst. Zum einen wird dadurch eine Abstraktion von den exakten Atompositionen erreicht, zum anderen kann mittels bestehender Algorithmen zum Vergleich von Aminosäuresequenzen *vor* der inhaltsbasierten Suche auf die Menge der Treffermoleküle eingeschränkt werden und so die Antwortzeit des Systems ggf. deutlich reduziert werden. Natürlich kann eine vorgeschaltete textuelle Metadatensuche auf gleiche Weise eingesetzt werden – unabhängig von dem jeweiligen Dokumententyp. Eine entsprechende Erweiterung der Suchalgorithmen wird in dieser Arbeit vorgestellt.
Durch die Möglichkeit von Rundungsfehlern bei der Berechnung der Verknüpfung zweier Gruppenelemente, werden auch Erweiterungen zu denen in [CK04a, CEMS00] vorgestellten Suchalgorithmen notwendig. Diese Applikation erfordert ebenfalls die Erweiterung des Retrievalsystems um neue Fehlertoleranzmechanismen, die durch die möglichen Unterschiede von Anfrage und Trefferdokument bestimmt sind. Auf diese Weise wurde die Bibliothek von verfügbaren Suchalgorithmen entsprechend erweitert, so dass dieses Wissen in anderen Anwendungen benutzt werden kann, die z.B. ebenfalls die euklidsche Gruppe $SE(3)$ zugrundelegen.

Implementierung

In Abbildung 1.1 wird deutlich, dass ein großer Teil der Retrievalsystems prinzipiell unabhängig ist von der Art der Dokumente in der Dokumentenkollektion. Abgesehen von speziell auf eine Anwendung zugeschnittenen Algorithmen, kann das Retrievalsystem und der Suchindex durch generische Programmierung sehr allgemein implementiert werden. Für eine konkrete Anwendung sind lediglich der Merkmalsextraktor zu implementieren, die Elemente des Repräsentantensystems und die der Gruppe zu realisieren. Die letzten beiden werden bei der im Rahmen dieser Arbeit entwickelten generischen C++ Umsetzung als sog. *Template-Parameter* dem zentralen Indexobjekt zum Zeitpunkt der Code-Übersetzung übergeben.
Zusätzlich zu der Programmierung der C++ Komponenten wurde eine auf Java-RMI basierendes „Framework“ [Rib06] implementiert, welches die Verteilung eines Suchindexes auf mehrere Rechner in einem Netzwerk ermöglicht. Auf diese Weise kann auch mit praxisrelevanten Datenmengen umgegan-

gen werden. Weiterhin ist über die RMI-Technologie eine Anbindung von Client-Programmen über das Internet an das Retrievalsystem möglich. So wurde für das Retrievalsystem für Moleküldaten ein bestehendes Visualisierungsprogramm [KN] für dreidimensionale Moleküle um Funktionen erweitert, die das Senden von Molekülinformationen an das Retrievalsystem ermöglichen und die Suchergebnisse entsprechend visualisieren. Im Falle der Audioidentifikation wird die gleiche Technologie im Rahmen des „Sync-Players“ verwendet, um ein Audioidentifikationsdienst über das Internet zu benutzen. Hierbei ist anzumerken, dass in der Regel der Merkmalsextraktor auf dem Client angesiedelt ist, um auf diese Weise lediglich die meist geringe Datenmenge der Merkmale über das Netzwerk transportieren zu müssen.

Gliederung

Die Arbeit gliedert sich wie folgt. In dem nachfolgenden Kapitel wird das Konzept der Konstellationssuche mittels G-invertierter Listen basierend auf [CK04a] kompakt vorgestellt und die wichtigsten Fehlertoleranzmechanismen sowie ein erster Suchalgorithmus zur Berechnung der Treffermenge beschrieben. In der Praxis hat sich gezeigt, dass dieser Algorithmus basierend auf dynamischer Programmierung nicht immer die beste Wahl darstellt.
Aus diesem Grund geht es im nachfolgenden Kapitel um neue Suchverfahren, die z.T. zu einer deutlichen kürzeren Antwortzeit führen können. Für angeordnete Gruppen wird ein neuer Algorithmus vorgestellt, der unter Ausnutzung der Ordnung auf den G-invertierten Listen Einträge in G-invertierten Listen unberücksichtigt lässt. Auf diese Weise reduziert sich je nach Anfrage die Anzahl der zu berechnenden Gruppenoperationen deutlich. Ein weiterer neu entwickelter Suchalgorithmus verwendet einen hierarchischen Ansatz, wobei die Treffermenge zunächst nur bzgl. einer vergröberten Auflösung berechnet wird. Die in diesem Schritt berechneten Trefferkandidaten werden dann in weiteren Schritten validiert. Ist der Speichplatzbedarf einer Realisierung der Gruppenelemente hinreichend groß, kann ein vorgestelltes Hashing-Verfahren die Suche beschleunigen, sofern eine geeignete Hashfunktion für die Gruppenelemente angegeben werden kann. Den Abschluss dieses Kapitels macht eine detaillierte Gegenüberstellung der vorgestellten Verfahren.
Kommerzielle Anwendung findet die inhaltsbasierte Suche bei Audioidentifikationsdiensten. Die detaillierte Beschreibung von kommerziellen Systemen soll zeigen, dass auch die Konstellationssuche basierend auf G-invertierten Listen zusammen mit denen in dieser Arbeit neu vorgestellten neuen Suchalgorithmen durchaus den kommerziell eingesetzten Systemen ebenbürtig, z.T. sogar überlegen ist. Ausserdem lässt sich das Konzept der Fetureextraktion sehr gut an Audiodokumenten motivieren. Es werden am Ende des Kapitels unterschiedliche Merkmalsextraktoren vorgestellt, die in unterschiedlichen Anwendungsszenarien Verwendung finden können.
Danach steht eine völlig anderer Dokumententyps im Mittelpunkt dieser Arbeit. An die Stelle von Audiosignalen treten nun dreidimensionale Strukturen von Makromolekülen (z.B. Proteine, DNS- und RNS-Sequenzen). Ziel der dort diskutierten Anwendung ist die inhaltsbasierte Suche in einer Datenbank von Molekülstrukturen. Auf der Webseite der „Protein Data Bank“, einer der weltweit wichtigsten Resourcen für Strukturdaten von biologischen Molekülen, werden derzeit nur Suchanfragen textueller Art angeboten. Die Möglichkeit eine Anfrage durch ein Molekülfragment zu formulieren ist dort derzeit noch nicht gegeben.
Ein wesentlicher Unterschied zur Audioidentifikation besteht darin, dass bei der inhaltsbasierten Suche in Moleküldaten Verschiebungen und Rotationen im dreidimensionalen Raum betrachtet werden müssen. Die Elemente der Gruppe $SE(3)$ können durch Matrizen sowie durch Quaternionen realisiert werden. Beide Ansätze werden miteinander verglichen, wobei sich die Verwendung von Quaternionen

anbietet, da deren Speicherplatzbedarf deutlich geringer ausfällt.
Einen Schwerpunkt in diesem Kapitel bildet die Entwicklung eines Merkmalsextraktors, der die gegebenen Atome eines Moleküls zu Dreiergruppen zusammenfasst, welche als Indexobjekt benutzt werden können. Der daraus resultierende Algorithmus führt auf das aus theoretischer Sicht interessante Problem der Überdeckung aller Knoten eines Graphen mittels sog. 2-Pfade. Für die spezielle Klasse von Graphen, wie sie im Anwendungsfall der Molekülsuche auftreten, wird ein neuer Algorithmus vorgestellt, der eine optimale Überdeckung berechnet. Im Hinblick auf die inhaltsbasierte Suche bedeutet hier optimal, dass eine möglichst geringe Anzahl von Dreiergruppen von Atomen bestimmt werden soll, die jedoch alle Atome einer Anfrage berücksichtigt.

Den Abschluss der Arbeit bildet die Beschreibung des implementierten Retrievalsystems basierend auf der Java-RMI Technologie und der Einbindung einer C++ Komponente, in der die eigentliche Suchfunktionalität enthalten ist.

Kapitel 2

Fehlertolerante Konstellationssuche

In diesem Kapitel soll kurz an die von Clausen et al. [CEMS00, CK04a] entwickelte Basistechnik zur inhaltsbasierten Suche in multimedialen Dokumenten erinnert werden. Diese Technik verbindet das algebraische Konzept der Operation einer Gruppe auf einer Menge mit der aus dem Textretrieval bekannten Idee von invertierten Listen und dient im weiteren Verlauf dieser Arbeit als Grundlage für effiziente Retrieval-Algorithmen.
Zunächst werden im ersten Abschnitt dieses Kapitels kurz die benötigten Begriffe und Fakten aus der Gruppentheorie zusammengestellt. Danach gehen wir auf fehlertolerante Konstellationssuche ein, diskutieren verschiedene Trefferbegriffe und erläutern all dies anhand mehrerer Beispiele. Im nächsten Kapitel wird durch geeigneten Ausbau dieser Konstellationssuche eine solide Grundlage geschaffen, um zwei ganz unterschiedliche Anwendungen in einheitlicher Weise in den Griff zu bekommen: einerseits geht es um die *robuste Audioidenti kation*, andererseits um die effiziente Suche nach angefragten Atomkonstellationen in großen *Proteinmolekülsammlungen*.

2.1 Algebraische Grundlagen

Eine *Gruppe* ist eine Menge G zusammen mit einer Abbildung $G \times G \quad (g,h) \quad g \cdot h \quad G$, die erstens assoziativ ist, d.h. für alle $g,h,k \quad G$ muss stets $(g \cdot h) \cdot k = g \cdot (h \cdot k)$ gelten, zweitens muss es ein neutrales Element 1_G geben, so dass $1_G \cdot g = g \cdot 1_G = g$ für alle $g \quad G$ gilt, und drittens muss es zu jedem Element $g \quad G$ ein inverses Element $g^{-1} \quad G$ geben, so dass $g \cdot g^{-1} = g^{-1} \cdot g = 1_G$ gilt. Eine Gruppe G heißt *abelsch* oder *kommutativ*, wenn für alle $g,h \quad G$ das Kommutativgesetz $g \cdot h = h \cdot g$ gilt. Im Falle einer abelschen Gruppe wird oft „+“ als Symbol für die Gruppenverknüpfung und 0_G oder 0 statt 1_G als Symbol für das neutrale Element verwendet. Oftmals wird das Verknüpfungssymbol ganz weggelassen und einfach gh statt $g \cdot h$ geschrieben. Eine für spätere Anwendungen wichtige Gruppe ist die sog. *symmetrische Gruppe* $\mathrm{Sym}(M)$ zu einer Menge M. $\mathrm{Sym}(M)$ besteht aus allen Permutationen von M mit der Komposition als Verknüpfung.
Eine nichtleere Teilmenge U von G heißt *Untergruppe* von G ($U \leq G$), wenn U unter Produkt- und Inversenbildung abgeschlossen ist. Das bedeutet, aus $x,y \quad U$ folgt auch $xy^{-1} \quad U$. Im Fall $U = G$ sprechen wir von einer *echten Untergruppe* ($U < G$). Sind G und H Gruppen, so heißt eine Abbildung $f\colon G \quad H$ ein *Gruppenmorphismus*, wenn f die Verknüpfung von G nach H transportiert, d.h. für $g,g' \quad G$ muss $f(gg') = f(g)f(g')$ gelten. In dem Fall ordnet f auch die neutralen Elemente einander zu: $f(1_G) = 1_H$.
Für die Basistechnik ist der Begriff der Operation einer Gruppe auf einer Menge von zentraler Bedeutung.

De nition 2.1 (Gruppenoperation auf einer Menge) *Die Gruppe G* operiert *auf einer Menge M vermöge*

$$G \times M \quad (g, m) \quad gm \quad M,$$

wenn für alle g, h G und alle m M

$$1_G m = m, \quad \textit{und} \quad (gh)m = g(hm)$$

*gilt. Dadurch wird M zu einer G-*Menge*. Im Fall einer additiven Gruppe $(G, +)$ schreibt man statt gm oft $g + m$.*

Zum Beispiel operiert die Gruppe $G = (\mathbb{Z}, +)$ auf der Menge $M = \mathbb{Z} \times \mathbb{Z}$ vermöge $g + (x, y) := (g + x, g + y)$. Diese Operation von G auf M modelliert die Verschiebung von ganzzahligen Punkten in der Ebene um ganzzahlige Werte g G.

Satz 2.1 *Die Gruppe G operiere auf der Menge M. Bei festem g G ist die Linksmultiplikation $\lambda_g : M$ M, m gm, eine Permutation von M. Weiterhin de niert die Zuordnung g $\lambda(g) := \lambda_g$ einen Gruppenmorphismus $\lambda\colon G$ Sym(M). Ist umgekehrt $\lambda\colon G$ Sym(M) ein beliebiger Gruppenmorphismus, so wird M vermöge (g, m) $\lambda(g)(m)$ zur G-Menge.*

Beweis: Wegen $(\lambda_g \circ \lambda_h)(m) = \lambda_g(\lambda_h(m)) = g(hm) = (gh)m = \lambda_{gh}(m)$ und $\lambda_{1_G}(m) = 1_G m = m$ für alle g, h G und alle m M ist λ ein Gruppenmorphismus G Sym(M). Die Umkehrung folgt analog. □

Ist M eine G-Menge, so wird vermöge ($m \sim_G m'$ gdw. ein g G existiert mit $gm = m'$) eine Äquivalenzrelation auf M definiert. Die zu m M gehörige Äquivalenzklasse ist gerade die sog. *G-Bahn*

$$Gm := \{gm \mid g \quad G\}$$

zu m. Demnach zerfällt M disjunkt in seine G-Bahnen. Ist R ein *Repräsentantensystem* der G-Bahnen, d.h. R enthält aus jeder G-Bahn genau ein Element, so gilt:

$$M = \bigcup_{m \in R} Gm.$$

Mit M G bezeichnen wir die Menge aller G-Bahnen von M.

Beispiel 2.1 Lässt man die Gruppe $G = \mathrm{SO}(2)$ aller ebenen Drehungen um den Koordinatenursprung in kanonischer Weise auf den Punkten der euklidischen Ebene operieren, so befinden sich in der G-Bahn eines Punktes m genau alle Punkte, die denselben Abstand wie m zum Ursprung haben. Demnach sind die G-Bahnen hier genau die konzentrischen Kreise um den Ursprung. Man beachte die Sonderrolle des Ursprungs: $\{0\}$ ist eine einelementige G-Bahn. Alle anderen G-Bahnen sind überabzählbar unendlich. Z.B. bildet die Menge aller reellen Zahlenpaare $(x, 0)$ mit $x \geq 0$ ein Repräsentantensystem R der G-Bahnen. Der zu $(x, y)^\top$ $\mathbb{R}^2$ gehörige Bahnenrepräsentant ist gegeben durch $(\sqrt{x^2 + y^2}, 0)^\top$, während umgekehrt durch die Drehmatrix

$$\frac{1}{\sqrt{x^2 + y^2}} \begin{pmatrix} x & -y \\ y & x \end{pmatrix}$$

der Repräsentant $(\sqrt{x^2+y^2},0)^\top$ in das Element $(x,y)^\top$ gedreht wird. Man beachte, dass die Faktorisierungsabbildung

$$\begin{pmatrix} x \\ y \end{pmatrix} \qquad \frac{1}{\sqrt{x^2+y^2}} \begin{pmatrix} x & -y \\ y & x \end{pmatrix}, \begin{pmatrix} \sqrt{x^2+y^2} \\ 0 \end{pmatrix} \qquad G \times R$$

hier leicht zu berechnen ist.

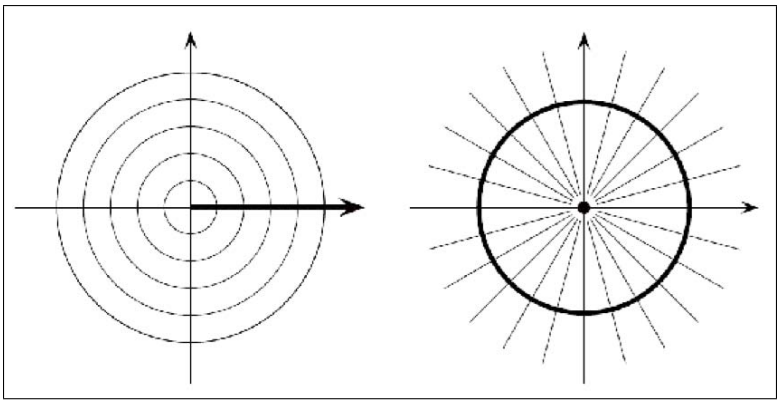


Abbildung 2.1: Die linke Zeichnung zeigt beispielhaft die Zerlegung des $\mathbb{R}^2$ in G-Bahnen. Aus diesem Bild motiviert sich auch der alternative Begriff *Orbit* für eine G-Bahn. Das Repräsentantensystem ist der hervorgehobene Bereich der x-Achse inklusive dem Nullpunkt. Die rechte Zeichnung veranschaulicht das Beispiel 2.2. Hier ist der Einheitskreis zusammen mit dem Nullpunkt als Repräsentantensystem gekennzeichnet.

Beispiel 2.2 Lässt man statt $G = \mathrm{SO}(2)$ die multiplikative Gruppe $H = \mathbb{R}_{>0}$ aller positiven reellen Zahlen via Skalarmultiplikation auf den Vektoren der euklidischen Ebene operieren, so besteht die H-Bahn eines von Null verschiedenen Vektors m gerade aus allen positiven Vielfachen dieses Vektors, stellt also einen von Null ausgehenden Strahl durch m dar (ohne die Null!). Demnach sind die H-Bahnen hier genau die vom Ursprung ausgehenden Strahlen. Man beachte auch hier die Sonderrolle des Ursprungs: $\{0\}$ ist eine einelementige G-Bahn. Alle anderen G-Bahnen sind überabzählbar unendlich. Z.B. bilden die Elemente des Einheitskreises zusammen mit dem Nullpunkt ein Repräsentantensystem R der H-Bahnen. Der zu $x \quad \mathbb{R}^2 \quad \{0\}$ gehörige Bahnenrepräsentant ist gegeben durch $x \quad x \ _2$, während umgekehrt aus dem Repräsentanten $x \quad x \ _2$ durch Multiplikation mit dem Skalar $x \ _2$ sich das Element rekonstruieren lässt. Auch hier ist die Faktorisierungsabbildung

$$0 = x \qquad (\ x \ _2 , x \quad x \ _2) \quad H \times R$$

leicht zu berechnen.

Eine Gruppe G operiert *transitiv* auf einer Menge M, wenn es nur eine einzige G-Bahn gibt, d.h. $M = Gm$ für beliebiges $m \quad M$. Ansonsten operiert G *intransitiv*. Ein wichtiges Beispiel für eine intransitive Operation ist das folgende: Es sei U eine echte Untergruppe der Gruppe G. Dann operiert die Gruppe U auf der Menge G durch Multiplikation: $U \times G \quad (u,g) \qquad ug \quad G$. Die U-Bahn von $g \quad G$ ist die sog. *Rechtsnebenklasse* $Ug = \{ug \mid u \quad U\}$. Diese Rechtsnebenklassen stehen alle in Bijektion zueinander: $U \quad u \qquad ug \quad Ug$. Ist R ein Repräsentantensystem der U-Bahnen, so gilt

also zusammen mit $G = \quad_{g\in R} Ur$ im Fall einer endlichen Gruppe G, wenn man zu den Kardinalitäten übergeht: $|G| = |R| \cdot |U|$, also $|R| := |G| \ |U| := [G : U]$. Dieser Quotient heißt der *Index* von U in G. Analog kann man U auf G operieren lassen vermöge $U \times G \quad (u, g) \quad gu^{-1}$. Die U-Bahnen sind hier die sog. *Linksnebenklassen* $gU := \{gu \mid u \quad U\}$, die alle wieder bijektiv zu U sind, so dass sich bei endlichem G eine disjunkte Zerlegung von G in $[G : U]$ viele Linksnebenklassen ergibt.
Jedes Element x einer G-Bahn Gm kann man in der Form $x = gm$ mit geeignetem $g \quad G$ schreiben. Später bei der Indexierung ist die Frage nach der Eindeutigkeit dieser Darstellung von x wichtig. Halten wir x und m fest und fragen nach der Menge aller $g \quad G$, die m nach x transportieren, dem sog. *Transporter* $\{g \quad G \mid gm = x\}$ von m nach x. Im Spezialfall $x = m$ ist dieser Transporter sogar eine Untergruppe von G, der sogenannte *Stabilisator* von m:

$$G_m := \{g \quad G \mid gm = m\}.$$

Im allgemeinen ist der Transporter von m nach x eine Linksnebenklasse des Stabilisators von m:

$$\{g \quad G \mid gm = x\} = hG_m,$$

wobei $h \quad G$ ein beliebiges Gruppenelement ist, das m nach x transportiert. Aus dieser letzten Formel ergibt sich bei endlichem G die wichtige Bahnenformel, die besagt, dass die Länge der G-Bahn zu m gleich dem Index des Stabilisators zu m ist:

$$|Gm| = [G : G_m].$$

Wir gehen kurz auf den Hintergrund dieser Aussage ein. Eine Abbildung $f\colon M \quad M'$ zwischen zwei G-Mengen heißt ein *G-Morphismus*, wenn $f(gm) = gf(m)$ für alle $g \quad G$ und alle $m \quad M$ gilt. Ist f zudem bijektiv, so liegt ein *G-Isomorphismus* vor. Ist m ein Element der G-Menge M, so definiert $gm \quad gG_m$ einen Isomorphismus zwischen den G-Mengen Gm und $G \ G_m := \{gG_m \mid g \quad G\}$. Insbesondere sind Gm und $G \ G_m$ gleichmächtig. Sind G und M beide endlich, so beschreibt das folgende Lemma die Anzahl der G-Bahnen in M.

Satz 2.2 (Lemma von Cauchy-Frobenius) *Die endliche Gruppe G operiere auf der endlichen Menge M. $M_g := \{m \quad M \mid gm = m\}$ bezeichne die Menge der Fixpunkte von $g \quad G$ in M. Dann ist die Anzahl der G-Bahnen von M gleich der mittleren Fixpunktanzahl:*

$$|M \ G| = \frac{1}{|G|} \sum_{g\in G} |M_g|. \tag{2.1}$$

Beweis: Zu $g \quad G$ und $m \quad M$ sei $\delta_{gm\,m} := 1$, falls $gm = m$ ist, ansonsten sei $\delta_{gm\,m} := 0$. Wegen

$$\begin{aligned} \frac{1}{|G|} \sum_{g\in G} |M_g| &= \frac{1}{|G|} \sum_{g\in G} \sum_{m\in M} \delta_{gm\,m} = \frac{1}{|G|} \sum_{m\in M} \sum_{g\in G} \delta_{gm\,m} = \frac{1}{|G|} \sum_{m\in M} |G_m| \\ &= \sum_{m\in M} \frac{1}{[G : G_m]} = \sum_{m\in M} \frac{1}{|Gm|} = |M \ G| \end{aligned}$$

gilt die Behauptung. □

2.2 Konstellationssuche

Unter einer *Konstellation* verstehen wir im folgenden allgemein eine beliebige (zumeist endliche) Teilmenge D einer G-Menge M. Aufgabe der G-Operation ist es, die *Äquivalenz* von Konstellationen zu spezifizieren: Teilmengen D und D' von M heißen *G-äquivalent*, wenn es ein $g \quad G$ gibt mit $gD := \{gd \mid d \quad D\} = D'$. Es folgen vier ganz unterschiedliche Beispiele für Konstellationen, die unter anderem die Reichweite dieses Konzepts zeigen sollen.

Beispiel 2.3 Wortkonstellationen: Es bezeichne W ein geeignetes Wortuniversum und $M := W \times \mathbb{Z}$. Ein Text, aufgefasst als Folge $(w_1, w_2, \ldots)$ von Wörtern, kann man dann als Teilmenge von M modellieren, indem man die jeweilige Wortposition explizit angibt und so von Folgen zu Mengen kommt, was uns erhebliche methodische Vorteile bieten wird: $\{(w_1, 1), (w_2, 2), \ldots\}$. Wenn wir nur an den relativen Positionsabständen von Wörtern in einer Wortfolge interessiert sind, modellieren wir dies durch die Operation der Gruppe $G = (\mathbb{Z}, +)$ auf der Menge M durch Positionsverschiebungen: $g+(w, i) := (w, g+i)$. Jede Teilmenge von M beschreibt dann eine Konstellation von Wörtern. Dabei können einerseits Positionslücken andererseits Wortanhäufungen an einer Position vorkommen, wie schon folgendes Beispiel zeigt:

$D =$	$(w_1, 1)$	$(w_2, 2)$	$(w_3, 3)$	$(w_4, 4)$	$(w_5, 5)$	$(w_6, 6)$	$(w_7, 7)$	$(w_8, 8)$	$(w_9, 9)$
Dokument:	Is	this	the	real	life	Is	this	just	fantasy

Q =	$(q_1, 1)$	$(q_2, 3)$	$(q_3, 4)$
Fuzzy-anfrage:	{Is}	{the, just}	{real, fantasy}

In dem obigen Beispiel wird eine sog. „Fuzzy“-Anfrage dargestellt. Dabei beschreibt die Anfrage Q im Prinzip eine Menge von Anfragen $Q_i \quad \mathbf{Q}$, welche sich aus den Alternativen ergeben:

1. $Q_1 = \{(\text{Is}, 1), (\text{the}, 3), (\text{real}, 4)\}$,
2. $Q_2 = \{(\text{Is}, 1), (\text{the}, 3), (\text{fantasy}, 4)\}$,
3. $Q_3 = \{(\text{Is}, 1), (\text{just}, 3), (\text{real}, 4)\}$ und
4. $Q_4 = \{(\text{Is}, 1), (\text{just}, 3), (\text{fantasy}, 4)\}$.

Nur für die Einzelanfragen Q_1 und Q_3 existieren Gruppenelemente $g_1 = 0$ und $g_3 = 5$, für die jeweils $g_i + Q_i \subseteq D$ gilt. Dieses Beispiel demonstriert die Flexibilität dieser Modellierung von Suchanfragen. Zum einen spielen Wortauslassungen keine Rolle (sofern dies in der Nummerierung der Anfrageworte berücksichtigt wird) und zum anderen können Alternativen von Worten an einer Position berücksichtigt werden.

Beispiel 2.4 Notenkonstellationen: $M = \mathbb{Z} \times \mathbb{Z}$ modelliere das (idealisierte) Notenuniversum zu den Parametern Einsatzzeit und Tonhöhe, $G = (\mathbb{Z}, +)$ operiert auf M durch Zeitverschiebung: $g + [t, p] := [g + t, p]$. Jede Teilmenge von M beschreibt dann eine Notenkonstellation. Sollen neben Zeitverschiebungen auch Tonhöhenverschiebungen erlaubt sein, lässt man statt G die additive Gruppe $\mathbb{Z} \times \mathbb{Z}$ auf M in kanonischer Weise operieren. An diesem Beispiel (vgl. Abbildung 2.2) sehen wir, dass man mit Variation der Gruppe zu unterschiedlichen, aufgabenspezifischen Äquivalenzbegriffen kommen kann.

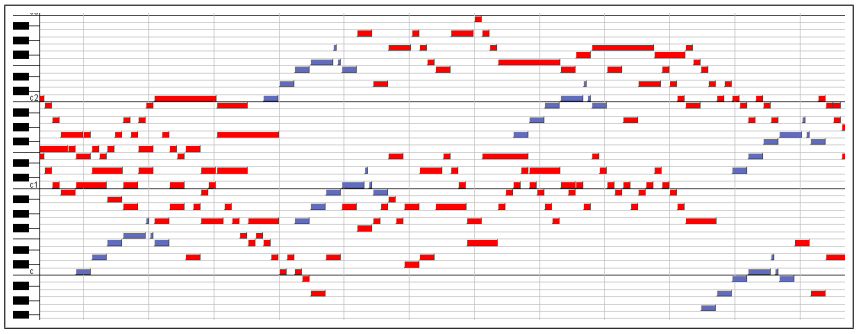


Abbildung 2.2: Beispiel Notenkonstellationen. Die Stellen im Dokument, an denen die angefragte Konstellation im Dokument enthalten ist, sind farblich hervorgehoben. In diesem Fall operiert die Gruppe $G = (\mathbb{Z}\times\mathbb{Z}, +)$ auf den Noten einer Konstellation, was zu einer zeit- und tonhöheninvarianten Suche führt.

Beispiel 2.5 Atomkonstellationen: $M = \mathbb{R}^3 \times \{\text{Elementname}\}$. Die euklidische Gruppe $G = E(3)$ besteht mengentheoretisch aus allen Paaren (A, a), wobei A SO(3) eine Drehmatrix und a $\mathbb{R}^3$ ein Translationsvektor ist. Die Verknüpfung in G ist definiert durch $(A, a)(B, b) := (AB, Ab + a)$. Diese Gruppe operiert auf M vermöge

$$(A, a)(x, \text{Elementname}) := (Ax + a, \text{Elementname}).$$

Jede Teilmenge von M beschreibt dann eine Atomkonstellation, wobei nicht alle Konstellationen chemisch sinnvoll sind. Im Detail wird auf diesen Anwendungsfall in Kapitel 6 eingegangen.

Beispiel 2.6 Menschliche Bewegungsdaten: Angenommen, die zeitlichen Veränderungen von k Körperpunkten eines Menschen, der sich in einer waagerechten xy-Ebene bewegt, werden gemessen und über ein (diskretes) Zeitintervall I aufgezeichnet. Dann wird die Bewegung der k Körperpunkte beschrieben durch ein zeitsynchrones Bündel $f\colon I$ $(\mathbb{R}^3)^k$ von k Trajektorien (vgl. Abbildung 2.3). Für t I beschreibt $f(t)$ die vom Menschen zum Zeitpunkt t eingenommene Pose. In diesem Fall kann man $M = \mathbb{R} \times (\mathbb{R}^3)^k$ wählen, wobei die erste Komponente den Zeitpunkt spezifiziert und die zweite die Pose. Geht man von $f\colon I$ $(\mathbb{R}^3)^k$ zum zugehörigen Graphen $\text{Im}(f) = \{(t, f(t)) \mid t$ $I\}$ über, so ist $\text{Im}(f)$ eine Teilmenge von M. Wollen wir zwei Bewegungen als äquivalent ansehen, wenn sie sich höchstens hinsichtlich Startzeitpunkt und Startausrichtung unterscheiden, so wird dies modelliert durch die Gruppe, die sich aus Zeittranslationen und Drehungen um die zur xy-Ebene senkrechten z-Achse zusammensetzt. Damit stellen menschliche Bewegungsdaten auch wieder Konstellationen dar.

2.2.1 Suche nach Konstellationen

Nachdem wir anhand mehrerer Beispiele Konstellationen kennengelernt haben, kommen wir nun auf das Problem der Konstellations*suche* zu sprechen. Wieder bezeichne M eine G-Menge. Bei der *exakten Konstellationssuche* geht man von einer Kollektion $\mathcal{D} = (D_1, \ldots, D_N)$ von Konstellationen $D_i \subseteq M$ aus. Weiterhin ist eine Anfrage $Q \subseteq M$ gegeben. Mit einem Gruppenelement g lässt sich nun die Anfrage Q modifizieren zu $gQ := \{gq \mid q$ $Q\}$. Ein *exakter* $(G, \mathcal{D})$*-Treffer zur Anfrage* Q wird beschrieben durch ein Paar (g, i), das aus einem Gruppenelement g und einer Dokumentennummer i $[1 : N]$ besteht, so dass die modifizierte Anfrage gQ ganz in der i-ten Konstellation D_i liegt:

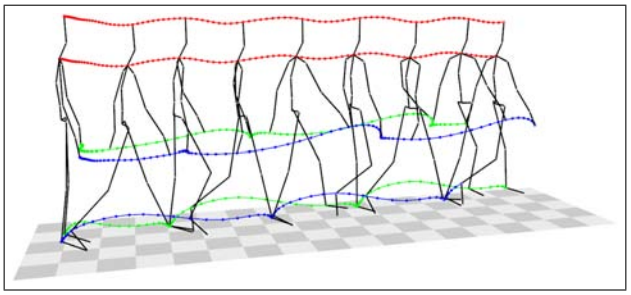


Abbildung 2.3: Unterschiedliche Bewegungsphasen einer gehenden Person. Die farbigen Linien beschreiben die Trajektorien einzelner Punkte am Körper der Person.

$gQ \subseteq D_i$. Fragt man nach allen derartigen Treffern, so wird dies durch die Menge

$$G_{\mathcal{D}}(Q) := \{(g, i) \mid gQ \subseteq D_i\}$$

aller $(G, \mathcal{D})$-*Treffer zur Anfrage* Q beschrieben.

Die Forderung $gQ \subseteq D_i$ an einen exakten $(G, \mathcal{D})$-Treffer (g, i) zur Anfrage Q ist für viele reale Retrievalszenarien zu restriktiv und zu unrealistisch. Benutzer machen bei Anfragen Fehler: das kann im Musikbereich etwa ein stellenweise falscher Rhythmus sein oder auch falsche Tonintervalle.
Eine erste Möglichkeit der *Fehlertoleranz* besteht im Aufweichen der Forderung, dass bei einem Treffer stets gQ komplett in D_i liegen muss. Diese Forderung kann man abschwächen, indem man zu einem vorgegebenen ganzzahligen Grenzwert $k \geq 0$ fordert, dass höchstens k Elemente von gQ außerhalb von D_i liegen, d.h. $|gQ \quad D_i| \leq k$. Derartige Paare (g, i) heißen $(G, \mathcal{D})$-Treffer mit bis zu k Fehlstellen. Im Grenzfall $k = 0$ erhalten wir wieder den exakten Trefferbegriff von oben. Die Gesamtheit dieser Treffer wird beschrieben durch die Menge

$$G_{\mathcal{D}\,k}(Q) := \{(g, i) \mid |gQ \quad D_i| \leq k\}.$$

Die bisher diskutierte Art von Fehlertoleranz gibt dem Benutzer keine Möglichkeit, selbst sein Wissen um mögliche Fehlerquellen oder sein unvollständiges, vages Wissen bei seiner Anfrage gewinnbringend mit einzubeziehen. Darum geht es als nächstes. Aufgrund des mengenbasierten Ansatzes besteht die Möglichkeit, dass der Benutzer Anfragepassagen, die ihm fehlerträchtig erscheinen, einfach wegzulassen (vgl. Abbildung 2.4). Dadurch entstehen erst gar keine Fehler, d.h. bei der Anfrage brauchen keine Fehlstellen zugelassen werden. Wie später noch erläutert wird, beeinflusst die Zahl der Fehlstellen massiv die Zeit, die zur Berechnung einer Treffermenge benötigt wird.
Vages Wissen um die eigentliche anzufragende Konstellation kann man mit sog. „Fuzzy"-Anfragen modellieren. Statt eine n-elementige Teilmenge Q von M anzufragen, geht man bei *Fuzzy-Anfragen* von einer Folge $F = (F_1, F_2, \ldots, F_n)$ von endlichen nichtleeren Mengen F_i von M aus. Jedes F_i stellt eine Menge von Alternativen dar. Eine solche Fuzzy-Anfrage F beschreibt ein ganzes Bündel $\mathbf{Q}(F)$ von gewöhnlichen Anfragen (vgl. Beispiel 2.3):

$$\mathbf{Q}(F) := \{\{q_1, \ldots, q_n\} \mid \quad i \quad [1 : n] : q_i \quad F_i\}.$$

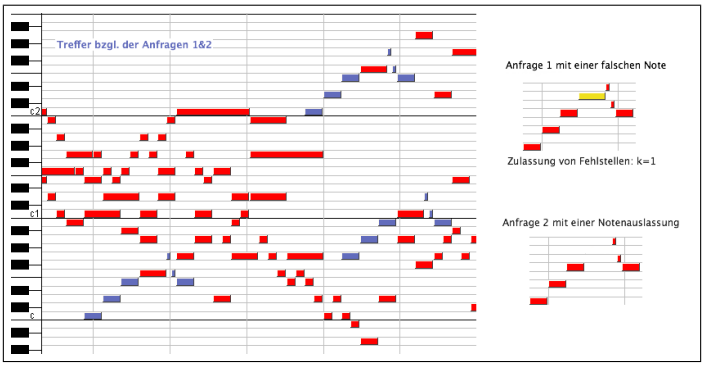


Abbildung 2.4: Beispiele für Anfragen, die Vorwissen eines Anwenders miteinbeziehen. In der ersten Anfrage ist eine Note (gelb) falsch angegeben. Durch das Erlauben einer Fehlstelle $(k = 1)$ werden dennoch die Treffer in der Partitur gefunden. Bei der zweiten Anfrage wurde eine Note ausgelassen, was ebenfalls zu den blau hervorgehobenen Trefferstellen in der Partitur führt.

Eine gewöhnliche Anfrage Q liegt bei paarweise disjunkten $F_1, \ldots, F_n$ genau dann in $\mathbf{Q}(F)$, wenn Q aus jedem F_i genau ein Element enthält. Die zur Fuzzy-Anfrage F gehörige Treffermenge ist definiert als

$$G_{\mathcal{D}}(F) := \bigcup_{Q \in \mathbf{Q}(F)} G_{\mathcal{D}}(Q).$$

In den seltensten Fällen ist es ratsam, erst alle $G_{\mathcal{D}}(Q)$ zu bestimmen und dann deren Vereinigung zu berechnen. Dies liegt schon an der möglicherweise viel zu großen Zahl von Elementen in $\mathbf{Q}(F)$: sind zum Beispiel alle F_i zweielementig und paarweise disjunkt, so hat $\mathbf{Q}(F)$ bereits 2^n viele Elemente. Die Frage, wie man hier geschickter vorgehen kann, werden wir gleich im Zusammenhang mit geeigneten Datenstrukturen und effizienten Algorithmen zur Berechnung von Treffermengen mitbehandeln. Es sollte noch gesagt werden, dass die „Fuzzy“-Suche mit dem Prinzip des Erlaubens von Fehlstellen kombiniert werden kann.

2.2.2 Berechnung der Treffermenge $G_{\mathcal{D}}(Q)$

Nachdem wir zuvor verschiedene Trefferbegriffe diskutiert haben, beginnen wir nun mit Anmerkungen zur effizienten Berechnung der Menge $G_{\mathcal{D}}(Q)$ aller exakten Treffer zur Anfrage Q. Hier schlagen Clausen et al. [CK04a] die Verwendung von sog. $(G, \mathcal{D})$-*invertierten Listen* vor, die zu einem $m \quad M$ protokollieren, welches Gruppenelement g das Element m in welches Dokument D_i transportiert:

$$G_{\mathcal{D}}(m) := \{(g, i) \mid gm \quad D_i\}.$$

Hier wird deutlich wie wichtig triviale Stabilisatoren für die Anwendbarkeit dieser Retrievaltechnik ist. Bei nicht trivialen Stabilisatoren gäbe es Mehrdeutigkeiten bzgl. des Gruppenelementes g, welches m in ein Dokument D_i transportiert. Mit Hilfe dieser Listen lassen sich alle exakten Treffer zur Anfrage Q durch Schnittbildung berechnen, wie folgendes Resultat zeigt.

Satz 2.3 *Es sei $\mathcal{D} = (D_1, \ldots, D_N)$ eine Folge von Konstellationen über der G-Menge M. Die Menge aller exakten $(G, \mathcal{D})$-Treffer zur Anfrage $Q \subseteq M$ ist dann der Durchschnitt aller $(G, \mathcal{D})$-invertierten*

Listen zu sämtlichen Elementen von Q:

$$G_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G_{\mathcal{D}}(q).$$

Beweis: (g,i) $G_{\mathcal{D}}(Q)$ gdw. $gQ \subseteq D_i$. D.h. für alle q Q liegt gq in D_i, was bedeutet, dass (g,i) im Durchschnitt aller $G_{\mathcal{D}}(q)$ liegt, wobei q alle Elemente aus Q durchläuft. □

Es sei ausdrücklich vermerkt, dass die invertierten Listen an sich unabhängig von einer Anfrage sind. Man kann diese Listen also vorab berechnen und abspeichern. Die konkrete Anfrage Q zeigt lediglich, welche Listen man heranzuziehen hat. Zur inhaltsbasierten Suche reicht es also aus, alle $(G,\mathcal{D})$-invertierten Listen auszurechnen und abzuspeichern. Dabei stoßen wir aber auf ein seriöses Problem: Wie die obigen Beispiele von Konstellationen gezeigt haben, ist die Grundmenge M meist unendlich. Wollten wir für alle Elemente m von M die Listen $G_{\mathcal{D}}(m)$ berechnen und abspeichern, so bräuchten wir sowohl unendliche Rechenzeit als auch unendlichen Speicher! In der Praxis würde dieses Problem eher dazu führen, dass in vielen Listen nur sehr wenige Einträge vorhanden wären. Somit stünde der Aufwand zur Verwaltung von G-invertierten Listen in keinem Verhältnis zum Nutzen. Zum Glück sind die Listen, wie folgendes fundamentale Resultat zeigt, nicht völlig unabhängig voneinander.

Satz 2.4 *Mit den Bezeichnungen des letzten Satzes gilt für g G und m M:*

$$G_{\mathcal{D}}(gm) = G_{\mathcal{D}}(m)g^{-1} := \{(hg^{-1}, i) \mid (h,i) \quad G_{\mathcal{D}}(m)\}.$$

Beweis: Wegen (h,i) $G_{\mathcal{D}}(m)$ hm D_i $(hg^{-1})(gm)$ D_i (hg^{-1},i) $G_{\mathcal{D}}(gm)$ gilt die Behauptung. □

Es besteht also eine sehr enge Beziehung zwischen den invertierten Listen zu den Elementen einer G-Bahn. Ist R ein Repräsentantensystem der G-Bahnen von M, so reicht es aus, die Listenfamilie $(G_{\mathcal{D}}(r))_{r \in R}$ abzuspeichern. Was ist damit wirklich erreicht? In der Praxis werden alle Konstellationen $D_1, \ldots, D_N$ endlich sein. Bezeichnet D deren Vereinigung, so ist auch D endlich, womit auch nur endlich viele G-Bahnen insgesamt involviert sein können. Selbst wenn M unendlich viele G-Bahnen enthält, können doch nur endlich viele der Listen in $(G_{\mathcal{D}}(r))_{r \in R}$ nichtleer sein. Natürlich werden nur diese abgespeichert. Die Anzahl der abzuspeichernden Listen ist also endlich.
Als nächstes wenden wir uns der Frage zu, wie lang diese Listen sind. Wir betrachten zunächst den Spezialfall, dass für alle m M der Stabilisator G_m trivial ist, also $G_m = \{1_G\}$ gilt. In diesem Fall hat die Liste zum Repräsentanten r R die Länge $\sum_{i=1}^{N} |D_i \setminus Gr|$. Die Gesamtlänge aller nichtleeren Listen ist also gleich $\sum_{i=1}^{N} |D_i|$. Mit anderen Worten benötigt die gesamte Indexierung einen Speicherbedarf, der linear in der Gesamtgröße aller Konstellationen in $\mathcal{D}$ ist. Nun nehmen wir an, dass der Stabilisator des Repräsentanten r R nichttrivial ist. Mit (g,i) liegen dann auch alle (gh,i) in $G_{\mathcal{D}}(r)$, wenn h alle Elemente des Stabilisators G_r durchläuft. Die invertierte Liste besteht also im wesentlichen aus $\sum_{i=1}^{N} |D_i \setminus Gr|$ vielen Nebenklassen des Stabilisators G_r, enthält also $(\sum_{i=1}^{N} |D_i \setminus Gr|) \cdot |G_r|$ viele Einträge. Da der Stabilisator unendlich sein kann, muss man sich notfalls nach anderen Beschreibungen dieser invertierten Listen umsehen. Hier bietet es sich an, pro Nebenklasse gG_r nur ein Paar der Form (g,i) abzuspeichern und global zudem eine endliche Beschreibung des Stabilisators G_r, sofern eine solche existiert. Eine andere Möglichkeit ist der Übergang zu anderen, mit M nahe verwandten G-Mengen wie M^2, M^3 oder 2^M, wo das Problem großer Stabilisatoren möglicherweise nicht mehr auftritt.

Beispiel 2.7 Sei $M = \mathbb{R}^2$ die Menge aller Punkte in der reellen Ebene. Die Gruppe $G = E(2)$ der Euklidischen Bewegungen in der Ebene operiere in diesem Beispiel auf den Elementen von M. In diesem Fall ist der Stabilisator G_m zu einem $m \quad M$ isomorph zur Rotationsgruppe $SO(2)$.
Der Übergang zu verschiedenen Paaren von Elementen von M führt auf endliche Stabilisatoren, wenn G auf $M_2 := M^2 \ \{(x, x) \mid x \quad M\}$ operiert. Denn das Element $(x, y) \quad M_2$ wird neben der Identität nur noch von der $180°$-Drehung um den Mittelpunkt von x und y festgelassen.

Diese Betrachtungen spielen bei der Suche in Proteinmolekülen, wie sie in Kapitel 6 vorgestellt, ebenfalls eine entscheidende Rolle bei der Modellierung der Menge M. Da in diesem Fall Rotationen und Translationen im $\mathbb{R}^3$ erlaubt sind, werden Atome zu Dreiergruppen zusammengefasst, um zu gewährleisten, dass die Stabilisatoren endlich sind.

Wir fassen die Schritte zusammen, die zur Berechnung aller exakten $(G, \mathcal{D})$-Treffer zur Anfrage Q führen: Zu jedem $q \quad Q$ hat man den zu q gehörigen Bahnenrepräsentanten r_q zu bestimmen sowie ein geeignetes Gruppenelement g_q, das den Repräsentanten r_q nach q transportiert: $q = g_q r_q$. Die Liste zu r_q hat man dann mittels g_q nachzujustieren um schließlich per Durchschnittsbildung die Menge aller exakten $(G, \mathcal{D})$-Treffer zu erhalten:

$$G_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G_{\mathcal{D}}(r_q) g_q^{-1}.$$

Diese Formel, die sich unmittelbar aus den beiden letzten Sätzen ergibt, ist die Grundlage für effiziente Konstellationssuche. Handelt es sich bei G um eine abelsche Gruppe mit additiv geschriebener Verknüpfung, so liest sich die letzte Formel entsprechend so:

$$G_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G_{\mathcal{D}}(r_q) - g_q.$$

Wie die beiden Gleichungen zeigen, ist das Hauptproblem die schnelle Durchschnittsbildung für nachzujustierende $(G, \mathcal{D})$-invertierte Listen. Zunächst einmal handelt es sich bei den invertierten Listen eigentlich um Mengen. Die Durchschnittsbildung ist aber einfacher zu realisieren, wenn man zu *sortierten Listen* übergeht, denn sortierte Listen lassen sich schnell schneiden und zwar durch eine Modifikation von Mergesort in einer Zeit, die linear in der Gesamtlänge beider Listen ist, oder bei wahlfreiem Zugriff per Binärsuche der Elemente der kürzeren Liste der Länge $\ _1$ in der längeren Liste der Länge $\ _2$ in einer Zeit $O(\ _1 \log\ _2)$.
Hier treten zwei Schwierigkeiten auf. Die Elemente der zu sortierenden Listen sind Paare (g, i). Zunächst einmal muss man auf derartigen Paaren eine Totalordnung einführen, was in konkreten Anwendungsbeispielen oft relativ unproblematisch ist. Im folgenden bezeichnen wir auch eine zu $G_{\mathcal{D}}(r)$ gebildete sortierte Liste ebenfalls mit $G_{\mathcal{D}}(r)$. Missverständnisse sind nicht zu befürchten. Die zweite Schwierigkeit ist seriöser: Selbst wenn die abgespeicherten Listen $G_{\mathcal{D}}(r) = ((g_{r1}, i_{r1}) < \ldots < (g_{rn_r}, i_{rn_r}))$ sortiert sind, kann die notwendige Nachjustierung mit einem Gruppenelement g^{-1} aus der sortierten Liste eine unsortierte Liste machen. In diesem Zusammenhang hilft dann folgendes Resultat aus [CK04a] weiter:

Lemma 2.1 *Sind a, b Elemente und A, B Teilmengen der multiplikativen Gruppe G, so gilt für eine Mengenoperation $\{\setminus, \ , \ \}$ die Formel*

$$Aa \quad Bb = (Aab^{-1} \quad B)b = (A \quad Bba^{-1})a.$$

Bei der Durchschnittsbildung kann man auf der Basis dieser Formel immer erreichen, dass zumindest die Sortierung von einer Liste ausgenutzt werden kann. Hier ein kurzes Beispiel:

$$H_1 g_1^{-1} \setminus H_2 g_2^{-1} \setminus H_3 g_3^{-1} = ((H_1 g_1^{-1} g_2 \setminus H_2) g_2^{-1} g_3 \setminus H_3) g_3^{-1}.$$

Bei dieser Art der Schnittmengenberechnung von drei Liste ist immer eine der beteiligten Listen geordnet, was die Suche von Elementen der ungeordneten Liste in der geordneten Liste mittels Binärsuche ermöglicht. Anzumerken ist jedoch, dass dieses Verfahren eine Implementierung von G-Listen voraussetzt, die die Binärsuche unterstützt.

2.2.3 Berechnung von $G_{\mathcal{D}\,k}(Q)$ mittels dynamischer Programmierung

Nun wenden wir uns dem Problem zu, die Menge $G_{\mathcal{D}\,k}(Q)$ aller Treffer mit bis zu k Fehlstellen effizient zu bestimmen. Clausen et al. [CK04a] schlagen vor, $G_{\mathcal{D}\,k}(Q)$ mittels dynamischer Programmierung wie folgt zu berechnen. Es sei $Q = \{q_1, \ldots, q_n\}$ und $G_j := G_{\mathcal{D}}(q_j)$ bezeichne die zu q_j gehörige $(G, \mathcal{D})$-invertierte Liste. Weiterhin bezeichne $\Gamma_j := G_1 \quad \ldots \quad G_j$ die Vereinigung der ersten j Listen. Induktiv werden nun Kreditfunktionen $C_j \colon \Gamma_j \quad \mathbb{Z}$ wie folgt definiert. $\Gamma_1(\gamma) := k + 1$, für alle $\gamma \quad \Gamma_1$. Für $j \quad [2 : n]$ setze

$$C_j(\gamma) := \begin{array}{ll} C_{j-1}(\gamma) & \text{falls } \gamma \quad \Gamma_{j-1} \setminus G_j \\ C_{j-1}(\gamma) - 1 & \text{falls } \gamma \quad \Gamma_{j-1} \quad G_j \\ k + 2 - j & \text{falls } \gamma \quad G_j \quad \Gamma_{j-1}. \end{array}$$

Dass man über diese Kreditfunktionen alle Treffer mit bis zu k Fehlstellen berechnen kann, zeigt folgendes Resultat aus [CK04a]:

Satz 2.5 *Die Elemente von Γ_n mit positivem Kredit sind genau die $(G, \mathcal{D})$-Treffer mit höchstens k Fehlstellen:*

$$G_{\mathcal{D}\,k}(Q) = \{\gamma \quad \Gamma_n \mid C_n(\gamma) > 0\}.$$

Beweis: Es sei $(g, i) \quad \Gamma_j$ und $Q_j := \{q_1, \ldots, q_j\}$. Durch Induktion nach j zeigen wir, dass $C_j(g, i) = k + 1 - |gQ_j \quad D_i|$. Der Start $j = 1$ ist klar, da nach Definition (g, i) in $G_{\mathcal{D}}(q_1)$ liegt, also $gq_1 \quad D_i$ gilt. Beim Induktionschritt $(j - 1 \quad j)$ unterscheiden wir drei Fälle.
Fall 1: $\gamma \quad \Gamma_{j-1} \setminus G_j$. Wegen $gq_j \quad D_i$ gilt $gQ_j \quad D_i = gQ_{j-1} \quad D_i$. Also folgt $C_j(g, i) := C_{j-1}(g, i) = k + 1 - |gQ_{j-1} \quad D_i| = k + 1 - |gQ_j \quad D_i|$.
Fall 2: $(g, i) \quad \Gamma_{j-1} \quad G_j$. Wegen $gq_j \quad D_i$ gilt $|gQ_j \quad D_i| = |gQ_{j-1} \quad D_i| + 1$. Also folgt $C_j(g, i) := C_{j-1}(g, i) - 1 = k + 1 - (|gQ_{j-1} \quad D_i| + 1) = k + 1 - |gQ_j \quad D_i|$.
Fall 3: $(g, i) \quad G_j \quad \Gamma_{j-1}$. Dann ist $gq_j \quad D_i$ aber für alle $[1 : j - 1]$ liegt gq nicht in D_i. Also ist $|gQ_j \quad D_i| = j - 1$. Somit ist $C_j(g, i) := k + 2 - j = k + 1 - (j - 1) = k + 1 - |gQ_j \quad D_i|$.
Da $Q_n = Q$ ist, erhalten wir $C_n(g, i) = k+1-|gQ \quad D_i|$ für alle $(g, i) \quad \Gamma_n$. Folglich ist $C_n(g, i) > 0$ gdw. $|gQ \quad D_i| \leq k$, d.h. $(g, i) \quad G_{\mathcal{D}\,k}(Q)$. □

2.2.4 Ef ziente Berechnung von Fuzzy-Anfragen

Wenden wir uns nun der Berechnung aller $(G, \mathcal{D})$-Treffer zur Fuzzy-Anfrage $F = (F_1, \ldots, F_n)$ zu. Wie schon erwähnt haben wir bei paarweise disjunkten Alternativmengen F_i insgesamt $\sum_{i=1}^{n} |F_i|$ viele gewöhnliche Anfragen simultan zu beantworten. Der folgende Satz zeigt, dass man die Menge $G_{\mathcal{D}}(F) := \bigcup_{Q \in \mathbf{Q}(F)} G_{\mathcal{D}}(Q)$ aller $(G, \mathcal{D})$-Treffer zur Fuzzy-Anfrage F viel effizienter als durch direkte algorithmische Umsetzung dieser definierenden Formel lösen kann.

Satz 2.6 *Für die Gesamtheit $G_{\mathcal{D}}(F)$ aller $(G,\mathcal{D})$-Treffer zur Fuzzy-Anfrage $F=(F_1,\ldots,F_n)$ gilt:*

$$G_{\mathcal{D}}(F)=\bigcap_{j=1}^{n}\Big(\bigcup_{q\in F_j} G_{\mathcal{D}}(q)\Big)=\bigcap_{j=1}^{n}\Big(\bigcup_{q\in F_j} G_{\mathcal{D}}(r_q)g_q^{-1}\Big).$$

Beweis: Es ist (g,i) $G_{\mathcal{D}}(F)$ gdw. $gQ\subseteq D_i$, für ein Q $\mathbf{Q}(F)$. Das bedeutet, es gibt zu jedem j $[1:n]$ ein Element q_j F_j mit gq_j D_i. Mit anderen Worten: für alle j liegt (g,i) in $\bigcup_{q_j\in F_j} G_{\mathcal{D}}(q_j)$. Das beweist die Behauptung. □

Beim Fuzzy-Retrieval sind also pro Alternativmenge F_j die $(G,\mathcal{D})$-invertierten Listen zu den Elementen von F_j zu vereinigen, bevor man dann die n vereinigten Listen schneidet. Diese Vorgehensweise macht die Fuzzy-Suche erst praktikabel.

2.2.5 Zusammenfassung

In diesem Kapitel wurde die Grundidee zur inhaltsbasierten Suche in unterschiedlichen Arten von Dokumenten unter Verwendung des algebraischen Prinzips der Operation einer Gruppe auf einer Menge vorgestellt, wie sie zum Beispiel in [CEMS00, KC01, CK04a, CKK03, CK04b] veröffentlicht ist. Ein wichtiger Aspekt dabei ist die Forderung nach endlichen Stabilisatoren der Gruppenoperation, um Mehrdeutigkeiten zu vermeiden. Weiterhin wurden zwei grundlegende Arten von Fehlertoleranzmechanismen vorgestellt. Zum einen das Prinzip von Fehlstellen und zum anderen das Prinzip der Fuzzy-Anfragen. Natürlich können beide Prinzipien miteinander verbunden werden. Ausserdem wurde ein Algorithmus vorgestellt, der unter Verwendung des Prinzips der dynamischen Programmierung, eine effiziente Berechnung von Treffermengen ermöglicht und dabei die Fehlertoleranzmechanismen berücksichtigt.
In den nachfolgenden Kapitel wird es insbesondere um andere Ansätze der Berechnung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ gehen. Es werden im Rahmen dieser Arbeit entwickelte Algorithmen und Datenstrukturen vorgestellt, die im Vergleich zu dem Algorithmus, der in diesem Kapitel vorgestellt worden ist, eine zum Teil deutliche schnellere Berechnung der Treffermenge zu einer Anfrage Q ermöglichen. Einige der Algorithmen zeichnen sich auch dadurch aus, dass sie zur Laufzeit einen nur sehr geringen Speicherplatzbedarf haben. Insbesondere bei einer großen Anzahl von Einträgen in den G-invertierten Listen kann dies erst einer Beantwortung der Anfrage ermöglichen, da nämlich der in diesem Kapitel vorgestellte Algorithmus z.B. bei k-Fehlstellen-Anfragen zur Laufzeit im schlimmsten Fall die Elemente von k G-invertierten Listen im Speicher vorhalten muss. Natürlich könnte eine Auslagerung auf den Hintergrundspeicher ins Auge gefasst werden, jedoch würde dies unweigerlich zu einer erheblichen Verzögerung bei der Bearbeitung einer Anfrage führen.

Kapitel 3

Alternative Ansätze und Algorithmen zur fehlertoleranten Trefferberechnung

Im vorhergehenden Kapitel wurde ein Algorithmus zur Trefferberechnung vorgestellt, der auf der Berechnung von Schnittmengen von justierten G-invertierten Listen beruht. In diesem Kapitel werden alternative Ansätze und Algorithmen zur effizienten Berechnung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ zu einer Anfrage Q diskutiert und verglichen. Dabei wird sich zeigen, dass es unter den vorgeschlagenen Methoden keine gibt, die in allen Anwendungsfällen den anderen Verfahren überlegen ist. Entscheidenden Einfluss auf die Wahl der jeweiligen Methode haben die Komplexität der Verknüpfung zweier Gruppenelemente, die Möglichkeit oder Unmöglichkeit eine mit der Gruppenoperation verträgliche Totalordnung auf G zu definieren, der benötigte Speicherplatz pro Gruppenelement, die Größe der Dokumentenkollektion, die mittlere Listenlänge, die Länge der Anfrage und die Anzahl von zugelassenen Fehlstellen.

Die zu lösende Fragestellung wurde im vorherigen Grundlagenkapitel bereits angedeutet. Gegeben sei eine Anfrage Q bestehend aus n Elementen der G-Menge M. Jedem q Q können wir ein Element r_q aus dem Repräsentantensystem R zuordnen. Zu eben jedem r_q wird in dem Suchindex eine Liste von Paaren (g, i) gespeichert, wobei $g \cdot r_q$ D_i gilt, d.h. die Einträge (g, i) spiegeln den Inhalt des i-ten Dokumentes normiert bzgl. eines Repräsentantensystems R wider. Da jedes Element aus einer Liste zum Repräsentanten r_q mit g_q^{-1} justiert werden muss, können wir im folgenden davon ausgehen, dass wir n viele bzgl. Q justierte G-invertierte Listen $G_1, \ldots, G_n$ vorliegen haben. Es gilt nun Elemente (g, i) in den *justierten* Listen zu finden, die in mindestens $n - k$ vielen Listen enthalten sind. Aufgrund der Justierung einer Liste bzgl. den Elementen einer Anfrage Q, macht es Sinn, von n Listen zu sprechen, auch wenn diese ggf. aus ein und derselben unjustierten Version einer G-invertierten Liste hervorgehen. In diesem Kapitel werden, falls es sich aus dem Kontext nicht erschliessen lässt, unjustierte G-invertierte Listen mit G'_i bezeichnet. Im Falle einer Fuzzy-Anfrage fassen wir gemäß den Aussagen im vorherigen Kapitel die Listen zu einem Fuzzy-Element als eine einzige Liste auf, die wir z.B. durch Vereinigung berechnen können – dies aber im folgenden aus Effizienzgründen vermeiden wollen.

Bei einigen der vorgestellten Algorithmen werden Trefferkandidaten sequentiell betrachtet und ggf. wieder verworfen, bevor der nächste Kandidat betrachtet wird. Dies wird für die Elemente von $k + 1$ beliebigen Listen durchgeführt und danach abgebrochen, denn kein Trefferkandidat aus der $(k + 2)$-ten Liste kann die geforderte Vielfachheit von $n - k$ erreichen. Dies ist ein großer Vorteil dieser Algorithmen, da die beliebige Wahl der Listen es ermöglicht, zunächst die $k + 1$ *kürzesten* Listen zu betrachten und somit die Zahl der durchzuführenden Operationen zu reduzieren.

Die Algorithmen, welche zu einem Trefferkandidaten (g, i) dessen Vielfachheit in Form eines Zählers c, also (g, i, c), speichern, beinhalten den Nachteil, dass die Elemente der G-invertierten Listen in eine neue Datenstruktur kopiert werden müssen, die eben gerade den zusätzlichen Zähler beinhaltet. Diese Kopieraktionen sind bei millionenfacher Ausführung ein wichtiger Aspekt, wenn es um die Laufzeit eines Suchalgorithmus geht. Denn der zusätzliche Zähler muss im Falle von Objektkopien ebenfalls kopiert werden.
Nach Möglichkeit sollten die Suchalgorithmen ohne dynamische Datenstrukturen wie R^*-Bäume etc. auskommen. Auch sind Einfüge- und Löschoperationen in Vektoren nach Möglichkeit zu vermeiden. Denn gerade diese Operationen können bei millionenfacher Ausführung die Suchzeit erheblich beeinflussen. Als Bespiel sei hier der Algorithmus 2.2.3 aus dem vorigen Kapitel genannt. Während der Aufbauphase, d.h. während die ersten $k + 1$ Listen unter der Berücksichtigung der Kreditfunktion Γ vereinigt werden, fallen viele Einfügeoperationen in die temporäre Trefferkandidatenliste an. Da bei der späteren Schnittmengenberechnung Binärsuche verwendet wird, wurde bei der Implementierung in [Wag03, RK02] Vektoren als Container für die Elemente von G-invertierten Listen verwendet, da nur diese wahlfreien Zugriff auf die Elemente ermöglichen. Eben die Einfügeoperationen sind verantwortlich für die in [Wag03, RK02] angegebenen Suchzeiten und als Konsequenz daraus das komplizierte Bewertungssystem für die Ergebnisse von Teilanfragen, da diese in einigermassen praxisrelevanten Zeit von dem System beantwortet werden konnten.
Ein Beispiel mag dies verdeutlichen. Gehen wir davon aus, dass wir $k + 1$ viele Listen zunächst unter Berücksichtigung von Γ zu einer temporären Liste von Trefferkandidaten vereinigen müssen. Jede der Listen enthalte 250.000 Elemente. Wenn wir von nur einem Treffer bei der Anfrage ausgehen, können wir annehmen, dass die temporäre Liste nach der Aufbauphase nur knapp weniger als $(k+1) \cdot 250.000$ Einträge enthält. Bei nur 10 erlaubten Fehlstellen erhalten wir nach der Aufbauphase eine Liste mit ca. 2.5 Millionen Elementen bei rund 2.25 Millionen Einfügeoprationen. Diese Liste wird mit der Bearbeitung der nachfolgenden Listen schnell auf den einen verbleibenden Trefferkandidaten zusammenfallen. Somit wird in der Praxis mit viel Aufwand eine temporäre Liste erstellt, von denen bei sinnvollen Anfragen mindestens 95% nach wenigen Schritten nach der Aufbauphase wegfallen.
Ein Sonderfall stellt eine exakte Anfrage, d.h. $k = 0$, dar. In einem solchen Fall muss ein Treffer (g, i) in *allen* justierten G-invertierten Listen, insbesondere auch in der kürzesten zu betrachtenden Liste G_1 vorkommen. Dies bedeutet, dass in G_1 bereits alle Trefferkandidaten aufgeführt sind. Nun kann man eine Abbildung $H\colon G \times [0 : N - 1] \qquad [0 : |G_1| - 1]$ definieren, die jedem Element in G_1 seine Position in der Liste zuordnet. Bei dem Durchlaufen der verbleibenden $n - 1$ vielen Listen muss für jedes justierte Element geprüft werden, ob es in G_1 vorhanden ist. Wenn nicht, kann es zu keinem Treffer gehören und kann vernachlässigt werden. Andernfalls lässt sich über $H(g, i)$ ein Array indizieren, welches die Vielfachheiten der Trefferkandidaten enthält. Dies stellt sicherlich den Idealfall einer Anfrage dar, der in der Praxis nur eine untergeordnete Rolle spielen sollte.
Betrachten wir im Moment nur das Gruppenelement als Treffer und vernachlässigen die Dokumenten-ID, so schränkt die Dokumentenkollektion $\mathcal{D}$ den Wertebereich von $g \quad G$ ein. Nehmen wir an, wir suchen einen Ausschnitt Q aus einem Audiosignal in einer Datenbank von Audiosignalen. Ferner beginne Q beim Zeitpunkt 0. Somit kommen nur Verschiebungen $t \geq 0$ in Frage, um Q auf der Zeitachse derart zu verschieben, dass es mit einem Teil eines Signals aus der Datenbank übereinstimmt. Weiterhin ist die maximale Verschiebung, die zu einem Treffer führen kann, in dem Beispiel durch die Länge des längsten Signals in der Datenbank gegeben. Beide Schranken können während einer Suche dazu benutzt werden, Paare (g, i), die aus justierten G-invertierten Listen hervorgegangen sind, vor einer weiteren Verarbeitung ausgeschlossen werden.
Bis auf die in Abschnitt 3.1 diskutierte Art der Trefferberechnung mittels relationalen Datenbanksystemen, folgen die anderen präsentierten Verfahren dem folgenden prinzipiell Aufbau:

Initialisierung von Datenstrukturen Vor dem eigentlichen Beginn der Suche sind bei den meisten Algorithmen Datenstrukturen zu initialisieren. Bei aufwändigerer Initialisierung bietet sich das Design Pattern der „Factory“ (siehe [GHRV94]) an, das über Referenzen Zugriff auf bereits initialisierte Datenstrukturen bietet. Dieses Modul sollte die eigentliche Initialisierung in einem eigenen Thread durchführen, was sich besonders bei Mehrprozessor-Systemem anbietet.

Aufbauphase Als *Aufbauphase* wird die Bearbeitung der ersten $k+1$ G-invertierten, i.d.R. auch kürzesten Listen bezeichnet. Denn potentielle Trefferkandidaten müssen in mindestens einer der $k+1$ justierten Listen $G_1, \ldots, G_{k+1}$ vorhanden sein, um die geforderte Vielfachheit erreichen zu können. Für die Algorithmen bedeutet dies, dass während dieser Phase Trefferkandidaten in ggf. dynamische Datenstrukturen eingefügt werden müssen. Zum Beispiel werden bei der Trefferberechnung mittels der Kostenfunktion Γ die ersten $k+1$ Listen bzgl. der Definition von Γ vereinigt. Bei einer hohen Anzahl von zugelassenen Fehlstellen führt dies schnell zu einer sehr langen Liste von Trefferkandidaten, was die Aufbauphase sehr verlangsamt.

Weitere Listenbearbeitung Dieser Abschnitt der Suche ist eng verwandt mit der Aufbauphase und unterscheidet sich bei den meisten Algorithmen nur durch die Tatsache, dass keine neuen Trefferkandidaten mehr den Datenstrukturen hinzugefügt werden müssen, sondern lediglich die Vielfachheitenzähler bereits vorliegender Trefferkandidaten modifiziert werden. Bei einigen Verfahren werden auch Trefferkandidaten verworfen, so dass sich die Zahl der zu prüfenden Trefferkandidaten während der Suche reduziert.

Aufbau der endgültigen Trefferliste Dieser Schritt ist nicht bei allen Algorithmen notwendig, da bereits während der Suche die Vielfachheit eines Trefferkandidaten bestimmt worden ist (z.B. Heap-Suche). Bei anderen Algorithmen wird ein Trefferkandidat der Resultatliste hinzugefügt, sobald er den Trefferschwellwert erreicht hat. Jedoch steht hier die korrekte Vielfachheit eines Treffers erst nach der Abarbeitung aller Listen fest, so dass am Ende der eigentlichen Suche für jeden Treffer die Vielfachheit aktualisiert werden muss.

Destruktion von Datenstrukturen Dieser Teil ist eng verknüpft mit der Initialisierung der Datenstruktur. Auch hier gilt, dass die ggf. aufwändige Destruktion bzw. Re-Initialisierung in einem eigenen Thread erfolgen sollte.

Zu Beginn widmen wir uns der Umsetzung des Suchverfahrens aus dem letzten Kapitel mittels relationalen Datenbanksystemen. Danach werden zwei neue Verfahren zur Berechnung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ zu einer Anfrage Q vorgestellt, die darauf bauen, dass die Elemente der G-invertierten Liste auch nach erfolgter Justierung nach wie vor geordnet sind. Daraufhin folgt die Beschreibung dreier Verfahren, die ohne diese Voraussetzung auskommen. Bevor diese Verfahren in einem praktischen Anwendungsfall miteinander verglichen werden, werden Möglichkeiten diskutiert, wie die vorgestellten Verfahren erweitert werden können, um auch Fuzzy-Anfragen effizient zu beantworten.

3.1 Audioidenti kation mittels relationaler Datenbanken

Bisher wurde ein Verfahren zur Treffermengenbestimmung vorgestellt, welches speziell auf den Umgang mit G-invertierten Listen ausgerichtet ist, indem es Schnittmengen von G-invertierten Listen berechnet. In diesem Abschnitt soll es am Beispiel der Audioidentifkation darum gehen, wie sich ein solches System auch mit Hilfe von relationalen Datenbanken umsetzen lässt. Wir gehen aus von einer Relation $\mathcal{R} = (\texttt{list\_id}, \texttt{g\_elem}, \texttt{doc\_id})$. Weiterhin nehmen wir an, dass die Umsetzung

der Relation $\mathcal{R}$ in einer Tabelle derartig erfolgt, dass eine virtuelle Unterteilung der Relation nach dem Attribut `list_id` erfolgt, d.h. bei Einschränkung einer Abfrage bzgl. `list_id` muss nicht jedes Tupel von $\mathcal{R}$ betrachtet werden, um die entsprechenden Tupel auszuwählen. Vielmehr liefert eine Index-Datenstruktur alle Tupel von $\mathcal{R}$, die die geforderte `list_id` aufweisen. Auf diese Weise kann man also davon ausgehen, dass die Einträge einer G-invertierten Liste prinzipiell in einer eigenen Relation abgelegt sind. Die Tupel der Relation $\mathcal{R}$ seien in der Tabelle `tb_glists` abgelegt.

Sei nun $Q = \{q_0, \ldots, q_{n-1}\} \subset M$ eine endliche Anfrage modelliert durch eine Teilmenge der G-Menge $M = \mathbb{Z} \times [0 : C-1]$, wobei C der maximale Klassenindex ist. Zu jedem Element $q_i \quad Q$ bezeichnen wir die korrespondierende G-invertierte Liste im Suchindex mit G_i. O.B.d.A. sei stets $|G_i| \leq |G_{i+1}|$.

Zunächst definieren wir die zu einem Element $q_i = (g_i, m_i) \quad Q$ zugehörige Relation $\mathcal{G}_i$, deren Tupel durch die folgende SQL-Abfrage berechnet werden:

$\mathcal{G}_i =$
```
SELECT
        g_elem ·g_i^-1  AS g_elem,
        doc_id          AS doc_id,
        1               AS score
FROM
        tb_glists
WHERE
        list_id = m_i;
```

Das Attribut `g_elem` der Relation $\mathcal{G}_i$ bezeichnet das bzgl. Q justierte Gruppenelement und das Attribut `score` die Vielfachheit des Tupels (`g_elem`, `doc_id`) bezogen auf $\mathcal{G}_i$. Da in einer G-invertierten Liste keine zwei gleichen Elemente vorliegen, ist die Vielfachheit für alle Tupel in $\mathcal{G}_i$ gleich 1. Allgemein gilt, die Anzahl von Tupeln in $\mathcal{G}_i$ ist kleiner oder gleich der Anzahl von Tupeln in $\mathcal{G}_{i+1}$.

3.1.1 Exakte Anfragen

Im Falle von exakten Anfragen enthält die Relation $\mathcal{G}_0$ bereits alle möglichen Trefferkandidaten. In diesem Fall werden zunächst $\mathcal{G}_0$ und $\mathcal{G}_1$ berechnet und dann in einer temporären Resultatrelation $\mathcal{T}_1 = \mathcal{G}_0 \setminus \mathcal{G}_1$ bestimmt. Die Relation $\mathcal{T}_1$ enthält danach maximal so viele Tupel wie $\mathcal{G}_0$, in der Praxis jedoch bereits meist viel weniger Trefferkandidaten. In nachfolgenden Schritten wird für alle $2 \leq i < n$ die Schnittmenge $\mathcal{T}_i$ der Relationen $\mathcal{T}_{i-1}$ und $\mathcal{G}_i$ berechnet. Gilt nach der Berechnung von $\mathcal{T}_{i-1} \setminus \mathcal{G}_i =$, so kann an dieser Stelle die Suche abgebrochen werden, da bereits zu einem Zeitpunkt i eine leere Resultatrelation $\mathcal{T}_{n-1}$ vorliegt.

Eine Installation von MySQL 5.0.18 auf einem PPC G4 mit 1.5 GHz Taktfrequenz und 1.25 GB RAM benötigt zur Berechnung einer Schnittmenge von einer Relation mit 750 Tupeln und einer mit 150.000 Tupeln ca. 8 Sekunden. Die Berechnung der Schnittmenge für zwei Relationen im Bereich von 150.000 Tupeln liegt mit rund 2 Minuten sehr hoch. Ohne der Auswertung der nachfolgend beschriebenen Suchalgorithmen vorgreifen zu wollen, kann jedoch gesagt werden, dass diese Bearbeitungszeiten für eine Anwendung in der Praxis zu hoch sind.

Zur Beschleunigung der Operation war ein Index über die Dokumenten-ID für alle Tabellen angelegt worden, der eine deutliche Beschleunigung der Vergleichsoperation bzgl. der Dokumenten-ID in dem folgenden SQL-Statement darstellt.

$\mathcal{T}_1$= **SELECT**
A.g_elem $\cdot g_0^{-1}$ **AS** g_elem,
A.doc_id **AS** doc_id,
FROM (
(**SELECT**
g_elem **AS** g_elem,
doc_id **AS** doc_id,
FROM
tb_glists
WHERE
list_id = m_0) **AS** A
INNER JOIN
(**SELECT**
g_elem $\cdot g_1^{-1}$ **AS** g_elem,
doc_id **AS** doc_id,
FROM
tb_glists
WHERE
list_id = m_1) **AS** B
ON
A.doc_id = B.doc_id **AND** A.g_elem = B.g_elem);

3.1.2 k-Fehlstellen- und Fuzzyanfragen

Die Beantwortung von k-Fehlstellen- und Fuzzyanfragen mittels relationaler Datenbanksysteme lässt sich ebenfalls sehr gut mittels der Zählung von Vielfachheiten erreichen. Wir erweitern dazu die Relationen $\mathcal{G}_i$ um das Attribut list_num, welches wir mit i identifizieren. Auf diese Weise erhalten alle Tupel einer Relation $\mathcal{S}_i$ eine ID, die angibt, aus welcher Relation das Tupel stammt. Diese ID ist notwendig, um bei der Berechnung von

$$\mathcal{T}' := \bigcup_{0 \leq i < n} \mathcal{S}_i$$

zu garantieren, dass im dem Fall der Übereinstimmung aller Attribute bis auf list_num ein Tupel in dem Zwischenergebnis $\mathcal{T}'$ enthalten ist. Wir können $\mathcal{T}'$ durch eine SQL-Abfrage der folgenden Art bestimmen:

```
T' =  SELECT
            g_elem ·g_0^-1  AS g_elem,
            doc_id          AS doc_id,
            1               AS score,
            0               AS list_num
      FROM
            tb_glists
      WHERE
            list_id = m_0;
UNION
            ...
UNION
      SELECT
            g_elem ·g_{n-1}^-1 AS g_elem,
            doc_id          AS doc_id,
            1               AS score,
            n − 1           AS list_num
      FROM
            tb_glists
      WHERE
            list_id = m_{n-1};
```

In einem nachfolgenden Schritt werden die Score-Werte pro Paar (g_elem, doc_id) bestimmt:

```
T'' =  SELECT
            g_elem      AS g_elem,
            doc_id      AS doc_id,
            SUM(score)  AS score
       FROM
            T'
       GROUP BY
            g_elem, doc_id;
```

An diesem Schritt wird deutlich, warum das zusätzliche Attribut list_num notwendig ist. Die Summierung würde ansonsten immer zu einem Score-Wert 1 führen. Als letzter Schritt sind in der Relation $\mathcal{T}''$ die Tupel zu bestimmen, deren Score-Wert größer oder gleich $n - k$ ist.
Das vorgestellte Konzept kann kanonisch um Fuzzyanfragen erweitert werden. Die Relationen $\mathcal{G}_i$, die zu einem Fuzzy-Objekt in der Anfrage gehören, gehen alle mit dem gleichen Wert für das Attribut list_num bei der Berechnung von $\mathcal{T}'$ ein. Damit ist sichergestellt, dass der Score-Wert eines Tupels in $\mathcal{T}'$ immer gleich 1 ist, auch wenn mehrere Alternativen zu einem Treffer führen würden.
Die Zeit, die das zuvor beschriebene RDBMS benötigt, um eine Anfrage der Länge 14 (d.h. 14 G-invertierte Listen waren zu betrachten) zu beantworten, liegt bei dem Szenario der Audioidentifikation im Bereich von rund 2 Sekunden. Dies ist deutlich schneller als die Anfragebeantwortung mittels Join-Operationen, jedoch immer noch deutlich langsamer als die Algorithmen, die nachfolgend vorgestellt werden.

3.2 Verfahren bei angeordneten Gruppen

Wenden wir uns nun wieder Algorithmen und Datenstrukturen zu, die speziell auf die Suche mittels G-invertierter Listen ausgelegt sind.
Eine Gruppe G heisst *angeordnet*, wenn es eine Totalordnung $<$ auf G gibt, die mit der Gruppenoperation verträglich ist, d.h. wenn für alle $g, x, y \quad G$ aus $x < y$ stets $xg < yg$ folgt. Beispielsweise ist

$(\mathbb{R}^n, +)$ bezüglich der lexikographischen Totalordnung angeordnet. Gleiches gilt für jede Untergruppe von $\mathbb{R}^n$, insbesondere also für $\mathbb{Z}^n$. Hingegen kann eine Gruppe G, die eine nichttriviale endliche Untergruppe U enthält, niemals angeordnet sein. Denn mit G wäre auch jede Untergruppe angeordnet. Es genügt also zu zeigen, dass eine nichttriviale endliche Gruppe G nicht angeordnet werden kann. Sei dazu $g_1 < \ldots < g_n$ eine Anordnung der endlichen Gruppe G, $n > 1$. Dann ist $g_1 < g_n$, aber für das Gruppenelement $g = g_1^{-1} g_n$ ist $g_n = g_1 g < g_n g$. Wegen $g_n g = g_n$ widerspricht dies der Annahme, dass g_n größtes Element ist. Da die n-ten Einheitswurzeln eine endliche Untergruppe der multiplikativen Gruppe des komplexen Zahlkörpers bilden, kann $(\mathbb{C}^*, \cdot)$ nicht angeordnet werden. Entsprechendes gilt für die Gruppen $SE(n)$.
In diesem Abschnitt legen wir immer eine angeordnete Gruppe G zugrunde, d.h. die G-invertierten Listen G_j sind auch nach der Justierung bzgl. der Elemente einer Anfrage $Q = \{q_1, \ldots, q_n\}$ sortiert. Die zugehörigen Listenlängen $\ _j$ seien aufsteigend sortiert. Dies ermöglicht es bei fast allen hier beschriebenen Verfahren die unjustierten und sortierten Listen nach und nach von der Festplatte zu laden und erst dann die Justierung durchzuführen. Bei großen Datenmengen kann dies sogar eine notwendige Vorgehensweise sein. In anderen Fällen kann die Sortierung einer G-invertierten Liste nach deren Justierung erfolgen, um auf diese Weise Algorithmen anwenden zu können, die eben dies voraussetzen.

3.2.1 Berechnung von $G_{\mathcal{D}\,k}(Q)$ mit der Kostenfunktion Γ

Im vorherigen Kapitel wurde ein Verfahren zur Berechnung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ vorgestellt, welches auf einer Kostenfunktion Γ beruht. Dabei wurde in der Aufbauphase beim Schnitt von $k+1$ Listen gezählt, wie oft ein Element einer Liste nicht im Schnitt vorkam. War dies in weniger als $k+1$ Fällen der Fall, war ein solches Listenelement ein Kandidat für einen k-Fehlstellentreffer.
Bei einer Implementierung dieses Verfahrens ist es nicht unbedingt notwendig, dass die jeweilige Gruppe angeordnet ist. Allerdings kann man in diesem Fall die Schnittoperation beschleunigen, indem man Binärsuche anwendet oder zwei Listen sequentiell durchläuft und aufgrund der auf den Listenelementen definierten Ordnung schnell entscheiden kann, ob ein Element im Schnitt liegt oder nicht. Wäre die Gruppe nicht angeordnet, wäre hier quadratischer Aufwand in der Länge der Listen zu erwarten, was das Verfahren unpraktikabel macht. Ist es möglich, die zu schneidenden Listen im Hauptspeicher zu verwalten, können die Listen nach der Justierung sortiert werden, was die Berechnung des Listenschnitts durch sequentielles Durchlaufen beider Listen ermöglichen würde. Alternativ könnte auch nur eine Liste sortiert werden (z.B. die längere von beiden) und die Elemente der anderen Liste in eben der sortierten Liste mittels binärer Suche gesucht werden.
Der größte Nachteil dieser Suchstrategie liegt in der Aufbauphase, d.h. während die ersten $k+1$ Listen bearbeitet werden. Im schlimmsten Fall erhält man nach deren Bearbeitung eine temporäre Liste von Trefferkandidaten, die nahezu alle Elemente der Einzellisten enthält. Um die Sortierung dieser temporären Liste von Trefferkandidaten zu erhalten, werden während der Aufbauphase Elemente an geeigneter Stelle in die Liste eingefügt und nicht einfach am Ende der Liste angefügt. Dies führt zu der Notwendigkeit einer Datenstruktur, die diese Operation effizient unterstützt. Eine verkettete Liste bietet sich dazu an. Allerdings wäre in diesem Fall keine Binärsuche nach einer Einfügeposition für ein neues Element mehr möglich. Daher werden meist Arrays als Container für die G-invertierten Listen und damit auch für die temporäre Liste der Trefferkandidaten benutzt, was zu einem erheblichen Mehraufwand bei den Einfüge- und Löschoperationen bedeutet. In jedem Fall bleibt dieses Verfahren auf solche Szenarien beschränkt, bei denen die Liste der Trefferkandidaten zu jeder Zeit im Hauptspeicher vorgehalten werden kann.
Ein weiterer Nachteil ist sicherlich der zu jedem Trefferkandidaten zu speichernde Kreditwert. Dies

entspricht prinzipiell der Erweiterung der Tupel (g, i), wie sie in den G-invertierten Listen enthalten sind um einen Wert $c \quad \mathbb{Z}_{\geq 0}$. Auf Seiten der Implementierung bedeutet dies, dass die Kreditwerte zu den Trefferkandidaten entweder in einem separaten Array gespeichert werden, welches jedoch bei jeder Einfüge- bzw. Löschoperation ebenfalls aktualisiert werden muss. Oder aber die Trefferkandidaten (g, i) werden in eine andere Datenstruktur kopiert, die eben die Speicherung eines Kreditwertes ermöglicht.
Nach dem Ende der Aufbauphase enthält die Liste der Trefferkandidaten alle Tripel (g, i, c) mit einem Kreditwert $c > 0$. Im nächsten Schritt wird diese Liste mit einer weiteren G-invertierten Liste geschnitten, jedoch können keine neuen Trefferkandidaten mehr hinzukommen. Allerdings werden – wenn man von einer sinnvollen Anfrage ausgeht – schon in diesem Schritt eine Vielzahl von Trefferkandidaten ausscheiden, nämlich eben gerade diese mit $c = 1$. D.h. die zuvor aufwändig erstellte Liste von Trefferkandidaten wird nach nur wenigen Listenschnitten auf eine Länge reduziert, die der Anzahl der Treffer entspricht. Betrachtet man die Berechnung der Treffermenge rückblickend, so kann gesagt werden, dass viele der während der Aufbauphase durchgeführten aufwändigen Einfügeoperationen unnötig gewesen sind.
Eben gerade die zuletzt dargestellte Problematik erklärt die Zeiten in [Wag03], die das System benötigt, um eine Anfrage zu beantworten. Besonders bei einer größeren Zahl von Fehlstellen und einer Anfrage Q mit einer Länge $|Q| \geq 10$ benötigte das Audioidentifikationssystem z.T. mehrere Sekunden zur Berechnung der Treffermenge. Aus diesem Grund wurde in [Wag03] eine Anfrage mit einer Länge größer als 10 in mehrere Teilanfragen kürzerer Länge unterteilt, deren Ergebnisse dann relativ aufwändig zu einem Gesamtergebnis verdichtet wurden. Bei der später in diesem Kapitel analysierten Implementation dieses Verfahrens wurde eine verkettete Liste verwendet, um die Trefferkandidaten während der Aufbauphase zu speichern. Allerdings wurde deswegen auf die Verwendung von Binärsuche verzichtet, d.h. im Falle von nur wenigen Trefferkandidaten, wurden dennoch die verbliebenen G-invertierten Listen linear durchlaufen, um zu prüfen, ob die Trefferkandidaten in diesen Listen enthalten sind.

3.2.2 Berechnung von $G_{\mathcal{D}\,k}(Q)$ mit n-Wege Merge-Sort

Da G angeordnet ist, liegen trotz Nachjustierung n sortierte Listen $G_1, \ldots, G_n$ vor, die zu einer Multimenge vereinigt werden müssen. Dies lässt sich mit einem Min-Heap, der zunächst aus den kleinsten Elementen der Listen aufgebaut wird, unter Berücksichtigung der Vielfachheiten realisieren. Das kleinste Element (g, i), das etwa aus der j-ten Liste stammt, wird vom Heap entfernt und zusammen mit der Vielfachheit in einer sortierten Liste L gespeichert: $L = ((g, i, 1))$. Aus der Liste G_j wird nun das folgende Element in den Heap eingefügt und die Heapeigenschaft durch „Versickern“ des neuen Elements hergestellt. Mit Hilfe des neu entstandenen minimalen Elements des Heaps (h, k) wird nun die Liste L aktualisiert: Ist $(g, i) = (h, k)$, so wird die Vielfachheit von (g, i) erhöht, d.h. $L = ((g, i, 2))$. Ansonsten wird das letzte Element in L aus der Liste entfernt und $(h, k, 1)$ an L angefügt. Allgemein gibt es nur diese beiden Fälle: Erhöhung der Vielfachheit des letzten Listenelementes von L oder das Anfügen eines neuen Elementes mit Vielfachheit 1 an L. Bezeichnet $\quad$ die Gesamtlänge der involvierten Listen $G_1, \ldots, G_n$, so beträgt der Gesamtaufwand $O(\ \log n)$. Im Fall der Erweiterung der Liste L ist das bis dahin letzte Element zu entfernen, wenn dessen Vielfachheit den Schwellwert $n - k$ unterschreitet. Man beachte, dass die Liste L, abgesehen vom jeweils letzten Element, nie länger als die endgültige Trefferliste ist.

Im Falle von nicht ordnungserhaltender Gruppenoperation könnte man den obigen Fall erzwingen, indem man die Listen $G_1, \ldots, G_n$ zunächst sortiert. Hat G_i die Länge $\ _i$, so geht dies insgesamt mit Aufwand $O(\ \ _i\ _i \log\ _i)$. Da typischerweise $\log n \ll \log\ _i$ gilt, ändert sich durch das anschließen-

Algorithmus 3.2.1: k-MISMATCHSEARCHWITHHEAP($Q = (q_1, \ldots, q_n), k$)

```
minheap ← ∅;
L ← ∅;   // Initialisierung Resultatliste
κ ← 0;   // Anzahl der abgearbeiteten Listen
for j ← 1 to n
  do { (g, i) ← G_D(q_j).REMOVE_FRONT();   // die zu q_j gehörige justierte G-invertierte Liste
       minheap.ADD((g, i, j));
while not minheap.EMPTY()
  do { (g, i, j) ← minheap.REMOVE_ROOT();   // (g, i) ist minimales Element in minheap
       if L = ∅
         then L.PUSH_BACK((g, i, 1))
         else { (g', i', c) ← L.REMOVE_BACK();
                if (g', i') = (g, i)
                  then L.PUSH_BACK((g', i', c + 1));
                  else { if c ≥ n − k
                           then L.PUSH_BACK((g', i', c));
                         if κ ≤ k
                           then L.PUSH_BACK((g, i, 1));
                           else return (L); }
       if G_D(q_j) ≠ ∅
         then { (g, i) ← G_D(q_j).REMOVE_FRONT();
                minheap.ADD((g, i, j)); }
         else κ ← κ + 1;
(g, i, j) = L.POP_BACK();
if j < n − k
  then L.REMOVE_BACK();
return (L);
```

Algorithmus 3.2.1: Berechnung einen Multimenge aus n vorsortierten Listen G_j mittels N-Wege Merge-Sort, wobei nur Elemente mit einer Mindestvielfachheit von $n - k$ in der Multimenge vorhanden sein dürfen. Vorbedingung ist, dass die Gruppenoperation ordnungserhaltend ist. Sind $\kappa > k$ viele Listen bereits abgearbeitet, können keine neuen Tripel hinzukommen, da die geforderte Mindestvielfachheit nicht mehr erreicht werden kann.

de n-Wege Merge-Sort die Größenordnung für den Gesamtaufwand nicht mehr, dieser beträgt also $O(\quad_i \; _i \log \; _i)$. Dies kann nach oben weiter durch $O(\; \log \; _n)$ abgeschätzt werden.

Bisher haben wir bei den obigen Aufwandsabschätzungen nur die Anzahl der Vergleichsoperationen betrachtet. Bei einer Implementierung sind aber noch weitere, möglicherweise viel entscheidendere Aspekte mit zu berücksichtigen. Betrachten wir zunächst die Gemeinsamkeiten. In beiden Fällen ist ein Heap mit n vielen Elementen aufzubauen und zu aktualisieren. Zusätzlich interessiert aktuell nur das jeweils letzte Element der Liste L. Daher spielt die Länge der Liste keine Rolle.

Die beiden Fälle unterscheiden sich jedoch bei der Berechnung der justierten Listen $G_1, \ldots, G_n$. Im Fall einer ordnungserhaltenden Gruppenoperation genügt es, den Anfang von jeder der n vielen unjustierten Listen (z.B. hardwareabhängig eine Seite) in den Hauptspeicher zu laden. Vor dem Einfügen in den Heap, werden die Elemente entsprechend der Anfrage justiert. Während des Aufbaus der Liste L sind bei Bedarf weitere Seiten der unjustierten Listen nachzuladen. Dies kann auf Mehrprozessorsystemen sogar parallel geschehen. Allerdings müssen bei diesem Vorgehen im schlimmsten Fall alle Listenelemente justiert und in den Heap eingefügt werden. Bei einer komplexen Gruppenoperation kann dies ein Nachteil gegenüber Verfahren sein, bei denen nach Re-Justierung Vergleiche in der unjustierten Liste durchgeführt werden. In günstigen Fällen kann es jedoch hier möglich sein, die Suche vorzeitig zu beenden, indem protokolliert wird, wie viele Listen bereits vollständig abgearbeitet wurden.

Im Gegensatz dazu sind im Fall einer nicht-ordnungserhaltenden Gruppenoperation die Listen G_j aus den unjustierten Listen zu berechnen und anschließend zu sortieren. In manchen Anwendungen (wie zum Beispiel bei der Suche in Moleküldaten) kann es vorkommen, dass nicht alle n Listen G_j gleichzeitig im Hauptspeicher vorgehalten werden können, weshalb diese anfrageabhängig justierten Listen zunächst gespeichert werden müssen. Insgesamt entsteht auf diese Weise z.T. erheblicher zusätzlicher Speicher- und Sortieraufwand.

3.2.3 Berechnung von $G_{\mathcal{D}\,k}(Q)$ mittels sortiertem Array

In diesem Abschnitt wird ein Algorithmus zur Trefferberechnung vorgestellt, der im Gegensatz zu dem vorherigen ohne einen Heap auskommt. Zusätzlich werden hier Elemente in den G-invertierten Listen übersprungen, wenn sie zu keinem Treffer mehr gehören können. Die Zahl der übersprungenen Elemente in den Listen hängt von der Anzahl der erlaubten Fehlstellen ab. Je weniger Fehlstellen zugelassen sind, desto mehr Elemente in den Listen können unberücksichtigt bleiben. Da für jedes übersprungene Listenelement eine Gruppenoperation entfällt, ist dieses Verfahren besonders dann anzuwenden, wenn die Verknüpfung in der Gruppe aufwändig ist. Da wir nicht für jeden Eintrag in einer G-invertierten Liste die jeweils mit einem Gruppenelement g justierte Version benötigen, verwenden wir in dem Algorithmus 3.2.3 auch die unjustierten Listen G'_j.

Das Grundprinzip des Verfahrens besteht darin, die jeweils zu betrachtenden Listen sequentiell zu durchlaufen und dabei zunächst die jeweils aktuellen Einträge in den G-invertierten Listen entsprechend der Anfrage Q zu justieren und in einem Array v der Länge $n = |Q|$ zu speichern. Die Einträge in $v = (v_0 \equiv (g_0, i_0), \ldots, v_{n-1})$ werden aufsteigend sortiert, bevor die nachfolgenden Überlegungen angestellt werden. Dieser Schritt wiederholt sich, bis entsprechend viele Listen vollständig abgearbeitet worden sind, so dass ein weiterer k-Fehlstellentreffer nicht mehr möglich ist.

Fall 1 (Treffer): Ein Treffer liegt dann vor, wenn v mindestens $n - k$ identische Einträge enthält. Aufgrund der Sortierung von v liegen diese Elemente in Form einer Sequenz vor. Ohne Beschränkung der Allgemeinheit gehen wir davon aus, dass diese *Treffersequenz* bei $v[0]$ beginnt und sich mindestens bis einschliesslich $v[|Q| - k - 1]$ erstreckt. Der Position $p = |Q| - k - 1$ mit $p > 0$, kommt eine Art

Schlüsselrolle zu. Sind die ersten $n-k$ Einträge in v identisch, so muss geprüft werden, ob die Sequenz ggf. mehr als die mindestens geforderten $n-k$ umfasst. Die Länge der Treffersequenz in v bezeichnen wir als *Score* des Treffers. In einem letzten Schritt können die Listenzeiger der G-invertierten Listen um eine Position verschoben werden, die an dem Treffer beteiligt sind.

Fall 2 (kein Treffer): In diesem Fall enthält v weniger als $n-k$ viele identische Einträge. Die Rolle der Schlüsselposition p in v ist hier besonders wichtig. Denn das an Position $v[p]$ gespeicherte Paar (g,i) stellt bzgl. der Ordnung $<$ das kleinste Element dar, welches überhaupt noch zu einem k-Fehlstellentreffer gehören kann. Sei (g',i') das Paar gespeichert an der Stelle $v[0]$. Dies ist aufgrund der aufsteigenden Sortierung der Einträge in v das kleinste Element. Da kein Treffer vorliegt, gilt $(g',i') < (g,i)$. Nun betrachten wir das nächste bgzl. Q justierte Element (h,j) aus der G-invertierten Liste, aus der (g',i') stammt. Dieses werde v anstelle des Elements (g',i') hinzugefügt. Auf jeden Fall gilt $(g',i') < (h,j)$. Gilt ausserdem $(h,j) < (g,i)$, kann erneut kein Treffer vorliegen, da die zuvor beschriebene Trefferbedingung nicht erfüllt sein kann. Erst wenn $(h,j) \geq (g,i)$ gilt, kann aufgrund der Sortierung von v das Paar (h,j) zu einem Treffer beitragen.
Diese Argumentation lässt sich analog für alle anderen Einträge in $v[\,], 0 \leq \quad < p$ und den dazu gehörigen G-invertierten Listen führen. Aus diesem Grund bietet es sich an, vor einer erneuten Sortierung von v dafür zu sorgen, dass für alle Einträge (h,j) in v gilt: $(h,j) \geq (g,i)$. D.h. die Listenzeiger der zu den Einträgen $v[\,], 0 \leq \quad < p$ G-invertierten Listen sind derart zu verschieben, bis ein entsprechendes Element (h,j) erreicht ist, so dass $(h,j) \geq (g,i)$ gilt. Dieses Element ersetzt dann das jeweils in v bereits vorhandene Element. Nun wird v erneut sortiert und es wird geprüft, ob v einen Treffer enthält.

An der Schlüsselposition p in v erkennt man auch direkt den Einfluss der Anzahl der Fehlstellen k auf die Laufzeit des Algorithmus. Ist $k = 0$, können die Listenzeiger aktualisiert werden, bis alle Listenzeiger auf Elemente verweisen, die größer oder gleich dem unjustierten Maximum in v sind. Ist $k > 0$ kann dies ein kleinerer Wert sein, so dass die Listenzeiger nicht in dem gleichen Maße zum Ende der Liste hin verschoben werden können. Ausserdem können mit steigendem k auch nur weniger Listenzeiger überhaupt verändert werden. Dies wird auch durch die Ergebnisse untermauert, wie sie in Abbildung 3.1 dargestellt werden.
Das bisher beschriebene Vorgehen hat zur Folge, dass wir nach der Sortierung von v die Frage, ob v einen Treffer enthält, dadurch beantworten können, indem wir die Elemente $v_0, \ldots, v_{n-k-1}$ auf Gleichheit prüfen. Enthalten eben diese Elemente von v den gleiche Werte, so müssen wir prüfen, ob noch weitere Elemente in v diesen Wert beinhalten, um die korrekte Vielfachheit zu bestimmen. Auch bei diesem Verfahren können wir die Suche beenden, sobald $k+1$ Listen ganz abgearbeitet worden sind. Dieses Verfahren benötigt im Vergleich nur sehr wenig Hauptspeicher in der Größenordnung der Länge der Anfrage Q.
Ein wichtiger Aspekt bei diesem Suchverfahren ist, dass das Element an der Schlüsselposition p in dem Array v dazu benutzt werden kann, die Listenzeiger von G-invertierten Listen derart zu verschieben, dass manche Elemente in den beteiligten Listen gar nicht erst betrachtet werden müssen. Gerade bei aufwändig zu berechnenden Gruppenverknüpfungen führt dies ggf. zu einer (deutlichen) Reduktion der Laufzeit. Dies ist natürlich abhängig von den Elementen in den G-invertierten Listen und damit von der zugrundeliegenden Dokumentenkollektion $\mathcal{D}$. Weiterhin beeinflussen die Elemente der Anfrage Q ebenfalls direkt die jeweilige Laufzeit, wie in Abbildung 3.1 deutlich zu erkennen ist. Die Anfrage Nummer sieben beinhaltet anscheinend Elemente, die zur Folge haben, dass bei keinen bis relativ wenigen Fehlstellen eine Vielzahl von Listenelementen übersprungen werden können. Hingen scheint dies bei der Anfrage fünf eben gerade nicht der Fall zu sein.
Sei nun $(h,i) = v[p]$ das Element an der Schlüsselposition p in v (vgl. Abbildung 3.2), wobei $j =$

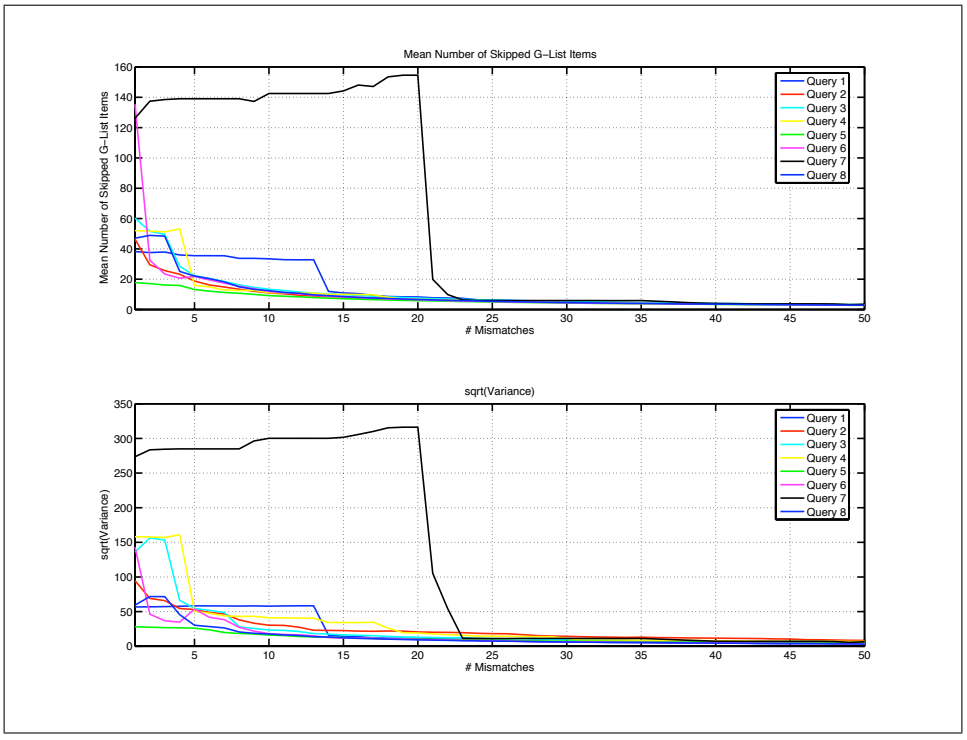


Abbildung 3.1: Die obere Abbildung zeigt die mittlere Anzahl von Listenelementen, die bei dem Aktualisieren der Listenzeiger während einer Suche übersprungen werden. Der Einfluss der Anfrage Q, sowie die Anzahl den zugelassenen Fehlstellen wird deutlich. Die untere Graphik zeigt die Standardabweichung.

$\pi(p)$ der Index der G-invertierten Liste G_j ist, aus der sich der Eintrag $v[p]$ durch Justierung eines Listenelements aus G'_j mit g_j ergeben hat: $(g,i) \quad G_j : h = g \cdot g_j^{-1}$. Sei nun $G \ , 0 \leq \ < j$, eine weitere G-invertierte Liste, deren Lesezeiger zum Listenende hin verschoben werden soll. Es wird also das kleinste Element (h',i') in G gesucht, für das $(h,i) \leq (h',i')$ gilt. Oder anders formuliert: $(g \cdot g_j^{-1}, i) \leq (f \cdot g^{-1}, i')$, mit $h' = f \cdot g^{-1}, f \quad G$. Da die Dokumenten-ID invariant unter der Gruppenoperation ist, richten wir den Fokus aus die G-Komponente in den Listenelementen: $g \cdot g_j^{-1} \leq f \cdot g^{-1}$. Durch Rechtsmultiplikation mit g erhalten wir: $g \cdot g_j^{-1} \cdot g \ \leq f$. D.h. anstelle die Elemente in G mit g^{-1} zu justieren und dann mit $(h,i) = v[p]$ zu vergleichen, wird (h,i) „de-justiert“ und die Elemente von G' mit diesem Wert verglichen. Auf diese Weise wird bei der Bestimmung der Verschiebung eines Listenzeigers nur eine Gruppenverknüpfung notwendig – die Einträge in den G-invertierten Listen brauchen für den Vergleich nicht justiert zu werden! Dies ist insofern wichtig, als dass die justierten Listen G_j in der Praxis nicht vorliegen, sondern während der Suchanfrage die Elemente aus G'_j einzeln justiert werden. Sobald also der Listenzeiger in einer Liste um zwei Positionen verändert wird, ist dieses Verfahren daher von Vorteil.

Beispiel 3.1 Dieses Beispiel soll die Vorgehensweise des Algorithmus 3.2.3 veranschaulichen. Um das Beispiel nicht unnötig kompliziert zu gestalten, gehen wir von nur einem Dokument in der Dokumentenkollektion $\mathcal{D}$ aus, d.h. wir können im folgenden auf eine Dokumenten-ID i verzichten. Zusammen mit der Gruppe $(\mathbb{Z},+)$ wird die Menge $\mathbb{Z} \times M = \{\circ, \square, \Diamond\}$ zur G-Menge. Betrachten wir

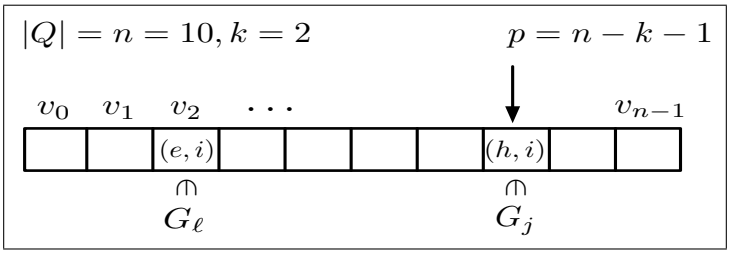


Abbildung 3.2: Verdeutlichung des Inhalts des Vektors v zur Laufzeit.

nur die drei folgenden unjustierten G-invertierten Listen:

$$\begin{aligned} G_0'^{\circ} &:= (2, 17, 19, 25, 30, 37, 45, 53), \\ G_1'^{\square} &:= (8, 19, 30, 35, 49), \\ G_2'^{\Diamond} &:= (2, 3, 5, 13, 21, 22, 25, 37, 40, 45, 68, 107). \end{aligned}$$

Weiterhin sei eine Anfrage $Q = \{(5, \circ), (10, \square), (20, \Diamond)\}$ gegeben. Dies bedeutet, die Elemente der Liste $G_1'^{\circ}$ sind mit dem inversen des Gruppenelements $g_1 = 5$ zu justieren. Analog ergibt sich $g_2 = 10$ und $g_3 = 20$. Wir lassen bei dem Beispiel keine Fehlstellen zu, d.h. $k = 0$ und $p = 2$. Betrachten wir nun die Schritte des Algorithmus 3.2.3 im Einzelnen, wobei wir bei der Bezeichnung der G-invertierten Listen im folgenden auf die Angabe der „Listenklasse“ $c_i \quad \{\circ, \square, \Diamond\}$ verzichten, da bereits die Listennummer eine eindeutige Zuordnung einer G-invertierten Liste zu einer Listenklasse beschreibt.

Schritt 1: Die Listenzeiger verweisen alle auf das jeweils erste Element in den beteiligten G-invertierten Listen: $\mathrm{lp} = (0, 0, 0)$. Dies führt zu $v = (2 - 5, 8 - 10, 2 - 20) = (-3, -2, -18)$. Nach der Sortierung von v gilt: $v = (-18, -3, -2)$ und $\pi = (2, 0, 1)$. Da bereits $v[0] = v[p]$ gilt, liegt kein Treffer vor. Somit sind nun die Listenzeiger der zu den Elementen $v[0]$ und $v[1]$ gehörenden G-invertierten Listen geeignet zu verschieben. Beginnen wir mit $v[0]$: Es ist[1] $h = v[p] = G_2[0] - g_2 = -2$. Der Index j der G-invertierten Liste, aus der der Eintrag $v[0]$ stammt, ergibt sich aus $j = \pi(0) = 2$. Dies bedeutet, dass das Gruppenelement h noch mit g_2 verknüpft werden muss, um den Wert von h der Justierung der Liste G_2 anzupassen: $h = -2 + 20 = 18$. Der Listenzeiger $\mathrm{lp}[2]$ kann nun solange zum Listenende hin verschoben werden, wie $G_2[\mathrm{lp}[2]] \leq h$ gilt. Existiert in G_2 kein Element $g = h$, so kann der Lesezeiger um eine weitere Position zum Listenende hin verschoben werden. In diesem Beispiel ergibt sich also nach dem ersten Schritt folgende Listenzeigerpositionen: $\mathrm{lp} = (1, 0, 4)$. In der Liste G_2 sind auf diese Weise drei Einträge übersprungen worden.

Schritt 2: $\mathrm{lp} = (1, 0, 4) \rightsquigarrow v = (12, -2, 1)$ bzw. $v = (-2, 1, 12), \pi = (1, 2, 0)$. Da auch hier bereits $v[0] = v[p]$ gilt, liegt erneut kein Treffer vor. In diesem Schritt werden die Listenzeiger der Listen G_1 und G_2 verschoben. Nach zu Schritt 1 analogen Betrachtungen erhalten wir $\mathrm{lp} = (1, 2, 7)$.

Schritt 3: $\mathrm{lp} = (1, 2, 7) \rightsquigarrow v = (12, 20, 17)$ bzw. $v = (12, 17, 20), \pi = (0, 2, 1)$. Da auch hier bereits $v[0] = v[p]$ gilt, liegt erneut kein Treffer vor. In diesem Schritt können die Listenzeiger der Listen G_0 und G_2 verschoben werden. Wir erhalten $\mathrm{lp} = (3, 2, 8)$.

[1] Wir verzichten auf die im Algorithmus vorkommende Dokumenten-ID d an dieser Stelle.

Schritt 4: $\mathrm{lp} = (3,2,8) \rightsquigarrow v = (20,20,20), \pi = (0,1,2)$. In diesem Fall enthält v eine Sequenz von identischen Einträgen der Länge $|Q|$, d.h. eine Verschiebung der Anfrage durch das Gruppenelement $g = 20$ führt zu einem Treffer. Alle Listenzeiger können daher um eine Position zum Listenende hin verschoben werden: $\mathrm{lp} = (4,3,9)$.

Schritt 5: $\mathrm{lp} = (4,3,9) \rightsquigarrow v = (25,25,25), \pi = (0,1,2)$. In diesem Fall enthält v ebenfalls eine Treffersequenz, d.h. eine Verschiebung der Anfrage durch das Gruppenelement $g = 25$ führt ebenfalls zu einem Treffer. Alle Listenzeiger können daher erneut um eine Position zum Listenende hin verschoben werden: $\mathrm{lp} = (5,4,10)$.

Schritt 6: $\mathrm{lp} = (5,4,10) \rightsquigarrow v = (32,39,48), \pi = (0,1,2)$. Da $v[0] = v[p]$ gilt, liegt kein Treffer vor. In diesem Schritt können die Listenzeiger der Listen G_0 und G_1 verschoben werden. Wir erhalten $\mathrm{lp} = (-1,-1,8)$, d.h. die Listen G_0 und G_1 sind bereits komplett durchlaufen worden. Da nur noch Elemente in einer Liste für weitere Treffer in Frage kommen, kann der geforderte Schwellwert von drei identischen Einträgen in v nicht mehr erreicht werden. Somit kann die Suche abgebrochen werden, ohne dass alle Elemente in G_2 betrachtet werden mussten.

Um den Algorithmus 3.2.3 nicht unnötig kompliziert darzustellen, wurde eine naheliegende Reduktion der zu berechnenden Gruppenverknüpfungen beim Einfügen neuer Einträge in das Array v nicht aufgeführt. Es liegt auf der Hand die Löschung aller Einträge zu Beginn eines jeden Durchlaufs der **while**-Schleife durch eine Aktualisierung der jeweils geeigneten Anzahl von Einträgen in v zu ersetzen. Liegt kein Treffer vor, werden lediglich die Listenzeiger der Listen $\pi(i), 0 \leq i < p$ verschoben, so dass nur die ersten $p-1$ Einträge in v einer Aktualisierung beim nächsten Schleifendurchlauf bedürfen. Im Falle eines Treffers bestehend aus s vielen Einträgen in v, sind lediglich die ersten s Einträge in v durch neue Elemente aus den jeweiligen justierten G-invertierten Listen zu ersetzen.

3.3 Verfahren bei nicht notwendigerweise angeordneten Gruppen

Nachdem wir bisher ausgenutzt haben, dass die Gruppe angeordnet werden kann, geht es nun um Verfahren, die auch bei nicht angeordneten Gruppen eine mehr oder weniger effiziente Treffermengenberechnung erlauben.

3.3.1 Berechnung von $G_{\mathcal{D}\,k}(Q)$ mittels baumbasierter Verfahren

Wir gehen von einer Totalordnung der Gruppenelemente aus (erweitert um die Dokumenten-ID), wobei wir nicht voraussetzen, dass diese Totalordnung mit der Gruppenverknüpfung verträglich ist. Bei dem nun diskutierten Verfahren werden die Vielfachheiten der Gruppenelemente unter Verwendung einer geeigneten Baumdatenstruktur (B-Bäume, R-Bäume) protokolliert, die sich durch die Justierung der G-invertierten Listen bzgl. einer Anfrage Q ergeben.
Bei typischerweise unendlicher Gruppe G und unendlicher Objektmenge M kann man nicht einen für alle denkbaren Anfragen Q gültigen, durch (g,i) parametrisierten Vielfachheitenvektor im Rechner vorhalten. Sind alle Stabilisatoren trivial und bezeichnet R eine Transversale der G-Bahnen von M, so ist die Anzahl der potentiellen Trefferkandidaten zur Anfrage Q und Dokumentenkollektion $\mathcal{D}$ nach oben beschränkt durch $\sum_{i=1}^{N} \sum_{r \in R} |Q \setminus Gr| \cdot |D_i \setminus Gr|$ und damit endlich. Allerdings variiert mit Q auch die Menge der Trefferkandidaten, so dass man a priori keine von Q unabhängige Datenstruktur zur Vielfachheitenzählung initialisieren kann. Wir benutzen zu diesem Zweck eben in Abhängigkeit von Q dynamische Baumdatenstrukturen. Es liegt auf der Hand, dass ein solches Suchverfahren nur dann effizient sein kann, wenn die Verwaltung der Trefferkandidaten im Hauptspeicher möglich ist.

Algorithmus 3.2.2: k-MISMATCHSEARCHLINEARARRAY($Q = (q_1 = (g_1\ c_1) \qquad q_n)\ k$)

```
L ← ∅;    // Resultatliste
κ ← 0;    // Zähler für vollständig abgearbeitete Listen
v[n];    // n Paare der Art (g i)
lp[n];    // Array von n Listenzeigern, initialisiert mit 0
p ← (|Q| − k − 1);    // Schlüsselposition im Array v

while κ ≤ k + 1
do { v ← ∅;
     for i ← 0 to n − 1
       do { if lp[i] ≥ 0
            then { (h d) ← G'_{c_i}[lp[i]];    // Das jeweils aktuelle Listenelement
                   v PUSH_BACK((h g_i^{-1} d));    // Justierung des jeweiligen Listenelementes
     π ← SORTASCENDING(v);    // Sortierung von v, Rückgabe der Permutation π
     f ← true; i ← 0; j ← (p);
     while i ≠ j and i < p and j > 0
       do { if v[i] ≠ v[j]
            then { f ← false; BREAK();    // Verlassen der Schleife
            i ← i + 1; j ← j − 1;
     if f = true    // Treffer gefunden?
       then { i ← (n − k);
              while v[i] = v[p]
                do i ← i + 1;
              s ← ((n − k) + (i − (n − k)));    // Vielfachheit bestimmen
              L PUSH_BACK((v[p] s));
              for i = 0 to (s − 1)    // Listenzeiger für Trefferlisten verschieben
                do { if lp[i] ≥ 0
                     then { lp[i] ← lp[i] + 1;
                            if lp[i] = |G'_{c_i}|
                              then { lp[i] ← −1;
                                     κ ← κ + 1;
       else { for i = 0 to (p − 1)    // Listenzeiger verschieben
              do { (h d) ← v[p];
                   j ← π(i);    // i bzgl. der Permutation π
                   v' ← (h g_j d);    // Re-Justierung zum Vergleich mit unjust. Elementen aus G'_{c_j}
                   while lp[j] ≥ 0 and G'_{c_j}[lp[j]] < v'
                     do { lp[j] ← lp[j] + 1;
                          if lp[j] = |G'_{c_j}|
                            then { lp[j] ← −1;
                                   κ ← κ + 1;
                   if lp[j] ≥ 0 and G'_{c_j}[lp[j]] < v'
                     then { lp[j] ← lp[j] + 1;
                            if lp[j] = |G'_{c_j}|
                              then { lp[j] ← −1;
                                     κ ← κ + 1;
return (L);
```

Algorithmus 3.2.2: Algorithmus zur Bestimmung der k-Fehlstellentreffer basierend auf einem sortiertem Array.

Bei der k-Fehlstellentreffersuche werden die Elemente der kürzesten $k+1$ Listen zunächst justiert. Für jedes Listenelement (g, i) wird dann geprüft, ob das Element bereits in der Baumdatenstruktur vorhanden ist. Ist dies nicht der Fall, wird das Listenelement in den Baum eingefügt und der zugehörige Vielfachheitenzähler auf Eins gesetzt. Andernfalls wird der bereits in der Datenstruktur vorhandene Vielfachheitenzähler um Eins erhöht. Nachdem $k + 1$ viele Listen bearbeitet worden sind, enthält der Baum alle möglichen Trefferkandidaten. Daher werden bei der Bearbeitung aller weiteren Listen keine neuen Elemente in den Baum mehr eingefügt. Hat ein Vielfachheitenzähler zu einem (g, i) den Trefferschwellwert $|Q| - k$ erreicht, ist (g, i) ein Element der Treffermenge. Allerdings können die tatsächlichen Vielfachheiten der Trefferelemente erst dann aus der Baumdatenstruktur abgelesen werden, nachdem alle Listen vollständig abgearbeitet worden sind.
Ein großer Nachteil dieses Verfahrens besteht darin, dass viel Rechenzeit darauf verwendet wird, die Baumdatenstruktur nach Einfügeoperationen zu aktualisieren. Ausserdem kann die Suche nicht nach der Bearbeitung von $k + 1$ Listen vollständig abgebrochen werden. Lediglich entfallen dann die Einfügeoperationen von neuen Elementen in die Datenstruktur, so dass mit logarithmischem Aufwand der zu einem (g, i) gehörige Zähler in dem Baum gefunden und erhöht werden kann. Es müssen jedoch für alle Elemente der zu der Anfrage Q korrespondierenden G-invertierten Listen die jeweils bzgl. Q justierten Elemente berechnet werden.

3.3.2 Berechnung von $G_{\mathcal{D}\,k}(Q)$ mittels Hashing

In diesem Abschnitt wird ein Suchverfahren vorgestellt, welches mittels einer Hashfunktion die dynamische Datenstruktur des zuvor beschriebenen Verfahrens ersetzt. Prinzipiell kommen alle $g \quad G$ eingeschränkt bzgl. der Dokumentenkollektion $\mathcal{D}$ als Bestandteil möglicher Trefferkandidaten zu einer Anfrage Q in Frage. Da dies in Abhängigkeit von Q immer noch unendlich viele g sein können, nutzen wir die Hashfunktion, um Trefferkandidaten ganzzahlige Werte aus einem zuvor vorgegebenen Intervall zuzuweisen. Dann können wir den Hashwert eines Trefferkandidaten als Adresse in einem Array ansehen. Es entfällt dadurch der Verwaltungsaufwand für eine dynamische Datenstruktur, was eine deutliche Beschleunigung im Vergleich zum vorhergehenden Verfahren nach sich ziehen kann. Dies ist abhängig von der Definition der Hashfunktion, der Anzahl unterschiedlicher Hashwerte und dem Datenbestand.
Wir definieren zunächst eine Hashfunktion $H\colon G \times [1 : N] \qquad [0 : M - 1]$, die einem Tupel und möglichen Treffer (g, i) einen ganzzahligen Hashwert im Bereich von 0 bis $M-1$ zuordnet. Natürlich sollte die Berechnung von $H(g, i)$ effizient möglich sein, da die Hashfunktion an allen Elementen aus den $k + 1$ kürzesten G-invertierten Listen ausgewertet werden muss. Der Wert von M hängt direkt von der Anzahl der Listenelemente in den $k + 1$ kürzesten Listen ab.
Genauer arbeiten wir mit einem Array v der Länge M, wobei jedem $v[j]$ zunächst eine leere Liste zugewiesen wird. Später enthält eine solche Liste Tripel der Art (g, i, c). Dabei bezeichne die Komponente c die Vielfachheit des Tupels (g, i). Zunächst werden die $k + 1$ kürzesten Listen bzgl. Q justiert. Zu jedem Element (g, i) aus diesen Listen wird der Hashwert $h := H(g, i)$ gebildet und in der Liste $v[h]$ nach einem Tripel der Form (g, i, c) gesucht. Dabei beschränkt sich der Vergleich auf die ersten beiden Komponenten. Gibt es bereits ein solches Tripel in $v[h]$, so wird der Wert von c um Eins erhöht. Ansonsten wird das Tripel $(g, i, 1)$ der Liste $v[h]$ angefügt. Nachdem die kürzesten $k + 1$ Listen vollständig abgearbeitet worden sind, können keine weiteren Listeneinträge hinzukommen, da diese niemals die geforderte Vielfachheit $n - k$ erreichen können. Somit geben die Elemente der restlichen $n - k - 1$ Listen höchstens Anlass zur Erhöhung der Vielfachheiten.
Die Vorteile dieses Verfahrens liegen in der Einfachheit seiner Implementierung und in der Anwendbarkeit im Falle von nicht angeordneten Gruppen. Allerdings ist anzumerken, dass die Initialisierung

des Vektors v vor jeder Suche ggf. viel Zeit in Anspruch nehmen kann. Daher macht es Sinn, diesen Schritt durch einen weiteren Prozess im Hintergrund durchzuführen. Die Streuung der verwendeten Hashfunktion kann dazu führen, dass zu einem Hashwert h eine Vielzahl von Kandidaten gespeichert werden müssen. Dies hat zur Folge, dass bei einem sehr wahrscheinlichen erneuten Auftreten des gleichen Hashwertes v eine Vielzahl von Trefferkandidaten mit den Einträgen in der Liste $v[h]$ verglichen werden müssen. Im Hinblick auf die Verwendung von Gruppen, deren Verknüpfung einen hohen Berechnungsaufwand erfordert, hat dieses Verfahren den Nachteil, dass für alle Einträgen in allen bzgl. Q beteiligten G-invertierten Listen eine solche Verknüpfung berechnet werden muss, bevor ein Treffer feststeht. Eine Vorauswahl von zu betrachtenden Listenelementen ist nicht möglich, da dazu zunächst der Hashwert eines justierten Listenelements vorliegen muss.

3.3.3 Berechnung von $G_{\mathcal{D}\,k}(Q)$ mittels Hyperwürfel

In einem vorhergehenden Abschnitt haben wir eine Baumdatenstruktur verwendet, in der die Trefferkandidaten und deren Vielfachheiten gespeichert wurden. Der Grund dafür ist die Tatsache, dass wir in der Regel mit Gruppen arbeiten, die unendlich viele Elemente beinhalten, so dass es unmöglich ist, entsprechend viele Vielfachheitenzähler wie es mögliche Tupel (g, i) gibt zu betrachten. Allerdings beobachten wir, dass der Suchindex zu einer Dokumentenkollektion $\mathcal{D}$ den Wertebereich für Tupel (g, i) einschränkt. Im Falle der Suche in MIDI-Partituren haben wir es mit N Dokumenten zu tun, wobei es zusätzlich eine durch $\mathcal{D}$ vorgegebene Note gibt, deren Einsatzzeit t_{max} in Bezug zu allen anderen Noten in $\mathcal{D}$ maximal ist. Somit kommen hier nur Trefferkandidaten in Frage, welche eine Translation im Bereich $[0 : t_{\text{max}}]$ beschreiben, wenn wir davon ausgehen, dass die erste Note einer Anfrage Q beim Zeitpunkt 0 beginnt. Sind t_{max} und N hinreichend klein, so können wir durchaus eine Matrix im Hauptspeicher nutzen, um sehr schnell die Vielfachheiten von Tupeln (g, i) zu bestimmen, da hier ein (g, i) einen direkten Verweis in die Vielfachheitenmatrix darstellt. In diesem Fall sind die k-Fehlstellentreffer sehr schnell zu bestimmen, da letztendlich die Gruppenverknüpfung einen mehr oder weniger großen Rechenaufwand verursacht und der Zugriff auf die Einträge der Matrix in konstanter anstelle von logarithmischer Zeit möglich ist. Eine aufwändige Aktualisierung einer Baumdatenstruktur entfällt ganz (vgl. Abbildung 3.3 auf Seite 40).

Ein rein technischer Vorteil ist auch, dass dieses Verfahren leicht zu parallelisieren ist. So könnten unterschiedlichen Prozessen bei der Abarbeitung von G-invertierten Listen disjunkte Teilintervalle von Dokumenten-IDs zugewiesen werden, was dazu führt, dass keine Synchronisation der Prozesse notwendig ist, d.h. eine Verdopplung der Anzahl der Prozessoren führt zu einer Halbierung der Laufzeit des Suchalgorithmus. Auch wenn es nicht notwendig ist, dass die Gruppenverknüpfung die Anordnung der Elemente in den G-invertierten Listen erhält, so kann dies jedoch von Vorteil sein, da aufgrund der Sortierung der Elemente in den G-invertierten Listen die Prozessor-Caches beim Zugriff auf Elemente des Hyperwürfels optimal genutzt werden können, denn aufgrund der Sortierung sind nur wenige „Cache-Page-Misses“ zu erwarten.

In den nachfolgenden Abschnitten wird eine Technik vorgestellt, die dieses Beispiel als Grundlage verwendet, jedoch das Verfahren anwendbar macht, wenn die Gruppe durch mehrdimensionale Vektoren über $\mathbb{R}$ oder $\mathbb{Z}$ parametrisiert wird. Prinzipiell wird hierbei zunächst die Auflösung des Suchverfahrens reduziert, d.h. nach einem ersten Suchschritt sind eine Reihe von Trefferkandidaten bestimmt worden, die in nachfolgenden Schritten bei höherer Auflösung und gleichzeitiger Beschränkung des Suchraums auf kleine Bereiche durch das Verfahren berechnet werden können.

3.3.3.1 Gruppenquantisierung

Unter anderem aus Speicherplatzgründen ist man daran interessiert, die Anzahl der während der Suche justierten und zu speichernden Gruppenelemente drastisch zu reduzieren. Ein bekanntes Beispiel aus dem Bereich der reellen Zahlen beschreibt eine sinnvolle Vorgehensweise, die wir später verallgemeinern werden. Dabei handelt es sich um den Abrundungsoperator, der jeder reellen Zahl x die größte ganze Zahl kleiner oder gleich x zuordnet: $x \quad x$. Dies kann man gruppentheoretisch so interpretieren: Die additive Gruppe $\mathbb{R}$ wurde mit Hilfe der Untergruppe $\mathbb{Z}$ sowie des Nebenklassenrepräsentantensystems (kurz: Transversale) $T = [0, 1)$ partitioniert:

$$\mathbb{R} = \bigcup_{t \in \mathbb{Z}} t + [0, 1).$$

Jedes Element von $\mathbb{R}$ liegt in genau einer geeignet verschobenen Version $T + t$ mit $t \quad \mathbb{Z}$. Abrundung bedeutet: Jedem $x \quad T + t$ wird als vergröbernder Wert die Zahl $t = x$ zugeordnet. Die n-dimensionale Abrundung basiert auf

$$\mathbb{R}^n = \bigcup_{t \in \mathbb{Z}^n} t + [0, 1)^n.$$

Wir gehen von einer Kollektion $\mathcal{D} = (D_0, \ldots, D_{N-1})$ aus, wobei die Dokumente D_i endliche Teilmengen von $M = \mathbb{Z}$ sind. Desweiteren operiere die Gruppe G auf M, wobei die Gruppenelemente selbst durch Elemente von $\mathbb{Z}^p$ parametrisiert sind. Beispielsweise könnte G die additive Gruppe $\mathbb{Z}^p$ sein, oder aber das semidirekte Produkt von $\mathbb{Z}^{p-1}$ mit den Drehungen um Vielfache von $90°$. Allgemein fordern wir, dass sämtliche Stabilisatoren der G-Operation endlich sind.
Identifizieren wir g mit seiner ganzzahligen Parametrisierung, $g = (g_1, \ldots, g_p)$, so bietet sich das ganzzahlige Tupel $(g_1, \ldots, g_p, i)$ als Verweis in ein Array V an, in der die Vielfachheit von (g, i) bezüglich einer Anfrage Q aufgebaut wird. Da die Dokumente und alle Stabilisatoren endlich sind, gibt es minimale $(c_1 - 1, \ldots, c_p - 1) \quad \mathbb{Z}^p_{\geq 0}$, so dass o.E. jedes Paar (g, i) aus einer invertierten Liste durch ein Element aus dem ganzzahligen Hyperwürfel $\mathcal{C} = [0 : c_1 - 1] \times \ldots \times [0 : c_p - 1] \times [0 : N-1]$ parametrisiert wird. Der Übergang von der Parametrisierung $(g_1, \ldots, g_p, i)$ zur entsprechenden Adresse $A(g, i)$ in der Datenstruktur ist (bzgl. p) in konstanter Zeit zu bestimmen:

$$A(g, i) = g_1 + \sum_{k=2}^{p} g_k \prod_{j=1}^{k-1} c_j + i \prod_{j=1}^{p} c_j.$$

Für $p \gg 1$ lässt sich die Berechnung von $A(g, i)$ effizient realisieren, indem man das Produkt $\pi_{k-1} := \prod_{j=1}^{k-1} c_j$ zwischenspeichert, um durch Multiplikation mit c_k das nächste Produkt π_k zu berechnen.
Nun wenden wir uns der Berechnung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ zu einer Anfrage $Q = \{q_1, \ldots, q_n\}$ der Länge n zu. Zunächst wird das Array V an allen Stellen mit 0 initialisiert. Dann berechnen wir zu jedem q_j die Darstellung $q_j = g_j r_j$, wobei $g_j \quad G$ und r_j der zu q_j gehörige Bahnrepräsentant ist. Sei nun G'_j die zu dem Bahnrepräsentanten r_j korrespondierende G-invertierte Liste. Für alle Elemente $(g', i) \quad G'_j$ berechnen wir nun die Adresse $a = A(g' g_j^{-1}, i)$ und erhöhen den Wert $V[a]$ um Eins. Ein Paar (g, i) mit $A(g, i) = a$ ist genau dann ein Treffer, wenn zu einem Zeitpunkt $V[a] = n - k$ gilt. Ist $V[a] = n - k$, wird das Tupel (g, i) der Resultatliste L hinzugefügt, obwohl der endgültige Wert von $V[a]$ im Fall $k > 0$ zum Einfügezeitpunkt noch nicht bekannt ist. Dies erspart ein abschließendes Testen *aller* $N \quad \prod_{i=1}^{p} c_i$ vielen Elemente in V auf $V[a] \geq n - k$. Dieses Vorgehen ist vorzuziehen, da der einfacherere Gleichheitstest $V[a] = n - k$ nur $\quad_j |G_j|$ mal durchgeführt werden muss. Die korrekte Vielfachheit des Treffers $(g, i) \quad L$ kann man, nachdem alle Listenelemente aus den G-invertierten Listen zur Anfrage Q abgearbeitet worden sind, am Eintrag $V[A(g, i)]$ ablesen.

Beispiel 3.2 Bei der Audioidentifikation operiert die Gruppe der ganzzahligen Translationen auf Elementen aus $\mathbb{Z}$. Hier ist $\mathcal{C} = [0 : t_{\max}] \times [0 : N-1]$. Dabei bezeichne $t_{\max}$ die maximale Zeitposition eines Eintrags (g, i) in einer der G-invertierten Listen zu den Bahnrepräsentanten.

3.3.3.2 Reduktion der Elemente in $\mathcal{C}$

In konkreten Anwendungen werden die Hyperwürfel bei selbst kleiner Anzahl von Dimensionen sehr groß und bei der Suche beinhalten die meisten Einträge in V den initialen Wert Null. Am Beispiel der Audioidentifikation soll dies verdeutlicht werden. Hier ist der zugehörige Hyperwürfel 2-dimensional. Bei einer hohen Zeitauflösung wäre der Hyperwürfel z.B. eine 10.000×200.000-Matrix. Selbst bei nur einem Byte Speicherplatzbedarf pro Matrixeintrag würde diese Matrix $2 \cdot 10^9$ Bytes (rund 2 GB) Speicherplatz belegen. In der Praxis würde dies Anfragen auf Längen unter 256 beschränken, da keine größeren Vielfachheiten gezählt werden können. Viel wichtiger jedoch ist, dass die Initialisierung einer solch großen Matrix im Vergleich zur eigentlichen Suche zu lange dauern würde, so dass der Vorteil des Zugriffs auf Elemente des Hyperwürfels in konstanter Zeit schnell verloren wäre. Je nach Länge der Anfrage Q und der Anzahl der Elemente in den zu berücksichtigenden Listen G_j würden auch nur sehr wenige Einträge in der Matrix einen Wert ungleich Null aufweisen.
Aus diesem Grund werden die Intervalle $[0 : c_j]$ in $\lfloor c_j / \rho_j \rfloor$ viele, etwa gleich große Intervalle partitioniert, wobei ρ_j die *Auflösung* bzgl. der j-ten Dimension spezifiziert. Im Gegesatz zu $\mathcal{C}$ besteht der so entstandene Hyperwürfel $\overline{\mathcal{C}}$ aus Folgen von Intervallen anstelle von Zahlen. Den Vektor $R = (\rho_1, \ldots, \rho_{p+1})$ bezeichnen wir als *Auflösungsvektor*. Im folgenden arbeiten wir nicht mit Intervallfolgen, sondern identifizieren jedes Intervall mit seiner unteren Grenze, wodurch wir wieder im obigen Fall von Zahlenfolgen sind. Allerdings ist jetzt der neue Hyperwürfel wesentlich kleiner.
Um nun die vergröberte Treffermenge zu bestimmen, werden wie zuvor die zu beachtenden G-invertierten Listen entsprechend den Elementen der Anfrage Q justiert. Sei $(g, i) \equiv (g_1, \ldots, g_p, i)$ ein solches Listenelement, welches bzgl. q_j justiert worden ist. Die folgende Formel beschreibt die vergröberte Version (g', i') von (g, i) bzgl. obiger Auflösung:

$$(g', i') = (\ \lfloor g_1 / \rho_1 \rfloor, \ldots, \lfloor g_p / \rho_p \rfloor,\ \lfloor i / \rho_{p+1} \rfloor\ ).$$

Nun können wir mit der zuvor bereits definierten Funktion A die Adressen von (g', i') in $\overline{\mathcal{C}}$ bestimmen und wie zuvor im Fall $\mathcal{C}$ die Vielfachheiten der in den G-invertierten Listen zu den Repräsentanten gespeicherten Paare (g', i') berechnen. $\overline{\mathcal{C}}(g', i') = V[A(g', i')] \geq n - k$ stellt eine *notwendige* Bedingung für einen $(G, \mathcal{D}, k)$-Treffer dar. In einem nachfolgenden Schritt muss für alle Adressen a mit $V[a] \geq n - k$ geprüft werden, ob tatsächlich ein $(G, \mathcal{D}, k)$-Treffer bzgl. einer höheren Auflösung vorliegt.
Allgemein können wir jedes ganzzahlige Tupel $(g_1, \ldots, g_p, i)$ eindeutig darstellen in der Form $(k_1\rho_1 + h_1, \ldots, k_{p+1}\rho_{p+1} + h_{p+1})$ mit $k_j = \lfloor g_j / \rho_j \rfloor$ und $h_j = g_j \mod \rho_j$ für $j \leq p$. Analoges gilt im Fall $j = p+1$. Im Zuge der zuvor angesprochenen Vergröberung haben wir $(g_1, \ldots, g_p, i)$ identifiziert mit $(k_1, \ldots, k_{p+1})$, d.h. die jeweiligen Reste h_j wurden vernachlässigt. Die Gesamtheit aller Elemente aus $\mathbb{Z}^{p+1}$, die auf $(k_1, \ldots, k_{p+1})$ abgebildet werden, bilden den Hyperwürfel

$$[k_1\rho_1 : (k_1 + 1)\rho_1) \times \ldots \times [k_{p+1}\rho_{p+1} : (k_{p+1} + 1)\rho_{p+1}).$$

Beispiel 3.3 *(Fortsetzung)* Betrachten wir nochmals das zuvor angesprochene Beispiel der Audioidentifikation, bei dem der Hyperwürfel $\mathcal{C}$ eine 10.000×200.000-Matrix darstellt. Der Auflösungsvektor sei $R = (100, 1.000)$, d.h. die zu $\overline{\mathcal{C}}$ gehörige Matrix ist nur noch eine 100×200-Matrix. Ein Eintrag dieser Matrix fasst demnach 100 Dokumenten-IDs und 1.000 Zeitpositionen zusammen. Entfallen

hinreichend viele Listeneinträge (g, i) nach Vergröberung auf einen Matrixeintrag, so ist in einem nachfolgenden „Dicing“-Schritt dieser Teil-Hyperwürfel mit einer höheren Auflösung genauer zu untersuchen.

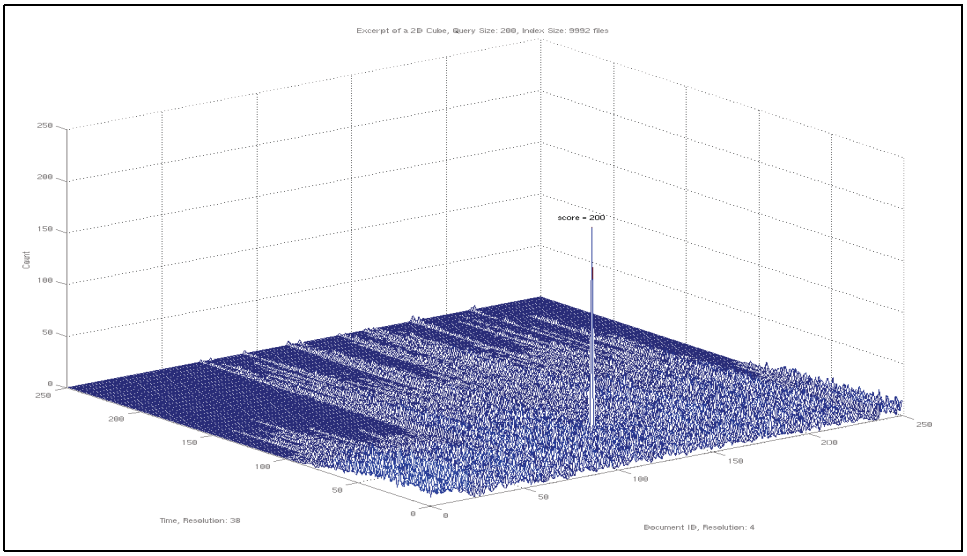


Abbildung 3.3: Die Abbildung zeigt einen Ausschnitt aus einer 2500×2500 Matrix, welche den 2-dimensionalen Hyperwürfel im Falle der Audioidentifikation darstellt. In dem Suchindex sind 9992 Dokumente enthalten, was zu einer Auflösung von vier Dokumenten pro Matrixeintrag führt. Für die Zeitachse ergibt sich aufgrund der maximalen Länge eines Musikstücks eine Auflösung von 38 Zeitpunkten pro Matrixeintrag. Bei einem Trefferschwellwert von hier 200, zeigt die Abbildung deutlich, dass es nur einen Matrixeintrag gibt, der zu einem Treffer führen könnte. Ausserdem sind deutlich die Unterschiede in den Längen der einzelnen Musikstücke zu erkennen.

Das bisher vorgestellte Vorgehen verlangt nach effizienten Methoden, den Inhalt einer G-invertierten Liste auf die Elemente einzuschränken, die einem bestimmten Hyperwürfel zugeordnet werden können. Somit sollte die Datenstruktur, in der die Elemente einer G-invertierten Liste gespeichert sind, $p+1$-dimensionale *Bereichsanfragen* unterstützen. Allerdings ist in der Praxis davon auszugehen, dass die Raumrichtung der Dokumenten-ID eine G-invertierte Liste bereits derart partitioniert, dass aus Sicht der Anwendung eine ausreichend kleine Zahl von Gruppenelementen pro Partition vorliegen.
Dies hängt auch direkt mit der Anzahl unterschiedlicher Listen im Suchindex zusammen. Je mehr G-invertierte Listen im Suchindex vorhanden sind, desto geringer wird die Anzahl von Elementen in einer Liste sein, die zu einem Dokument gehören. Geht man also von einer Sortierung der Elemente in einer G-invertierten Liste aus, die zunächst nach Dokumenten-IDs sortiert und dann in der Reihenfolge der Dimensionen, so können die Elemente in einer G-invertierten Liste, die zu einem bestimmten Teil des Hyperwürfels gehören, hinreichend schnell gefunden werden. Die Dokumenten-ID bietet sich besonders an, da diese unter der Gruppenoperation invariant ist.

Beispiel 3.4 *(Fortsetzung)* Die G-invertierten Listen, die bei der Audioidentifikation verwendet werden, speichern aufsteigend sortiert die in der Liste vorhandenen Dokumenten-IDs. Zu jeder dieser IDs werden dann ebenfalls aufsteigend sortiert die Zeitpositionen gespeichert, an denen das jeweilige Merkmal im Dokument vorkommt. Zusätzlich dazu wird in einer weiteren Liste festgehalten, welche

Dokumenten-IDs in der G-invertierten Liste vorkommen und an welcher Position im Speicher die Daten zum Dokument mit einer bestimmten ID beginnen. Auf diese Weise können die zur Beantwortung einer Bereichsanfrage gehörenden Gruppenelemente schnell bestimmt werden. Nebenbei spart diese Speicherung auch Speicherplatz, da eine Dokumenten-ID nur einmal pro G-invertierter Liste abgelegt werden muss. Zusammen mit differenzieller Speicherung der Gruppenelemente und dem damit verbundenen geringerem Speicherplatzbedarf pro Gruppenelement, belegt so zum Beispiel ein Suchindex für rund 10.000 Musikstücke nur 40 MB Hauptspeicher.

Begnügt man sich mit der zuvor im Beispiel beschriebenen Partitionierung der G-invertierten Listen aufgrund der Dokumenten-ID, so können die G-invertierten Listen in einfachen Feldern gespeichert werden. Es sind also keine komplizierten Datenstrukturen wie Bäume etc. notwendig, was ebenfalls der Ausführungsgeschwindigkeit zu Gute kommt, denn das lineare Durchlaufen von Feldern wird auch von den Cache-Mechanismen der Harware sehr gut unterstützt.
Im Vergleich zur Schnittmengenberechnung ist dieses Verfahren auch sehr leicht parallelisierbar. Zusammen mit der Partitionierung der Listen aufgrund der Dokumenten-ID lässt sich erreichen, dass die einzelnen parallel arbeitenden Prozesse *keiner* Synchronisierung bedürfen, da auf die G-invertierten Listen nur lesend zugegriffen wird und bei auflösungsabhängiger Wahl des zu bearbeitenden Dokumenten-ID-Intervalls niemals zwei Prozesse in die gleiche Speicherstelle des Hyperwürfels schreiben werden.

3.3.3.3 Ef ziente Durchführung von Detailanalysen

Es wurde ja zuvor bereits angedeutet, dass die Initialisierung eines Hyperwürfels zur Vielfachheitenzählung einen großen Zeitaufwand darstellen kann. Bei der Verwendung von Hyperwürfeln zur Treffermengenberechnung wird zunächst eine vergröberte Trefferkandidatenmenge berechnet. Bei der Audioidenfikation kann nach dem ersten Suchschritt z.B. gesagt werden, dass sich ein Treffer in den Dokumenten 10 bis 19 und im Bereich von 2000 bis 2099 auf der Zeitachse (Samples, Frames etc.) befinden kann. In der Regel ergibt sich auf diese Weise eine Liste von näher zu untersuchenden Teilen des Hyperwürfels.
Um nun für einen solchen Trefferkandidaten zu entscheiden, ob sich in dem Bereich des Lösungsraums wirklich ein Treffer befindet, wird das Verfahren rekursiv angewendet. Dabei werden nur die Elemente aus den G-invertierten Listen benötigt, die justiert in eben diesen Teil des Lösungsraums fallen, der gerade von Interesse ist. Für alle Trefferkandidaten müssen im Prinzip zwei Schritte durchgeführt werden:

1. Initialisierung eines Hyperwürfels $\mathcal{B}$ mit den Ausmaßen eines Teilwürfels mit Nullwerten zur Vielfachheitenzählung,

2. Durchführung des Vielfachheitenzählung.

Je nachdem wie viele Einträge der verwendete Hyperwürfel aufweist, kann dessen Initialisierung mit Nullwerten die Suche insgesamt verlangsamen. Bei Mehrprozessorsystemen könnte man das Design Pattern „Factory“ wie in [GHRV94] beschrieben benutzen, um eine Referenz auf eine bereits initialisierte Hyperwürfel-Datenstruktur zu erhalten. Es ist die Aufgabe der „Factory“, die Initialisierung durchzuführen, was z.B. in einen eigenständigen Thread ausgelagert werden kann.
Bei Systemen mit nur einem Prozessor(-kern), ist dieses Vorgehen nicht sinnvoll. In diesem Fall kann ein weiterer Hyperwürfel $\mathcal{B}'$ gleicher Gestalt, der mit Nullwerten initialisiert wurde, genutzt werden, um zu protokollieren, welche Zellen in $\mathcal{B}$ einer Initialisierung bedürfen. Wir nehmen dazu an, die zu

überprüfenden Trefferkandidaten seien aufsteigend von 1 an nummeriert. Wird gerade der i-te Trefferkandidat betrachtet, muss lediglich vor dem Erhöhen eines Vielfachheitenzählers in $\mathcal{B}$ geprüft werden, ob an der gleichen Position in $\mathcal{B}'$ ein Wert ungleich i steht. Ist dies der Fall, ist der jeweilige Vielfachheitenzähler in $\mathcal{B}$ auf den Wert 1 zu setzen und in $\mathcal{B}'$ an der gleichen Stelle der Wert i einzutragen. Ansonsten wird der jeweilige Vielfachheitenzähler in $\mathcal{B}$ um 1 erhöht.
Obwohl dieses Vorgehen eine Abfrage der Form $\mathcal{B}'[a] = i$ für jedes zu betrachtende justierte Element (g, i) erfordert, ist dieses Verfahren vorzuziehen, da zum einen die Überprüfung auf Ungleichheit effizient umgesetzt werden kann und zum anderen die Anzahl der Initialisierungen deutlich reduziert wird.

3.3.3.4 Vorberechnung von Listen im Modulo-Raster

Wie wir zuvor gesehen haben, kommt es bei einer geringen Auflösung des Hyperwürfels dazu, dass zu unterschiedlichen Einträgen in der gleichen G-invertierten Liste die gleiche Adresse berechnet wird. Es stellt sich daher die Frage, ob es möglich ist, die G-invertierten Listen vorab geeignet vergröbert abzuspeichern, um auf diese Weise die Anzahl der Elemente in den zu betrachtenden Listen zu reduzieren. Dies würde nicht nur dazu führen, dass der Zugriff auf die Liste an sich schneller möglich ist, auch die Anzahl der durchzuführenden Justierungen wird reduziert. Ausserdem entspricht es auch mehr der Philosophie einer gering aufgelösten Suche, wenn man dazu nicht die Detaildaten verwenden muss.
Dieser Abschnitt befasst sich zunächst theoretisch mit den notwendigen Voraussetzungen, um dann das zuvor angesprochene Beispiel der Audiosuche fortzuführen.

Sei U eine Untergruppe von G von endlichem Index $[G : U]$, d.h. mit einer endlichen Zahl von Nebenklassen, was zu einem endlichen Nebenklassenrepräsentantensystem Γ (mit o.B.d.A. $1 \quad \Gamma$) führt. Wir können G darstellen als disjunkte Vereinigung

$$G = \bigcup_{\gamma \in \Gamma} U\gamma.$$

Weiterhin ist jedes $g \quad G$ eindeutig darstellbar als $g = u \cdot \gamma$ mit $u =: g \operatorname{div}(U, \Gamma) \quad U, \gamma =: g \mod (U, \Gamma) \quad \Gamma$. Zur Abkürzung definieren wir $\operatorname{div}(g, i) := (g \operatorname{div}(U, \Gamma), i \operatorname{div} \rho)$. Dabei sei $i \operatorname{div} \rho = i \operatorname{div}(\rho\mathbb{Z}, [0 : \rho - 1])$. Wir können anschaulich davon sprechen, dass die Elemente in U *vergröberte* Darstellungen der Elemente in G sind. Im folgenden seien alle Stabilisatoren der G-Menge $M = \bigcup_{r \in R} Gr$ trivial und $\rho \quad \mathbb{Z}_{\geq 1}$ ein Vergröberungsparameter für die Dokumenten-IDs. Dann können wir zu $h \quad G$ und $q \quad M$ eine Funktion

$$G^h_{\mathcal{D}}(q) \colon U \times [0 : N - 1 \operatorname{div} \rho] \quad \mathbb{Z}_{\geq 0}$$

als Summe von Indikatorfunktionen zu vergröberten G-invertierten Listen wie folgt definieren:

$$G^h_{\mathcal{D}}(q) := \sum_{(g\ i) \in G_{\mathcal{D}}(q)} \chi_{\operatorname{div}(gh^{-1}\ i)}.$$

Im Fall $h = 1$ verzichten wir auf die Angabe des Parameters h und schreiben kurz $G_{\mathcal{D}}(q)$.

Lemma 3.1 *Für alle $h \quad G$ und alle $q \quad M$ gilt $G_{\mathcal{D}}(hq) = G^h_{\mathcal{D}}(q)$.*

Beweis: Es gilt

$$G_{\mathcal{D}}(hq) = \sum_{(g\ i)\in G_{\mathcal{D}}(hq)} \chi_{\mathrm{div}(g\ i)} = \sum_{(k\ i)\in G_{\mathcal{D}}(q)} \chi_{\mathrm{div}(kh^{-1}\ i)} = G^h_{\mathcal{D}}(q).$$

□

Lemma 3.2 *Für alle $h = \gamma u$ gilt $G_{\mathcal{D}}(hq) = G^{u^{-1}\gamma u}_{\mathcal{D}}(q)u^{-1}$.*

Beweis: Es gilt

$$\begin{aligned} G_{\mathcal{D}}(hq) &= G_{\mathcal{D}}(\gamma(uq)) = G^{\gamma}_{\mathcal{D}}(uq) \\ &= \sum_{(g\ i)\in G_{\mathcal{D}}(uq)} \chi_{\mathrm{div}(g\gamma^{-1}\ i)} \\ &= \sum_{(k\ i)\in G_{\mathcal{D}}(q)} \chi_{\mathrm{div}(ku^{-1}\gamma^{-1}\ i)} \\ &= \sum_{(k\ i)\in G_{\mathcal{D}}(q)} \chi_{\mathrm{div}(k(u^{-1}\gamma^{-1}u)u^{-1}\ i)} \\ &= \sum_{(k\ i)\in G_{\mathcal{D}}(q)} \chi_{\mathrm{div}(k(u^{-1}\gamma u)^{-1}\ i)u^{-1}} \\ &= G^{u^{-1}\gamma u}_{\mathcal{D}}(q)u^{-1}. \end{aligned}$$

□

Ist Γ invariant unter Konjugation mit Elementen aus U, so genügt es aufgrund des letzten Satzes lediglich alle Listen der Form $G^{\gamma}_{\mathcal{D}}(r)$ vorzuberechnen und abzuspeichern. Damit beträgt die Anzahl der zu speichernden Listen $|R| \cdot |\Gamma|$. Obige Invarianz ist z.B. bei abelschen Gruppen, bei direkten Produkten sowie bei semidirekten Produkten gegeben. Die Invarianz ist auch lokal immer dann trivialerweise gültig, wenn der Zentralisator von $\gamma \quad \Gamma$ in U identisch U ist.

Damit können wir nun den zentralen Satz formulieren, auf dem unser Ansatz beruht. Wie zuvor bei den G-invertierten Listen, müssen wir auch hier nur eine begrenzte Anzahl von Listen vorberechnen und speichern. Aus diesen Listen können wir dann mittels Justierung die Darstellung gewinnen, die wir zur Berechnung von $G_{\mathcal{D}}(Q)$ benötigen. Anzumerken ist, dass die vergröberten Listen weniger Elemente enthalten, so dass die Anzahl der Justierungen geringer wird.

Satz 3.1 *Die Invarianz von Γ unter der U-Konjugation vorausgesetzt, gilt*

$$G_{\mathcal{D}}(Q) := \sum_{q\in Q} G_{\mathcal{D}}(q) = \sum_{q\in Q} G^{\gamma_q}_{\mathcal{D}}(r_q)u_q^{-1}.$$

Beweis: Die Behauptung folgt direkt aus dem Lemma 3.2. □

3.3.3.4.1 Beispiel: Suche in Audiodaten In diesem Abschnitt greifen wir das Beispiel aus dem Abschnitt über die Suche mittels Hyperwürfel auf. Die Gruppe $G = (\mathbb{Z}, +)$ operiert auf $\mathbb{Z}$. In einem ersten Suchschritt interessieren wir uns für Trefferkandidaten, die eine um $n = 3$ reduzierte Zeitauflösung aufweisen. Konzentrieren wir uns hier zunächst nur auf die Zeitachse und vernachlässigen die DokumentenID vorerst.

Die Auflösungsreduktion modellieren wir durch die Untergruppe $U := 3\mathbb{Z} < \mathbb{Z}$. Da $G = (\mathbb{Z}, +)$ abelsch ist, können wir das Ergebnis aus Satz 3.1 anwenden. Das NKLRS Γ ist die Menge $\{0, 1, 2\}$. Ein beliebiges Gruppenelement $g \quad \mathbb{Z}_{\geq 0}$, welches in einem Tupel in der G-invertierten Liste $G_{\mathcal{D}}(0, c)$ enthalten ist, lässt sich schreiben als $g = u + \gamma, u \quad U, \gamma \quad \Gamma$, wobei $u = g \operatorname{div} 3$ und $\gamma = g \mod 3$ gilt.
Sei nun $G_{\mathcal{D}}(r) = \{1, 2, 3, 5, 7, 8, 11, 13\}$ eine G-invertierte Liste aus dem Suchindex. Eine Vergröberung der Zeitauflösung um 3 führt zu der Multi-Menge $\{0, 0, 1, 1, 2, 2, 3, 4\}$. Diese schreiben wir im folgenden mit Hilfe von Tupeln, wobei die erste Komponente den Wert $u \quad U$ bezeichnet und der zweite Wert die Vielfachheit des Wertes u in der Multimenge. Damit können wir nun die Funktionen $G^{\gamma}_{\mathcal{D}}(r)$ dadurch beschreiben, indem wir die Stellen angeben, an denen die Funktion einen Wert ungleich Null hat:

$$\begin{aligned} G^0_{\mathcal{D}}(r) &= \{(0,2),(1,2),(2,2),(3,1),(4,1)\} \\ G^1_{\mathcal{D}}(r) &= \{(0,3),(1,1),(2,2),(3,1),(4,1)\} \\ G^2_{\mathcal{D}}(r) &= \{(-1,1),(0,2),(1,2),(2,1),(3,2)\} \end{aligned}$$

Im Fall $\gamma = 2$ erhalten wir Elemente aus U mit negativem Vorzeichen. Da wir jedoch den Beginn einer Anfrage immer am Nullpunkt der Zeitachse ansiedeln können, sind wir nur an Translationen mit positivem Vorzeichen interessiert. Aus diesem Grund könnte man das erste Element aus $G^2_{\mathcal{D}}(r)$ hier auch entfernen. Betrachtet man nun

$$G^3_{\mathcal{D}}(r) = \{(-1,2),(0,2),(1,2),(2,1),(3,1)\},$$

ist sofort zu sehen, dass die jeweiligen Vielfachheiten denen in $G^0_{\mathcal{D}}(r)$ entsprechen. Lediglich die jeweils erste Komponente der Elemente von $G^3_{\mathcal{D}}(r)$ ist um 1 verringert.

Betrachten wir nun ein $q = (7, r)$ aus einer Anfrage Q. Zunächst haben wir die Elemente aus $G_{\mathcal{D}}(r)$ mit -7 zu justieren und dann zu vergröbern:

$$G_{\mathcal{D}}(r) - 7 = \{-6, -5, -4, -2, 0, 1, 4, 6\}.$$

Nach Vergröberung ergibt sich

$$\{(-2,3),(-1,1),(0,2),(1,1),(2,1)\} = G^1_{\mathcal{D}}(r) - 2.$$

Dies entspricht prinzipiell der Liste $G^1_{\mathcal{D}}(r)$, wobei die Untergruppenelemente mit -2 zu justieren sind. Die Parameter ergeben sich aber auch direkt aus q durch $u = 7 \operatorname{div} 3 = 2$ und $\gamma = 7 \mod 3 = 1$. Anstelle $G^1_{\mathcal{D}}(r) - 2$ aus $G_{\mathcal{D}}(r)$ zu berechnen, genügt es die Liste $G^1_{\mathcal{D}}(r)$ vom Hintergrundspeicher zu laden und die Untergruppenelemente zu justieren. Da $G^1_{\mathcal{D}}(r)$ in der Praxis meist deutlich weniger Elemente als $G_{\mathcal{D}}(r)$ enthält, kann auf diese Weise die Zeit, die zur Anfragebearbeitung benötigt wird, deutlich reduziert werden. Nachteilig ist lediglich der deutlich höhere Speicherplatzbedarf an Hintergrundspeicher, auch wenn die Listen zu den Funktionen $G^{\gamma}_{\mathcal{D}}$ weniger Elemente enthalten.

3.3.3.5 Nutzt eine Vergröberung der Anfrage Q?

Im vorherigen Abschnitt wurde eine Technik vorgestellt, die es erlaubt, Trefferbegriffe zu niedrigerer Auflösung zu formulieren und die Trefferberechnung mit vergröberten G-invertierten Listen zu realisieren. Auf den ersten Blick mag es sinnvoll erscheinen, diese Technik auch auf eine Anfrage Q selbst anzuwenden.

Analog zu dem vorherigen Beispiel soll die Zeitauflösung der Anfrage um den Faktor n reduziert werden, um in einem ersten Schritt näherungsweise Trefferkandidaten zu bestimmen. In einem Vorverarbeitungsschritt vor der eigentlichen Suche überführen wir Q in Q, wobei letzteres die vergröberte Version darstellt. Q enthält im Gegensatz zu Q nun Quadrupel q der Form

$$q \quad Q \subset U \times \Gamma \times [0 : C-1] \times \mathbb{Z}_{>0}.$$

Die erste Komponente von q ist ein Element aus U, die zweite ein Element aus dem NKLRS Γ, das dritte ist ein Element des Repräsentantensystems R der G-Bahnen und die vierte Komponente gibt die Vielfachheit von q bzgl. Q an. Die Vielfachheit kann als Faktor bei der Bestimmung von Vielfachheiten im Hyperwürfel eingehen.
Der nachfolgende Satz besagt jedoch, dass $|Q| = |Q|$ für beliebige Anfragen Q gilt. Die Vergröberung der Anfrage liefert demnach lediglich die benötigte Darstellung $g = u\gamma$ der Elemente aus G bzgl. U. Eine gewünschte Reduktion der Anzahl der Elemente in Q gegenüber Q ist mit dieser Vorgehensweise nicht zu erreichen.

Satz 3.2 *Für eine beliebige Anfrage Q existiert kein $q \quad Q$, dessen Vielfachheit > 1 ist.*

Beweis: Seien $q = (g, c)$ und $q' = (g', c)$ aus $Q, q = q'$. Dann ist $q = (g \operatorname{div} n, g \mod n, c, 1)$ und $q' = (g' \operatorname{div} n, g' \mod n, c, 1)$. Aus den Forderungen $g \operatorname{div} n = g' \operatorname{div} n$ und $g \mod n = g' \mod n$ folgt direkt $g = g'$, was ein Widerspruch zur Annahme ist. □

3.3.3.6 Kaskadierung unterschiedlicher Au ösungen

Ausgangspunkt ist eine Kette $G = U_0 > U_1 > \ldots > U_m$ von Untergruppen U_j der Gruppe G, sowie eine Kette $\{1\} = N_0 \subset N_1 \subset \ldots \subset N_m$ von zugehörigen Rechtsnebenklassenrepräsentantensystemen N_j. Es gilt also

$$G = \bigcup_{n \in N_j} U_j n \quad \text{sowie} \quad G = \bigcup_{u \in U_j} u N_j.$$

Während die links stehende Partitionierung von G in Rechtsnebenklassen in der Gruppentheorie häufig betrachtet wird, spielt die andere Partitionierung von G in U_j-geshiftete Versionen des Rechtsnebenklassenrepräsentantensystems N_j hier eine fundamentale Rolle. Die Elemente uN_j stellen vergröberte Sichtweisen aller an uN_j beteiligten Gruppenelemente dar. Mit steigendem Index j steigt auch der Grad der Vergröberung. Jedes Gruppenelement $g \quad G$ kann bei vorgegebenem Index j in der Form $g = u_{jg} n_{jg}$ mit eindeutig bestimmten $u_{jg} \quad U_j$ und $n_{jg} \quad N_j$ geschrieben werden. Dies ist Grundlage der Vergröberungsabbildung $_j \colon G \quad \{uN_j \mid u \quad U_j\}$, die durch $_j(g) := u_{jg} N_j$ definiert ist.
Eine Anfrage Q bearbeiten wir nicht sofort über den Listenschnitt aller $G_{\mathcal{D}}(q), q = g_q r_q \quad Q$, sondern wir bilden zunächst zu j die vergröberte „Treffermenge“

$$G^j_{\mathcal{D}}(Q) := \bigcup_{q \in Q} \{(\ _j(h g_q^{-1}), i) \mid (h, i) \quad G_{\mathcal{D}}(r_q)\}.$$

Bei der obigen Schnittmengenbildung ist zu beachten, dass zwei verschobene Versionen von N_j entweder gleich oder disjunkt sind. Mit $[G^j_{\mathcal{D}}(Q)]$ sei die Menge aller Paare (g, i) bezeichnet, zu denen es ein Element $(uN_j, i) \quad G^j_{\mathcal{D}}(Q)$ gibt. Damit gilt folgender Satz.

Satz 3.3 *Für eine Anfrage Q gilt*

$$G_{\mathcal{D}}(Q) = [G^0_{\mathcal{D}}(Q)] \subseteq [G^1_{\mathcal{D}}(Q)] \subseteq \ldots \subseteq [G^m_{\mathcal{D}}(Q)].$$

Beweis: Sei $(g,i) \quad G_{\mathcal{D}}(Q)$. Dann gilt $(g,i) \quad G_{\mathcal{D}}(q) = G_{\mathcal{D}}(r_q)g_q^{-1}$ für alle $q \quad Q$. Also existiert $(h_q,i) \quad G_{\mathcal{D}}(r_q)$ mit $(g,i) = (h_q g_q^{-1}, i)$. D.h. $(\ _j(g),i) = (\ _j(h_q g_q^{-1}),i)$ liegt in $G_{\mathcal{D}}^j(Q)$. Wegen $g \quad _j(g)$ liegt (g,i) in $\{(g',i) \mid g' \quad _j(g)\}$. □

Man beginnt zur Trefferbestimmung mit $G_{\mathcal{D}}^m(Q)$ und bekommt dadurch die Information, welche Verschiebungen des Nebenklassenrepräsentantensystems N_m überhaupt Trefferkandidaten enthalten können. Mit dieser Information wird im nächsten Schritt eine entsprechende *Teil*menge von $G_{\mathcal{D}}^{m-1}(Q)$ bestimmt. Aufgrund der obigen Satz bewiesenen Inklusionskette erreicht man im letzten Schritt genau die gesuchte Menge $G_{\mathcal{D}}(Q)$.
Je nach Anwendung kann es sinnvoll sein, nicht nur in G zu vergröbern, sondern auch die Menge $[1:N]$ aller Dokumenten-IDs zunächst hierarchisch in Teilintervalle zu partitionieren.

Eine weitere Modifikation von obigem Prinzip wenden wir im Fall der euklidschen Bewegungsgruppe $G = SE(3)$ an. Diese Gruppe lässt sich durch 7 reelle Parameter stetig repräsentieren. G kann damit als Teilmenge des $\mathbb{R}^7$ angesehen werden. Aufgrund der Stetigkeit der Parametrisierung können wir nun die additive Gruppe $(\mathbb{R}^7,+)$ und geeignete Ketten von Untergruppen nebst zugehörigen Nebenklassen verwenden. Die Stetigkeit gibt letztlich die gesuchten $SE(3)$-Treffer. Genauer gehen wir darauf im Abschnitt 6 über die Suche in Proteinstrukturdaten ein.

3.4 Fuzzyanfragen

Bisher haben wir in diesem Kapitel unser Augenmerk auf die k-Fehlstellensuche gerichtet. Nun wenden wir uns der Erweiterung der in diesem Kapitel vorgestellten Algorithmen zu, um auch Fuzzyanfragen effizient beantworten zu können. Wie in Abschnitt 2.2.4 bereits diskutiert wurde, ist typischerweise die Anzahl der durch eine Fuzzyanfrage spezifizierten Einzelanfragen so groß, dass eine getrennte Bearbeitung der Einzelanfragen aus Effizienzgründen nicht in Frage kommt. Im vorhergehenden Kapitel wurden daher bei der Fuzzysuche die Vereinigung der Listen, die an einem Fuzzyelement beteiligt sind, vorgeschlagen. Auf diese Weise erhält man pro Fuzzyelement wieder eine justierte G-invertierte Liste, die dann kanonisch in den k-Fehlstellensuchprozess einzubinden ist. Dabei ist der Aufwand zur Vereinigung von G-invertierten Listen zu einer neuen Fuzzyliste nicht zu vernachlässigen, wenn eine Vielzahl von Fuzzyelementen in einer Anfrage vorhanden sind. Ausserdem ist dieses Verfahren nur bei Gruppen sinnvoll, die die Sortierung der G-invertierten Listen nicht zerstören, wenn eine solche Liste justiert wird, denn ansonsten ist die Bestimmung der Vereinigungsmenge nur mit großem Aufwand möglich. So könnte man alle Elemente in eine Liste kopieren, diese sortieren und Vielfachheiten eliminieren. Dies würde einen Aufwand von $\Theta(n \log n)$ bedeuten, wobei n die Anzahl der Elemente in den beteiligten G-invertierten Listen ist. Dieses Vorgehen setzt allerdings voraus, dass die Vereinigung der aktuellen G-invertierten Listen im Hauptspeicher möglich ist, da ansonsten der Paging-Aufwand die Berechnung unpraktikabel werden lässt.

3.4.1 Fuzzyanfragenbearbeitung mittels Vielfachheitenzählung

Hier stellen wir nun eine relative einfache Erweiterung der k-Fehlstellenalgorithmen vor, die die Trefferberechnung auf Basis der Zählung von Vielfachheiten durchführen. Dabei entfällt die Berechnung der Vereinigungsmenge aus G-invertierten Listen, die zu einem Fuzzyelement gehören. Durch Vereinigungsbildung soll garantiert werden, dass pro Fuzzyelement der Anfrage ein Paar (g,i) nur einmal berücksichtigt wird.
Bei k-Fehlstellenalgorithmen, die auf der Bestimmung von Vielfachheiten beruhen, kann dieses Ziel auch erreicht werden, indem wir die Datenstrukturen zur Zählung der Vielfachheiten um einen ganz-

zahligen Wert ergänzen, der protokolliert, welches Fuzzyelement zur Erhöhung des Vielfachheitenzählers beigetragen hat. Dies setzt voraus, dass jedem Fuzzyelement der Anfrage eine eindeutige ganzzahlige ID zugewiesen wurde. Werden nun die Listen, die zu einem Fuzzyelement gehören, nacheinander abgearbeitet, kann anhand der gespeicherten Fuzzy-ID festgestellt werden, ob dieses Fuzzyelement bereits zur Erhöhung des Vielfachheitenzählers beigetragen hat. Ist dies der Fall, darf der Zähler nicht erhöht werden. Auf diese Weise ist garantiert, dass höchstens eine Alternative eines Fuzzyelements zur Erhöhung eines Vielfachheitenzählers beitragen kann. Dies entspricht eben gerade dem Prinzip, welches sich hinter der Vereinigung von G-invertierten Listen verbirgt. Der zusätzliche Aufwand betrifft hauptsächlich den Speicherplatzbedarf. Im Falle von Hyperwürfeln erhöht sich der Bedarf um mindestens 1 Byte pro Hyperwürfel-Element.

3.4.2 Fuzzyanfragenbearbeitung mittels n-Wege Merge Sort

Sei $\mathbf{F} = (F_1, \ldots, F_n)$ eine Fuzzyanfrage mit insgesamt $m := \sum_h |F_h|$ Anfrageelementen, wobei q_h das $(h, \ )$-te Element aus F_h bezeichnet. Die zugehörige justierte G-invertierte Liste sei mit G_h bezeichnet. Zur Erweiterung des Min-Heap-Suchverfahrens auf die Fuzzysuche benötigen wir einen Zähler c, der die Anzahl der bisher betrachteten Trefferkandidaten protokolliert und mit 0 initialisiert wird. Weiterhin wird ein Array V der Länge n mit -1 initialisiert.
Während der Suche werden Elemente $(g, i, (h, \ ))$ vom Heap genommen, die aus der $(h, \ )$-ten G-invertierten Liste stammen. Nun wird der Bestandteil (g, i) mit dem aktuellen Trefferkandidaten (g', i') verglichen. Sind diese verschieden, so wird der Zähler c um Eins erhöht und das Paar (g, i) wird zum Trefferkandidaten mit momentaner Vielfachheit $s_c = 1$. Hat das Paar (g', i') eine Vielfachheit $s_{c-1} \geq n - k$, so verbleibt dieses Paar in der Resultatliste L und (g, i) wird als potentieller Kandidat an L angefügt. Ansonsten wird (g', i') aus L entfernt.
Falls $(g, i) = (g', i')$ gilt, muss geprüft werden, ob dies bereits für ein anderes Listenelement zu einem Element aus F_h galt. Aus diesem Grund wird c mit $V[h]$ verglichen. Ist $V[h] < c$, so wurde bisher noch kein Listenelement zu einem Element aus F_h betrachtet, für das $(g, i) = (g', i')$ galt. Der Vielfachheitenzähler s_c wird um Eins erhöht und wir setzen $V[h] = c$, um anzuzeigen, dass bereits ein Listenelement zu einem Element aus F_h berücksichtigt wurde. Ist $|F_h| = 1$, wird s_c sofort um Eins erhöht.
Die Implementierung arbeitet mit einer linearen Anordnung der Doppelindizes $(h, \ )$. Ist $(h, \ )$ das j-te Element dieser Anordnung, so wird dies in einem Array P an der Stelle j protokolliert: $P[j] := h$ falls $|F_h| > 1$ ansonsten wird $P[j] := -1$ gesetzt. Damit gilt $V[h] = V[P[j]]$, für $P[j] = -1$. Ist $P[j] = -1$, so zeigt dies an, dass das zugehörige F_h die Kardinalität Eins hat.
Der zusätzliche Aufwand beschränkt sich auf die Initialisierung der beiden Arrays V und P. Durch die Verwendung des Zählers c entfällt eine Re-Initialisierung von V während der Suche.

3.4.3 Fuzzyanfragenbearbeitung mittels sortiertem Array

Bei dem zuvor diskutierten Verfahren wurde ein Heap verwendet, um die Elemente in den G-invertierten Listen in eine Reihenfolge bzgl. der Ordnung $<$ auf den Paaren (g, i) zu bringen. Daher genügte es bei der Betrachtung des aktuellen Trefferkandidaten zu prüfen, ob bereits eine Liste zu einem Fuzzyelement einen Beitrag geliefert hat oder nicht.
Hier verhält es sich etwas anders, da die Elemente der G-invertierten Listen nicht durch einen Heap sortiert vorliegen. Wie im Abschnitt zuvor gehen wir von einer linearen Anordnung der Doppelindizes aus. Das Array P wird analog initialisiert. Im Unterschied zu vorher enthält das Array V der Länge n

zu Beginn an allen Stellen das Paar $(g,i)_{\max} = \max \ \ _{j=1}^{m} G'_j$. Die Initialisierung von V muss vor der Bestimmung der Elemente in v (siehe Algorithmus 3.2.3) wiederholt werden.
Um nun das Array v mit den jeweils aktuellen Elementen aus den unjustierten Listen G'_j zu füllen, werden die Paare (g,i) aus G'_j gemäß der aktuellen Positionen der Lesezeiger entnommen. Im Fall $P[j] = -1$ wird (g,i) bzgl. der Anfrage justiert und in v eingefügt. Ist hingegen $P[j] = -1$, muss geprüft werden, ob (g,i) nach Justierung das minimale Element zu den aktuellen und justierten Elementen ist, die zu demselben F_h gehören. Dazu wird das justierte (g,i) mit dem Element $V[P[j]]$ verglichen und ggf. der Wert von $V[P[j]]$ aktualisiert. Sind alle m Listen betrachtet worden, wird v um die in V gespeicherten Elemente ergänzt, sofern $V[h] = (g,i)_{\max}$ gilt. Dies gewährleistet, dass immer nur ein Eintrag in v existiert, der zu einem der F_h gehört. Anschliessend wird dann der zuvor beschriebene Algorithmus fortgeführt.
Der Mehraufwand zur Berechnung von Fuzzytreffern liegt wesentlich in der Initialisierung des Arrays V, bevor neue Werte in v eingefügt werden können. Da $|V| \ll m$ gilt, ist dieser Aufwand vernachlässigbar, was sich auch auf das Einfügen der Werte von V in v bezieht. Die Initialisierung von P braucht auch nur einmal durchgeführt zu werden.
Ist die Anzahl k der erlaubten Fehlstellen kleiner als die Anzahl der einelementigen Fuzzymengen F_h, kann für die Aktualisierung der Einträge in V als untere Schranke das minimale aktuelle Element in den unjustierten Listen der einelementigen F_h genommen werden.

3.5 Vergleich der vorgestellten Verfahren

In diesem Abschnitt werden die zuvor beschriebenen Verfahren hinsichtlich der Zeit, die benötigt wird, um eine Anfrage Q zu beantworten, verglichen. Dazu dient ein Audioidentifikationszenario unter der Verwendung des in [Wag03] verwendeten Datensatzes. Dies ermöglicht ausserdem, die hier vorgestellten Fragen in Relation zu den in der Arbeit von Wagener angegebenen Zeiten zu setzen. Allerdings sollte man dabei beachten, dass in [Wag03] unterschiedliche Hardware verwendet worden ist.
Der angesprochene Datensatz wurde aus 9992 MP3-Dateien gewonnen und umfasst 20540372 Paare $(g,i) \quad \mathbb{Z} \times [0:N-1]$ verteilt auf 4095 viele G-invertierte Listen. Die Längen der Listen sind in Abbildung 3.4 dargestellt. Es fällt auf, dass es wenige Listen mit vielen Einträgen gibt.
Die Zahl der Listen ist im Vergleich zur Summe der Einträge in den Listen sehr gering und wird sich konstruktionsbedingt im Falle einer Vergrößerung der Dokumentenkollektion nicht erhöhen. Dies hätte zur Folge, dass selbst die kürzesten Listen viele Elemente enthalten, was direkte Auswirkungen auf die Laufzeit der Suchalgorithmen hat. Würde man einen Suchindex für eine Dokumentenkollektion im Bereich von 16 Millionen Audiostücken aufbauen, käme man auf rund 3.3 Milliarden Einträge in den G-invertierten Listen. Ausgehend von einer Gleichverteilung über alle Listen, würden pro Liste rund 8 Millionen Einträge vorliegen. Da die Listen in der Praxis aber eben nicht alle über die gleiche Anzahl von Einträgen verfügen, wird es Listen geben, die über deutlich mehr Einträge verfügen werden. Hier hilft die generelle Anmerkung, dass ein Audioidentifikationsdienst ohne Weiteres auf r Rechner verteilt werden kann. Über ein die Rechner verbindendes Netzwerk würde nur die Anfrage und Ergebnismengen transportiert werden. Eine Kommunikation der beteiligten Rechner ist während der Suche nicht notwendig, so dass eine sog. „Shared Nothing“-Architektur einer verteilten Datenbank vorliegt. Somit macht es Sinn, die Suchzeiten für den oben beschriebenen Datensatz zu bestimmen als auch für die etwa 10-fache Menge an Einträgen (g,i).
Neben dem Vergleich der Zeit, die ein bestimmtes Verfahren benötigt, um eine Anfrage zu beantworten, macht es Sinn, ein abstraktes Kostenmodell einzuführen und die Algorithmen damit zu verglei-

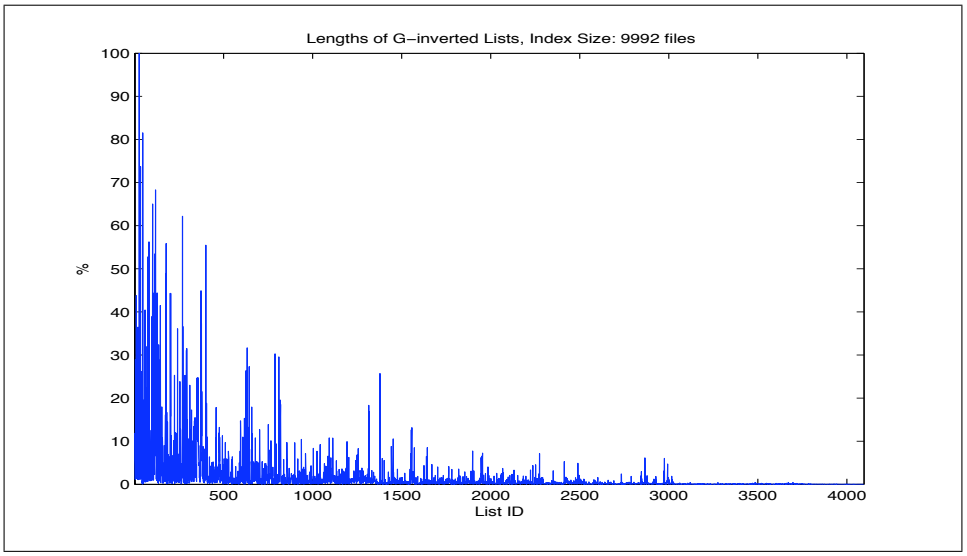


Abbildung 3.4: Länge der Listen in dem Suchindex zur Audioidentifikation. Dieser Suchindex basiert auf 9.992 MP3 Dateien mit Längen um jeweils 3.5 Minuten. Insgesamt wurden aus den MP3 Dateien 20540372 viele Merkmale extrahiert, was zur gleichen Anzahl von Indexeinträgen der Form (g, i) $\mathbb{Z} \times [0 : N - 1]$ führt

chen. Denn z.B. ist der Aufwand zur Berechnung des Produkts zweier Gruppenelemente direkt von G abhängig. Im Falle der Audioidentifikation ist dies lediglich die Addition zweier ganzer Zahlen und damit schnell berechenbar. D.h. die Anzahl der Verknüpfungsoperationen spielt bei der Audioidentifikation nur eine untergeordnete Rolle. Ganz anders verhält es sich bei der später vorgestellten Suche in Proteinmoleküldaten, wo die Verknüpfung zweier Gruppenelemente Gleitkommaarithmetik erfordert und aus mehreren Multiplikationen und Additionen besteht.

Ein weiterer wichtiger Punkt ist die Frage, ob es ausreicht mit Referenzen auf die Einträge in den G-invertierten Listen zu arbeiten, oder müssen die Elemente kopiert werden, z.B. in eine andere Datenstruktur wie einen Heap. Dabei ist es auch entscheidend wie viel Speicher während der Suche vom Algorithmus benötigt wird. Müssen ggf. vor Beginn der Suche aufwändig Datenstrukturen initialisiert werden? Sind während der Suche Datenstrukturen dynamisch anzupassen, wie es z.B. bei der Verwendung von Baumdatenstrukturen der Fall ist? Müssen alle justierten Versionen der Elemente in den Listen als Kopie im Speicher vorgehalten werden?

In Abbildung 3.5 sind die Laufzeiten der vorgestellten Verfahren für unterschiedliche Anfragelängen und Fehlstellenquote dargestellt. Dabei wurden Laufzeiten von über 2.5 Sekunden auf eben diesen Maximalwert festgesetzt. Zusätzlich sind Ergebnisse für den Algorithmus mit dem Label „Vector+Sort“ aufgeführt. Bei diesem Verfahren werden alle justierten Einträge der zu betrachtenden Listen in einen Vektor eingefügt, dieser sortiert und durch sequenzielles Abzählen die jeweiligen Häufigkeiten der Einträge in dem Vektor die Treffermenge bestimmt. Dieses Verfahren ist natürlich nur dann sinnvoll einsetzbar, wenn alle justierten Listenelemente im Hauptspeicher zu verwalten sind.

Im Falle der exakten Anfrage, d.h. es sind keine Fehlstellen zugelassen, ist die Suche mittels Listenschnitt am schnellsten. Dies erklärt sich dadurch, dass die kürzeste zu betrachtende G-invertierte Liste alle Trefferkandidaten beinhalten muss. Bereits nach der Berechnung der Schnittmenge zwischen den

beiden kürzesten Listen, enthält der Schnitt meist nur wenige Elemente. Dies ist gleichbedeutend mit der Anzahl der Trefferkandidaten, so dass im weiteren Verlauf des Algorithmus Schnittmengen zwischen Listen mit nur sehr wenigen Elementen und Listen mit vielen Elementen zu berechnen sind. Dies ist sehr effizient durchführbar, indem die Elemente der kürzeren Liste justiert werden, bevor diese mit einer weiteren G-invertierte Listen geschnitten wird. Würde man wie in dem Kapitel zuvor beschrieben vorgehen, würde alle Einträge in der G-invertierten Liste justiert und geprüft, ob dieses justierte Element einem Trefferkandidaten entspricht.
Ähnlich verhält es sich bei kurzen Anfragen mit dem baumbasierten Ansatz (hier implementiert durch eine STL-Map). Die aufwändige Phase, in der Elemente in die Baumdatenstruktur eingefügt werden, beschränkt sich auf die Abarbeitung der Elemente der kürzesten zu betrachtenden G-invertierten Liste. Bei der Bearbeitung der nachfolgenden Listen erfolgen keine Einfügeoperationen mehr, sondern es muss lediglich für jedes Element der noch zu betrachtenden Listen geprüft werden, ob dieses Element in dem Baum vorhanden ist oder nicht. Erst bei längeren Anfragen summiert sich die Aufwände für die Beantwortung dieser Frage auf und führen zu einer längeren Bearbeitungsdauer. Anzumerken ist noch, dass in diesem Fall für alle zu betrachtenden Paare (g, i) aus den Listen im Suchindex eine justierte Version zu berechnen ist.
Dies gilt auch für das Verfahren basierend auf Hyperwürfeln, welche im Fall der Audioidentifikation Matrizen über $\mathbb{Z} \times \mathbb{Z}$ entsprechen. Die kürzere Bearbeitungsdauer im Falle von langen Anfragen lässt sich durch den effizienten Zugriff auf die in der Matrix abgelegten Vielfachheitenzähler zurückführen. Dass die Laufzeit dieses Verfahrens mehr oder weniger konstant ist bzgl. der Anzahl der erlaubten Fehlstellen, ergibt sich aus der Tatsache, dass unabhängig von der Anzahl von Fehlstellen die gleiche Anzahl von Gruppenoperationen berechnet werden. Lediglich eine größere Anzahl von Trefferkandidaten kann bei großer Fehlertoleranz zu einer Vielzahl von zusätzlichen Prüfungen der Trefferkandidaten führen. Hier spielt es eine große Rolle, wie individuell ein Audiostück durch Merkmale dargestellt wird. Weisen die gewählten Merkmale einen entsprechend geringen Informationsgehalt auf, so kann es dazu führen, dass bei großer Fehlertoleranz im ersten Schritt eine Vielzahl von Trefferkandidaten berechnet berechnet werden, für die jeweils zu überprüfen ist, ob sich bei einer höheren Auflösung dort wirkliche Treffer finden lassen.
Die anderen Verfahren bauen mehr oder weniger auf die Verwendung von dynamischen Datenstrukturen bzw. auf die Sortierung eines Arrays mit vielen Einträgen. Da in dem hier betrachteten Fall die Verknüpfung zweier Gruppenelemente nur wenig Rechenaufwand erfordert, fallen die Aufwände zur Aktualisierung der Datenstrukturen hier besonders ins Gewicht. Dies zeigt sich z.B. sehr deutlich an dem Heap-basierten Verfahren, das bei diesem Szenario am langsamsten ist, obwohl im Falle von keinen erlaubten Fehlstellen die vollständige Abarbeitung einer beliebigen Liste dazu führt, dass der Algorithmus vorzeitig beendet werden kann.
Mit steigender Anzahl von zugelassenen Fehlstellen und Anfragelänge, verlieren jedoch die Verfahren, die in einer ersten Phase Listen oder Baumdatenstrukturen mit Trefferkandidaten füllen, an Boden (z.B. „Map“, „List Intersect“). Hingen bleiben die Laufzeiten der anderen Verfahren nahezu konstant bzw. steigen im Vergleich nur unwesentlich. Lediglich das Verfahren „Linear+Array“ (siehe 3.2.3), welches auf einem sortierten Array beruht, benötigt deutlich mehr Zeit mit steigender Zahl von Fehlstellen. Der Grund dafür ist, dass mit steigender Fehlstellenanzahl, die Zahl von Listenelementen, die übersprungen werden können, abnimmt.
Die Abbildung 3.6 verdeutlicht den gleichen Sachverhalt wie die Abbildung 3.5 zuvor, jedoch mit dem Unterschied, dass hier eine aufwändigere Berechnung der Verknüpfung zweier Gruppenelemente simuliert wurde. Da jedoch ansonsten alle Abläufe gleich sind, lässt sich sehr gut der Einfluss des nötigen Aufwands zur Verknüpfung zweier Gruppenelemente erkennen. Simuliert wird neben der Operation von $G = (\mathbb{Z}, +)$ auf Elementen aus $\mathbb{Z}$, die multiplikative Verknüpfung zweier Elemente

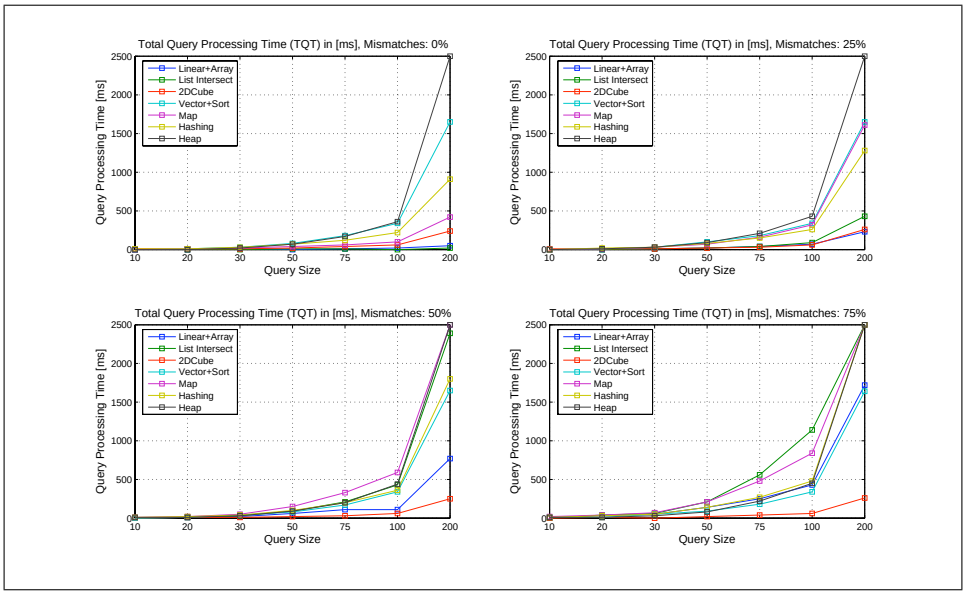


Abbildung 3.5: Dargestellt sind die Laufzeiten der bisher vorgestellten Algorithmen für die Anfragelängen 10, 20, 30, 40, 50, 75, 100 und 200, wobei jeder der Teilabbildungen eine andere Quote von erlaubten Fehlstellen zugrunde liegt. Erwartungsgemäß sind kurze Anfragen für alle Verfahren keine große Hürde. Erst mit steigender Länge der Anfragen und höherer Fehlstellenquote beginnen sich die Laufzeiten deutlich zu unterscheiden. Es operiert $G = (\mathbb{Z}, +)$ auf Elementen aus $\mathbb{Z}$.

der Gruppe $SE(3)$ basierend auf Quaternionen, wie sie in Abschnitt 6.4.4 beschrieben werden. Der zusätzliche Zeitaufwand ist demnach praktisch durchaus relevant, wie wir später noch sehen werden. Deutlich zu erkennen ist der Einfluss der komplexeren Berechnung der Gruppenoperation an den deutlich gestiegenen Laufzeiten aller Algorithmen, die jeweils alle Elemente der G-invertierten Listen betrachten, die durch eine Anfrage Q bestimmt werden. Ohne Fehlertoleranz und bei 25% Fehlstellen bezogen auf die Länge der Anfrage, führen die Treffermengenberechnung mittels Schnittmengenberechnung und mittels sortiertem Array zu den kürzesten Antwortzeiten. Bei einer komplexen Gruppenverknüpfung ist es umso wichtiger, dass nicht alle Elemente einer G-invertierten Liste betrachtet werden müssen bzw. dass die Suche beendet werden kann, sobald eine Liste ganz abgearbeitet worden ist.
Lässt man rund 50% Fehlstellen zu, beginnt sich der zuvor beschriebene Vorteil zu relativieren, indem die Verwaltung von Trefferkandidaten („List Intersect“) bzw. die Abarbeitung der zu betrachtenden G-invertierten Listen in kleineren Schritten erfolgt („Linear+Array“), was wiederum zu einer steigenden Anzahl von zu berechnenden Gruppenverknüpfungen führt. Bei einer noch größeren Anzahl von erlaubten Fehlstellen übersteigen die Zeiten der Anfragebearbeitung aller Verfahren die des Hyperwürfel-basierten Ansatzes, da hier nahezu die gleiche Anzahl von Gruppenverknüpfungen berechnet werden.

3.5.1 Vergrößerung der Datenbasis

Um festzustellen, wie sich die Antwortzeiten der Verfahren bei größeren Datenmengen verhalten, wurde ein Suchindex mit der doppelten Anzahl von Dokumenten aufgebaut, indem jedes Dokument zweimal der Dokumentenkollektion $\mathcal{D}$ hinzugefügt wurde. Auf diese Weise ändert sich die Verteilung

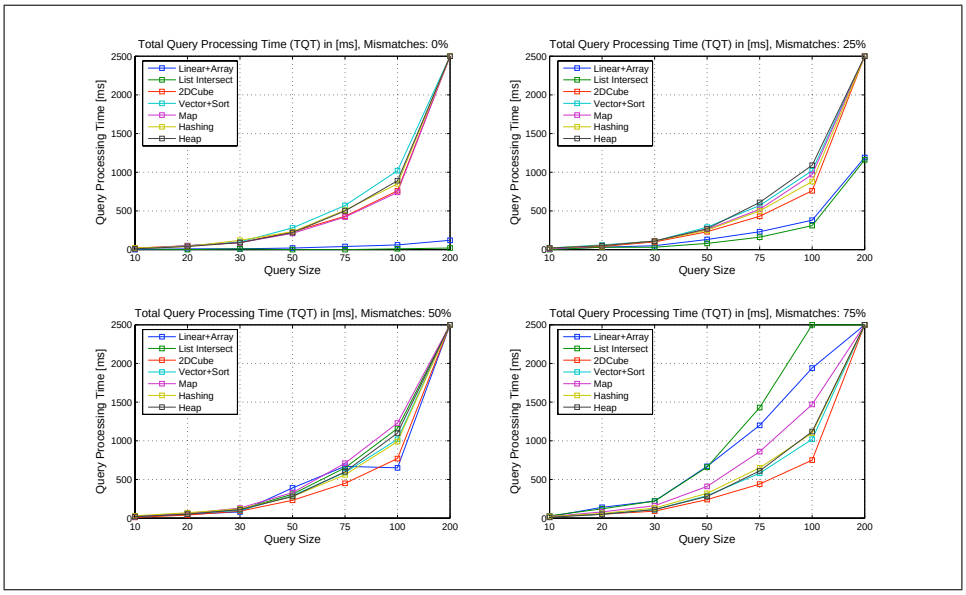


Abbildung 3.6: Der Inhalt dieser Abbildung ist im Prinzip identisch mit dem der Abbildung 3.5. Es wurde exakt die gleichen Anfragen berechnet, jedoch mit dem Unterschied, dass eine aufwändig zu berechnende Gruppenverknüpfung simuliert wurde. Da die einzelnen Algorithmen identisch ablaufen, kann man daran den Einfluss der Berechnungskomplexität der jeweiligen Gruppenverknüpfung erkennen.

der Listenlängen nicht, lediglich die Anzahl der Listeneinträge verdoppelt sich.
Vergleicht man die Abbildungen 3.5 und 3.7 miteinander, so fällt auf, dass sich bei allen Algorithmen die Verdopplung der Listenlängen auch in etwa in einer Verdopplung der benötigten Antwortzeit widerspiegelt. Besonders das Listenschnitt-Verfahren leidet darunter, dass die Anzahl der Trefferkandidaten bei hoher Fehlstellenquote zu einer großen Anzahl zu verwaltender Trefferkandidaten während der Suche führt.

3.5.2 Vergleich der vier wichtigsten Algorithmen

In diesem Abschnitt sollen vier wichtige Algorithmen nochmals im Detail betrachtet werden. Dazu wurden Anfragen der Länge 200 durchgeführt, und dabei die Anzahl der Fehlstellen schrittweise von Null auf 150 erhöht. Ausserdem ist hier die Variante des Hyperwürfel-Verfahrens mit erfasst, bei der die Elemente der G-invertierten Listen bereits vergröbert vorliegen.
Deutlich ist in Abbildung 3.8 das stetige Ansteigen der Antwortzeit bei dem Listenschnittverfahren zu sehen. Jede zugelassene Fehlstelle mehr führt dazu, dass die Elemente einer weiteren und aufgrund der Sortierung der Anfrageobjekte in der Regel auch längeren G-invertierten Liste justiert und in eine temporäre Liste von Trefferkandidaten eingefügt werden müssen, ohne dabei die Sortierung diese Liste zu verletzen. Da in den Experimenten, aus denen die Daten für die Abbildung gewonnen worden sind, die Berechnung der Gruppenverknüpfung nur aus der Subtraktion zweier ganzer Zahlen besteht, gibt es kaum Unterschiede für die beiden Hyperwürfel-Verfahren. Der plötzliche Anstieg der Laufzeit des „Linear+Array“-Verfahrens erklärt sich dadurch, dass ab diesem Zeitpunkt sehr lange Listen komplett verarbeitet werden müssen, die zuvor nur teilweise gelesen worden waren, da zuvor zu dem Zeitpunkt bereits ausreichend viele Listen ganz abgearbeitet worden waren, um die Suche zu

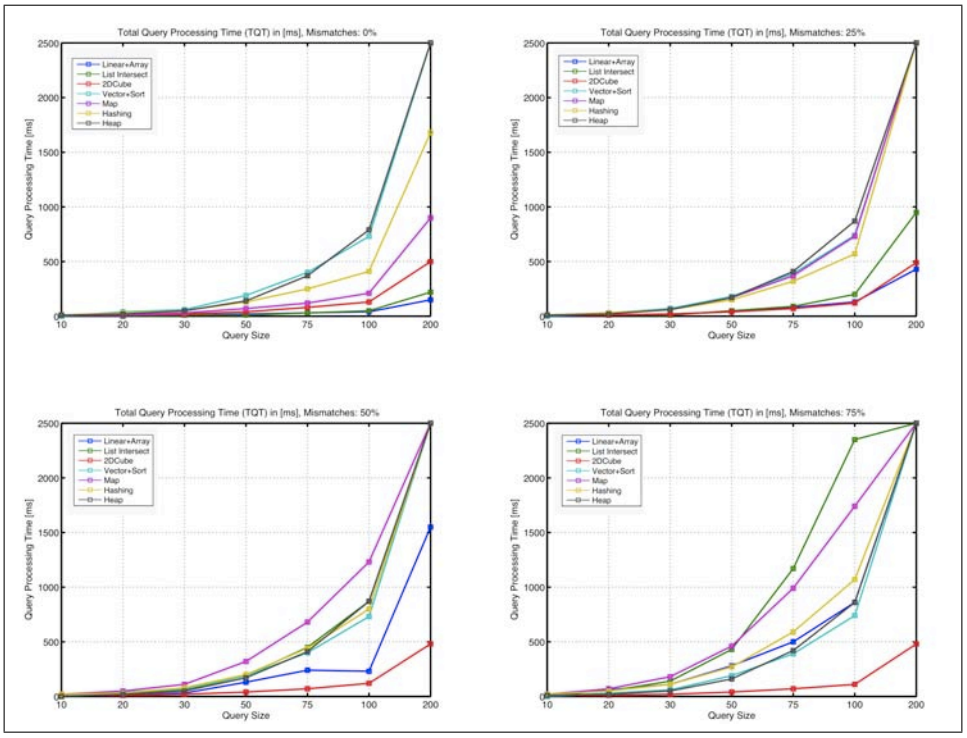


Abbildung 3.7: Wie in Abbildung 3.5, sind hier die Zeiten zur Beantwortung von Anfragen unterschiedlicher Länge und bei unterschiedlichen Fehlstellenquoten dargestellt. Der Unterschied besteht darin, dass hier nun die doppelte Anzahl von Dokumenten im Suchindex abgelegt wurden, indem jedes Dokument zweimal in der Dokumentenkollektion $\mathcal{D}$ vorhanden ist. D.h. die G-invertierten Listen haben die doppelte Länge.

beenden.
Aufgrund der Simulation einer aufwändig zu berechnenden Gruppenverknüpfung ist in Abbildung 3.9 ein deutlicher Geschwindigkeitsvorteil bei der Verwendung von vorberechneten und vergröberten Listen zu erkennen. Die Anzahl der zu berechnenden Verknüpfungen zweier Gruppenelemente ist deutlich geringer als bei dem Hyperwürfel-Verfahren ohne Vorberechnung und Vergröberung. Das Listenschnittverfahren leidet ebenfalls unter der aufwändigeren Berechnung. Die Abbildung 3.10 zeigt diesen Sachverhalt nochmal im einzelnen. Die jeweils rot dargestellten Kurven entsprechen der Laufzeit der Verfahren bei der Simulation einer komplexeren Verknüpfung in der zugrundeliegenden Gruppe G.

3.6 Einbindung von Metadaten

Im bisher beschriebenen Ansatz zur Suchindexstruktur mittels G-invertierter Listen sind bisher *Metadaten*, die zu jedem Dokument vorliegen, nicht berücksichtigt worden. Vielmehr wurden die Dokumente auf eine ID $[0 : N-1]$ reduziert, die keinerlei Information zu Metadaten eines Dokumentes mehr enthält. Allerdings ist in vielen Szenarien denkbar, dass der Benutzer durchaus in der Lage ist, Metainformationen seiner Anfrage hinzuzufügen, etwa über eine Suchmaske, in der der Benutzer zu vorgegebenen Kategorien wie Name des Künstlers, des Albums etc. textuelle Informationen eingeben

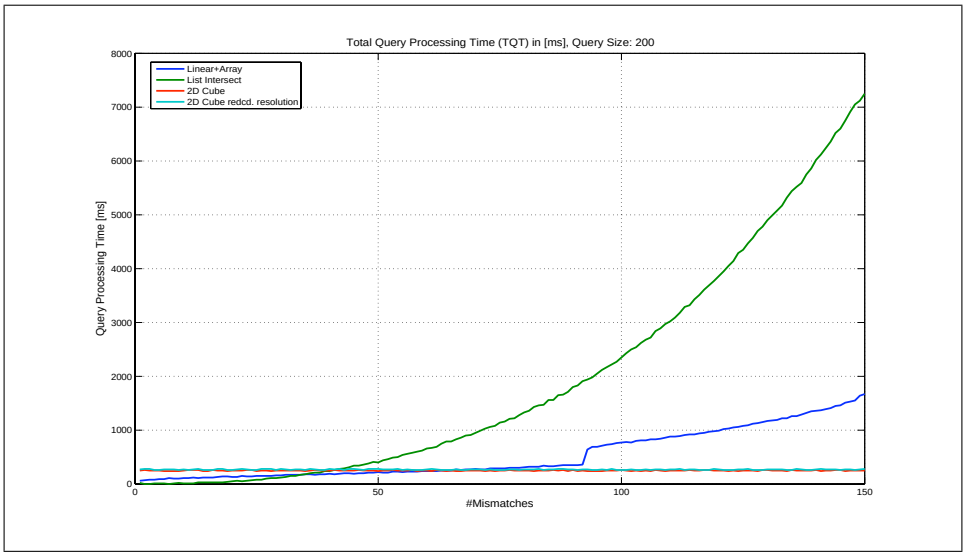


Abbildung 3.8: Die Abbildung zeigt für eine Anfrage der Länge 200 und für die Anzahl von erlaubten Fehlstellen im Bereich von 0 bis 150 die Zeit, die benötigt wurde, um die Anfrage zu bearbeiten. Dabei operiert $G = (\mathbb{Z}, +)$ auf Elementen aus $\mathbb{Z}$, d.h. die Verknüpfung zweier Gruppenelemente ist nicht aufwändig.

kann. Umgekehrt kann auch das System vorhandene Metainformationen dem Benutzer präsentieren, damit dieser eine geeignete Auswahl treffen kann.

Für die inhaltsbasierte Suche sind Metainformationen von enormen Wert, da sie in der Regel den Suchraum a priori auf geeignete Dokumente einschränkt. Dies kann die Anzahl der zu berechnenden Gruppenoperationen reduzieren, was gleichzeitig eine Reduktion der vorzuhaltenden Zwischenergebnisse bewirken würde. Nebenbei sinkt die Gefahr von „False-positive"-Treffern, was auf der anderen Seite die Anwendung von Fehlertoleranzmechanismen stärkt. Die Integration von Metainformationen bei der inhaltsbasierten Suche ist daher wichtig.

Die Einbindung von Metainformationen in die im vorherigen und in diesem Kapitel vorgestellten Suchtechnologie erfolgt durch einen zusätzlichen Vorverarbeitungsschritt, in dem die zu einer Anfrage Q gehörigen Metainformationen $\mathcal{M}_Q$ mit den Metainformationen der Dokumentenkollektion $\mathcal{M}_\mathcal{D}$ verglichen werden. Das Ergebnis dieses Vorverarbeitungsschritts ist eine Indexmenge I von Dokumenten-IDs

$$I := \{i \quad [0 : |\mathcal{D}| - 1] | \mathcal{M}_Q \subseteq \mathcal{M}_{D_i}, D_i \quad \mathcal{D}\}.$$

Die Tatsache, ob die Metadaten einer Anfrage zu denen eines Dokumentes „passen", wird durch eine Teilmengenrelation beschrieben.

Im vorherigen Kapitel wurde die inhaltsbasierte Suche durch die Berechnung einer Schnittmenge von G-invertierten Listen beschrieben. Wir erweitern nun die Definition der G-invertierte Liste zu einem Repräsentanten r wie folgt zur *I-eingeschränkten* G-invertierten Liste:

$$G^I_\mathcal{D}(r) := \{(g, i) \quad G_\mathcal{D}(r) \,|\, i \quad I\}.$$

Damit ergibt sich die folgende, auf Dokumente mit IDs aus I eingeschränkte Treffermenge zur An-

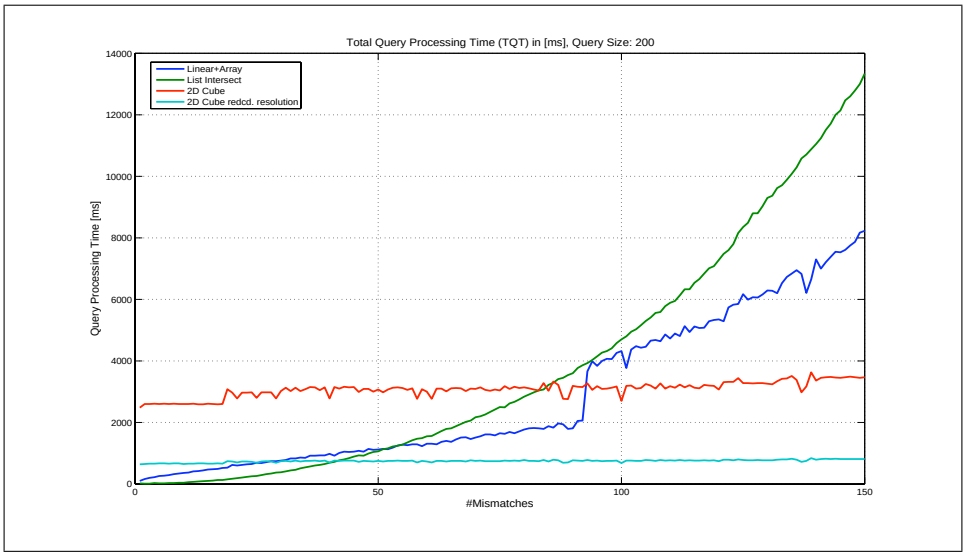


Abbildung 3.9: Die Abbildung zeigt für eine Anfrage der Länge 200 und für die Anzahl von erlaubten Fehlstellen im Bereich von 0 bis 150 die Zeit, die benötigt wurde, um die Anfrage zu bearbeiten. Im Gegensatz zur Abbildung 3.8 zuvor, wurde hier eine aufwändige Berechnung der Gruppenverknüpfung simuliert.

frage Q als

$$G^I_{\mathcal{D}}(Q) = \bigcap_{q \in Q} G^I_{\mathcal{D}}(r_q) g_q^{-1}.$$

Zur Vorabbestimmung der Menge I möglicher Treffer-Dokumenten-IDs anhand der Metadatenteilanfrage $\mathcal{M}_Q$ bieten sich relationale Datenbanksysteme an.

3.6.1 Negativliste

Im vorherigen Abschnitt wurde auf der Basis von Metainformationen die Idee einer Positivliste von Dokumenten-IDs vorgestellt, um die Anzahl der zu betrachtenden Einträge in den G-invertierten Listen auf eine bestimmte Menge von Dokumenten einzuschränken.
Gehen wir nun von einer Anfrage Q und Metainformationen $\mathcal{M}_Q$ aus. Allerdings enthalte $\mathcal{M}_Q$ solche Metaddaten, die dazu führen, dass in der Indexmenge I mehr als 50% der Dokumenten-IDs enthalten sind. In diesem Fall macht es Sinn zu der Komplementärmenge, der *Negativliste* $\overline{I}$, überzugehen, da $|\overline{I}| < |I|$ gilt. Somit ist die Frage, ob ein Eintrag (g, i) in einer der G-invertierten Listen betrachtet werden muss, schneller zu beantworten, da weniger Elemente in $\overline{I}$ enthalten sind:

$$\overline{G}^{\overline{I}}_{\mathcal{D}}(r) := \{(g, i) \quad G_{\mathcal{D}}(r) \mid i \quad \overline{I}\} = G^I_{\mathcal{D}}(r).$$

Die Treffermenge $G^I_{\mathcal{D}}(Q)$ lässt sich alternativ über die Formel

$$G^I_{\mathcal{D}}(Q) = \bigcap_{q \in Q} \overline{G}^{\overline{I}}_{\mathcal{D}}(r_q) g_q^{-1}$$

berechnen.

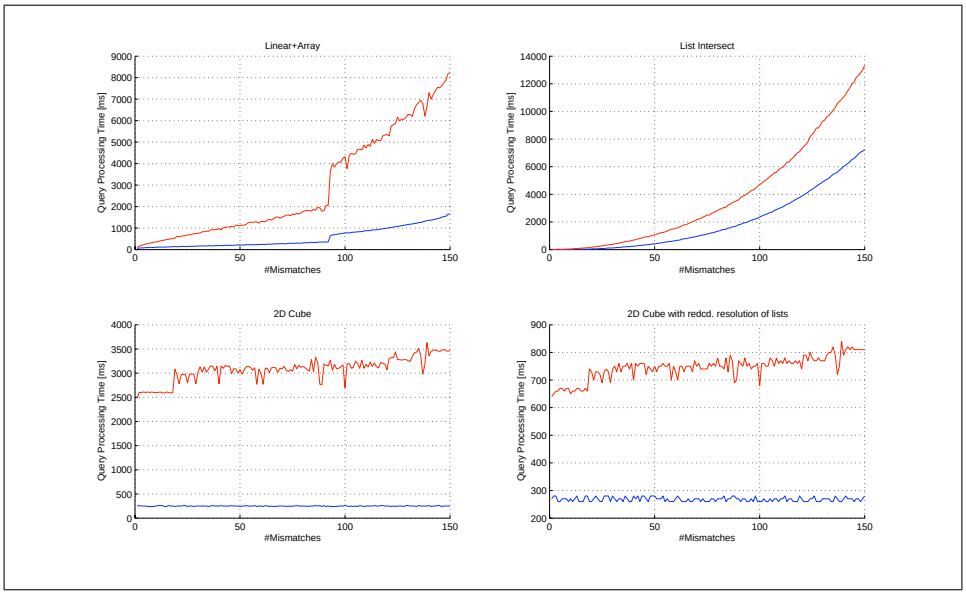


Abbildung 3.10: Diese Abbildung ermöglicht den direkten Vergleich der Laufzeiten der einzelnen Algorithmen in Bezug auf die zugrunde liegende Gruppenoperation bzw. auf deren Berechnungskomplexität. Der rot gezeichnete Kurve dabei liegt eine berechnungsaufwändigere Gruppenoperation zugrunde.

Sind in einem Suchindex nur wenige Dokumente gespeichert, kann man die Positivliste von Dokumenten-IDs als einen Bitvektor auffassen, so dass die Frage, ob ein Element (g, i) bei der Suche berücksichtigt werden muss, in konstanter Zeit beantwortet werden kann.

3.7 Zusammenfassung

In diesem Kapitel wurden unterschiedliche Verfahren zur Berechung einer Treffermenge $G_{\mathcal{D}\,k}(Q)$ zu einer Anfrage Q vorgestellt. Dabei wurde unterschieden, ob die einem Suchindex zugrundeliegende Gruppe G mit der darin definierten Verknüpfung die Ordnung der Elemente einer G-invertierten Liste erhält oder nicht, wenn alle Elemente (genauer gesagt die G-Komponente der Paare (g, i)) der G-invertierten Liste mit einem $g_q^{-1} \quad G$ justiert werden.
Bleibt die Ordnung der Listenelemente erhalten, bieten sich Verfahren an, die eben diese Eigenschaft nutzen und ggf. während der Suche in der Lage sind zu entscheiden, ob nachfolgend überhaupt noch ein Treffer gefunden werden kann oder nicht. Dies kann zu signifikanten Laufzeitverbesserungen führen, da die Suche relativ früh abgebrochen werden kann, d.h. nicht alle Elemente der G-invertierten Listen werden betrachtet. Gerade wenn die in der Gruppe definierte Verknüpfung aufwändig zu berechnen ist, ist dies ein Vorteil gegenüber anderen Verfahren.
Die zweite Klasse von Suchverfahren erfordert keine vorsortierten Listen. Dass dies eine wichtige Klasse ist wird zum Beispiel im Fall $G = SE(3)$ deutlich. Multipliziert man alle G-Komponenten g der Paare (g, i) einer G-invertierten Liste mit einem Element $g_q^{-1} \quad G$, so führt dies in der Regel dazu, dass die Sortierung der Elemente der G-invertierten Liste nicht mehr sortiert vorliegen.
Gemein ist solchen Verfahren, dass sie alle Einträge (g, i) in den bzgl. Q zu betrachtenden Listen mindestens einmal betrachten, d.h. eine justierte Version (g', i) berechnet werden muss. Bei einer

aufwändigen Verknüpfung bzgl. G stellt dies einen Nachteil in Bezug auf die Laufzeit des Suchalgorithmus dar.

3.7.1 Verfahren bei angeordneten Gruppen

3.7.1.1 Listenschnitt

Dieses Verfahren bietet sich nur dann an, wenn bei Anfragen nur wenige Fehlstellen zugelassen werden, da ansonsten die sortierte Speicherung von Trefferkandidaten viel Speicherplatz und Rechenzeit erfordert. Bei exakten Anfragen sicherlich das Verfahren der Wahl, da hier die kürzeste Liste automatisch der Ausgangspunkt für die Liste der Trefferkandidaten ist. Nur für deren Elemente müssen justierte Versionen der Einträge berechnet werden.

3.7.1.2 Sortiertes Array

Dieses Verfahren sollte angewendet werden bei Anfragen mit wenigen zugelassenen Fehlstellen. Durch die Möglichkeit, Einträge in den Listen zu überspringen und dadurch die Anzahl von Gruppenoperationen zu reduzieren, macht dieses Verfahren besonders dann interessant, wenn die durch die Gruppe gegebene Verknüpfung aufwändig zu berechnen ist.
Bei steigender Anzahl von Fehlstellen ist die Zahl der zu überspringenden Listeneinträge nur sehr gering, dafür muss jedoch häufiger ein Array mit so vielen Einträgen wie die Anfrage Elemente enthält sortiert werden. Dies kostet entsprechend viel Rechenzeit und einen nicht zu vernachlässigenden Aufwand an Kopieraktionen im Speicher. Jedoch ist im Gegensatz zu dem Verfahren zuvor der Speicherverbrauch deutlich geringer, da immer nur $|Q|$ viele Listenelemente im Hauptspeicher zu halten sind.

3.7.1.3 Heap

Das letzte Argument von zuvor gilt auch im Falle des Heap-basierten Suchverfahrens. Allerdings hat sich in der Praxis das häufig notwendige „versickern“ von neu in den Heap eingefügten Elementen als zu aufwändig erwiesen. Daher zeigt dieses Verfahren leider keine gute Performanz, obwohl auch dieses Verfahren einen vorzeitigen Abbruch der Suche ermöglicht, wenn keine Treffer mehr zu finden sind.

3.7.2 Verfahren bei nicht notwendigerweise angeordneten Gruppen

3.7.2.1 Hyperwürfel

Dieses Verfahren ist eines derer, welches in nahezu allen Fällen sinnvoll einsetzbar ist. Da es nahezu unabhängig ist von der Anzahl der erlaubten Fehlstellen und Fuzzy-Anfragen kanonisch eingebunden werden können, ist die Laufzeit sehr gut abschätzbar. Der sicherlich größte Nachteil, dass alle zu bearbeitenden Listen ganz gelesen werden müssen (inkl. Justierung der Listeneinträge bzgl. Q), kann dadurch relativiert werden, dass auf Kosten von Hintergrundspeicher G-invertierte Listen in unterschiedlichen Auflösungsstufen vorgehalten werden können. Dies reduziert bei aufwändig zu berechnenden Verknüpfungen von Gruppenelementen die Laufzeit deutlich. Ist die Verknüpfung zweier Gruppenelemente wie im Fall der Audioidentifikation sehr schnell zu berechnen, gibt es bei großen Anfragelängen und einer großen Zahl von zugelassenen Fehlstellen kaum eine andere Alternative.

Der Speicherplatzbedarf zur Laufzeit ist je nach Größe der verwendeten Hyperwürfel nicht unerheblich, ist aber unabhängig von Anfragen und damit an die entsprechenden Umgebungen (Indexgröße, Speicherplatz vs. Bearbeitungsgeschwindigkeit etc.) gut anpassbar.

3.7.2.2 Baumbasierte Verfahren

Die Bestimmung der Treffermenge mittels eines baumbasierenden Verfahrens ist sehr ähnlich zu dem Listenschnittverfahren. Bei k zugelassenen Fehlstellen werden die Elemente der $k+1$ kürzesten Listen, die bei einer Anfrage Q zu betrachten sind, justiert und in die Baumdatenstruktur eingefügt, nachdem geprüft worden ist, ob ein Element nicht schon vielleicht in dem Baum vorhanden ist. Dieser Teil des Algorithmus ist recht aufwändig, da Einfügeoperationen in die Baumdatenstruktur dazu führen können, dass Knoten in dem Baum die maximale Anzahl von Nachkommen überschreiten. Dies hat zur Folge, dass ggf. ein großer Teil des Baumes zusätzlich umstrukturiert werden muss. Sind die $k+1$ Listen in den Baum eingefügt worden, werden die Elemente der verbliebenen zu betrachtenden Listen justiert, um dann zu prüfen, ob ein entsprechendes Element in dem Baum vorhanden ist. Dies erfordert jedesmal logarithmischer Zeitaufwand.
Neben der Tatsache, dass der Aufbau des Baumes relativ viel Zeit in Anspruch nehmen kann, ist der Speicherplatzbedarf ebenfalls durch die jeweilige Anfrage Q bestimmt. Bei größeren Anfragelängen müssen entsprechend viele Trefferkandidaten in dem Baum gespeichert werden. Ein Vorteil ist sicherlich die sehr einfache Implementierung des Suchverfahrens an sich, wenn man auf entsprechende Bibliotheken für die Baumdatenstruktur zurückgreift. Dieses Verfahren hat sich auch aus diesem Grund als Referenzverfahren bewährt, um die Treffermengen, die durch andere Verfahren berechnet worden sind, zu validieren.

3.7.2.3 Suche mittels Sortierung und Abzählung

Dieses Verfahren (in den Graphiken als „Vector+Sort“ bezeichnet) wurde bisher nur kurz vorgestellt. Der Speicheraufwand ist beträchtlich und abhängig von der Anfrage Q, da alle Elemente der zu betrachtenden G-invertierten Listen justiert und in ein Array kopiert werden, welches anschliessend sortiert wird. Die eigentliche Suche nach Treffern (g, i) erfolgt dann durch sequentielles Durchlaufen des Arrays, wobei nach hinreichend langen Sequenzen von gleichen Elementen gesucht wird, die aufgrund der Sortierung aufeinanderfolgen. Die Berechnung von Fuzzyanfragen ist kanonisch möglich, indem auch nach Sequenzen in dem Array gesucht wird, die länger als die Anfrage Q sind. In einem solchen Fall sind mehrere Alternativen an einem Treffer beteiligt.
Der Vorteil ist sicherlich die einfache Implementierung, wobei dieser bei großen Datenmengen schnell durch den zu erwartenden Speicher- und Sortieraufwand während der Suche relativiert wird.

3.7.2.4 Hashing

Diese Vorgehensweise zur Bestimmung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ zeigte bei den durchgeführten Test relativ lange Antwortzeiten. Die Abhängigkeit der benötigten Zeit des Algorithmus und der verwendeten Hashfunktion zeigt die Teilabbildung oben links in Abbildung 3.11 . Bei dem verwendeten Testdatensatz galt es eine Hashfunktion H zu definieren, die ein Paar (g, i) $[0 : 92622] \times [0 : 9992]$ nach $\mathbb{Z}$ abbildet:

$$H_s(g, i) := (g \cdot 2^s \mid i) \,\&\, (2^{19} - 1), \tag{3.1}$$

wobei „|“ für die bitweise Oder-Verknüpfung und „&“ für die bitweise Und-Verknüpfung steht. Mit Hilfe des Parameters s $[0 : 32]$ kann gesteuert werden, in wie weit sich die Binärdarstellungen

von g und i überlappen. Je größer s gewählt wird, desto mehr Bits der Binärdarstellung von g werden zur Berechnung des Hashwerts $h(g,i)$ herangezogen. Um den Wertebereich der Hashfunktion auf 2^{19} unterschiedliche Hashwerte einzuschränken, wird das Ergebnis der Oder-Verknüpfung mit einer entsprechenden Bitmaske verknüpft.

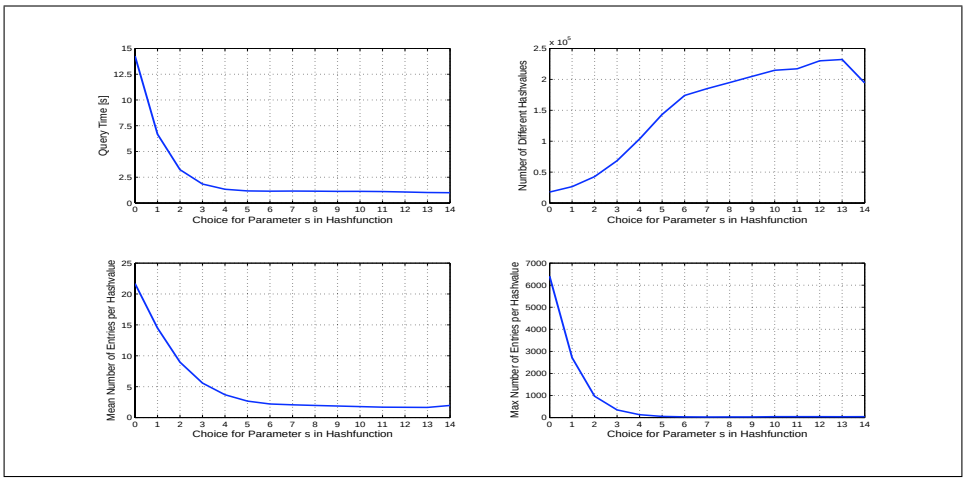


Abbildung 3.11: Auswirkungen des Parameters s auf die Hashfunktion (3.1). Eine Multiplikation des Gruppenelements g eines jeden Paares (g,i) mit dem Faktor 2^{13} zeigt die besten Ergebnisse. Die Anfrage enthielt dabei 100 Paare der Form (g,i) und es waren 75% Fehlstellen erlaubt.

Die durchschnittliche Anzahl von Paaren (g,i), die auf ein und denselben Hashwert abgebildet werden, ist von enormer Bedeutung. Denn zu jedem Hashwert wird eine *unsortierte* Liste von Gruppenelementen geführt, deren Vielfachheiten während der Suche gezählt werden. Je mehr Paare durch H_s auf einen Hashwert abgebildet werden, desto mehr Vergleiche müssen durchgeführt werden, um festzustellen, ob ein $g \quad G$ bereits in der Liste zu einem Hashwert vorhanden ist oder nicht. Auf die Sortierung der Gruppenelemente in der Liste zu einem Hashwert kann dann verzichtet werden, wenn nur sehr wenige Einträge pro Liste zu speichern sind.

3.7.3 Abschliessende Bemerkungen

Dieses Kapitel hat gezeigt, dass es eine breite Auswahl an effizienten Algorithmen zur Berechnung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ gibt. Eine generelle Empfehlung für einen Algorithmus zu geben ist nahezu unmöglich, da die Wahl in Abhängigkeit der jeweiligen Anwendung getroffen werden muss. Allerdings ist die Wahl des passenden Algorithmus zur Trefferbestimmung nur ein Teil, der zu einem Gesamtsystem beiträgt. Die Wahl der „richtigen“ Realisierung der Gruppe G bei einer Implementierung in Bezug auf Speicherplatzbedarf und Aufwand zur Verknüpfung zweier Gruppenelemente ist ebenso entscheidend. In Kapitel 6 wird dies deutlich. Aber den größten Einfluss auf die Performanz hat die Wahl der Elementarobjekte, aus denen ein Suchindex basierend auf G-invertierten Listen aufgebaut wird. Hier ist es von entscheidender Bedeutung, dass die Anzahl der Einträge in den G-invertierten Listen nicht zu groß sein sollte, um von vornherein nur einen möglichst kleinen Teil des Suchindex überhaupt betrachten zu müssen. Im Falle der Audioidentifikation wird diese Fragestellung in Kapitel 5 im Detail diskutiert.

Kapitel 4

Grundbegriffe der Signalverarbeitung

In diesem Kapitel werden in Abschnitt 4.1 die in dieser Arbeit verwendeten Begriffe und Konzepte der Audiosignalverarbeitung erläutert. Eine kurze Einführung in die Theorie der „Hiden Markov Models“ wird in Abschnitt 4.2 gegeben, während in Abschnitt 4.3 ein kurzer Einblick in das MP3-Format zur verlustbehafteten Audiokompression gegeben wird.

4.1 Signalverarbeitung

Audiosignale sind im Alltag allgegenwärtig, sei es als Radioprogramm, welches uns durch Frequenz- oder Amplitudenmodulation erreicht oder in digitaler Form auf einer CD. In diesem Abschnitt sollen die Grundlagen für die Verarbeitung von digitalen Audiosignalen erläutert werden. Dies kann hier natürlich nur in aller Kürze geschehen. Mehr Details findet man z. B. in [PM96].
Eine gängige Darstellungsart eines Audiosignals ist die sog. *Wellenformdarstellung*, die den zeitlichen Amplitudenverlauf eines Signals, interpretierbar als die Schwingungen einer Lautsprechermembran, darstellt. Dieser Darstellung kann man direkt die Lautstärke zu einem Zeitpunkt ablesen. Am plötzlichen Ansteigen der Amplitude erkennt man zum Beispiel sehr gut den Zeitpunkt, an dem ein Instrument einsetzt. Die Wellenformdarstellung ist nicht sehr robust gegenüber verlustbehafteter Kompression, was bedeutet, dass die Wellenform eines MP3-kodierten und wieder dekodierten Signals sich deutlich von der Wellenform des Ursprungssignals unterscheidet, obwohl sich beide Signale für den Menschen gleich anhören, sofern eine entsprechend hohe Bitrate benutzt wurde. Die Wellenformdarstellung verbirgt jegliche Frequenzinformation. Aus diesem Grund setzen viele Algorithmen im Multimedia Information Retrieval-Bereich eine Transformation des Signals in die zweidimensionale *Zeit-Frequenz-Darstellung* oder *Phasenraumdarstellung* als einen ersten Verarbeitungsschritt ein. Dieser Darstellung kann man entnehmen, welche Frequenzen in dem Signal zu einem Zeitpunkt t mit welcher Amplitude vertreten sind. Jedoch ist diese Darstellung mit Vorsicht zu interpretieren, da Schwingungen ein Zeitphänomen sind und es somit keinen Sinn macht, von Frequenzen zu einem Zeitpunkt t zu sprechen. Auch setzt die Heisenbergsche Unschärferelation dieser Darstellung deutliche Grenzen in der Art, dass eine hohe Frequenzauflösung mit einer geringeren zeitlichen Auflösung einhergeht und umgekehrt.
Um ein Signal in den *Frequenzraum* zu transformieren, benutzt man die *Fouriertransformation* (FT). Wie später in diesem Kapitel noch genauer erläutert wird, misst die FT die Intensität, mit der eine Sinusschwingung der Frequenz im Signal enthalten ist. Die FT liefert diese Information gemittelt über das gesamte Signal. Daher ist diese Information nur bedingt verwendbar, da sich Signale mit der Zeit stark ändern. Um nun die Phasenraumdarstellung zu erhalten, wird die *gefensterte Fou-*

riertransformation (WFT) benutzt. Grob gesprochen wird das Signal mittels einer Fensterfunktion in aufeinanderfolgende Teilsignale unterteilt, für die jeweils die FT berechnet wird.

4.1.1 Signale und Signalräume

Mathematisch ist ein analoges Audiosignal f eine Funktion, die einem Zeitpunkt t $\mathbb{R}$ einen komplexen oder reellen Amplitudenwert zuordnet, also $f : \mathbb{R}$ $\mathbb{C}$ oder $f : \mathbb{R}$ $\mathbb{R}$. Je nach Wertebereich spricht man von einem *komplexwertigen* oder *reellwertigen* Signal. Der Raum $\mathcal{L}^2(\mathbb{R})$ aller Lebesgue-messbaren Funktionen $f\colon \mathbb{R}$ $\mathbb{C}$ mit $f\ _2 := (\ _{\mathbb{R}} |f(u)|^2 du)^{1\ 2} <$ ist ein komplexer Vektorraum. $N_2 := \{f$ $\mathcal{L}^2(\mathbb{R}) \mid f\ _2 = 0\}$ ist ein Unterraum von $\mathcal{L}^2(\mathbb{R})$ und der Quotientenraum

$$L^2(\mathbb{R}) := \mathcal{L}^2(\mathbb{R})\ N_2$$

ist der bekannte L^2-Raum. Dieser Raum wird zum vollständig normierten Raum, also zum Banachraum, vermöge der wohldefinierten Festsetzung $f + N_2\ _2 := (\ _{\mathbb{R}} |f(u)|^2 du)^{1\ 2}$. Im folgenden schreiben wir oft f statt $f + N_2$, haben aber immer im Hinterkopf, dass es sich bei den Elementen von $L^2(\mathbb{R})$ eigentlich um Äquivalenzklassen von Funktionen handelt. Die Norm stammt vom Skalarprodukt

$$f, g \quad := \quad \int_{\mathbb{R}} f(t)\overline{g(t)}dt,$$

denn $f, f \ = \ f\ _2^2$. Somit ist $L^2(\mathbb{R})$ sogar ein Hilbertraum, der auch Raum der *Energiesignale* genannt wird, da die L^2-Norm der *Signalenergie* entspricht. Dieser Raum lässt sich verallgemeinern, indem man für $1 \leq p <$ den Raum $\mathcal{L}^p(\mathbb{R})$ aller Lebesgue-messbaren Funktionen $f\colon \mathbb{R}$ $\mathbb{C}$ mit $f\ _p := (\ _{\mathbb{R}} |f(u)|^p du)^{1\ p} <$ bildet. Eine analoge Quotientenbildung zu oben führt auf den Raum

$$L^p(\mathbb{R}) = \mathcal{L}^p(\mathbb{R})\ N_p,$$

der durch $f + N_p\ _p := (\ _{\mathbb{R}} |f(u)|^p du)^{1\ p}$ zum Banachraum wird.
Um Signale mit dem Rechner verarbeiten zu können, geht man über zu Signalräumen, die $\mathbb{Z}$ als Definitionsbereich haben, d.h. das Signal ist nur an diskreten Stellen definiert. Analog zu oben definieren wir für p $[1,$ $)$

$$^p(\mathbb{Z}) := \{x : \mathbb{Z} \quad \mathbb{C} \mid \sum_{i\in\mathbb{Z}} |x(i)|^p < \ \}$$

als die Menge aller p-summierbaren Folgen von $\mathbb{Z}$ $\mathbb{C}$. Signale x mit endliche Länge N, also z.B. $x : [0 : N-1]$ $\mathbb{C}$, können durch $x(i) = 0$ für alle i $\mathbb{Z}$ $[0 : N-1]$ auf ganz $\mathbb{Z}$ fortgesetzt werden. Die Räume $^p(\mathbb{Z})$ sind ebenfalls Banachräume, $^2(\mathbb{Z})$ ist sogar ein Hilbertraum. Wichtig im Zusammenhang mit sog. linearen *Faltungs ltern* ist der Raum $^1(\mathbb{Z})$ der absolut-summierbaren diskreten Signale. Diesen Faltungsoperatoren wenden wir uns nun zu.
Zu x $^p(\mathbb{Z})$ und h $^1(\mathbb{Z})$ wird, wie man leicht zeigt, durch

$$x * h = C_h[x] := (\mathbb{Z} \quad n \quad \sum_{i\in\mathbb{Z}} x(n) \cdot h(n-i))$$

wieder ein Element $x * h$ aus $^p(\mathbb{Z})$ definiert, die *Faltung* von x mit h. Fasst man beide Größen als variabel auf, so handelt es sich bei der Faltung um eine bilineare Abbildung $^p(\mathbb{Z}) \times\ ^1(\mathbb{Z})$ $^p(\mathbb{Z})$. In den meisten Anwendungen liegt jedoch ein festes h vor. Dann ist C_h, der *Faltungsoperator* zu festem h $^1(\mathbb{Z})$, eine lineare Abbildung $C_h\colon$ $^p(\mathbb{Z})$ $^p(\mathbb{Z})$. (Die Zahlen $h(i)$ heißen *Faltungskoef zienten*.) Im folgenden interessieren wir uns besonders für den Fall $p = 2$. Faltung hängt eng mit Filterung zusammen. Der tiefere Zusammenhang wird erst durch eine Frequenzanalyse des Signals deutlich. Damit beschäftigt sich der nächste Abschnitt.

4.1.2 Frequenzanalyse

Wenn man in den Definitionen der L^p-Räume überall den Definitions- und Integrationsbereich durch das endliche Intervall $[0,1]$ ersetzt, so erhält man wieder Banachräume und im Fall $p = 2$ sogar wieder einen Hilbertraum. Diese Räume kann man ansehen als Räume gewisser 1-periodischer Signale. Hilberträume besitzen *Orthonormalbasen*. Für den Raum $L^2([0,1])$ ist zum Beispiel

$$\{e_k := ([0,1] \quad t \quad e^{2\pi ikt}) | k \quad \mathbb{Z}\}$$

eine ON-Basis. Somit lässt sich jedes Signal $f \quad L^2([0,1])$ als Linearkombination dieser Basisfunktionen darstellen: $f = \quad_{k\in\mathbb{Z}} \ f|e_k \ e_k$. Dies ist die sog. *Fourierreihenentwicklung* von $f \quad L^2([0,1])$. Der k-te Fourierkoeffizient $f|e_k$ beschreibt die Intensität, mit der die k-te komplexe Exponentialfunktion, die sich ja wegen der für reelle ϕ gültigen Eulerschen Formel $e^{i\phi} = \cos(\phi) + i\sin(\phi)$ aus Cosinus- und Sinusschwingungen zusammensetzt, an dem Signal f beteiligt ist. Die Zuordnung, die einem $f \quad L^2([0,1])$ die Folge seiner Fourierkoeffizienten zuordnet, ist ein Isomorphismus $L^2([0,1]) \qquad ^2(\mathbb{Z})$ von Hilberträumen. Nachdem wir damit die Frequenzanalyse für den Fall periodischer, quadrat-integrierbarer Signale besprochen haben, kommen wir nun zum wesentlich schwierigeren Teil, der Frequenzanalyse im nichtperiodischen Fall. Eigentlich müssten wir uns erst mit der Fouriertransformation für den Raum $L^1(\mathbb{R})$ beschäftigen, diese dann auf den Raum $L^1(\mathbb{R}) \setminus L^2(\mathbb{R})$, der dicht in $L^2(\mathbb{R})$ liegt, einschränken, um dann diese Einschränkung geeignet auf ganz $L^2(\mathbb{R})$ fortzusetzen. Wir überspringen diese Reihe technischer Subtilitäten und führen die Fouriertransformation auf $L^2(\mathbb{R})$ etwas hemdsärmlig ein.
Die *Fouriertransformierte* $F[f] = f$ eines Signals $f \quad L^2(\mathbb{R})$ ist an der Stelle $\qquad \mathbb{R}$ definiert durch

$$f(\ \ ) = F[f](\ \ ) \ := \qquad_{\mathbb{R}} f(t)e^{-2\pi i \ \ t}dt.$$

Mit f liegt auch f in $L^2(\mathbb{R})$, genauer ist die Fouriertransformation $F := (f \qquad f)$ nach dem Satz von Plancherel ein Automorphismus des Hilbertraums $L^2(\mathbb{R})$, also eine unitäre Transformation. Insbesondere ist diese Transformation energie- und winkelerhaltend, denn $\ f\ _2 = \ f\ _2$ und $f|g = f|g$, für alle $f, g \quad L^2(\mathbb{R})$. Aus f lässt sich f wie folgt rekonstruieren:

$$f(t) = F^{-1}[f](t) \ = \qquad_{\mathbb{R}} f(\ \ )e^{2\pi i \ \ t}d \ \ .$$

Für die Anwendungen in der Audiosignalverarbeitung hat der Operator F den Nachteil, dass er keine zeitliche Lokalisierung aufweist. Vielmehr sind die Werte $f(\ \ )$ als Mittelung über das gesamte Signal f zu sehen, d.h. in dem Signal f ist die Frequenz im Mittel mit der Intensität $f(\ \ )$ enthalten. Die als nächstes zu besprechende gefensterte Fouriertransformation stellt ein wichtiges Werkzeug zur besseren Zeit-Frequenz-Lokalisierung dar. Hier geht man von einer sog. *Fensterfunktion* $g \quad L^2(\mathbb{R})$ aus, von der man rein technisch nur $\ g\ _2 = 0$ fordern muss. Typischerweise handelt es sich aber um eine um den Nullpunkt konzentrierte (glockenartige) Funktion, die nach beiden Seiten schnell nach Null abfällt. Signale $f \quad L^2(\mathbb{R})$ werden nun mit Hilfe von um t verschobenen und mittels modulierten Versionen von g, also mittels

$$g \ \ _t := (\mathbb{R} \quad u \quad e^{2\pi iu}g(u-t)$$

analysiert, indem man für $(\ \ ,t) \quad \mathbb{R}^2$ das Skalarprodukt

$$f(\ \ ,t) := \ f|g \ \ _t$$

bildet. Wegen $g_{\ 2} = g_{\ t\ 2}$ liegen auch alle $g_{\ t}$ in $L^2(\mathbb{R})$. Die Funktion $f =: \mathrm{WFT}_g[f]$ heißt die *gefensterte Fouriertransformierte* von f (bzgl. g). Eine andere Sichtweise ist möglich: Durch punktweise Multiplikation des Signals f mit der um t verschobenen Fensterfunktion g wird ein Signalauschnitt von f berechnet, welcher um t lokalisiert ist. Man betrachtet also statt $u \quad f(u)$ jetzt $u \quad f(u) \cdot g(u-t)$. Für diesen Signalausschnitt wird dann die Fourieranalyse durchgeführt.
Beispiele für häufig genutzte Fensterfunktionen sind das *Hamming-Fenster* der Länge N (Abb.4.1), welches außerhalb des Intervalls $[-N\ 2, N\ 2]$ verschwindet und innerhalb dieses Intervalls definiert ist als

$$w_N(t) \quad := \quad 0.54 + 0.46 \cos\left(\frac{2\pi t}{N}\right).$$

Das *Hanning-Fenster* (Abb. 4.1) der Länge N ist entsprechend definiert als

$$w_N(t) \quad := \quad \frac{1}{2}\left[1 + \cos\left(\frac{2\pi t}{N}\right)\right].$$

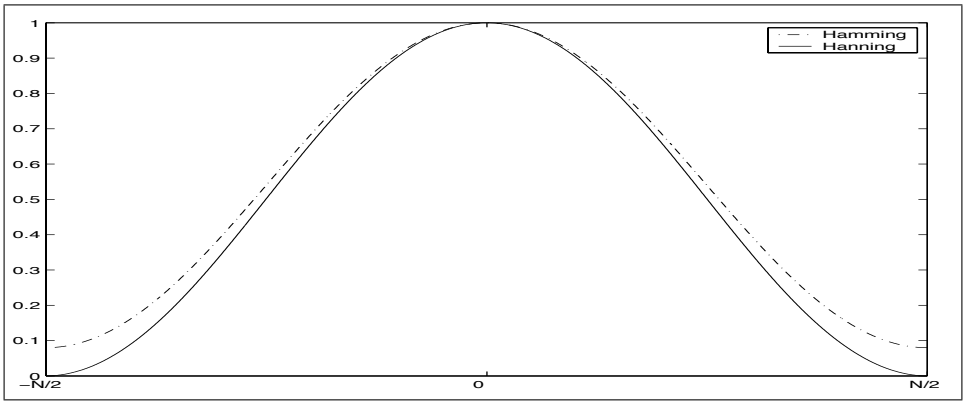


Abbildung 4.1: Darstellung des Hamming- und des Hanning Fensters der Länge $N = 1024$.

Die Vorteile dieser Fensterfunktionen liegen darin, dass Signalwerte, welche am Rande des Fensters liegen, ein geringeres Gewicht bekommen. Auf diese Weise lassen sich *Randeffekte* reduzieren. Anschaulich beschrieben bildet der WFT-Operator ein Signal f auf eine Folge von Spektren ab. Da das menschliche Gehör relativ unempfindlich gegenüber der Phase eines Monosignals ist, nutzt man in der Praxis häufig das *Energiespektrum*, welches man durch die Quadrierung der Absolutbeträge der einzelnen Fourierkoeffizienten erhält. Die dadurch entstehende Folge von reellwertigen Spektren wird als *Spektrogramm* bezeichnet und bietet eine gute Visualisierung der in einem Signal auftretenden Frequenzen über die Zeit.
Um zumindest manche Signale aus $L^2(\mathbb{R})$ mit dem Rechner verarbeiten zu können, wird ein Signal $f \quad L^2(\mathbb{R})$ durch *Abtastung (Sampling)* in ein Signal $x \quad {}^2(\mathbb{Z})$ überführt. Um die Abtastung sinnvoll definieren zu können, benötigt man den Begriff der *Bandbeschränktheit* eines Signals. Ein Signal f $L^2(\mathbb{R})$ heißt Ω-*bandbeschränkt*, wenn für alle $\mathbb{R}$ mit $|\ | > \Omega$ stets $f(\ ) = 0$ gilt. Nach dem Satz von Bernstein liegt in der Äquivalenzklasse $f + N_2$ eines Ω-bandbegrenzten Signals genau ein stetiger Repräsentant, den wir wieder mit f bezeichnen. Wenn wir von Abtastung sprechen, meinen wir im

folgenden immer diesen stetigen Repräsentanten f. Nun besagt das Abtasttheorem von Shannon[1], dass eine Ω-bandbeschränkte Funktion f aus der Abtastfolge $(f(n\ 2\Omega))_{n\in\mathbb{Z}}$ über die Formel

$$f(t) = \sum_{n\in\mathbb{Z}} f(\frac{n}{2\Omega})\text{sinc}(2\Omega t - n)$$

rekonstruiert werden kann, wobei die sinc-Funktion definiert ist durch $\text{sinc}(t) := \sin(\pi t)\ (\pi t)$. Signale auf einer CD liegen mit einer Rate von $F_s = 44100$ Abtastungen pro Sekunde vor. Dies bedeutet, dass auf einer CD Signale Frequenzen bis zu 22050 Hz enthalten können. Die Frequenz Ω wird *Nyquist-Frequenz* genannt. Genügt die Abtastung eines analogen Signals den Voraussetzungen des Abtasttheorems, so bedeutet dies, dass durch die Abtastung kein irreversibler Verlust an Information stattfindet.
Bisher wurde die Frequenzanalyse für Signale aus $L^2([0,1])$ und $L^2(\mathbb{R})$ betrachtet. Für den Fall eines endlichen Signals, also etwa $x \quad \mathbb{C}^N$, geschieht die Fouriertransformation durch Multiplikation des Spaltenvektors $x = (x_k)_{0\le k<N}$ mit der sog. DFT_N-Matrix

$$\text{DFT}_N \quad := \quad \frac{1}{\overline{N}}(W_N^{kn})_{0\le k\ n<N},$$

wobei $W_N := e^{-2\pi i\ N}$ eine *primitive N-te Einheitswurzel* ist. Für diese speziellen Matrix-Vektor-Multiplikationen gibt es schnelle Algorithmen, die unter dem Namen *Fast Fourier Transforms* (FFT) bekannt sind, siehe z.B. [CB93]. Diese FFT-Algorithmen erlauben die Berechnung der Fouriertransformierten $x = \text{DFT}_N \cdot x$ mit nur $O(N \log N)$ arithmetischen Operationen.
Ausgehend von der DFT_N kann nun auch die diskrete Version der WFT definiert werden als

$$\text{WDFT}_{g\ s}[x](k,n) \quad := \quad \sum_{=0}^{N-1} g(\ )x(ns+\ )W_N^k$$

mit einer diskreten Fensterfunktion $0 = g \quad \mathbb{C}^N$ und einer *Schrittweite* s.

4.1.3 Quantisierung

Ein zentraler Bestandteil beim Umgang mit Audiosignalwerten im Rechner ist deren *Quantisierung*. Auf einer Audio-CD sind pro Sekunde und Kanal 44100 Abtastwerte mit einer Auflösung von je 16 Bit gespeichert. Dies bedeutet, dass bei der Abtastung des analogen Signals $f \quad L^2(\mathbb{R})$ der Abtastwert $f(t)$ auf einen Wert im Intervall $[0 : 2^N - 1]$, hier $N = 16$, abgebildet wird. Allgemein ist ein *Quantisierungsoperator* mit einer *N-Bitau ösung* eine ordungserhaltende surjektive Abbildung $Q\colon \mathbb{R} \quad [0 : 2^N - 1]$. Mit $I_j := \{r \quad \mathbb{R} \mid Q(r) = j\}$, für $j \quad [0 : 2^N - 1]$, zerfällt $\mathbb{R}$ also disjunkt in die Intervalle $I_0, \ldots I_{2^N-1}$ und für $p \quad I_j$ sowie $q \quad I_{j+1}$ gilt stets $p < q$. Die Breite Δ der Intervalle I_j wird als *Quantisierungsschrittweite* bezeichnet. Abbildung 4.2 zeigt beispielhaft die lineare Quantisierung einer Cosinusschwingung.
Bei der linearen Quantisierung eines reellwertigen Signals zu einem Signal mit einem Wertebereich von z.B. $[0 : 2^N - 1]$ entsteht ein Fehler, der als *Quantisierungsrauschen* bezeichnet wird. Der sog. *Signal-Rauschabstand (SNR)* ist ein Maß für den Fehler, der bei der Quantisierung entsteht. Der SNR ist das Verhältnis von Signalenergie und der Energie eines Fehlersignals. Sei $x_q(t) = Q[x](t)$ die quantisierte Version des analogen Signals $x(t) = A\cos(\Omega t)$ (hier $A = \Omega = 1$). Unter der Annahme,

[1] Dieses Theorem geht eigentlich auf folgende Personen zurück: Shannon (1949), Kotelnikov (1933), Whittaker (1915) und de la Vallée Poussin (1908).

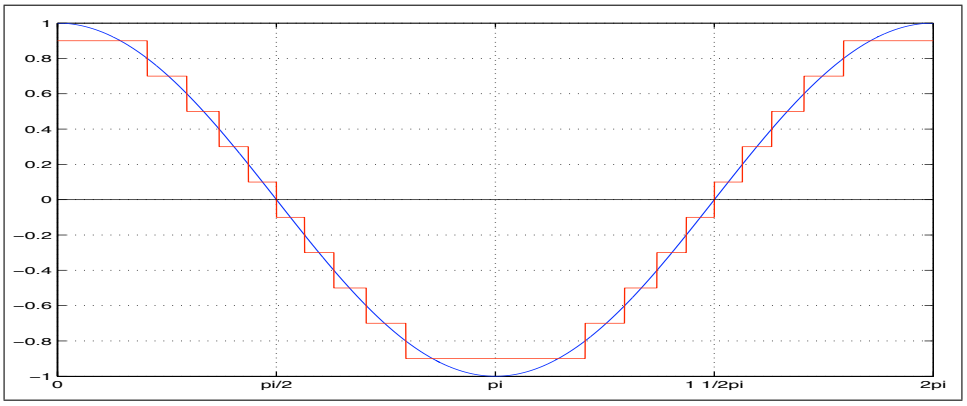


Abbildung 4.2: Beispiel für eine lineare Quantisierung einer Cosinusschwingung. Das Intervall $[-1, 1] \subset \mathbb{R}$ wird auf $N = 10$ Quantisierungsstufen $(\pm 0.1, \pm 0.3, \pm 0.5, \pm 0.7, \pm 0.9)$ quantisiert. Im Falle der Audio-CD existieren 2^{16} Quantisierungsstufen für das Intervall $[-1, 1]$.

dass durch das Abtasten kein Fehler erzeugt wird (s.o.), kann das Fehlersignal $x_e(t)$ definiert werden als Differenz zwischen dem analogen und dem quantisierten Signal $x_e(t) := x(t) - x_q(t)$. Die Energie des Fehlersignals x_e ergibt sich aus

$$P_e = \frac{1}{2\tau} \int_{t_0-\tau}^{t_0+\tau} x_e^2(t)\, dt \tag{4.1}$$

$$= \frac{1}{\tau} \int_{0}^{\tau} x_e^2(t)\, dt \tag{4.2}$$

wobei $x(t_0) = Q[x(t_0)]$ und $x_q(t) = Q[x(t_0)]$ für alle $t \quad [t_0 - \tau, t_0 + \tau]$ gilt. Die Grenzen des Integrals in (4.1) beschreiben somit gerade einen Signalausschnitt, der auf die gleiche Quantisierungsstufe abgebildet wird. Im folgenden wird vereinfachend angenommen, dass der $t_0 = 0$ Mittelpunkt des Intervalls ist. Bei hinreichender Anzahl von Quantisierungsstufen kann man weiterhin annehmen, dass das Signal x in dem Intervall nahezu linear verläuft. Mit diesen Argumenten lässt sich $x_e(t)$ schreiben als $x_e(t) = (\Delta \ 2\tau) \cdot t$, d.h. der Quantisierungsfehler ist in der Nähe des Intervallmittelpunktes t_0 geringer als an den Rändern des Intervalls. Damit lässt sich P_e ausdrücken als

$$\begin{aligned} P_e &= \frac{1}{\tau} \int_{0}^{\tau} \left(\frac{\Delta}{2\tau}\right)^2 t^2\, dt \\ &= \frac{\Delta^2}{12} \\ &= \frac{A^2 \ 3}{2^{2b}} \quad (\text{mit } \Delta = 2A \ 2^b). \end{aligned}$$

Die mittlere Signalenergie ist

$$\begin{aligned} P_x &= \frac{1}{T_p} \int_{0}^{T_p} (A \cos \Omega t)^2\, dt \\ &= A^2 \ 2. \end{aligned}$$

Somit kann man nun den SNR berechnen als Verhältnis von P_x und P_e:

$$\begin{aligned} \text{SNR} &= \frac{P_x}{P_e} = \frac{3}{2} \cdot 2^{2b}, \\ \text{SNR[dB]} &= 10 \log_{10} \text{SNR} = 1,76 + 6,02 \cdot b. \end{aligned}$$

Aus (4.3) folgt direkt, dass jedes Bit mehr oder weniger Auflösung bei der Quantisierung zu einer Änderung des SNR von ca. 6 dB führt. Bei der Audio-CD ($b = 16$) beträgt der SNR ungefähr $16 \cdot 6 = 96$ dB. Im professionellen Bereich wird mit einer Auflösung von 24 Bit je Abtastwert gearbeitet, was einem SNR von ca. 144 dB entspricht.
Im Gegensatz zu der bisher diskutierten linearen Quantisierung wird in den Kodierungsverfahren MP3 und AAC ein *nicht-linearer Quantisierer* eingesetzt, um die MDCT-Koeffizienten (siehe hierzu 4.3) zu quantisieren, bevor sie Huffman-codiert werden. Dies ist darin begründet, dass in diesem Fall besonders betraglich kleine Koeffzienten mit einer höheren Genauigkeit dargestellt werden müssen als betraglich große (vgl. Abbildung 4.3 auf Seite 74).

4.1.4 Merkmale im Spektralbereich

Wie zuvor bereits angesprochen, ist die Wellenformdarstellung für die Audioidentifikation und für viele andere Anwendungen nicht geeignet, denn obwohl zwei Signale sich „sehr ähneln“, kann deren Wellenformdarstellung stark voneinander abweichen. So wird zum Beispiel bei der MP3-Kompression das Frequenzspektrum auf 16 kHz bandbeschränkt, da Maskierungseffekte die höheren Frequenzen, welche im Vergleich zu tieferen Frequenzen lauter sein müssen um wahrgenommen zu werden, und somit eine Kodierung von Frequenzen > 16 kHz nicht sinnvoll ist. So klingen, eine entsprechende Bitrate bei der Kodierung vorausgesetzt, CD-Aufnahme und MP3-komprimierte und bandbeschränkte Version derselben Interpretation eines Musikstücks für einen ungeübten Hörer gleich. Die Wellenformdarstellungen sind jedoch deutlich voneinander verschieden!
Aus diesem Grund werden in vielen Applikationen im Bereich Music-IR und der Sprachverarbeitung spektrale Eigenschaften von Musik- bzw. Sprachsignalen verwendet. Einige dieser Eigenschaften sollen hier vorgestellt werden, da sie in den in Kapitel 5 diskutierten Systemen Verwendung finden.

4.1.4.1 Mel Frequency Cepstral Coef cients (MFCC)

Besonders im Bereich der Sprachverarbeitung und -erkennung werden die MFCC verwendet. In [Log00] motiviert die Autorin die Verwendung von MFCC auch für Musiksignale. Grundlage für die MFCC bildet die sog. *Mel-Skala*, welche das Frequenzspektrum in Anlehnung an die menschliche Hörwahrnehmung auf eine logarithmische Skala vermöge

$$\text{MEL}(f) = 2595 \cdot \log \left(1 + \frac{f}{700} \right)$$

abbildet, wobei hier f ein Frequenzwert gemessen in Hertz ist. Dieser Definition liegt die Beobachtung zu Grunde, dass das menschliche Gehör Frequenzen nicht linear analysiert. Vielmehr werden mit steigender Frequenz immer größere Frequenzbereiche zusammen analysiert. Für Frequenzwerte unter 1000 Hz ist die Abbildung (4.3) nahezu linear, danach logarithmisch.

4.1.4.1.1 Berechnung der MFCC Die MFCC-Berechnung erfolgt Frame-basiert. Die einzelnen Frames werden durch Fensterung des Signals $x \quad ^2(\mathbb{Z})$ mit einem Hamming-Fenster gewonnen.

Durch das Hamming-Fenster werden Randeffekte reduziert. Meist weisen die Frames eine anwendungsabhängig gewählte Überlappung auf. Für jeden Framevektor $v \quad \mathbb{R}^N$ werden die folgenden Berechnungsschritte durchgeführt:

DFT Jeder Framevektor wird mittels der DFT in den Frequenzbereich transformiert. Anstatt das komplexwertige Spektrum zu betrachten, wird auf die Phaseninformation verzichtet und stattdessen zum Betrag des DFT-Ergebnisses, dem sog. *Amplituden-Spektrum*, übergegangen.

Log Motiviert durch die Tatsache, dass das Lautstärkeempfinden beim Menschen logarithmischer Natur ist und zum Beispiel deshalb die dB-Skala logarithmisch ist, wird der Logarithmus des Amplituden-Spektrums berechnet.

Mel-Scaling/Smoothing Gemäß der Mel-Skala werden Frequenzen zu B Bändern zusammengefasst und für jedes Band der Mittelwert bestimmt. Auf diese Weise erhält man eine geglättete und kompakte Darstellung des Frequenzspektrums mit Vektoren aus $\mathbb{R}^B_{\geq 0}$.

Dekorrelation Die einzelnen MFCC-Vektoren aus $\mathbb{R}^B_{\geq 0}$ sind stark korreliert.[2] Durch eine dekorrelierende Transformation werden diese Redundanzen entfernt und die Dimension der MFCC-Vektoren auf $B' < B$ reduziert.

Abschließend kann die Berechnung der MFCC Vektoren für einen Framevektor $v \quad \mathbb{R}^N$ geschrieben werden als

$$\text{MFCC}[v] \quad := \quad \text{DCT}_{B'} \circ \text{MEL}_B \circ \log |\text{DFT}_N[v]|.$$

4.1.4.1.2 Hauptachsen-Transformation Eine theoretisch optimale Dekorrelation einer *Beobachtungsmatrix* $X \quad \mathbb{R}^{M \times N}$, deren M Zeilen Beobachtungen von N Zufallsvariablen darstellen, ist die *Hauptachsen-Transformation*. Ist μ_i der Mittelwert der Einträge der i-ten Zeile von X, so wird in einem ersten Schritt die Matrix $X = (X_{ij})$ zentriert, indem man $X^* = (X^*_{ij})$ mit $X^*_{ij} := X_{ij} - \mu_i$, bildet und dann die *Kovarianzmatrix* $\text{Cov}(X) \quad \mathbb{R}^{M \times M}$ durch

$$\text{Cov}(X) \quad := \quad \frac{1}{M-1} \cdot X^* \cdot (X^*)^t$$

berechnet. Die Kovarianzmatrix kann als reelle symmetrische Matrix durch Konjugation mit einer geeigneten orthogonalen Matrix P in eine Diagonalmatrix D überführt werden: $D = P^{-1}\text{Cov}(X)P$. Dabei bestehen die Spalten von P aus Eigenvektoren der Kovarianzmatrix und die Diagonaleinträge von D sind die entsprechenden Eigenwerte. Die Größe eines Eigenwerts gibt die „Wichtigkeit“ des jeweiligen Eigenvektors an. Beschränkt man sich nun auf die L signifikantesten Eigenvektoren, $L < M$, und fasst diese in einer Matrix $Q \quad \mathbb{R}^{M \times L}$ zusammen, so stellt $X := Q^t \cdot X^* \quad \mathbb{R}^{L \times N}$ eine zentrierte, dimensionsreduzierte und vergröberte Version von X dar. Diese Vorgehensweise wird auch als *Principal Component Analysis (PCA)* bezeichnet.
In der Praxis hingegen wird die PCA durch die *Diskrete Cosinus Transformation (DCT)* approximiert, da die Berechnung der PCA zu aufwendig ist. Die ersten m Cosinus-Basisfunktionen der Länge N der DCT-Matrix berechnen sich wie folgt:

$$\begin{aligned} \text{DCT}_{1\,i} &= \sqrt{1\ N}, && 1 \leq i \leq N, k = 1 \\ \text{DCT}_{k\,i} &= \sqrt{2\ N} \cdot \cos \frac{\pi(2i-1)(k-1)}{N}, && 1 \leq i \leq N, 2 \leq k \leq m. \end{aligned}$$

[2] Für stochastische Grundlagen sei auf den folgenden Abschnitt verwiesen.

Die DCT ist eng verwandt mit der DFT. Sei $X := (x_0, \ldots, x_{N-1}, x_{N-1}, \ldots, x_0) \quad \mathbb{R}^{2N}$ das zu $x = (x_0, \ldots, x_{N-1}) \quad \mathbb{R}^N$ gehörige Palindrom. Durch die Symmetrie von X sind die daraus resultierenden $2N$ Fourierkoeffizenten reellwertig und es gilt (bei geeigneter Normalisierung):

$$\mathrm{DCT}_N[x] = (\mathrm{DFT}_{2N}[X])\Big|_{[0:N-1]}.$$

Somit ist es durchaus sinnvoll von einem *DCT-Spektrum* zu sprechen.

4.1.4.2 Spektrale Glattheit

Ein in MPEG-7 vorgeschlagener „Low Level Descriptor" für die Beschreibung des Inhalts von Audisignalen ist das *Spectral Flattness Measure*, dem Verhältnis von geometrischen zu arithmetischen Mittel über Bänder des Enegiespektrums eines Signals. Es ist ein Maß für die Glattheit des Spektrums. Es hat sich herausgestellt, dass dies eine durchaus charakteristische Eigenschaft eines Signals ist. Sei X das Enegiespektrum zu einem Signal x. Dann ist

$$\mathrm{SFM}[X] \quad = \quad \frac{\sqrt[N]{\prod_{i=0}^{N-1} X_i}}{\frac{1}{N}\sum_{i=0}^{N-1} X_i}$$

Häufig wird das SFM für einzelne Frequenzbänder berechnet, um die Signalcharakteristika in unterschiedlichen Frequenzbereichen getrennt voneinander analysieren zu können.

4.1.4.3 Spectral Crest Factor

Eine ähnliche Eigenschaft hat der sog. *Spectral Crest Factor*. Sei X das Enegiespektrum zu einem Signal x. Dann ist

$$\mathrm{SCF}[X] \quad = \quad \frac{\max_{0 \le i < N} X_i}{\frac{1}{N}\sum_{i=0}^{N-1} X_i}$$

das *Spectral Crest Factor* für das Energiespektrum X.
Auch der SCF wird in Anwendungen meist für einzelne Frequenzbänder berechnet.

4.2 Hidden Markov Modelle

Ein im folgenden Kapitel beschriebener Ansatz zur Audioidentifikation setzt auf die Verwendung von Hidden Markov Modellen (HMMs), um damit die benötigte Fehlertoleranz zu erreichen. Die Gruppe um Cano et al. versuchen mittels der HMMs zu erreichen, dass deren Audioidentifikationssystem auch Anfragen erlaubt, bei denen das Signal zeitlich verzerrt, z.B. um einige Prozent schneller abgespielt, vorliegt.
Um den Ansatz vorstellen zu können, folgt an dieser Stelle eine kurze Einführung in HMMs aufbauend auf [Rab89]. Auf das Training der Modelle, d.h. die Anpassung der Modellparameter auf eine Menge von Testeingaben, soll hier nicht näher eingangen werden. Ein Grund ist, dass die Angaben in [CBMN02] bzgl. des Trainingprozesses für ihr Audioidentifikationssystem recht allgemein gehalten sind.
Intuitiv handelt es sich bei HMMs um zwei homogene Markov-Prozesse. Der unsichtbare Prozess (im folgenden modelliert durch einen Markov-Prozess auf Zuständen) wird durch einen sichtbaren

Markov-Prozess, in dem Symbole beobachtet werden, überlagert. Man versucht anhand beobachteter Symbolsequenzen Rückschlüsse auf die Zustandssequenzen zu ziehen.

Ein (N, M)-HMM λ besteht formal aus

- $S = \{S_1, \ldots, S_N\}$, einer Menge von N Zuständen,
- $V = \{v_1, \ldots, v_M\}$, einer Menge von M beobachtbaren Symbolen und aus
- $2N + 1$ Wahrscheinlichkeitsverteilungen, die wie folgt bezeichnet werden:
  - P_0^s,P_1^s, ..., P_N^s Wahrscheinlichkeitsverteilungen auf S und
  - P_0^v,P_1^v, ..., P_N^v Wahrscheinlichkeitsverteilungen auf V

Die Bedeutungen der Mengen S,V sind intuitiv klar und bedürfen keiner weiteren Erläuterung. Anders verhält es sich mit den Wahrscheinlichkeitsverteilungen. Die Verteilung $P_0^s := (\pi_1, \ldots, \pi_N)$ besagt, dass sich das HMM zu Anfang mit der Wahrscheinlichkeit π_i in dem Zustand S_i befindet. $P_i^s := (a_{i1}, \ldots, a_{iN})$ bezeichnet die Wahrscheinlichkeit dafür, dass das HMM von dem Zustand S_i in den Zustand S_j übergeht. Mit $P_j^v := (b_{j1}, \ldots, b_{jM})$ werden die Wahrscheinlichkeitsverteilungen über die Menge der beobachtbaren Symbole in Abhängigkeit vom Zustand S_j bezeichnet.
Die Wahrscheinlichkeiten $P_1^s, \ldots P_N^s$ werden in der Literatur als zeilen-stochastische Matrix $A = (a_{ij}) \quad [0,1]^{N \times N}$ zusammengefasst. Analog werden die Wahrscheinlichkeiten $P_1^v, \ldots P_M^v$ zu einer zeilenstochastischen Matrix $B = (b_{jk}) \quad [0,1]^{N \times M}$ zusammengefasst. Geht man davon aus, dass die Menge der Zustände und die zu beobachteten Symbole durch die Anwendung vorgegeben sind, kann man ein (N, M)-HMM als Tripel $(A, B, \pi) schreiben.$
In den folgenden beiden Abschnitten werden zwei zentrale Fragen im Zusammenhang mit HMMs diskutiert. Zum einen interessiert, mit welcher Wahrscheinlichkeit ein gegebenes HMM eine Beobachtungssequenz erzeugt. Zum anderen ist von Interesse, welche Zustandsfolge am wahrscheinlichsten ist, wenn eine Beobachtungssequenz gegeben ist.

4.2.1 Berechnung der Wahrscheinlichkeit einer Beobachtungssequenz O

Seien ein (N, M)-HMM $\lambda = (A, B, \pi)$ und eine Beobachtungssequenz $O = (v_{j_1}, \ldots, v_{j_T}) \quad V^T$ gegeben. Von Interesse ist die Wahrscheinlichkeit $P_\lambda(O)$, mit der λ die Sequenz O erzeugt. Diese Wahrscheinlichkeit berechnet sich als Summe über alle Wahrscheinlichkeiten mit der ein beliebiger Zustandspfad $I = (i_1, \ldots, i_T) \quad [1:N]^T$ (hier als Indexmenge zu S geschrieben) die Sequenz O erzeugt:

$$P_\lambda(O) := \sum_{(i_1 \quad i_T) \in [1:N]^T} (\pi_{i_1} \cdot b_{i_1 j_1}) \cdot (a_{i_1 i_2} \cdot b_{i_2 j_2}) \cdot \ldots \cdot (a_{i_{T-1} i_T} \cdot b_{i_T j_T}) \tag{4.3}$$

$$= \sum_{(i_1 \quad i_T) \in [1:N]^T} (\pi_{i_1} \cdot b_{i_1 j_1}) \cdot \prod_{=2}^{T} a_{i_{\ell-1} i_\ell} \cdot b_{i_\ell j_\ell} \tag{4.4}$$

Die Wahrscheinlichkeit $P_\lambda(O)$ kann effizient berechnet werden, indem die Variable α in Abhängigkeit

von $(j_1, \ldots, j_T)$) wie folgt induktiv definiert wird:

$$\alpha_1(i) = \pi_i b_{ij_1}, \tag{4.5}$$

$$\alpha_{t+1}(j) = \sum_{i_t=1}^{N} \alpha_t(i_t) \cdot a_{i_t j} \cdot b_{jj_{t+1}}, \tag{4.6}$$

$$P_\lambda(O) = \sum_{i=1}^{N} \alpha_T(i) \tag{4.7}$$

Zu Beginn der Induktion (4.5) wird die Wahrscheinlichkeit berechnet, dass sich das HMM λ im Zustand i befindet und dass dabei das Symbol v_{j_1} beobachtet wird. In (4.6) wird für jeden Zustand j zum Zeitpunkt $t+1$ die Wahrscheinlichkeit dafür berechnet, dass zum Zeitpunkt $t+1$ das Symbol $v_{j_{t+1}}$ beobachtet wird und dass sich λ in dem Zustand j befindet. Diese setzt sich aus der Beobachtungswahrscheinlichkeit $b_{jj_{t+1}}$ und der Wahrscheinlichkeit dafür, dass λ sich im Zustand j befindet zusammen. Schliesslich ergibt sich die Wahrscheinlichkeit, mit der eine gegebene Beobachtungsfolge O von λ erzeugt wird, als Summe über die N $\alpha_T(i)$ (4.7). Die Berechnung von $P_\lambda(O)$ erfolgt in $O(N^2T)$ Schritten, wie aus (4.6) und (4.7) folgt.

4.2.2 Berechnung der wahrscheinlichsten Zustandssequenz zu einer Beobachtungssequenz O

Eine weitere wichtige Frage in diesem Zusammenhang ist die Frage nach der *optimalen Zustandssequenz* $I_{\text{opt}} = (i_1, \ldots, i_T) \quad [1:N]^T$ bei vorgegebener Beobachtungssequenz $O = (v_{j1}, \ldots, v_{jT})$ V^T. Optimal bedeutet hier, dass λ die Beobachtungssequenz O mit größtmöglicher Wahrscheinlichkeit erzeugt, wenn die Zustandssequenz I_{opt} der Beobachtungssequenz O zugrunde liegt:

$$I_{\text{opt}} := \underset{(i_1 \quad i_T)}{\arg\max}(\pi_{i_1} \cdot b_{i_1 j_1}) \cdot \prod_{=2}^{T} (a_{i_{\ell-1} i_\ell} \cdot b_{i_\ell j_\ell})$$

Zur Berechnung von I_{opt} dient der *Viterbi-Algorithmus* (4.2.1). Mittels dynamischer Programmierung werden in dem Algorithmus die Werte

$$\delta_t(i) = \max_{(i_1 \quad i_t)} P_\lambda(i_1, \ldots, i_t)$$

zu gegebener Beobachtungssequenz O berechnet. Die Werte $\delta_t(i)$ bezeichnen die maximale Wahrscheinlichkeit dafür, dass sich λ im Schritt t im Zustand $i = i_t$ befindet und dass dabei die Sequenz $(v_{j_1}, \ldots, v_{j_t})$ beobachtet worden ist. In Algorithmus 4.2.1 wird zusätzlich in jedem Schritt in der Variablen $\quad_t(j)$ der Zustand gespeichert, welcher bzgl. O der wahrscheinlichste Vorgängerzustand zu j ist. Im vierten Schritt des Viterbi-Algorithmus wird die gesuchte optimale Zustandssequenz mit Hilfe der Information in $\quad_t(j)$ bestimmt.

4.3 MP3-Kompression

In dem nachfolgenden Kapitel wird häufig die MP3-Kompression [ISO93] stellvertretend für verlustbehaftete Audiocodierer erwähnt. Verlustbehaftet bedeutet hier eine Kompression eines Audiosignals

Algorithmus 4.2.1: VITERBI-ALGORITHMUS($\lambda = (A, B, \pi), O = (v_{j_1}, \ldots, v_{j_T}) \quad V^T$)

1. Initialisierung:

$\delta_1(i) = \pi_i b_{ij_1}, \quad 1 \leq i \leq N$

$_1(i) = 0.$

2. Rekursion:

$\delta_t(j) = \max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij})\, b_{ij_t}, \quad 2 \leq t \leq T,\, 1 \leq j \leq N$

$_t(j) = \arg\max_{1 \leq i \leq N} (\delta_{t-1}(i) a_{ij})\,.$

3. Rekursionsende:

$P^* = \max_{1 \leq i \leq N} (\delta_T(i)),$

$I_T^* = \arg \max_{1 \leq i \leq N} (\delta_T(i)).$

4. Sequenzaufbau:

$q_t^* = \ _{t+1}(q_{t+1}^*),\ t = T-1, \ldots, 1.$

Ausgabe: $I_{\text{opt}} = q_1^* q_2^* \ldots q_T^*$.

Algorithmus 4.2.1: Der Viterbi-Algorithmus aus [Rab89] zur Bestimmung von I_{opt}, einem maximalen Summanden in (4.4)

in der Art, dass das ursprüngliche Audiosignal *nicht* mehr aus dem komprimierten Signal rekonstruiert werden kann. Die Grundlage für diese Art von Audiokompressionsalgorithmen liefert die Psychoakustik. Dieser Wissenschaftszweig befasst sich mit der menschlichen Hörwahrnehmung [ZF90]. Audiosignale in CD-Qualität weisen eine Datenrate von rund 1,4 Mbps auf. Das MP3-Kompressionsverfahren bietet bei 64 kbps pro Kanal nahezu CD-Qualität. Dies entspricht einer Kompressionsrate von 1:11. Je nach der Qualität des verwendetem Codierers kann es möglich sein, dass die Mehrzahl der Menschen das Originalsignal und das verlustbehaftet komprimierte Signal nicht mehr unterscheiden können. Ist dies gegeben spricht man von *psychoakustischer Transparenz*. Realistischerweise sollte man jedoch bei MP3 von einer Datenraten von 192 kbps (stereo) ausgehen, wenn von psychoakustischer Transparenz ausgegangen werden soll.
Die zentrale Idee bei *psychoakustischen Transformationskodierern* ist, ein zu komprimierendes Audiosignal in kurze Zeitabschnitte, sog. *Frames*, zu unterteilen und diese Frames mehr oder weniger getrennt zu kodieren. Für jeden Frame wird durch ein *psychoakustisches Modell* die sog. *Maskierungsschwelle* im Frequenzbereich berechnet, die angibt wie laut ein Ton in Abhängigkeit von seiner Frequenz sein muss, damit dieser vom menschlichen Gehör wahrgenommen wird. Diese Information wird vom Codierer genutzt, um die zur Codierung zur Verfügung stehende Anzahl von Bits so auf Teilbereiche des Frequenzspektrums zu verteilen, dass möglichst in keinem Bereich des Spektrums das erzeugte *Quantisierungsrauschen* hörbar wird. Der Pegel des Quantisierungsrauschens sollte im gesamten Spektralbereich unterhalb der Maskierungsschwelle liegen.

4.3.1 Akustische Maskierung

Das Phänomen der Maskierung ist jedem aus unzähligen Alltagssituationen bekannt. Das Telefongespräch, welches durch einen vorbeifahrenden LKW unmöglich gemacht wird, da man „sein eigenes Wort nicht mehr versteht“. Das laute Signal, welches von dem LKW erzeugt wird, „schluckt“ die ge-

sprochenen Worte, was man allgemein als *Maskierung* bezeichnet. Man muss zwei Arten der (akustischen) Maskierung unterscheiden. Zum einen maskieren Töne im Spektrum benachbarte Frequenzen. Der Einfluss eines *Maskierers*, Töne anderer Frequenz zu überdecken, nimmt mit dem Frequenzabstand der Töne ab. Zum anderen gibt es auch die zeitliche Maskierung, welche sich durch eine gewisse Trägheit des Gehörs und der Verarbeitung der Sinnesreize im Gehirn erklären lässt. Ein lauter Ton maskiert dabei einen leiseren Ton, auch wenn der leisere Ton den Hörer *vor* dem lauteren erreicht. Dieser Effekt ist nur sehr schwach, denn der Abstand des Einsetzens der beiden Töne sollte dazu 20 ms nicht überschreiten. Ausgeprägter ist da der Effekt, dass das Gehör nach dem Abklingen eines lauten Tons erst wieder nach etwa 150 ms in der Lage ist, leise Töne wahrzunehmen. Umgangssprachlich kann man den Maskierungseffekt als „akustischen Schattenwurf“ eines Tons im Zeit- wie im Frequenzbereich ansehen.
Das psychoakustische Modell (PM) eines MP3-Codierers versucht nun das menschliche Hörverhalten zu simulieren, um für jede Frequenz im Bereich von 0 Hz bis 20.000 Hz die Amplitude zu bestimmen, ab der ein Ton wahrgenommen werden würde. Das PM ist ein zentraler Bestandteil eines jeden Codierers und wird deshalb in seinem Detailaufbau auch nur selten offengelegt. Nachfolgend soll kurz eine Einführung gegeben werden. Mehr Details finden sich z.B. in [Rib00, ISO93].

4.3.2 Der Codiervorgang

Bei der MP3-Codierung werden nicht die Abtastwerte $x(n)$ eines digitalen Audiosignals $x \quad ^2(\mathbb{Z})$ direkt quantisiert, sondern das Signal wird durch eine 32-Band Multiratenfilterbank zunächst in 32 Subbandsignale unterteilt und diese durch eine modifizierte diskrete Cosinus-Transformation (MDCT) (siehe z.B. [PB86]) in den Frequenzbereich transformiert. Dabei werden verschiedene Arten der MDCT angewendet, um der Signalcharakteristik gerecht zu werden. Dies ist besonders bei Signalteilen wichtig, die sich innerhalb von wenigen Millisekunden stark ändern.

Die dabei berechneten MDCT-Koeffizienten werden nach Möglichkeit mit einem *logarithmischen Quantisierer* so quantisiert, dass das durch die Quantisierung hervorgerufene Rauschen unterhalb der Maskierungsschwelle liegt. Stehen dazu bei geringen Bitraten nicht genügend Bits zur Verfügung, können *Kompressionsartefakte* hörbar werden. Gelingt es jedoch ein MDCT-Spektrum mit weniger Bits als vorgesehen psychoakustisch transparent zu codieren, können die nicht genutzten Bits über ein *Bitreservoir* bei der Codierung nachfolgender Frames genutzt werden. In einem abschließenden Schritt werden die quantisierten MDCT-Koeffizienten mittels Huffman-Codierung zusätzlich verlustfrei komprimiert, wobei die Codetabellen standardisiert [ISO93] sind und damit nicht übertragen zu werden brauchen.

Um die Quantisierung nicht global für alle Frequenzbereiche durchführen zu müssen, ist das Frequenzspektrum in Bänder ähnlich den sog. *kritischen Bändern* unterteilt, die mit steigender Frequenz breiter werden. Das menschliche Gehör fasst im Prinzip alle Frequenzen in einem kritischen Band zu einer Hörwahrnehmung zusammen, weshalb sich diese Unterteilung des Frequenzspektrums anbietet.

Durch Steuern der Quantisierungsauflösung in jedem der Bänder, kann bei ausreichender Anzahl von zur Verfügung stehender Bits das Quantisierungsrauschen auf diese Weise unterhalb der Maskierungsschwelle gehalten werden. Jedem der Frequenzbänder ist ein sog. *Skalierungsfaktor* zugewiesen, der die Quantisierungsauflösung q_s in dem Band steuert. (Die folgenden Formeln werden weiter unten

näher erläutert.)

$$q_s := \frac{1}{4}(g_{\text{global}} - g_{\text{subblock}}) - \lambda(\text{scf}_s + \delta_{\text{pretab}} \cdot \lambda_{\text{pretab}}), \tag{4.8}$$

$$Q_{\text{MP3}}(x_i, q_s) := \text{nint}\left[\left(\frac{x}{2^{q_s}}\right)^{\frac{3}{4}} - 0.0964\right], \tag{4.9}$$

$$x \qquad Q_{\text{MP3}}(x, q_s) = x. \tag{4.10}$$

Nach (4.8) ergibt sich die Quantisierungsauflösung eines Skalierungsfaktorbandes aus verschiedenen Parametern. Die Variable $g_{\text{global}} \quad [-210 : 45] \subset \mathbb{Z}$ ist die für ein sog. *Granule* global geltende Quantisierungsauflösung, in Relation dazu die Skalierungsfaktoren zu sehen sind. Um die Kodierung besser an ein Audiosignal anpassen zu können, wird ein Frame in zwei zeitlich aufeinander folgende Granules unterteilt, die jeweils für sich kodiert werden. Da sich in der Regel ein Signal nicht sehr stark innerhalb eines Frames ändert, können Kodierungsparameter des ersten Granules eines Frames auch im zweiten verwendet werden, was zur Einsparung von Bits im Datenstrom führt. Diese eingesparten Bits stehen zusätzlich zum normalen Bitbudget bei der Kodierung der Frequenzinformation zur Verfügung.
Auf den Skalierungsfaktor scf_s hat die sog. *Preemphasis* λ_{pretab} Einfluss, d.h. die generelle Verstärkung hoher Frequenzen, welche in der Stärke in [ISO93] festgelegt ist. Diese wird durch $\delta_{\text{pretab}} \quad \{0, 1\}$ ein- bzw. ausgeschaltet. Der Faktor $\lambda \quad \{\ \overline{2}, 2\}$ gibt die Größenordnung der Preemphasis vor, um auf diese Weise eine kürzere Binärdarstellung der Skalierungsfaktoren zu erreichen. Insgesamt gilt, je kleiner q_s ist, desto mehr Quantisierungsstufen stehen zur Verfügung und desto genauer werden die Frequenzinformationen dargestellt.

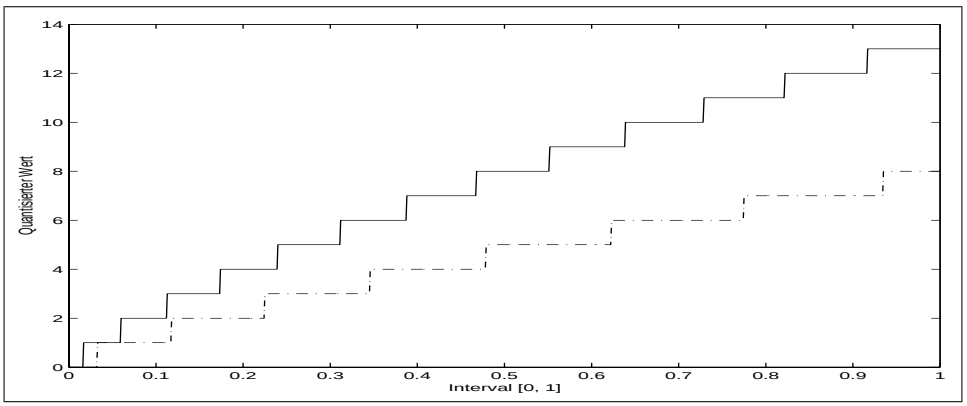


Abbildung 4.3: Quantisierung des Intervals [0,1] mit $q = -5$ (durchgezogene Linie) und $q = -4$ (gestrichelte Linie). Die zunehmede Breite der „Treppenstufen“ zeigt an, dass kleine Werte x genauer dargestellt werden als größere Werte.

Für die *Requantisierungsoperator* gilt

$$Q^{-1}_{\text{MP3}}(x, q_s) := x^{4\ 3} \cdot 2^{q_s}.$$

Die derart quantisierten MDCT-Koeffizienten werden mit standardisierten Huffmancodes verlustfrei kodiert und durch einen Multiplexer in einen Datenstrom (vgl. Abbildung 4.4) transformiert.
Durch welchen Algorithmus die Kodierungsparameter bestimmt werden, ist nicht durch den MPEG-Standard festgelegt. Dieser beschreibt lediglich den Datenstrom und den Dekodierungsprozess. Auf

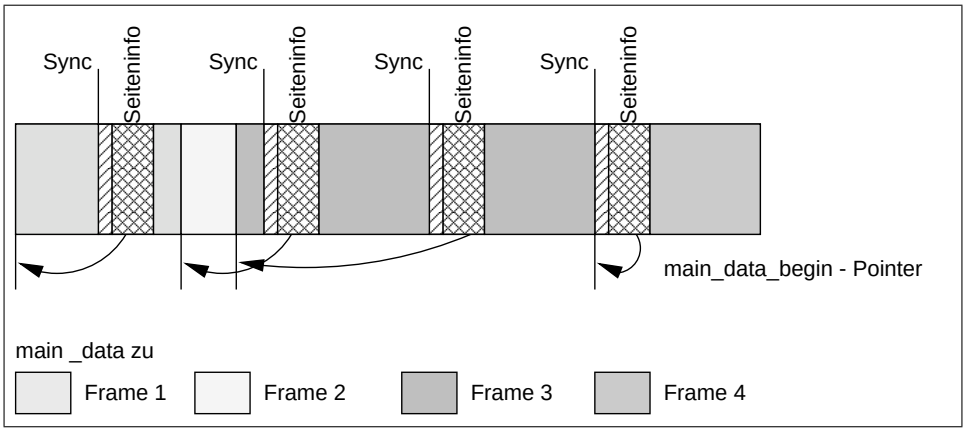


Abbildung 4.4: Ausschnitt aus einem MP3-Datenstrom (nach [ISO93]).

diese Weise können immer ausgefeiltere Kodierungsalgorithmen und psychoakustische Modelle entwickelt werden. Solange der dabei erzeugte Datenstrom dem Standard entspricht, kann das komprimierte Signal von jedem Dekoder dekomprimiert werden.

4.3.3 Decodierung

Die Dekodierung des Bitdatenstroms lässt sich in mehreren Stufen bis hin zu den Abtastwerten im Zeitbereich beschreiben.

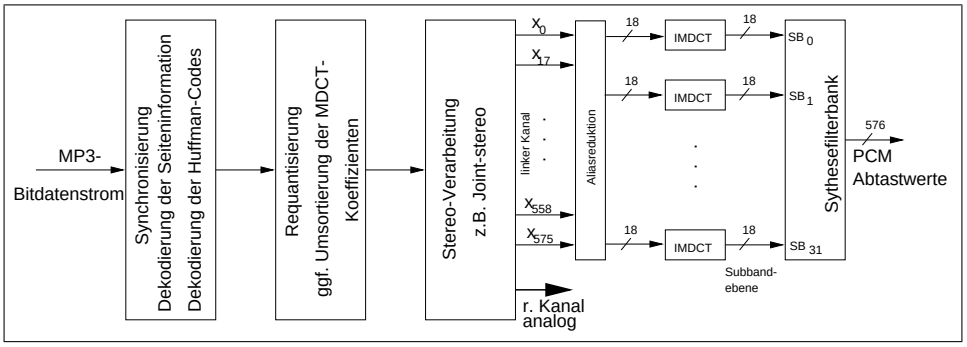


Abbildung 4.5: Decodierung des MP3-Datenstroms (nach [ISO93]).

Wie in Abbildung 4.5 ersichtlich, beginnt die Decodierung damit, den Beginn eines Frames zu lokalisieren und die im Datenstrom gespeicherten Informationen zu diesem Frame auszulesen. Die Seiteninformationen ermöglichen die Huffman-codierten MDCT-Koeffizienten zu decodieren und zu requantisieren. Danach werden diese getrennt nach Kanal mittels der IMDCT mit vorgeschalteter *Aliasreduktion* in Subbandsignale transformiert. Die Ausgabe der IMDCT besteht aus 32 Subbandsignalen zu je 18 Abtastwerten. Die Subbandsignale sind die Eingangsignale der Synthesefilterbank, die die Subbandsignale in eine Folge von 576 Abtastwerten im Zeitbereich transformiert.
Jeder Frame-Beginn im MPEG-Datenstrom ist mit dem *Synchronisationswort* 0xFFF markiert. In den auf das Sync-Word folgenden 20 Bits sind Informationen zur Bitrate, Abtastrate etc. als Tabellenindizes codiert. Optional folgt diesen vier Bytes ein 16-Bit Block zur Erkennung von Fehlern (CRC)

im Datenstrom. Die folgenden 32 Bytes (bei Stereosignalen) enthalten die *Seiteninformationen*, die zur Decodierung der eigentlichen Frequenzinformation in einem Frame notwendig sind. Besonders zu erwähnen ist der sog. „main_data_begin“-Zeiger, welcher als negativer Offset im Datenstrom ab dem Beginn des Sync-Wortes zu verstehen ist. Dieser gibt an, wo sich im Datenstrom die eigentliche Frequenzinformation und Skalierungsfaktoren zu einem Frame befinden. Durch diesen Zeiger wird das Bitreservoir bei der Codierung realisiert, da die Frequenzinformationen zu einem Frame auch mehr Speicherplatz beanspruchen können als zwischen zwei Sync-Words und Header zur Verfügung stehen. Für den Zeiger sind jeweils 12 Bit reserviert, was auf einen maximalen Abstand von Sync-Word zu den Frequenzinformationen von 4095 Bytes im Datenstrom bedeutet. Dabei werden die Header- und Seiteninformation nicht mitgezählt.
Die Abbildung 4.5 verdeutlicht den mehrstufigen Dekodierungsprozess. In folgenden Kapitel wird es unter anderem auch darum gehen, Merkmale aus MP3-Datenströmen zu extrahieren, die sich zur Audioidentifikation nutzen lassen. Da nur der Bitdatenstrom und das Dekodierungsverfahren standardisiert sind, können unterschiedliche Enkoder ein und dasselbe Audiosignal in unterschiedlichen MP3-Datenströmen überführen. Aus Sicht der Audioidentifikationsapplikation ist es daher interessant zu prüfen, auf welcher Stufe der Dekodierung sich Merkmale gewinnen lassen, die auch über unterschiedliche Enkoder hinweg nur wenig unterscheiden.

Kapitel 5

Inhaltsbasierte Suche in Audiodaten

Bereits im vorhergehenden Kapitel wurde zur Evaluierung von Suchalgorithmen ein *Audio-Index* benutzt, d.h. ein Suchindex basierend auf dem Prinzip der G-invertierten Listen, wie er im ersten Kapitel vorgestellt worden ist. Eine Anwendung für einen solchen Suchindex ist die sog. *Audioidenti kation*. Nach einem unbekannten Audiosignal(-fragment) wird in einer Kollektion von Audiosignalen gesucht. Wird ein Treffer gefunden, kann dem unbekannten Signal oder Signalausschnitt ein Name gegeben werden. Neben dem Namen bietet diese *inhaltsbasierte* Identifikation des angefragten Signals die Möglichkeit, weitere Metainformationen zu dem eigentlichen Signal dem Benutzer zur Verfügung zu stellen und so eine Sammlung von Audiosignalen mit hochwertigen Metadaten zu versehen, um danach mit Hilfe der Metadaten eine Navigation durch den Datenbestand zu ermöglichen. Bei einem eindeutigen Ergebnis der Identifikation kann z.B. einem Kunden direkt eine Kaufoption für den Titel oder dem gesamten zugehörigen Album angeboten werden.

Aufgrund des kommerziellen Einsatzes von Audioidentifikationssystemen, werden in diesem Kapitel einige Systeme vorgestellt, die von unterschiedlichen Forschergruppen entwickelt worden sind. Besonders erfolgreich am Markt behaupten sich die Systeme von Shazam und Philips, weshalb diese auch ausführlich vorgestellt werden. Dies bildet die Grundlage für den Vergleich dieser kommerziell erfolgreichen Systeme mit dem von der AG Clausen entwickelten Systems. Es wird sich zeigen, dass das System der AG Clausen sich keinesfalls verstecken muss.

Einige Ansätze verfolgen das Ziel, auch bei sehr schlechter Signalqualität auf Seiten der Anfrage (Umgebungsgeräusche, Aufnahme mit dem Mobiltelefon aus der Entfernung, verlustbehaftete Signalkomprimierung etc.) ein eindeutiges Identifikationsergebnis bei einem großen Datenbestand von möglichen Audiosignale zu liefern. Andere Verfahren wiederum widmen sich dem speziellen Problem der Tempoveränderung eines Signals, bevor es z.B. im Radio gesendet wird. Diese Fähigkeit wird i.A. als *Robustheit* bezeichnet.

Allen Verfahren ist gemein, dass die Audiosignale geeignet vorverarbeitet werden, und die Suche dann auf Basis dieser Darstellung durchgeführt wird. Dabei wird vor allem die Menge der zu verarbeitenden Daten reduziert. Im Rahmen dieser Arbeit sprechen wir bei diesem Vorverarbeitungsschritt von der *Merkmalsextraktion*.

Dieses Kapitel gliedert sich wie folgt. Nachdem einige Systeme in Abschnitt 5.1 vorgestellt worden sind, wird das Vorgehen bei der Audioidentifikation im Falle eines G-invertierten Suchindexes vorgestellt. Nicht zuletzt aus dem Grund, da dieses Verfahren das sicherlich am allgemeinsten einsetzbare Verfahren ist, wie das vorherige Kapitel gezeigt hat und das nachfolgende noch zeigen wird. Wohingegen einige der vorgestellten Systeme eben gerade „nur“ die Aufgabe der Audioidentifikation lösen und eine Anpassung an andere Aufgaben sicherlich als aufwändiger einzuschätzen ist. Den Abschluss

dieses Kapitels bildet die Vorstellung des „SyncPlayer“-Projektes, in dem eine Abspielsoftware für Audiodateien entwickelt worden ist, die dem Benutzer synchron zum abgespielten Musikstück Noten, Texte etc. anzeigt. Dabei spielt eine in der AG Clausen entwickelte Audioidentifikationskomponente basierend auf G-invertierten Listen eine entscheidende Rolle, da mit Hilfe dieses Dienstes Metadaten sicher dem gerade spielenden Stück zugeordnet werden können. Ausserdem erlaubt die durch die Audioidentifikation ermittelte Zeitposition eine exakte zeitliche *Synchronisierung* zwischen der Abspielsoftware und den Zeitinformationen in den Metadaten.

5.1 Systeme zur Audioidenti kation

5.1.1 Shazam

Die Firma Shazam [Sha03] bietet in einigen europäischen Ländern, darunter England, einen Audioidentifikationsservice per Mobiltelefon an. Der Kunde ruft das Rechenzentrum der Firma an und übermittelt per Mobiltelefon einen bis zu 30 Sekunden langen Ausschnitt einer gerade spielenden Musik z.B. aus dem Radio. Laut den Aussagen in [Wan03] reichen meist schon 15 Sekunden und zum Teil auch weniger aus, um ein Audiosignal zu identifizieren. Dabei unterliegt das übermittelte Signal einer Reihe von Störeinflüssen, bis es im Rechenzentrum verarbeitet werden kann. So kann es in manchen Anwendungsfällen sehr schwer sein, das Mikrofon des Mobiltelefons so dicht an einen Lautsprecher heranzubringen, was eine generell schlechtere Qualität zur Folge hat. Aber auch die dabei aufgenommenen Umgebungsgeräusche überlagern sich mit dem eigentlichen Signal. Daneben wird das Musiksignal mit einem verlustbehafteten Sprachcodierer, in der Regel GSM [RWO95] codiert, um die Sprachinformation über das Mobilfunknetz senden zu können. Da der Codierer primär dafür ausgelegt ist Sprache effizient zu codieren, lässt dessen Leistung bei Musik zu wünschen übrig. Weiterhin kann es im Mobilfunknetz auch zu Paketverlusten kommen, die das Signal zerstückeln. Insgesamt kann davon ausgegangen werden, dass die Signalqualität, die zur Identifikation zur Verfügung steht, in der Regel sehr schlecht ist. Dennoch muss eine Identifikation in der Hauptzahl der Anfragen korrekt sein, um einen kommerziellen Erfolg zu ermöglichen. Entsprechend hoch sind die Anforderungen an das System.

5.1.1.1 Merkmalsextraktion

Die Extraktion von stabilen Merkmalen ist bei dieser Applikation besonders wichtig, da die zu erwartende Signalqualität sicherlich gering sein wird. Dazu wird zu einem zu indexierenden diskreten Signal $x \quad ^2(\mathbb{Z})$ das sog. *Spektrogramm* $(S_i)_{i\in I}$, berechnet. Wie in Kapitel 4 definiert, ist dies eine Folge von $|I|$ Spektren, die mittels einer gefensterten Fouriertransformation berechnet werden. Jedes Spektrum S_i beschreibt die Frequenzanteile des Signals x um eine gewisse Zeitposition herum. $S_{i\,j}$ bezeichnet die j-te Frequenzkomponente des i-ten Spektrums.
Zur Merkmalsberechnung wird nun in der Matrix $(S_{i\,j})$ mit einem Maximum-Operator $\mathcal{M}$ nach lokalen Maxima gesucht. Dass diese sich als Merkmale eignen, wird z.B. in [Yan01] beschrieben. Beispiele für einen solchen Operator sind $\mathcal{M}_{\square}^{\gamma\ \delta}[S]$, der an der Stelle (i,j) den Wert 1 berechnet, wenn in der Umgebung $[i-\gamma : i+\gamma] \times [j-\delta : j+\delta]$ von (i,j) alle betreffenden Werte in S kleiner sind als $S_{i\,j}$. Anstelle dieser rechteckigen Umgebung um (i,j), werden im Falle $\mathcal{M}_{\circ}^{\rho}[S]$ die Stellen in S mit $S_{i\,j}$ verglichen, der Euklidscher Abstand von dem Punkt (i,j) kleiner oder gleich ρist.
Neben der einfachen Suche nach einem Maximum in dem Bereich, der durch ein gewähltes Fenster definiert wird, kann zusätzlich ein Schwellwert T festgelegt werden, der einen Mindestabstand

des Maximums von seiner Umgebung fordert. Auf diese Weise lassen sich lokale Maxima bestimmen, die mit größerer Wahrscheinlichkeit auch bei schlechterer Signalqualität bei Anwendung des gleichen Maximum-Operators wieder bestimmt werden. Im Gegenzug reduziert sich die Anzahl der extrahierten lokalen Maxima auf diese Weise. Dies kann zur Folge haben, dass ein Signal, welches eine Suchanfrage darstellt, länger sein muss, um eine bestimmte Mindestanzahl von lokalen Maxima extrahieren zu können.
Die Art und die Größe des gewählten Fensters haben entscheidenden Einfluss auf die Robustheit der extrahierten Maxima. Durch die Extraktion von lokalen Maxima erhält man, je nach Wahl des Fensters oder eines Schwellwerts, eine nur dünn besetzte Matrix $(\mathcal{S}_{i\,j}) := \mathcal{M}[S]$, die an den Stellen einen Eintrag $= 0$ hat, an denen ein lokales Maximum gefunden wurde. Über die Fenstergröße und über Schwellwerte lässt sich die Anzahl zu extrahierender Merkmale pro Sekunde sehr gut steuern. Auf diese Weise kann die Merkmalsdichte der jeweiligen Signalqualität gemäß der Aufgabenstellung angepasst werden. So benötigt man bei Anfragen hoher Qualität nur eine geringe Anzahl von Merkmalen, womit sich die Größe der Datenbank und der benötigte Zeitaufwand zur Suche verringert.

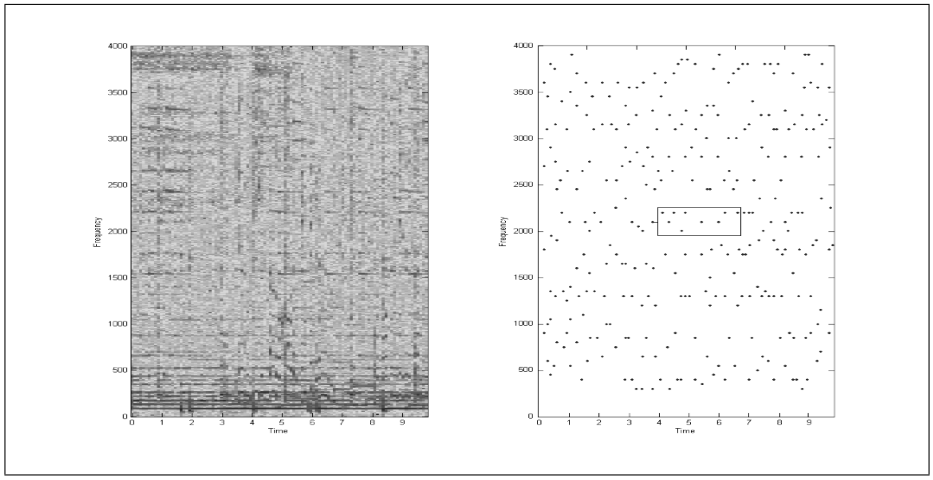


Abbildung 5.1: Die Abbildung links zeigt das Spektrogramm eines 10 Sekunden langen Audiosignals, die Abbildung rechts die daraus extrahierten Maxima. Der rechteckig markierte Bereich beschreibt den sog. *Zielbereich* zu einem Punkt.

Prinzipiell ergibt sich hier nun die Aufgabenstellung, in einer großen Sammlung von Punktkonstellationen in der Zeit/Frequenz-Ebene nach einer angefragten Konstellation zu suchen. Bei einer Anzahl von ca. 20.000 Musikstücken beträgt die Anzahl von Punkten, die in der Datenbank gespeichert werden müssen, schon viele Millionen. Fasst man die Punktmengen der in der Datenbank abgelegten Stücken als ein großes Dokument auf bei dem die einzelnen Stücke aneinander gereiht werden, würde man bei einer naiven Vorgehensweise die Anfragekonstellation Q entlang der Zeitachse in der Datenbank verschieben und zu jedem Zeitpunkt $t \quad [0 : n]$ prüfen, in wie weit die Anfragekonstellation an der Stelle t mit der Punktkonstellation der Datenbank übereinstimmt. Ist die Anfrage in mindestens einem Musikstück aus der Datenbank enthalten, dann ist an entsprechenden Stellen t die Übereinstimmung signifikant größer als an anderen Stellen. Diese Ansatz erfordert $O(|Q| \cdot n)$ viele Überprüfungen, ob ein Punkt der Anfrage mit einem Punkt in der Datenbank zusammenfällt. Zusätzlich muss die Zeitkoordinate für jeden Punkt in der Anfrage Q nach einem durchgeführten Vergleich der Punktkonstellationen um eine Stelle verschoben werden. Dies entspricht zusätzlich $|Q| \cdot (n-1)$

vielen Additionen. In der Praxis ist n im hohen zweistelligen Millionenbereich anzusiedeln, $|Q|$ im Bereich von 50 bis 100.
Einzelne Merkmale werden als Tupel (f, t) bestehend aus einem Frequenzindex f und einem Zeitindex t aufgefasst. Da eine Anfrage Q aus einer beliebigen Stelle eines Audiosignals bestehen kann, ist die absolute Zeitangabe t zunächst ohne direkten Wert, da die Zeitpunkte in Q nicht mit den Zeitpunkten der Punkte in der Datenbank übereinstimmen müssen. Schließlich kann Q ja auch ein Ausschnitt aus der Mitte eines Musikstücks, z.B. ab der Position t', sein. Die zeitliche Verschiebung $t' - t$ von Q gegenüber der zugehören Position in einem Datenbankdokument ist also durch das Identifikationsverfahren zu bestimmen.
Geht man von einer Frequenzauflösung von 1024 Frequenzkomponenten in jedem Spektrum S_i aus, so entspricht der Informationsgehalt eines jeden Punktes gerade einmal zehn Bit, sofern überhaupt alle Frequenzkomponenten gleichmäßig vorkommen. Es fällt auf, dass bei diesem rein punktbasierten Ansatz die Anordnung der Punkte zueinander nicht berücksichtigt werden, obwohl diese sicherlich die wichtigste Eigenschaft einer Konstellation von Punkten darstellt. Um diese Information mit bei der Suche zu berücksichtigen, werden bei dem Verfahren von Shazam aus Paaren von Punkten sog. *Hashwerte* berechnet, in die auch die Lage eines Punktepaares zueinander eingehen.

5.1.1.1.1 Berechnung der Hashwerte Eine Konstellation Q besteht aus endlich vielen Punkten $q_i = (f_i, t_i)$ aus $\mathcal{U} = [0 : f_{\text{max}}] \times \mathbb{Z}_{\geq 0}$. Da nicht bekannt ist, welche dieser Punkte bei einer späteren Maximaextraktion erhalten bleiben, z.B. nachdem das Signal verlustbehaftet komprimiert worden ist, müssten für jeden Punkt q_i die Lage bezüglich aller anderen Punkte in der Konstellation berechnet und abgespeichert werden, was einen quadratischen Speicheraufwand erfordert. Für $|Q| = 100$ würde dies also bedeuten, dass einige zehntausend Hashwerte berechnet und gespeichert werden müssten. Um diesen Aufwand drastisch zu reduzieren und das Verfahren dadurch erst praktikabel zu machen, führen die Autoren zu einem Element $a = (f_a, t_a)$ Q, dem *Ankerpunkt*, und zu festen positiven reellen Parametern δ, ϵ den sog. *Zielbereich* $Z_{a\,Q}$ ein, der aus allen Elementen aus Q besteht, die im Intervall $[f_a - \delta\ \ 2, f_a + \delta\ \ 2] \times (t_a, t_a + \epsilon]$ liegen:

$$Z_{a\,Q} := Q \setminus [f_a - \delta\ \ 2, f_a + \delta\ \ 2] \times (t_a, t_a + \epsilon].$$

Abbildung 5.1, rechts, zeigt einen solchen Zielbereich. Man beachte, dass in der zweiten Komponente nur das halboffene Intervall $(t_a, t_a + \epsilon]$ benutzt wird, was eine doppelte Indexierung des gleichen Punktepaares vermeidet.
Die Einschränkung auf einen Zielbereich führt dazu, dass zu jedem Punkt q Q eine viel geringere Anzahl von Punkten betrachtet wird als bei der Variante, bei der die Relationen eines Punktes zu allen anderen Punkten betracht werden. Außerdem muss nicht jeder Punkt in Q als Ankerpunkt aufgefasst werden, was eine weitere Möglichkeit darstellt, die Anzahl der zu speichernden Hashwerte zu reduzieren. Ist z.B. bekannt, dass bei einer Applikation ein gewisser Frequenzbereich nicht von Degenerationen betroffen ist, würde es Sinn machen, nur diese Punkte als Ankerpunkte zu betrachten. Leider machen die Autoren hierzu keine weiteren Angaben, so dass im folgenden davon ausgegangen wird, dass jeder Punkt in Q als Ankerpunkt betrachtet wird.
Sei a Q ein Ankerpunkt. Zu jedem Element $q = (f_q, t_q)$ aus dem Zielbereich $Z_{a\,Q}$ wird nun ein dreikomponentiger *Hashwert* $h(a, q)$ wie folgt gebildet:

$$h(a, q) := (f_a, f_q, t_q - t_a) \quad [0 : f_{\text{max}}] \times [0 : f_{\text{max}}] \times \mathbb{Z}_{\geq 0} =: \mathcal{U}_H.$$

Mit $\mathcal{U}_H$ wird also die Menge der möglichen Hashwerte bezeichnet. Die beiden Frequenzkomponenten f_a und f_q sowie der zeitliche Abstand $t_q - t_a$ werden zu einem 32-Bit Hashwert zusammengefasst.

Zusätzlich zum eigentlichen Hashwert werden in weiteren 32 Bits die absolute Zeitposition des Ankerpunktes t_a und eine Signal-ID gespeichert. Damit wird ein kompletter Hashwert durch 64 Bit codiert. Im folgenden werden wir unter Hashwert dreierlei verstehen: erstens das Tripel $(f_a, f_q, t_q - t_a)$, zweitens die 32-Bit-Codierung dieses Tripels und drittens die gerade angesprochene 64-Bit-Codierung. Da aus dem Kontext stets hervorgeht, welche Art von Hashwert gemeint ist, werden wir auf die jeweilige Art nicht mehr gesondert hinweisen. Die Hashwertbildung findet immer auf der Basis eines festen Parametersatzes P (bestehend aus Fensterbreite, Schrittweite, Schwellwerten, $f_{\max}$, δ, ϵ usw.) statt. Genauer sprechen wir daher auch von P-Hashwerten. Zu konkretem P wird ein Operator $\mathcal{F}_P$ definiert, der jedes Signal $x \quad ^2(\mathbb{Z})$ auf eine Folge $\mathcal{F}_P[x]$ von P-Hashwerten aus $\mathcal{U}_H$ abbildet. Einen solchen Operator bezeichnen wir als *Merkmalsextraktor*, da dieser aus einem Signal wesentliche Eigenschaften (sog. Merkmale oder engl. „Features“) extrahiert.
Als Grund für den Übergang zu Hashwerten aus drei Komponenten von zwei Punkten geben die Entwickler an, dass sich der Informationsgehalt eines Hashwerte in etwa verdopple, indem man die Relation zwischen zwei Punkten in der Ebene als Grundlage für den Hashwert nimmt.

5.1.1.1.2 Identi kationswahrscheinlichkeit Ausgehend von einem Audiosignal $x \quad ^2(\mathbb{Z})$ und einer degenerierten Version x', die aus x z.B. durch verlustbehaftete Kompression (MP3, GSM,...) gewonnen worden ist, wird eine Wahrscheinlichkeit dafür ermittelt, dass ein im Spektrogramm von einem Signal x gefundenes Maximum ebenfalls im Spektrogramm von einer degenerierten Version x' vorhanden ist. Diese sog. *Überlebenswahrscheinlichkeit* wird mit p bezeichnet. In der Praxis hängt diese Wahrscheinlichkeit von der Position des Maximums im Spektrogramm ab. Die Tiefpassfilterung eines Audiosignals vor der MP3-Kompression ist hier ein gutes Beispiel. Im Originalsignal x können lokale Maxima im Spektrogramm durchaus oberhalb von 11 kHz auftreten. Aufgrund der Tiefpassfilterung während der MP3-Kompression enthält x' diese Frequenzkomponenten nicht mehr. Somit ist $p = 0$ für alle lokalen Maxima im Spektrogramm von x, die sich im Bereich über 11 kHz befinden. Die Autoren nehmen jedoch vereinfachend an, dass p unabhängig von der Frequenzkomponente eines Punktes aus $\mathcal{U}$ ist.
Da zur Berechnung eines Hashwertes aus $\mathcal{U}_H$ zwei Punkte verwendet werden, ist die Überlebenswahrscheinlichkeit eines Hashwertes aus $\mathcal{U}_H$ gleich $p_H = p^2$, denn beide Maxima müssen in beiden Spektrogrammen an der gleichen relativen Position vorliegen. Somit ergibt sich auf den ersten Blick eine deutliche Reduktion der Überlebenswahrscheinlichkeit p_H gegenüber der entsprechenden Wahrscheinlichkeit für nur einen Punkt p. Vereinfachend nehmen Shazam et al. an, dass die Überlebenswahrscheinlichkeiten unabhängig sind.
Zieht man jedoch die weit größere Anzahl von Hashwerten im Vergleich zur Anzahl der einzelnen Punkte in einer Konstellation mit in die Überlegung ein, kann dies unter gewissen Randbedingungen die Reduktion der Überlebenswahrscheinlichkeit aufwiegen. Dazu bezeichnen die Autoren mit F die mittlere Anzahl von Punkten im Zielbereich über alle betrachteten Ankerpunkte, der sog. *fan-out*. Die Wahrscheinlichkeit dafür, dass mindestens ein übereinstimmender Hashwert zu einem Ankerpunkt aus x und aus x' ermittelt werden kann, kann geschrieben werden als $p[1 - (1 - p)^F]$. Es gilt für $F > 10$ und $p > 0.1$

$$p \quad \approx \quad p[1 - (1 - p)^F].$$

Durch die Verwendung von Punktepaaren zur Berechnung von Hashwerten („combinatorial hashing“) erreicht man bei in etwa gleicher Identifikationswahrscheinlichkeit und einem 10-mal größeren Speicheraufwand einen Geschwindigkeitsgewinn von einem Faktor von $2^{20} \quad F^2$. Letzterer ergibt sich aus der Überlegung, dass durch die 20 zusätzlichen Bits an Information pro Hashwert gegenüber dem

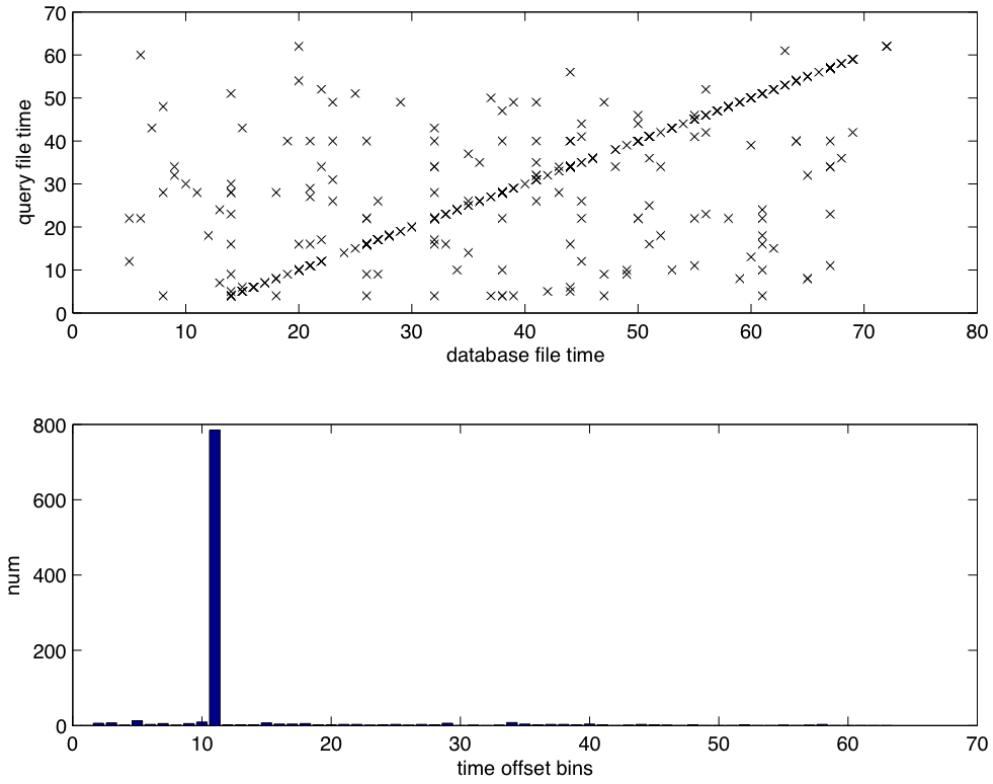


Abbildung 5.2: Visualisierung eines Anfrageergebnisses

Informationsgehalt nur eines Punktes eine Beschleunigung um den Faktor 10^6 erreicht wird. Da jedoch in der Datenbank F-mal so viele Hashwerte abgelegt sind und die Anfrage aus F-mal so vielen Hashwerten besteht, reduziert sich der Beschleunigungsfaktor um F^2. Die Wahl der Punktdichte und die Wahl der Größe des Zielbereichs sind anwendungsabhängig und haben einen direkten Einfluss auf die Performanz des Gesamtsystems.

5.1.1.2 Die Shazam-Suche

Um einen Signalausschnitt x in einer Datenbank zu finden, müssen für x zunächst die Hashwerte $Q = \mathcal{F}_P[x]$ berechnet werden. Für jeden Hashwert h Q wird in der zuvor aufgebauten Hashtabelle nachgeschlagen, an welchen Positionen und in welchen Musikstücken dieser Hashwert h berechnet wurde. In der Abbildung 5.2 (oben) sind für jeden zwischen der Anfrage und einem Musikstück aus der Datenbank übereinstimmenden Hashwert die jeweiligen Zeitpositionen der Hashwerte gegeneinander aufgetragen. Die in der Abbildung deutlich erkennbare Diagonale zeigt an, dass die Anfrage in der Datenbank enthalten ist, denn in diesem Fall treten übereinstimmende Hashwerte an den gleichen relativen Zeitpositionen in beiden Signalen auf. Allerdings ist die Suche einer Diagonalen recht aufwändig, weshalb von der folgenden Beobachtung Gebrauch gemacht wird.

Für die Zeitpositionen t_k und t'_k zweier übereinstimmender Hashwerte gilt $t'_k = t_k + g$, für ein g $\mathbb{Z}$. D.h. die Zeitposition t_k eines Hashwertes der Anfrage Q wird um eine Anzahl $g \geq 0$ von Zeiteinhei-

ten an die Stelle t'_k verschoben, an der der jeweilige Hashwert in der Datenbank vorkommt[1]. Um für eine Anfrage g zu bestimmen, wird ein Histogramm erstellt, in dem die Häufigkeiten möglicher Verschiebungen $\delta t_k = t'_k - t_k$ gezählt werden. Die Abbildung 5.2 (unten) zeigt ein solches Histogramm. An der Stelle $g = 11$ zeigt das Histogramm einen signifikant größeren Wert. Dies lässt den Schluss zu, dass die Anfrage in dem Musikstück um 11 Zeiteinheiten verschoben enthalten ist. Stammt die Anfrage nicht aus dem betrachteten Musikstück, so gibt es nach Angaben der Autoren diesen signifikanten Peak nicht. Für jede Dokumenten-ID, die bei der Suche nach Anfrage-Hashwerten gefunden wird, wird ein solches Histogramm berechnet. Nachdem alle Anfrage-Hashwerte bearbeitet worden sind, muss entschieden werden, welches Histogramm den signifikantesten Peak aufweist.
Durch die Durchführung von Anfragen mit Signalfragmenten, welche in der Datenbank enthalten sind und solchen, wo dies nicht der Fall ist, kann statistisch ein Schwellwert bestimmt werden, so dass eine gewünschte „False-Positive-Rate“ erreicht werden kann. Diese Wahl des Schwellwertes ist stark von der jeweiligen Applikation bzw. der jeweiligen Signalqualität, die der Berechnung der Konstellation Q zugrunde liegt, abhängig.
Nach Angaben der Autoren, unterstrichen durch deren kommerzielles Engagement, ermöglicht das vorgestellte System die Identifikation von Signalen, die in lauter Umgebung per Mobiltelefon aufgenommen wurden. Als Beispiel wird genannt, dass ein 15 Sekunden langes Musikstück, welches starke Störungen aufweist, zu einem signifikanten Peak in einem der Histogramme führt, wenn lediglich 1-2% der berechneten Hashwerte übereinstimmen. Für ausführlichere Angaben sei auf [Wan03, WS00, WR01] verwiesen.

5.1.1.3 Geometric Hashing (GH)

Der von Wang et al. gewählte Ansatz ist eng verwandt mit dem GH (siehe z.B. [WR97]). Dort geht es ebenfalls um die Suche nach Punktkonstellationen in einer Datenbank von Punktkonstellationen. Dabei kann die Anfrage auch nur eine Teilmenge von Punkten einer Konstellation sein. Jedoch wird beim GH ebenfalls berücksichtigt, dass die angefragte Konstellation von Punkten gegenüber einer Konstellation in der Datenbank gedreht, skaliert und beliebig verschoben sein darf. Somit stellt das Indexierungsverfahren von Shazam eine Einschränkung von GH auf die ganzzahlige Verschiebung der Punktmenge parallel zur x-Achse des Koordinatensystems dar. Aus diesem Grund wird an dieser Stelle kurz der GH-Ansatz vorgestellt, um auf die Gemeinsamkeiten hinzuweisen.
GH kann durch eine Problemstellung aus der Robotik bzw. „Computer Vision“ motiviert werden, bei der aus einem Kamerabild von unterschiedlichen Objekten Eckpunkte extrahiert werden, anhand derer das Objekt erkannt werden soll. Um nun z.B. mit einem Roboterarm nach diesem Objekt zu greifen, ist die Lage des Objekts von entscheidender Bedeutung. Folglich müssen skalierte, translatierte und rotierte Versionen einer Punktkonstellation in einer Objektdatenbank gefunden werden können. Je nach Lage der Objekte ist die aus dem Kamerabild extrahierte Punktkonstellation lediglich verschoben gegenüber dem Prototypen in der Datenbank, meist sind die Objekte jedoch zusätzlich gedreht. Außerdem kann es bei bestimmten Drehungen durch Verdeckungen im Bild zur Auslöschung einzelner Punkte einer Konstellation kommen.
Die Grundidee des GH ist die, dass jeder Punkt in der Ebene eindeutig durch zwei Koordinaten beschrieben werden kann, weshalb sich die Koordinaten, ggf. quantisiert, als Schlüssel für eine Hashtabelle eignen, sofern das zugrundeliegende Koordinatensystem bekannt ist. Um dieses zu erreichen, werden die Konstellationspunkte bezüglich mehrerer Koordinatensysteme indexiert. Dies stellt auch

[1] Anmerkung: Die Verschiebung g entspricht dem Gruppenelement eines Treffertupels (g, i), wie in Kapitel 2 beschrieben.

gleich den größten Nachteil dieses Ansatzes dar, da die Wahl einer Vielzahl von Koordinatensystemen die Anzahl der zu speichernden Hashwerte drastisch erhöht.
Sei nun $D \subset \mathbb{R} \times \mathbb{R}$ eine endliche Menge von Punkten in der Ebene.

1. Zur Wahl eines neuen, potentiellen Koordinatensystems werden zwei Punkte $p_1, p_2 \quad D$ gewählt. Die gesamte Konstellation wird so skaliert, dass die Länge der Strecke von p_1 nach p_2 gleich Eins ist. Die neue Menge nennen wir wieder D.

2. Durch Translation von D um den Vektor $-(p_1 + p_2)\ \ 2$ wird die Mitte der Strecke von p_1 nach p_2 zum Ursprung des neuen Koordinatensystems.

3. Die aktuelle Punktkonstellation wird so gedreht, dass die Gerade $\mathbb{R}(p_2 - p_1)$ zur neuen x-Achse wird. Die neue y-Achse steht senkrecht auf der neuen x-Achse. Die resultierende Konstellation werde mit D' bezeichnet. Insgesamt erhalten wir so eine Bijektion $p \quad p'$ zwischen der ursprünglichen Punktkonstellation D und dem neuen D'.

4. Jeder Punkt $p \quad D$, dessen zugehöriger Bildpunkt $p' \quad D'$ bzgl. des neuen Koordinatensystems die Koordinaten (p'_x, p'_y) hat, wird als Schlüssel in einer Hashtabelle benutzt. Die zu diesem Schlüssel gehörige Liste wird um den Eintrag [Dokumenten-ID, p_1, p_2] erweitert.

Diese Schritte werden für alle möglichen Paare von Punkten wiederholt, da bei einer Anfrage z.B. aufgrund der Auslöschung von Punkten nicht ein Punktepaar fest als Bezugspaar p_1, p_2 gewählt werden kann. Zusätzlich kann im vierten Schritt noch eine Quantisierung der Koordinaten stattfinden, um so dem Verfahren Robustheit gegenüber Rauschen hinzuzufügen. Insgesamt gibt es $\binom{|D|}{2}$ viele unterschiedliche Punktepaare, die als Basis betrachtet werden müssen! Eine graphische Veranschaulichung findet sich in [WR97].
Bei der Suche wählt man ebenfalls ein Punktepaar aus Q zur Gewinnung einer Basis eines Koordinatensystems, berechnet die Koordinaten der anderen Punkte bezüglich dieses Koordinatensystems und führt ggf. eine Quantisierung durch. Die so gewonnenen Hashwerte dienen als Schlüssel für die Hashtabelle. Alle Einträge in der Hashtabelle zu einem Hashwert bekommen während der Suche eine Stimme. Dies wiederholt man für jede der möglichen Basen in Q. Die Konstellation in der Datenbank mit den meisten Stimmen wird mit großer Wahrscheinlichkeit die gesuchte Konstellation sein. Die Autoren in [WR97] schlagen jedoch das GH nur als Vorverarbeitungsschritt vor, dessen Aufgabe es ist, die Kandidaten in der Datenbank zu identifizieren, bei denen eine genauere Betrachtung sinnvoll ist. Die oben beschriebene Vorgehensweise ist invariant gegenüber Rotation, Skalierung und Translation, was direkt aus der Wahl der Koordinatensysteme folgt.
Der von Wang et al. beschriebene Hashing-Algorithmus weist lediglich die für die Applikation nötige Invarianz gegenüber Translationen auf. Übertragen auf das GH bedeutet dies, dass nur ein Punkt aus der zu indexierenden Konstellation gewählt werden muss. Der Koordinatenursprung wird auf jeden Punkt gelegt und die Positionen der anderen Punkte bezüglich dieses Ankerpunktes als Koordinatenursprung, bestimmt und ggf. quantisiert als Hashtabellenindex verwendet. Aufgrund der reduzierten Anzahl von Freiheitsgraden genügt auch das Ergebnis des „Voting-Algorithmus“, um ein Audiosignal zu identifizieren. Auf einen Verifikationsschritt wird bei Wang et al. verzichtet. Aus der Anwendung der Technologie bei der Audioidentifikation folgt die Einschränkung der zu indexierenden Punkte auf den Zielbereich und die Hinzunahme des Abstandes der jeweiligen Punkte auf der x-Achse.

5.1.2 AudioHashing der Firma Philips

Ein Forschergruppe der Firma Philips hat sich ebenfalls der Suche in großen Audiodatenbanken gewidmet. Der in [HKO01, HK02, OKH02] vorgestellte Ansatz ist im wesentlichen durch die Daten-

struktur motiviert. Kalker et al. verwenden eine Hashtabelle als Indexdatenstruktur. Die verwendeten Merkmale bestehen aus 32-Bit Vektoren, die ein sehr grobes Abbild eines sehr kurzen Ausschnitts eines Audiosignals darstellen. Durch den Vergleich vieler solcher Vektoren sollen die Stellen in Musikstücken in der Datenbank gefunden werden, welche eine möglichst hohe Übereinstimmung mit einer Position in der Datenbank aufweisen. Aufgrund der sehr groben Darstellung als Bitvektor ist dieses Verfahren als relativ robust gegen gängige Signaldegenerationen zu bezeichnen. Allerdings ist der Hauptnachteil der große Speicheraufwand, da auf die Information, d.h. die Bitvektoren, auf zweifache Art zugegriffen werden muss: Zusätzlich zur Hashtabelle müssen die Bitvektoren auch als Sequenz vorliegen, um Teilsequenzen von aufeinanderfolgenden Bitvektoren mit einer Anfragesequenz vergleichen zu können. Prinzipiell verdoppelt sich so der schon ohnehin recht große Speicherplatzbedarf.
Dennoch wird dieses Verfahren kommerziell eingesetzt. So bietet die Firma *Gracenote* [webb], bekannt durch die CDDB zur automatischen Abfrage von Titelnamen einer Musik-CD, einen Audioidentifikationsdienst an, um qualitativ hochwertige Metadaten dem Kunden zur Verfügung zu stellen. Zusammen mit einem der führenden Hersteller von Mobiltelefonen wird dieser Dienst auch über das Mobilfunknetz angeboten.

5.1.2.1 Die Merkmalsextraktion

In einem ersten Schritt wird ein Audiosignal $x \quad ^2(\mathbb{Z})$ in sich überlappende „Frames“ von je 0,37 Sekunden Länge unterteilt. Zusammen mit der Überlappung aufeinanderfolgender Frames von 31/32 der Framelänge und der Fensterung mit einem Hanning-Fenster wird alle 11.6 ms ein 32-Bitvektor als Merkmal berechnet. Die Autoren bezeichnen einen solchen Bitvektor als *Sub-Fingerprint* s_i $\{0,1\}^{32}$. Eine Folge von 256 aufeinanderfolgenden Sub-Fingerprints wird als *Fingerprintblock* bezeichnet. Die große Überlappung zweier aufeinanderfolgender Frames hat zur Folge, dass im schlimmsten Fall ein Anfragesignal gegenüber dem korrespondierenden Signal in der Datenbank um 5,8 ms verschoben ist. Daher ist in der Praxis davon auszugehen, dass die Bitvektoren sich stark ähneln, obwohl die Signale zeitlich um maximal 5,8 ms gegeneinander verschoben sind.
Für jeden Frame, der die Basis eines 32-Bitvektors (*Sub-Fingerprints*), darstellt, wird eine Fourier-Transformation berechnet. Da die menschliche Hörwahrnehmung unsensitiv gegenüber der Phase ist, genügt es hier den Absolutbetrag des Spektrums, die sog. *power spectrum density*, zu betrachten. Für die Berechnung des Sub-Fingerprint Bitvektors wird das Spektrum eines Frames in *Frequenzbänder* gemäß der Bark-Skala (siehe [ZF90]) eingeteilt. Diese Skala unterteilt das Spektrum in Bänder mit logarithmisch wachsender Breite, die den sog. *kritischen Bändern* des menschlichen Gehörs annähernd entsprechen. Grob gesprochen erzeugen mehrere Töne, deren Frequenzen in demselben kritischen Band liegen, im Gehirn nur ein Hörerlebnis. D.h. das Gehör unterscheidet diese Töne nur begrenzt. Aus diesem Grunde wird diese Skala in vielen Anwendungen im Bereich Musik-IR verwendet. Kalker et al. unterteilen den Frequenzbereich zwischen 300 und 2000 Hz in 33 nicht überlappende Bänder, da dieser Bereich in der Hörwahrnehmung beim Menschen am wichtigsten ist.
Bezeichne $E^m(n)$ die Energie des m-ten Bandes des n-ten Frames. Das m-te Bit des n-ten Sub-Fingerprints berechnet sich als

$$F^m(n) := \begin{cases} 1 & \text{falls} \quad E^m(n) - E^{m+1}(n) - [(E^m(n-1) - E^{m+1}(n-1)] > 0, \\ 0 & \text{falls} \quad E^m(n) - E^{m+1}(n) - [(E^m(n-1) - E^{m+1}(n-1)] \leq 0. \end{cases} \tag{5.1}$$

Ein Sub-Fingerprint hat genau dann an Bitposition m den Wert 1, wenn die Energiedifferenz des m-ten und des $m+1$-ten Bandes größer ist als im Frame zuvor. Die Berechnung der Subfingerprints ist nach

Angabe der Autoren auch auf mobilen Geräten möglich, da nur Frequenzen ≤ 2000 Hz betrachtet werde und somit Abtastraten von 4000 Hz ausreichend sind. Die Berechnung der Fouriertransformation für jeden Sub-Fingerprint kann durch die Verwendung von Fixpunktarithmetik ebenfalls ausreichend schnell erfolgen.

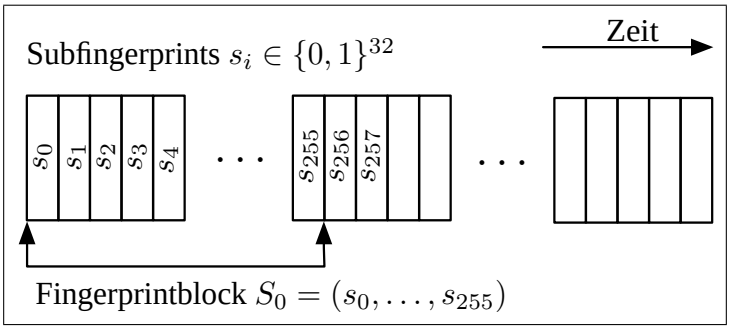


Abbildung 5.3: Verdeutlichung der Unterteilung eines Audiosignals in Sub-Fingerprints s_i $\{0,1\}^{32}$. Eine Folge von 256 aufeinanderfolgenden Sub-Fingerprints wird als Fingerprintblock $S_j = (s_j, \ldots, s_{j+255})$ bezeichnet.

5.1.2.2 Suche in der Datenbank

Die berechneten Sub-Fingerprints dienen als Index für den Zugriff auf eine Hashtabelle, wo zu jedem Hashwert eine Liste von Zeigern verwaltet wird, die auf die Stellen in der Sequenz von Sub-Fingerprints zu einem Musikstück verweisen, an denen dieser Sub-Fingerprint extrahiert worden ist. Über diese Hashtabelle kann also schnell bestimmt werden, in welchen Musikstücken und an welchen Positionen ein extrahierter Sub-Fingerprint vorkommt.
Sei $S_i = (s_i, \ldots, s_{i+N-1})$ ein sog. *Fingerprintblock*, d.h. eine Folge von aufeinanderfolgende Sub-Fingerprints, die aus einem Anfragesignal extrahiert worden sind. Die Autoren geben den Wert $N = 256$ in ihren Arbeiten an. Um nun zu entscheiden, ob das Anfragesignal in einem Stück, welches in der Datenbank gespeichert ist, enthalten ist, verwenden die Autoren die *Bit Error Rate, (BER)*, als Ähnlichkeitsmaß. Da dabei der Vergleich der lediglich 32 Bit eines Sub-Fingerprints nicht aussagekräftig genug wäre, wird daher die BER für zwei Fingerprintblocks berechnet.

De nition 5.1 (Bitfehlerrate, BER) *Die Bitfehlerrate von A, B $\{0,1\}^N$ de niert eine Abbildung*

$$\textit{BER} : \{0,1\}^N \times \{0,1\}^N \qquad [0,1] \subset \mathbb{R}$$

mit

$$\textit{BER}(A,B) \;:=\; \frac{1}{N} \sum_{i=0}^{N-1} |a_i - b_i|.$$

Beginnend mit dem ersten Sub-Fingerprint s_0 eines Fingerprintblocks, werden per Zugriff auf die Hashtabelle die Positionen in Musikstücken in der Datenbank bestimmt, an welchen s_0 vorkommt. Für jede dieser Kandidatenpositionen wird nun die BER zwischen dem Anfrage- und der Datenbank-Fingerprintblock berechnet. Da der erste Sub-Fingerprint ja bereits Bitfehler aufgrund von Signaldegeneration enthalten kann, ist nicht sicher, ob s_0 überhaupt zu dem richtigen Musikstück in der

Datenbank geführt hat. Aus diesem Grund wird die Suche für jeden Sub-Fingerprint s_i eines Anfrage-Fingerprintblocks durchgeführt. Laut Aussage der Autoren ist die Wahrscheinlichkeit bei moderater Signaldegeneration sehr hoch, dass mindestens ein Sub-Fingerprint eines Fingerprintblocks keine Bitfehler aufweist und damit jedes Musikstück gefunden werden kann. Ein Treffer liegt dann vor, wenn eine Position mit einer BER von weniger als 0,35 (siehe [HK02]) gefunden worden ist. Wird keine solche Position gefunden, wird davon ausgegangen, dass das Fragment nicht in der Datenbank vorhanden ist.
Ausgehend von einer Datenbankgröße von 250 Millionen Sub-Fingerprints und der Annahme, dass alle Sub-Fingerprints mit der gleichen Wahrscheinlichkeit vorkommen, ergibt sich eine mittlere Anzahl von Einträgen in einer Liste zu einem Hashwert von $250 \cdot 10^6 \;\; 2^{32} = 0,0582$. Somit müssen im Mittel pro Vergleich zweier Fingerprintblocks lediglich $256 \cdot 0,0582 \approx 15$ Sub-Fingerprints verglichen werden. Allerdings weisen die Autoren darauf hin, dass in der Praxis keine Gleichverteilung über alle möglichen Sub-Fingerprints beobachtet werden kann, weshalb sich die Anzahl der Sub-Fingerprint Vergleiche pro Identifikation eines Fingerprintblocks auf 300 erhöht. Jedoch auch der Vergleich von 300 Bit-Vektoren der Länge 32 ist auf einem modernen Rechner mit einem Zeitaufwand im Bereich einer Millisekunde möglich.
Bei starker Signaldegeneration, wie sie zum Beispiel bei der Mobiltelefon-Applikation auftritt, ist die zur Identifikation notwendige Bedingung, dass mindestens ein Sub-Fingerprint pro Fingerprintblock ohne Fehler aus dem Anfragesignal extrahiert werden konnte, in der Regel nicht erfüllt. Um die Robustheit des Verfahrens zu erhöhen, schlagen Kalker et al. daher vor, nicht nur nach Vorkommen des Sub-Fingerprints zu suchen, sondern auch die Sub-Fingerprints zu beachten, die einen Hamming-Abstand von 1 zum angefragten Sub-Fingerprint haben. Die führt zu einer Erhöhung der Anzahl der Sub-Fingerprint Vergleiche um den Faktor 33 (original 32 Bit-Vektor und 32 Vektoren mit Hamming-Abstand 1) , was von den Autoren als immer noch akzeptabel in Hinblick auf die Laufzeit der Suche beschrieben wird. Um jedoch die für die Mobiltelefon-Applikation notwendige Robustheit zu erreichen, müssten alle Sub-Fingerprints mit Hamming-Abstand 3 zum angefragten Sub-Fingerprint ebenfalls betrachtet werden, was jedoch zu inakzeptablen Suchzeiten führt. Um die Suchzeiten für diesen Fall zu reduzieren, gehen die Autoren davon aus, dass nicht jedes Bit eines Sub-Fingerprints mit der gleichen Wahrscheinlichkeit falsch extrahiert wird. Bei der Berechnung von (5.1) lässt sich die *Güte* eines Bits durch den Abstand vom Schwellwert in (5.1) beschreiben. Ist der Abstand groß, ist auch die Wahrscheinlichkeit groß, dass dieses Bit korrekt extrahiert worden ist. Auf diese Weise kann man eine Rangliste erstellen, und darauf basierend bei jedem Sub-Fingerprint die $k < 32$ unzuverläßigsten Bits verändern, anstelle alle Sub-Fingerprints mit Hamming-Abstand k betrachten zu müssen. Für die Mobiltelefon-Applikation schlagen die Autoren $k = 10$ vor, d.h. zu jedem Sub-Fingerprint werden 1024 Sub-Fingerprints berechnet, die zu Kandidatenpositionen führen. Folglich muss pro Fingerprintblock an $256 \cdot 1024 = 262.144$ vielen Kandidatenpositionen die BER berechnet werden. In der Praxis, so geben die Autoren an, führt dies zu einer deutlichen Verbesserung des Systems bei immer noch akzeptablen Laufzeitverhalten.

5.1.2.3 „False-Positive“ Analyse

Gegeben seien zwei Fingerprintblöcke $G, H \qquad \{0,1\}^{32\times 256}$. Als Maß für die Ähnlichkeit zweier solcher Blöcke nutzen die Autoren die *Bitfehlerrate*, d.h. die Anzahl der Bits, in denen sich G und H unterscheiden, normiert mit der Gesamtanzahl N der betrachteten Bits, hier $N = 32 \cdot 256 = 8192$. Zwei Fingerprintblöcke heißen gleich, wenn

$$\text{BER}(G,H) < T \quad [0;1] \subset \mathbb{R}, \quad \text{geschrieben als} \quad A \overset{\text{BER}}{=}_T B.$$

gilt. Dies bedeutet, dass die beiden Fingerprintblöcke dem gleichen Signal zuzuordnen sind. T ist ein applikationsabhängig gewählter Schwellwert. Wird T zu groß gewählt, wächst die Wahrscheinlichkeit dafür, dass zwei Fingerprintblöcke als gleich gelten, obwohl diese nicht aus dem gleichen Signal berechnet worden sind. In diesem Fall liefert das System „False Positive"-Treffer. Wählt man T zu klein, erhöht sich die Wahrscheinlichkeit dafür, dass zwei Fingerprintblöcke als ungleich erkannt werden, obwohl sie aus den inhaltlich gleichen Signalen berechnet worden sind. „False Negative"-Ergebnisse sind die Folge.
Um einen Wert für T festlegen zu können, interessiert die Frage, wie groß die Wahrscheinlichkeit ist, dass $G \overset{\text{BER}}{=}_T H$ gilt, obwohl die Signale, aus denen G und H extrahiert worden sind, nicht gleich sind. Zunächst gehen wir davon aus, dass alle Bits eines Fingerprintblocks zufällig und voneinander statistisch unabhängig gewählt sind. Demnach kann ein Fingerprintblock als Folge von Zufallsvariablen $(G_i)_{i\in I}, I = [0 : N-1]$ mit folgenden Eigenschaften modelliert werden:

$$\begin{array}{lll} i \quad I : G_i : \{0,1\} & & \{-1,1\}, \text{ mit} \\ \quad G_i(0) := -1 & \text{und} & G_i(1) := 1. \end{array}$$

Zusätzlich nehmen wir an, dass jede Zufallsvariable gleichverteilt ist, d.h. $i \quad I : P(G_i = -1) = P(G_i = 1) = \frac{1}{2}$. Betrachten wir nun die Zufallsvariablen $C_i := G_i \cdot H_i$ mit $C_i : \{-1,1\}$ $\{0,1\}, i \quad I$. Es gilt:

$$P(C_i = -1) = P(C_i = 1) \quad = \quad \frac{1}{2}.$$

Der Erwartungswert der Zufallsvariablen C_i ergibt sich zu

$$i \quad I : \mathrm{E}[C_i] = (-1) \cdot \frac{1}{2} + 1 \cdot \frac{1}{2} \quad = \quad 0. \tag{5.2}$$

Für die Varianz der Zufallsvariablen C_i gilt im unnormalisierten Fall

$$\begin{array}{rcl} i \quad I : \mathrm{Var}[C_i] & = & \mathrm{E}[(C_i - \mathrm{E}[C_i])^2] \\ & = & \mathrm{E}[C_i^2] \\ & = & 1. \end{array}$$

Im Falle einer Normalisierung mit N gilt

$$i \quad I : \mathrm{Var}[C_i \ N] \quad = \quad \frac{1}{N^2}\mathrm{Var}[C_i] = \frac{1}{N^2}. \tag{5.3}$$

Von Interesse ist nun die Zufallsvariable

$$\begin{array}{rcl} S & := & \sum_{i=0}^{N-1} \frac{C_i}{N}, \\ S : \{0,1\}^N & & [-1;1] \subset \mathbb{R} \quad \text{mit} \\ \mathrm{E}[S] & = & \sum_{i=0}^{N-1} \mathrm{E}[C_i \ N] \overset{(5.2)}{=} 0, \\ \mathrm{Var}[S] & = & \sum_{i=0}^{N-1} \mathrm{Var}[C_i \ N] \overset{(5.3)}{=} N \cdot 1 \ N^2 = 1 \ N. \end{array}$$

die die übereinstimmenden und nicht übereinstimmenden „Bits“ aufsummiert. Im Falle von vielen unterschiedlichen Bits liegt der Wert von S nahe bei -1 und umgekehrt. S unterliegt einer Binomialverteilung, d.h. die Wahrscheinlichkeit, dass sich G und H an k Stellen unterscheiden ist

$$P(S = (N-k)\ \ N) \quad = \quad \binom{N}{k} \cdot P(G_i = H_i)^k \cdot (1 - P(G_i = H_i))^{(N-k)}.$$

Da S eine Summe von unabhängigen Zufallsvariablen ist, die alle die gleiche Varianz und Erwartungswert aufweisen, kann die Binomialverteilung von S für große Werte N nach dem *lokalen Grenzwertsatz von de Moivre-Laplace* durch die Normalverteilung Φ mit dem Mittelwert μ und der Varianz σ^2 approximiert werden. Da von einer Gleichverteilung der Bits ausgegangen wird, gilt $p = 0.5$. Wenn $F(x)$ die Verteilungsfunktion der Zufallsvariablen S ist, dann bedeutet dies

$$\lim_{N\to\infty} F(x) = \Phi(x, 0, 1\ \ N) = \frac{N}{\overline{2\pi}} \int_{-\infty}^{x} e^{-\frac{u^2}{2/N^2}} du.$$

Die BER wird ebenfalls als Zufallsvariable B in Abhängigkeit von S modelliert:

$$\begin{aligned} B : [-1;1] \qquad & [0;1] \subset \mathbb{R} \\ B \quad := \quad & \frac{1-S}{2}. \end{aligned}$$

Für B gilt

$$\begin{aligned} \mathrm{E}[B] \quad &= \quad \mathrm{E}\left[\frac{1-S}{2}\right] = \mathrm{E}\left[\frac{1}{2} - \frac{S}{2}\right] = \frac{1}{2} \cdot \underbrace{\mathrm{E}\left[-\frac{S}{2}\right]}_{=0} = \frac{1}{2}, \\ \mathrm{Var}[B] \quad &= \quad \mathrm{Var}\left[\frac{1-S}{2}\right] = \mathrm{Var}\left[-\frac{1}{2} + \frac{S}{2}\right] = \mathrm{Var}\left[\frac{S}{2}\right] = \frac{1}{2^2}\mathrm{Var}[S] = \frac{1}{4N}. \end{aligned}$$

Damit kann nun die Wahrscheinlichkeit dafür, dass die BER unter einem Wert α liegt, durch eine Normalverteilung approximiert werden:

$$P(B \le \alpha) = \Phi\left(\alpha, \frac{1}{2}, \frac{1}{2\ \ \overline{N}}\right) = \frac{2\ \ \overline{N}}{\overline{2\pi}} \int_{-\infty}^{\alpha} e^{-\frac{(u-\frac{1}{2})^2}{\frac{1}{2N}}} du.$$

Durch Integration der Verteilungsfunktion und durch geeignete Abschätzungen erhält man

$$P(B \le \alpha) \le \frac{1}{2}\mathrm{erfc}\left(\frac{(1-2\alpha)}{\overline{2}} \quad \overline{N}\right). \tag{5.4}$$

Dabei bezeichnet erfc die komplementäre Fehlerfunktion definiert als

$$\mathrm{erfc}(z) := \frac{2}{\overline{\pi}} \int_{z}^{\infty} e^{-t^2}\,\mathrm{d}t.$$

Bisher wurde davon ausgegangen, dass die Bits aufeinanderfolgender Sub-Fingerprints in einem Fingerprintblock unkorrelliert sind. Aufgrund der Tatsache, dass sich die, den Fingerprintblöcken zugrundeliegenden Audiosignale, nur langsam mit der Zeit ändern und die Frames stark überlappen,

ist diese Annahme in der Praxis unhaltbar. Vielmehr muss davon ausgegangen werden, dass die Bits zweier aufeinanderfolgender Sub-Fingerprints stark korrelliert sind. Dies hat natürlich Auswirkungen auf die zuvor gemachten Aussagen über $P(B \leq \alpha)$. Mit der folgenden Argumentation zeigen Kalker et al., dass sich durch die Korrelation aufeinanderfolgender Sub-Fingerprints lediglich die Standardabweichung gegenüber (5.4) vergrößert. In einem Versuch wurden 100.000 Fingerprintblocks zufällig aus einer Datenbank ausgewählt. Die anhand dieses Datensatzes bestimmte Standardabweichung ist um den Faktor $2,7$ größer als die von unkorrellierten Fingerprintblocks. Daher wurde als empirisch bestimmter Faktor 3 in (5.4) eingefügt, um der Korrelation von Sub-Fingerprints zu genügen:

$$P_{\text{mod}}(B \leq \alpha) = \frac{1}{2}\text{erfc}\left(\frac{(1-2\alpha)}{3\ \ \overline{2}}\ \ \overline{N}\right). \tag{5.5}$$

Aus (5.5) folgt für den von den Autoren vorgeschlagenen Wert $\alpha = 0.35$ ein Wahrscheinlichkeit für einen FP-Treffer von $3.6 \cdot 10^{-20}$. Bei einer Übereinstimmungsquote der Bits von zwei Fingerprinblöcken von 65% dürfen höchstens 2867 Bits fehlerhaft sein, um eine Identifikation mit Fehlerwahrscheinlichkeit zu ermöglichen.

5.1.2.4 Performanz

In [HK02] haben die Autoren die Leistungsfähigkeit ihres Systems untersucht, indem sie Audiosignale unterschiedlich stark modifiziert haben. Dabei wurde die BER zwischen dem Original und den modifizierten Versionen gemessen. Lag diese unter dem Schwellwert $\alpha = 0.35$, so folgern die Autoren, sollte das System die jeweilige schlechte Qualität bei einer Suchanfrage kompensieren können. Nach Angaben der Autoren konnten Suchanfragen, die durch einen GSM-Codec komprimiert worden sind, erkannt werden. Gängige Kompressionsverfahren für Audiosignale wie MP3 und RealAudio verursachten keine Probleme. Lediglich eine lineare Veränderung der Abspielgeschwindigkeit, welche eine Änderung der Tonhöhen von $\pm 4\%$ verursacht, führte zu Störungen, die nicht durch das vorgestellte Verfahren kompensiert werden konnten.

5.1.3 AudioDNA

Besonders dem im vorherigen Abschnitt angesprochenen Problem der linearen Veränderung der Abspielgeschwindigkeit widmet sich die Gruppe um Battle et al. In [CBMN02, NMB01, BMG02] stellen sie ein Audioidentifikationssystem vor, welches aufgrund der Merkmalgewinnung speziell für die Erkennung von Audiosignalen, welche über das Radio übertragen werden, optimiert ist. Denn Radiostationen spielen Musik häufig aus zwei Gründen etwas schneller. Der sicherlich einfachste Grund dafür ist der, dass mehr Musik pro Stunde gespielt werden kann. Zum anderen erzeugt das beschleunigte Abspielen beim Hörer einen Höreindruck, worüber Radiostationen eine Bindung der Hörer an den Sender zu erreichen. Daneben werden Bässe verstärkt und das Signal so verändert, dass in typischen Situationen, in denen Radio gehört wird (z.B. im Auto), der Stereo-Effekt verstärkt wird.
Dem Problem der skalierungsinvarianten Suche mittels G-invertierter Listen widmet sich aus den gleichen Gründen die Diplomarbeit von Rolf Bardeli. Er hat in [Bar03] den in Kapitel 2 vorgestellten Ansatz erweitert, so dass auch linear skalierte Versionen eines Audiosignals identifiziert werden konnten.

5.1.3.1 Merkmalsextraktion

Die zuvor beschriebenen Verfahren haben grob gesprochen eine Frequenzanalyse des Audiosignals durchgeführt und spektrale Eigenschaften, die von gängigen Signalstörungen zu großen Teilen un-

berührt bleiben, benutzt, um daraus Merkmale, d.h. Indexe in eine Hashtabelle, zu generieren. Hier gehen Battle et al. einen anderen Weg. Im Hinblick auf die Zielsetzung, linear skalierte Versionen von Audiosignalen identifizieren zu können, benutzen sie Hidden Markov Models (HMMs, [Rab89]), um sog. *AudioGenes* aus den Audiodaten zu extrahieren. Die Analogie zu Gensequenzen wie sie aus der Biologie bekannt sind, besteht darin, dass die Suche in der Datenbank in einer ersten Stufe ähnlich dem FASTA-Algorithmus [PL88] erfolgt. Dieser Algorithmus wird in der Biologie genutzt, um in Gensequenzen nach Teilsequenzen fehlertolerant zu suchen. Die Autoren fassen ein Audiosignal oder einen Audiostream als Sequenz von Buchstaben auf, welche jeweils einem AudioGene entsprechen. Motiviert wird dieser Ansatz aus der Spracherkennung, wo es darum geht, ein Sprachsignal in eine Sequenz von Phonemen zu überführen. Auch hier kommen meist HMMs zum Einsatz. Im Gegensatz zur Biologie ist das Alphabet, welches den AudioGene-Sequenzen zugrunde liegt, hier umfangreicher. Wie bei der Benutzung von HMMs üblich, benötigt man vor der eigentlichen Verwendung der HMMs eine *Trainigsphase*, in der die Parameter der einzelnen HMMs bestimmt werden. Und zwar in einer Art, dass jedes verwendete HMM die Audiodaten zu einem AudioGene möglichst gut widerspiegelt. Auf diese Weise soll während der späteren Anwendung genau das HMM eine hohe Wahrscheinlichkeit für die eingehenden Audiodaten liefern, welches in der Trainigsphase auf diese Audiodaten „trainiert“ worden ist.
Sowohl in der Trainingsphase als auch später während der Anwendung wird ein eingehendes Audiosignal zunächst in Frames fester Länge unterteilt. Um Randeffekte zu vermindern, werden die Audiodaten in einem Frame mit einem Hamming-Fenster gewichtet. In [CBMN02] geben die Autoren an, dass die Frames einen Abstand von ca. 10ms aufweisen und durch Überlappung einem Zeitraum von 25ms entsprechen. Für jeden Frame F_i wird das Fourier-Spektrum

$$F_i = \text{DFT}[F_i]$$

berechnet. Aus den Frequenzinformationen werden die sog. *Mel-Frequency Cepstral Coef cients* (MFCC) (siehe Definition 4.3), wie sie in der Sprachverarbeitung oft benutzt werden, berechnet. In einem letzten Schritt werden die MFCCs durch eine DCT-Transformation dekorrelliert. Da zwischen den DCT-Koeffizienten zweier aufeinanderfolgender Frames in der Regel große Ähnlichkeit besteht, fügen die Autoren neben den MFCCs noch die erste und zweite diskrete Ableitung der MFCC-Folge dem sog. *akustischen Merkmalsvektor* an. Diese Vektoren dienen dann als Eingabe für die HMMs, die die AudioGenes-Sequenz berechnen. Die Autoren deuten an, dass sie noch weitere bzw. andere Signaleigenschaften an dieser Stelle benutzen.
In [BMG02] beschreiben die Autoren einen weiteren Vorverarbeitungsschritt, um die Störungen, die ein Radiosignal im Vergleich zu dem korrespondierenden Signal auf einer CD aufweisen kann, vor der Extraktion der HMM-Eingabevektoren zu reduzieren. Dazu modellieren die Autoren das Radiosignal $y(n)$ als Produkt aus dem korrespondierenden Originalsignal $x(n)$ und einem linearen Filter mit der Frequenzantwort $H(\ )$ im Frequenzbereich:

$$\begin{aligned} y(\ ) &= H(\ )x(\ ), \\ \ln|y(\ )| &= \ln|H(\ )| + \ln|x(\ )|. \end{aligned}$$

Da jedoch nur $y(n)$ vorliegt, wird $H(\ )$ als störender Kanal angenommen, der sich über die Zeit jedoch nur langsam ändert. Durch die Filterung des Radiosignals mit einem solchen Filter, erhöht sich nach Angabe der Autoren die Identifikationswahrscheinlichkeit deutlich.

5.1.3.1.1 Berechnung der AudioGenes Um aus einem Audiostück die Audiogenes zu extrahieren, wird das Signal in eine Folge Y von akustischen Merkmalsvektoren überführt. Sei $(\lambda_i)_{0\leq i<N}$ die

Folge von HMMs, die jeweils einem AudioGene entsprechen. Für jeden Merkmalsvektor $y \quad Y$ wird nun durch

$$\text{AudioGene}(y) = \arg\max_{0 \leq i < N} P(y|\lambda_i)$$

der Index des jeweiligen HMMs berechnet, welches mit größter Wahrscheinlichkeit diesen akustischen Merkmalsvektor y erklärt. Dazu verwenden die Autoren den Viterbi-Algorithmus (Algorithmus 4.2.1). Dazu wird aus den einzelnen λ_i ein HMM aufgebaut, in dem intuitiv gesprochen alle λ_i HMMs derart miteinander verbunden sind, dass die Zustände, an denen ein λ_i „endet“ mit den Startzuständen eines jeden $\lambda_j, j = i$, mit einer positiven Übergangswahrscheinlichkeit verknüpft werden. Bei jedem Wechsel von einem λ_i zu einem λ_j wird ein neues AudioGene erzeugt. Leider geben die Autoren keine weiteren Details an, die eine genauere Beschreibung ermöglichen würden. Durch die Benutzung einer Variante des Viterbi-Algorithmus wird auch eine Verarbeitung von Streaming-Audio möglich. Die Ausgabe dieses Extraktionsschrittes ist eine Sequenz von AudioGenes, die zusätzlich zu den Gensequenzen in der Biologie, noch Informationen über die jeweilige Startposition und der Dauer des jeweiligen AudioGenes enthalten. In der Testumgebung geben die Autoren eine Rate von durchschnittlich 800 AudioGenes pro Minute Audio ($\approx$ 13 AudioGenes / Sekunde) an. Wobei sie darauf hinweisen, dass die Rate und die Anzahl der möglichen AudioGenes in Bezug auf die jeweilige Applikation gewählt werden können.

5.1.3.2 AudioDNA Matching

In dem Matching-Schritt, in dem eine angefragte Sequenz von AudioGenes in der Datenbank gesucht wird, unterscheiden sich die in [CBMN02] und [BMG02] vorgestellten Systeme deutlich. In dem in der ersten Veröffentlichung beschriebenen System kommt ein approximatives Stringmatching-Verfahren zum Einsatz. Dabei wird in einem ersten Schritt die Anfragesequenz in kurze Sequenzen unterteilt und nach exakten Treffern in der Datenbank gesucht. Diese Treffer werden in einer balancierten Baumstruktur gespeichert, wobei in jedem Knoten die exakten Treffer zu einem Musikstück zusammengefasst werden. Dabei wird ein exakter Treffer nur dann zu einem Knoten hinzugefügt, wenn dessen Zeitinformation, die zu jedem AudioGene gehören, in den jeweiligen Kontext passt. Entfallen auf einen Knoten eine gewisse Anzahl von exakten Treffern, so wird unter der Benutzung der Zeitinformation ein approximatives Matching Verfahren angewendet, um die gesamte Anfragesequenz in der zugehörigen Datenbanksequenz zu lokalisieren. Als Ähnlichkeitsmaß S verwenden die Autoren das Verhältnis von N gefundenen Zeitintervallen mit übereinstimmenden AudioGenes, Δt_e^i, und der Dauer der angefragten Sequenz Δt_o:

$$S(\Delta t_o, \Delta t_e^i) := \frac{\sum_{i=0}^{N-1} \Delta t_e^i}{\Delta t_o}.$$

In [BMG02] wird ein anderer Matching-Algorithmus vorgestellt. Anstatt des zweistufigen Matchingprozesses wird hier auch im Matching-Schritt ein HMM und der Viterbi-Algorithmus benutzt. Das sog. „Matching-HMM“ wird aus allen Datenbanksequenzen aufgebaut, wobei jeder Zustand des HMM s einem Namen eines AudioGene s entspricht. Die Zustände werde als Kette nach der zeitlichen Abfolge der AudioGenes durch entsprechende Übergangswahrscheinlichkeiten verbunden. Jede dieser Zustandsketten entspricht der AudioGenes-Sequenz eines Musikstückes. Alle Zustandsketten werden zu einem Ring zusammengefasst. Die Autoren bezeichnen diese Verbindungen zwischen den HMM-Zuständen als *externe Verbindungen (external links)*. In einem applikationsabhängig gewählten Zeitabstand werden Zustandsübergänge zwischen den Zustandsketten ermöglicht, um auf diese

Weise auch akustische Übergänge zwischen zwei Musikstücken modellieren zu können. Mit Hilfe des Viterbi-Algorithmus wird für eine Anfragesequenz O die Zustandsfolge Q ermittelt, so dass $P(Q|O, \lambda)$ maximiert wird. Laut [BMG02] sollen die folgenden drei Fälle durch das HMM modelliert werden:

Identi kation eines Musikstücks In diesem Fall besteht die Zustandsfolge Q nur aus internen Zustandsübergängen, d.h. die Zustände, die das HMM mit größter Wahrscheinlichkeit durchlaufen wird, um die O zu erzeugen, gehören zu genau einem Musikstück.

Übergang zwischen zwei Musikstücken Es werden bis auf einen nur interne Zustandsübergänge in Q zu finden sein. Der eine externe Zustandsübergang entspricht der Verbindung von zwei Modellen und damit der akustischen Verbindung von zwei Musikstücken.

Unbekanntes Material Da eine Anfragesequenz auch zu Musikstücken gehören können, die nicht in der Datenbank verzeichnet sind, würde hier die Zustandsfolge aus vielen externen gemischt mit internen Übergängen bestehen. Durch das Fehlen einer klaren Struktur von Q kann darauf geschlossen werden, dass die Anfragesequenz O zu unbekanntem Material gehört.

Leider geben die Autoren in der Veröffentlichung keinerlei Details zum Training der HMMs und z.B. zur Bestimmung der Übergangswahrscheinlichkeiten bei internen Verbindungen an. Hier fließt sicherlich eine Menge von empirischen Erfahrungen ein. Daher ist eine Verifikation der Ergebnisse nur schwer möglich. In ihren Tests haben die Autoren etwa 250 Stunden Musik als Datenbasis genutzt. Bereinigt um „False Positive“-Treffer, die z.B. durch mehrfaches Vorhandensein eines Musikstücks in der Datenbank („Greatest Hits -Problem“) entstehen, zeigen die Autoren, dass das zuletzt beschriebene System eine höhere Fehlerrate am Beginn und am Ende von Musikstücken aufweist. An diesen Stellen fällt die Klassifikation mittels HMMs schwer, da z.B. aufgrund von Fading-Effekten die „Informationsmenge“ in dem Signal nur langsam zu- bzw. abnimmt.

5.1.4 AudioID

In [AHH+01] stellen Allamanche et al. einen Ansatz zur Audioindentifikation basierend auf Vektorquantisierung und Nächster-Nachbar-Suche (NN-Suche) vor. Mittels einer WFT wird das Spektrogramm $(S_i)_{0\leq i<N}$ eines Eingabesignals berechnet. Da das menschliche Gehör unempfindlich gegenüber der Phase ist, wird das *Energiespektrum* $(S_i)_{0\leq i<N}$ genutzt. Der Frequenzbereich von 300 Hz bis 6000 Hz wird auch bei diesem Ansatz in eine Folge disjunkter Frequenzbänder $(B_j)_{0\leq j<M}$ unterteilt. Die Autoren haben Tests mit $M = 4$ und $M = 16$ durchgeführt. Aus den Subbändern B_j eines Spektrums S_i wird ein Merkmalsvektor F^i durch Berechnung eines der folgenden Signalcharakteristika

$$F^i_{\text{SFM}}(j) = \text{SFM}[B(j)], \quad 0 \leq i < N, 0 \leq j < M, B_j \quad S_i \tag{5.6}$$
$$F^i_{\text{SCF}}(j) = \text{SCF}[B(j)], \quad 0 \leq i < N, 0 \leq j < M, B_j \quad S_i \tag{5.7}$$
$$F^i_{\text{LOD}}(j) = \text{LOD}[B(j)], \quad 0 \leq i < N, 0 \leq j < M, B_j \quad S_i \tag{5.8}$$

gewonnen. Je nach Anwendung kann die Wahl unterschiedlich ausfallen. In dem MPEG-7-Standard (siehe [ISO00]) ist jedoch als „Low Level Descriptor“ (LLD) nur die Funktion SFM vorgesehen.

5.1.4.1 Anfragebearbeitung und Datenbankaufbau

Jedes Audiosignal $x \quad ^2(\mathbb{Z})$ wird durch die zuvor beschriebenen Schritte in eine Folge $F = (F^i)_{0 \leq i < N}$ von Merkmalsvektoren aus dem $\mathbb{R}^M$ überführt. Da bei der Anwendung eine der Berechnungen aus (5.6) - (5.8) fest gewählt wird, schreiben wir F^i anstelle einer der zuvor erwähnten Operatoren. Durch ein *Clustering-Verfahren* wird versucht, die Menge von Punkten durch eine kleinere Anzahl von Vektoren c^j zu approximieren. Diese Vektoren bilden ein sog. *Codebuch*, welches für jedes Musikstück chrakteristisch sein soll.

5.1.4.1.1 k-Means Clustering Die Grundlage beim k-*Means Clustering* bildet eine Fehlerfunktion R, die misst, wie gut die Codebuchvektoren c^j die Datenpunkte F^i approximieren. Ziel des Verfahrens ist es, die Fehlerfunktion R zu minimieren. Die Autoren verwenden die *Root Mean Square Error*-Funktion, um den Abstand von Codebuch zu den Datenvektoren zu messen. Sei

$$C_j \quad := \quad \{F^i : |F^i - c_j| < |F^i - c_k|, \quad 0 \leq k < M, k = j\}$$

die Menge aller Datenpunkte, die minimalen Abstand zu Codebucheintrag c_j unter aller anderen Codebuchvektoren haben. Die Menge C_j wird *Cluster* genannt. Zusammen mit der Zuordnungsfunktion

$$\begin{aligned} Z : \mathbb{R}^M & \qquad [0 : M-1] \\ Z(f) & := \arg\min_{0 \leq j < M} \{|f - c_j|\}. \end{aligned}$$

die jedem Datenvektor den Codebuchvektor (oder Cluster) zuordnet, zu dem der Datenvektor minimalen Abstand hat, definiert sich die RMSE-Funktion als

$$\text{RMSE}(F, C) \quad := \quad \sqrt{\frac{1}{n} \sum_{i=0}^{N-1} (F^i - c_{Z(F^i)})^2}.$$

Der k-Means Clustering Algorithmus versucht iterativ das Codebuch an die Datenvektoren anzupassen. Da das Ergebnis von der initialen Bestimmung eines Codebuchs abhängt, kann nicht garantiert werden, dass der Algorithmus ein globales Minimum findet. Eine gute Einleitung zu dem Gebiet des Clustering und ein Vergleich unterschiedlicher Verfahren findet sich z.B. in [JMF99].
Der Algorithmus 5.1.1 entspricht der einfachsten Version dieses Verfahrens. Bei der Initialisierung wird ein Codebuch zufällig gewählt. Dies kann durch die zufällige Wahl von Vektoren aus $\mathbb{R}^M$ oder aber auch aus der Menge von Datenvektoren F^i geschehen. Für eine zuvor festgelegte maximale Anzahl von Iterationsschritten wird zunächst der Fehler bestimmt, mit dem das Codebuch die Datenvektoren approximiert. In einem nächsten Schritt werden neue Codebuchvektoren als Mittelwerte der jeweiligen Cluster gewählt und die Zuordnung von Datenvektoren zu Codebuchvektoren neu berechnet. Unterscheidet sich der Fehler r_t zum Zeitpunkt t um weniger als ϵ von dem Fehler zum Zeitpunkt $t-1$, kann die Iteration vorzeitig abgebrochen werden, da das Codebuch die Daten mit genügender Genauigkeit approximiert.
In der bis hier vorgestellten Version des k-Means Clustering war die Anzahl k der Codebuchvektoren fest vorgegeben. Jedoch ist die Anzahl der zur Verfügung stehenden Codebuchvektoren ein sehr wichtiger Parameter. Daher wird meist zusätzlich erlaubt, dass die Anzahl der Cluster während der Laufzeit variieren darf. Dazu legt man einen weiteren Parameter fest, der angibt, ab welchem Fehler ein Cluster in zwei Cluster aufgeteilt wird. Dazu startet man den Algorithmus mit einem Codebuch, welches nur ein Codewort enthält. Mit der Karhunen-Loeve-Transformation kann man die Hauptachsen der

Algorithmus 5.1.1: k-MEANS-CLUSTERING ALGORITHMUS(F, k, ϵ)

Initialisierung:
let $M = \dim(F^i), N = |F|$
Wähle $C = \{c_j, 0 \leq j < k\}$ Codebucheinträge zufällig aus $\mathbb{R}^M$ oder F)
for $i = 0$ **to** $N - 1$
  do Bestimme $Z(F^i)$ aufgrund des Codebuchs C
let $r_0 = \max\{|F^i - c_j|, 0 \leq i < N, 0 \leq j < M\}$
for $t = 1$ **to** MaxStep
  $r_t = \text{RMSE}(F, C)$
  if $|r_t - r_{t-1}| \leq \epsilon$
    then exit for //Abbruchkriterium erfüllt
  do
    for $j = 0$ **to** $M - 1$
      do $c_j = \text{MEAN}(C_j)$ //Bestimmung neuer Codebuchvektoren
    else
    for $i = 0$ **to** $N - 1$
      do Bestimme $Z(F^i)$ aufgrund des neuen Codebuchs C
Ausgabe: C, Z, r_t.

Algorithmus 5.1.1: Der k-Means Clustering Algorithmus.

Datenvektoren eines Clusters bestimmen. Entlang der dominanten Hauptachse, in deren Richtung die Datenpunkte die größte Varianz aufweisen, wird dann das Cluster geteilt, so dass die Varianz in den beiden neuen Clustern etwa die Hälfte der Varianz des Originalclusters beträgt. Bei dieser Variante wird die Iteration abgebrochen, sobald der Fehler hinreichend klein ist oder eine zuvor festgelegte Zahl von Iterationen durchgeführt worden ist.
Für jedes Musikstück $D_i \quad \mathcal{D}$ erhält man auf diese Weise ein Codebuch C^i mit Codebuchvektoren c^i_j, welche die Verteilung der Datenpunkte im Raum $\mathbb{R}^M$ approximiert. Die Codebücher bilden die Einträge in die Datenbank, in der später nach einem unbekannten Musikstück gesucht werden soll.

5.1.4.1.2 Matching-Algorithmus Für ein Anfragemusiksignal werden die Merkmalsvektoren F^i bestimmt. Für jedes Codebuch C^i wird nun der Fehler zwischen den Merkmalsvektoren und den Codebuchvektoren c^i_j bestimmt. Das Codebuch, welches die Merkmalsvektoren am besten approximiert, ist mit großer Wahrscheinlichkeit aus dem Musikstück D_i berechnet worden, aus dem das Anfragesignal stammt. In einer neueren Version des Systems werden nicht mehr die Merkmalsvektoren selber, sondern statistische Eigenschaften der Vektoren benutzt, um das Clustering durchzuführen. Leider geben die Autoren keine weiteren Details bekannt.

5.1.5 MusicDNS

In 2006 gestartet, bietet MusicDNS eine open-source Variante eines Audioidentifikationsdienstes an. In [mus06b, mus06a] wird dieser Dienst beschrieben, welcher auch kommerziellen Anwendern kostenpflichtig zugänglich ist. Laut der Webseite MusicDNS.org sind zum Zeitpunkt der Erstellung dieser Arbeit rund 16 Millionen Titel in der Datenbank erfasst. Dies ist sicherlich eine der größten Datenbanken im Bereich Audiofingerprinting. Den zentralen Nutzen sehen die Entwickler darin, dass es möglich ist, automatisch qualitativ hochwertige Metainformationen zu den eigenen Musikstücken zu

bekommen und auf diese Weise die Ordnung einer umfangreichen Sammlung von Musiktiteln zu erleichtern.
In [mus06b] werden einige technische Details zu dem Verfahren erläutert. Grundsätzlich arbeitet das Verfahren auf der PCM-Darstellung eines Musikstücks, d.h. Musik, die in einem komprimierten Format (MP3, ACC, etc.) vorliegt, ist vor der Identifikation zu dekodieren. Hierzu sind pro Musikstück sicherlich einige Sekunden Zeit nötig, bevor die eigentliche Generierung des Fingerprints erfolgen kann. Der Vorteil ist, dass das Verfahren dadurch unabhängig von einem verwendeten Kompressionsverfahren ist. Lediglich ein entsprechender Dekoder muss vorhanden sein, um ein komprimiertes Signal wieder in die PCM-Darstellung zu überführen.
Prinzipiell sei der Dienst so ausgelegt, dass beliebige Segmente eines Musikstücks als Anfrage formuliert werden können, allerdings werde derzeit lediglich die Generierung von Fingerprints auf den ersten zwei Minuten eines Musikstücks verfolgt. Ist das Stück kürzer als zwei Minuten, wird der Fingerprint aus dem kürzeren Signal berechnet. Die Autoren führen an, dass dies die Qualität der Identifikation erhöhe, da ein großer Teil eines Stücks die Basis dafür bilde. Die Signalqualität, welche zu einer erfolgreichen Identifikation notwendig sei, wird von den Autoren mit der Qualität eines MP3-komprimierten Musikstücks mit einer Bitrate von 64 kbps angegeben, was im Vergleich zu den zuvor beschriebenen Arbeiten als eher qualitativ hochwertig zu bezeichnen ist. Allerdings ist dies mit der zuvor angesprochenen Zielsetzung dieser Applikation vereinbar.

5.1.5.1 Generierung eines Fingerprints

Der „Open Fingerprint“ ist ein Array mit 516 Bytes, welcher die ersten zwei Minuten eines Musikstücks eindeutig beschreiben soll. Berechnet wird dieser Fingerprint in drei Schritte:

1. Im folgenden gehen wir der Einfachheit halber davon aus, dass das zu verarbeitende Signal die Länge von zwei Minuten aufweist. In einem ersten Schritt wird das PCM-Signal normalisiert, d.h. durch Änderung der Lautstärke des Signals wird die maximale Amplitude des Signals auf einen festgelegten Wert gebracht. Ggf. werden die Samplingrate und die Anzahl der Kanäle angepasst, damit die nachfolgenden Operationen auf vergleichbaren Daten erfolgen können. Nach diesem Schritt liegt ein mono Signal mit einer Abtastrate von 44.1 kHz vor. Ausserdem wurden ggf. am Anfang und am Ende des Stücks Passagen der Stille entfernt, da diese keinen signifikanten Beitrag zum Fingerprint liefern würden.

2. Im zweiten Schritt wird das mono Signal mit einer nicht überlappenden, gefensterten Fouriertransformation der Länge 8192 in den Frequenzraum überführt. Zur Weiterverabeitung wird zum Absolutbetrag des Spektrums übergegangen. Als Fensterfunktion wird ein Hamming-Fenster verwendet. Das vorläufige Ergebnis ist eine Matrix A, deren Zeilen jeweils der ^{1}-Norm eines berechneten Spektrums entsprechen. Jedes dieser Spektren umfasst 4096 Frequenzkomponenten.

   Mit Hilfe der sog. *Singulärwertzerlegung* (z.B. in [SW93]) kann die $m \times n$-Matrix A im Fall von $m \geq n$ geschrieben werden als Produkt $A = U \cdot S \cdot V^\top$, wobei U eine orthogonale $m \times m$-Matrix, S eine $m \times n$-Diagonalmatrix und V eine orthogonale $n \times n$-Matrix ist. Die Einträge $s_{ii} = s_i$ auf der Hauptdiagonalen der Diagonalmatrix S bilden eine monoton steigende Folge von Werten größer oder gleich Null. Ausserdem sind die Werte s_i charakteristisch für eine gegebene Matrix A, was für die Matrizen U, V nicht gilt. Die Einträge s_i bilden aus diesem Grund den zentralen Bestandteil des Fingerprints. Insgesamt stehen 512 Bytes in Analogie zur Eigenwertzerlegung für die ersten s_i zur Verfügung.

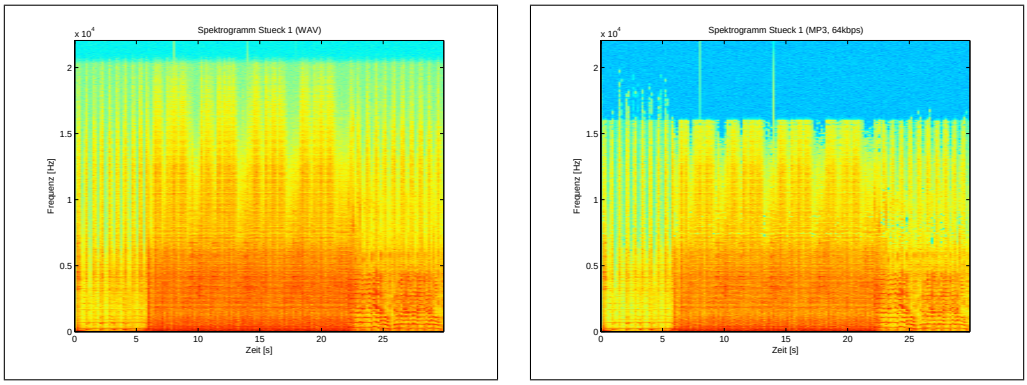


Abbildung 5.4: Die Abbildung zeigt zwei Spektrogramme des gleichen 30 Sekunden langen Audiostücks, wobei das rechte Spektrogramm der MP3-komprimierten (64kbps, mono) Version entspricht. Links ist das Originalsignal, wie es auf CD vorliegt. Deutlich zu erkennen ist die Verwendung eines Tiefpassfilters vor der MP3-Kompression. Frequenzen um 16 kHz wurden so vor der Komprimierung entfernt

3. Zusätzlich zu diesen 512 Bytes werden vier weitere Bytes dazu genutzt, die in dem Stück „auffälligsten“ Noten der MIDI-Tonleiter zu speichern. Genutzt wird diese Information, um zu Beginn der Suche eines Fingerprints in einer Fingerprint-Datenbank die Kandidaten von der Suche auszuschliessen, die über deutlich andere, auffällige Noten verfügen. Um die vier wichtigen Noten zu identifizieren, wird erneut eine Folge von Frequenzspektren berechnet, diesmal jedoch mit einer 80%-gen Überlappung der FFT-Frames. Danach wird prinzipiell über die Spektren iteriert und pro Note festgehalten, ob diese in einem Spektrum vertreten ist oder nicht. Die Dauern werden aufsummiert und am Ende werden die vier wichtigsten Noten bestimmt.

Schlussendlich wird der 516 Bytes umfassende Fingerprintvektor in die Base64-Darstellung transformiert und dieser ASCII-String wird über das Internet an die Suchmaschine übertragen. Diese Kodierung benutzt lediglich die Zeichen A-Z, a-z, 0-9, + und /, was eine Verarbeitung auf unterschiedlichen Plattformen ermöglichen soll.

5.1.5.2 Bearbeitung der Suchanfrage

Die eigentliche Suche beruht auf einer Vorauswahl von Kandidaten-Fingerprints basierend auf der übertragenden Noteninformation und einem Vergleich der Kandidaten-Fingerprints mit dem angefragten Fingerprint. In [mus06b] wird in diesem Kontext lediglich von „. . . server-tuned tolerances“ gesprochen, die bei dem Vergleich der Singulärwerte Verwendung finden. In [mus06b] wird weiterhin von einer Serverfarm ausgegangen, in der jeder Server in der Lage sein soll, pro Sekunde rund 100 Fingerprints zu validieren, wobei die Fingerprint-Informationen im RAM vorgehalten werden.

Das Ergebnis einer Suchanfrage ist ein XML-Dokument, welches neben einer eindeutigen ID (32 Bytes), Informationen wie Titel des Musikstücks und das jeweilige Album bzw. den Interpreten enthält. Die Toleranzen, die zum Vergleich von Fingerprints herangezogen werden, sind so gewählt, dass das resultierende XML-Dokument Metadaten zu nur einem Musikstück enthält.

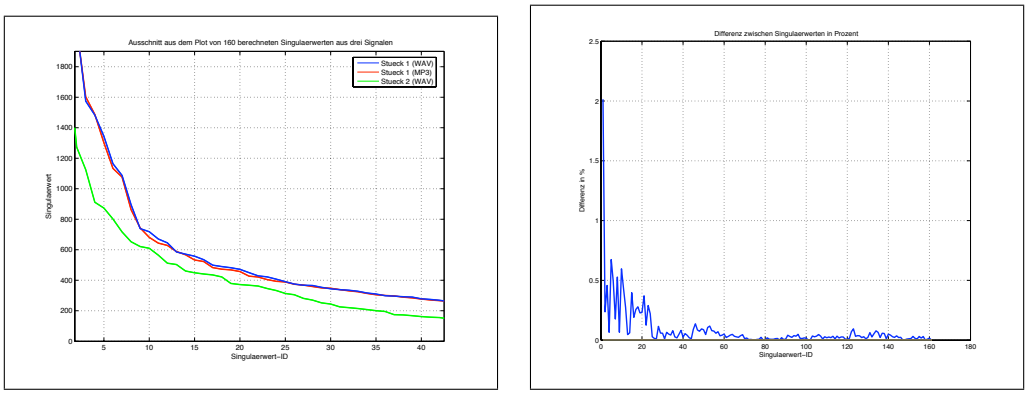


Abbildung 5.5: Die Abbildung zeigt links die durch die SVD berechneten Singulärwerte für drei Musikstücke. Die in blau und rot dargestellten Werte stammen von den in Abbildung 5.4 dargestellten Musikstücken. Die grün dargestellte Kurve zeigt die Singulärwerte eines anderen Musikstücks zum Vergleich. Die rechte Abbildung zeigt prozentual die punktweise Differenz zwischen Singulärwerten gewonnen aus dem Originalsignal und der komprimierten Version

5.2 Audioidenti kation mittels Suchindex basierend auf G-invertierten Listen

Im Vergleich zu den bisher vorgestellten Systemen zur Audioidentifikation ist das bereits in Kapitel 3 angesprochene Verfahren basierend auf der Verwendung von G-invertierten Listen flexibel an unterschiedliche Anwendungsszenarien anzupassen. Neben der inhaltsbasierten Suche in Audiosignalen, wurden in der Arbeitsgruppe Clausen Systeme entwickelt, die die Suche in Partituren [CEMS00, Mue05], Bildern [Röd02] oder auch „Motion Capture“ Daten [MRC05] ermöglichen.
Ermöglich wird dies durch die anwendungsabhängige Wahl der Gruppe G und der Menge M. Dabei ist die bei einer konkreten Anwendung zu benutzende Gruppe G meist fest vorgegeben, jedoch können unterschiedliche Realisierungen bzw. Implementierungen von G dazu genutzt werden, die Bearbeitung einer Suchanfrage zu beschleunigen. Insbesondere kann der Gleichheitsoperator von G anwendungsspezifisch implementiert werden.
Deutlich mehr Wahlmöglichkeiten bietet die Menge M, deren Elemente den Inhalt der Dokumente D_i $\mathcal{D}$ beschreiben. In diesem Abschnitt befassen wir uns mit unterschiedlichen Mengen M, die zur Audioidentifikation benutzt werden können. Dazu führen wir den Begriff *Merkmalsextraktor* ein. Dies ist ein Operator, der das Ausgangsdokument D_i auf eine Folge von *Merkmalen* D_i abbildet, die wiederum die Grundlage für den Aufbau eines G-invertierten Suchindex darstellt. Daher ist der Merkmalsextraktor die zentrale Komponente, wenn das Prinzip des G-invertierten Suchindexes auf eine neue Klasse von Dokumenten angewendet werden soll.
Die Benutzung des algebraischen Prinzips der Operation einer Gruppe auf einer Menge bietet ein mathematisches Grundgerüst, welches es erlaubt, die Korrektheit von Suchalgorithmen zu beweisen und gleichzeitig auf abstrakte Art und Weise neue Algorithmen zu entwickeln bzw. bestehende Algorithmen an gegebene Aufgabenstellungen anzupassen. In dieses Grundgerüst zur inhaltsbasierten Suche ist anwendungsabhängig ein als Merkmalsextraktor bezeichnetes Modul einzufügen, dessen Aufgabe es ist, die Dokumenteninformation derart aufzubereiten, dass sie zusammen mit dem Grundgerüst zur inhaltsbasierten Suche verwendet werden können. Meist beinhaltet dieses Modul auch eine drastische Reduktion an Informationen, die die inhaltsbasierte Suche in manchen Fällen erst praktisch möglich

machen. Je nach Art der Dokumente, die die Dokumentenkollektion $\mathcal{D}$ bilden, ist es meist die schwierigste Aufgabe ein solches Modul zu entwickeln, das die Anforderungen der jeweiligen Anwendung erfüllt.
Im folgenden werden einige Beispiele von Merkmalsextraktoren anhand des Szenarios der Audioidentifikation vorgestellt, d.h. die Ausgangsdokumente $D_i \quad ^2(\mathbb{Z})$. Die dabei zur Anwendung kommende Gruppe G ist $(\mathbb{Z}, +)$.

5.2.1 Merkmalsextraktion am Beispiel Audiosignalen

Ein auf einer CD gespeichertes Audiosignal besteht aus 44.100 Abtastwerten pro Sekunde und Stereo-Kanal. Dies entspricht einer Datenrate von $\approx 1,4MBit\ s$. Gehen wir von einer Sammlung von 100 randvollen AudioCDs aus, so benötigt die Dokumentenkollektion $\mathcal{D}$ einen Speicherplatz von rund 65 GB. Es gilt nun dieses Datenbestand in eine Suchindexdatenstruktur zu überführen.
Betrachtet man das Anwendungsszenario, dass ein Signalfragment von 10 Sekunden Länge in dieser Dokumentenkollektion lokalisiert werden soll, stellt sich die Frage, ob zu diesem Zweck überhaupt mit den Originalsignalen gearbeitet werden muss. Ggf. genügt es, lokale Eigenschaften eines jeden Signals D_i aus der Dokumentenkollektion zu berechnen und die eigentliche Suche auf diesen berechneten Daten durchzuführen. Auf diese Weise könnte man die Datenmenge, die in den Suchindex eingefügt werden muss, deutlich reduzieren, da wir anstelle von Audiosignalen Folgen von Merkmalen der Art $((\mathrm{tpos}_i, \mathrm{classification}_i))_{i \in I}$. Dies bedeutet anschaulich, dass zum Zeitpunkt $\mathrm{tpos}_i \quad \mathbb{Z}_{\geq 0}$ das Signal Eigenschaften aufweist, welche zusammengefasst als $\mathrm{classification}_i \quad [0 : C-1], C \quad \mathbb{Z}_{\geq 0}$, gespeichert wird. Dieses Prinzip lässt sich auch auf andere Arten von Dokumenten ausdehnen, wie das Kapitel 6 zeigen wird.
Die folgenden Anforderungen sind nicht nur im Falle von Audiosignalen an Merkmalsextraktoren und die von ihnen berechneten Merkmalsfolgen zu stellen:

Informationsgehalt: Die aus einem Dokument extrahierte Merkmalsfolge soll das zugehörige Dokument möglichst eindeutig beschreiben.

Robustheit: Ein Merkmalsextraktor sollte für jedes Dokument, welches die durch die Anwendung definierten Eigenschaften besitzt, die gleiche Merkmalsfolge berechnen. Im Fall von Audio könnte dies z.B. bedeuten, dass ein mit Bitraten größer oder gleich 64 kBit/s (mono) MP3-komprimiertes Signal möglichst die gleiche Merkmalsfolge ergibt, wie das Originalsignal von einer CD.

Einfachheit: Die Berechnung der Merkmale sollte in kurzer Zeit möglich sein, um auch große Datenmengen verarbeiten zu können bzw. um die Antwortzeit eines inhaltsbasierten Suchsystems zu reduzieren, da aus dem angefragten Dokument ebenfalls Merkmale extrahiert werden.

Translationsinvarianz: Sollen auch Fragmente von Dokumenten als Suchanfrage möglich sein, so sollte die Merkmalsfolge, die aus einem eben solchen Fragment berechnet worden ist, eine Teilfolge der Merkmalsfolge des korrespondierenden Gesamtdokuments sein.

Anzahl: Ein Merkmalsextraktor sollte eine möglichst große Zahl von unterschiedlichen Merkmalen berechnen, da diese der Anzahl der G-invertierten Listen im Suchindex entspricht.

Es liegt auf der Hand, dass die oben aufgelisteten Forderungen z.T. in gegenseitigem Widerspruch zueinander stehen. So liessen sich z.B. sehr robuste Merkmale definieren, indem man die Anzahl der möglichen Merkmale deutlich begrenzt. Allerdings steigt damit die Wahrscheinlichkeit, dass zwei

unterschiedliche Merkmalsfolgen D_i und D_j über Teilsequenzen verfügen, die in der jeweils anderen Folge enthalten sind. Die Beschreibung eines Dokuments durch die zugehörige Merkmalsfolge ist nicht mehr individuell genug, d.h. die Wahrscheinlichkeit für False-Positive Treffer steigt. Gleichzeit ist in einem solchen Fall mit einem Anstieg der Suchzeit zu rechnen, da in dem Suchindex nur wenige G-invertierte Listen mit vielen Elementen enthalten sind.
Erhöht man umgekehrt die Anzahl der sog. *Featurklassen*, d.h. die Anzahl unterschiedlicher Merkmale, ist mit einer größeren Zahl von kurzen G-invertierten Listen im Suchindex zu rechnen. Es ist also zu erwarten, dass Anfragen in kürzerer Zeit beantwortet werden können. Jedoch sinkt mit dem gestiegenen Informationsgehalt eines jeden Merkmals dessen Robustheit, so dass der zuvor beschriebene Zeitvorteil bei der Anfragebearbeitung dadurch zunichte gemacht wird, dass entsprechend viele Fehlstellen zugelassen bzw. Fuzzy-Elemente betrachtet werden müssen.
Besonders im Falle von Audiosignalen ist es unumgänglich, dass eine Anfrage eben nur einen Teil eines gesuchten Audiosignals entspricht. Dies ist bei der Entwicklung eines Merkmalsextraktors von Nachteil, da z.B. die Lautstärke eine Anfrage z.B. nicht normalisiert werden kann.
Die Wahl eines Merkmalsextraktors beeinflußt essentiell den Trefferbegriff. Im Fall von wenig deskriptiven Merkmalen können zwei Merkmalsfolgen zu unterschiedlichen Dokumenten gleiche Teilsequenzen enthalten, was zu einer größeren Treffermenge $G_{\mathcal{D}\,k}(Q)$ führt. D.h. die Dokumente werden bzgl. einer Anfrage Q als ähnlich angesehen, obwohl diese sehr unterschiedlich sein können.

5.2.1.1 Abtastwerte als Merkmale

Der sicherlich einfachste Merkmalsextraktor benutzt die einzelnen Abtastwerte als Merkmal. Einer jeden Zeitposition im Signal wird eine Merkmalsklassenindex $c \quad C = [0 : 65535]$ zugewiesen (16 Bit Auflösung pro Abtastwert). Die Suche nach einem Fragment liefert die samplegenaue Position des Fragments im Treffersignal. Diesen beiden Vorteilen folgen eine Vielzahl von Nachteilen. So wird keine Datenreduktion erreicht, d.h. der Suchindex belegt mindestens den gleichen Speicherplatz wie die Dokumentenkollektion $\mathcal{D}$. Es ist daher davon auszugehen, dass bei einer Anfrage eine große Anzahl von Listenelementen betrachtet werden muss. Ausserdem sind diese Merkmale nicht robust gegenüber den kleinsten Signalstörungen. Der aus diesen Merkmalen resultierende Trefferbegriff ist sehr restriktiv, da samplegenaue Übereinstimmung von Anfrage und Dokument gefordert werden.

5.2.1.2 Lokale Maxima im tiefpassge lterten Signal

Einen anderen Ansatz wurde von Kurth vorgeschlagen. Das Eingabesignal $x \quad ^2(\mathbb{Z})$ wird zunächst mit einem Tiefpassfilter f gefaltet: $C_f[x] := f * x$. Das resultierende Signal $C_f[x]$ enthält nur noch tiefe Frequenzen und erscheint damit geglättet. Nachfolgend werden in dem geglätteten Signal signifikante lokale k-*Maxima* gesucht. Ein lokales k-Maximum an der Stelle t liegt genau dann vor, wenn gilt:

$$C_f[x](t-k) < \ldots < C_f[x](t) > \ldots > C_f[x](t+k)$$

für ein fest gewähltes $k \geq 0$. Der Operator M_k setzt alle Signalwerte auf 0, an denen kein lokales k-Maximum vorliegt.
Zur weiteren Reduktion der Anzahl von Merkmalen wird im nächsten Schritt eine k'-Maximaberechnung $M'_{k'}$ durchgeführt, jedoch werden diesmal nur die Stellen im Signal betrachtet, die einen Wert ungleich 0 aufweisen. Durch das verwendete Tiefpassfilter und die Parameter k und k' ist es möglich, die Anzahl der lokalen Maxima pro Sekunde zu steuern. Um nun aus diesem stark ausgedünnten Signal eine Merkmalsfolge zu berechnen, werden die Abstände in Samples zwischen den einzelnen Maxima durch den Operator δ bestimmt. Dieser Abstand entspricht dann geeignet quantisiert durch

den Operator Q_c der Klassifikation zum Zeitpunkt t. Dieser Merkmalsextraktor $F_{\max}$ lässt sich schreiben als Verkettung von einzelnen Operatoren:

$$F_{\max}[x] := Q_c \circ \delta \circ M'_{k'} \circ M_k \circ C_f[x].$$

Im Gegensatz zu dem zuvor beschriebenen simplen Merkmalsextraktor, bestimmt hier das Signal x zu welchen Zeitpunkten t ein Merkmal extrahiert wird. Somit gewinnt hier die Zeitkomponente mehr Gewicht, da die Abstände zwischen den Merkmalen an sich schon sehr charakteristisch sein können. Allerdings verlangt dieser Ansatz gemäß der Wahl der Parameter eine entsprechende Länge der Anfrage, so dass genügend Merkmale extrahiert werden können. Allerdings zeigt die Praxis [KC01], dass in aller Regel genügend Merkmale gewonnen werden können. Anzumerken ist noch, dass die Berechnung der Faltung mittels FFT-Algorithmen sehr schnell erfolgen kann.

5.2.1.3 Lautstärke basierte Merkmale

In [RK02] wird ein weiterer Merkmalsextraktor vorgestellt, der Merkmale in der Zeitbereichsdarstellung des Audiosignals lokalisiert, jedoch eine höhere Robustheit der Merkmale gegenüber Signaldegeneration (z.B. MP3-Komprimierung) garantiert.
Anstelle das Audiosignal direkt mit einem Tiefpassfilter zu glätten, wird bei diesem Merkmalsextraktor die mittlere Lautstärke eines kurzen Signalausschnitts bestimmt. Da das dazu verwendete Analysefenster über das Signal bewegt wird, entsteht auf diese Weise ein neues Signal, welches die mittlere Lautstärke des Eingangssignals über die Zeit beschreibt. Dieses Signal wird vor der bereits zuvor beschriebenen Maximasuche durch Tiefpassfilterung geglättet.

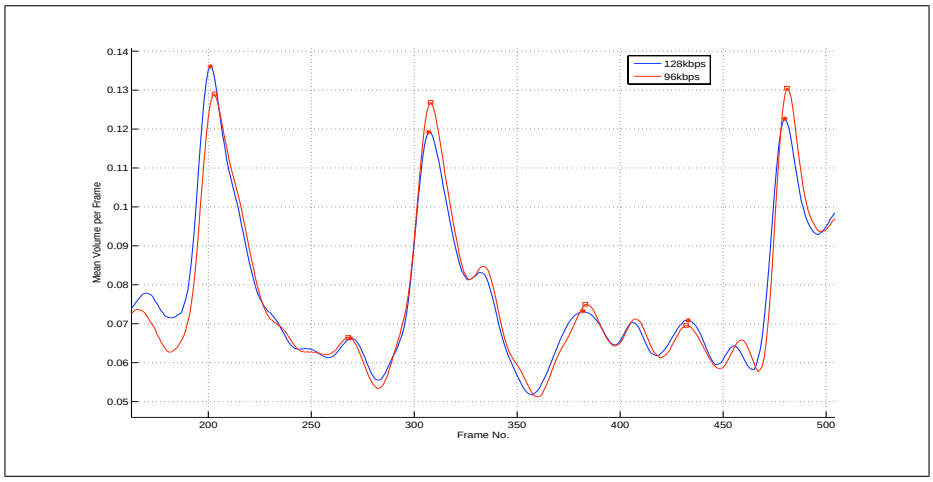


Abbildung 5.6: Ausschnitt aus zwei Signalen der mittleren Lautstärke für zwei Qualitäten desselben Audiosignals.

5.2.1.4 Code-basierte Merkmale

In [Kur02] beschreibt Kurth einen anderen Ansatz, der besonders im Hinblick auf die Suche mit Fehlstellen die Laufzeit des Suchalgorithmus reduzieren soll. Nach einer Tiefpassfilterung wird das

Signal mit der Fensterfunktion w mit kompaktem Träger und Schrittweite s in überlappende Frames unterteilt, wobei in jedem Frame die Signalenergie bestimmt wird:

$$V_s[x] := k \qquad |\tau_{ks}[w](\ )x(\ )|^2.$$

Dabei beschreibet der Operator $\tau_t[x] : k \quad x(t+k)$ eine Verschiebung des Signals $x \quad ^2(\mathbb{Z})$ um $t \quad \mathbb{Z}$ Abtastwerte. Streng genommen verletzt die Verwendung einer Fensterfunktion die Forderung nach der Translationsinvarianz eines Merkmalsextraktors. Allerdings kann bei geeigneter Wahl der Schrittweite s und der Länge des Fensters w eine Art Pseudotranslationsinvarianz erreicht werden, da z.B. bei genügend kleiner Schrittweite s das Audiosignal sich nur wenig zwischen zwei aufeinanderfolgenden Frames ändert. Das durch die Anwendung von V_s gewonnene Signal wird nun grob quantisiert: $\Delta[x] : k \quad \text{sign}(x(k+1)-x(k))$ wobei $\text{sign}(x) := 0$, wenn $x \leq 0$ gilt und $\text{sign}(x) := 1$ in allen anderen Fällen.
Die so erhaltene Binärsequenz wird nun durch den Operator $\mathcal{C}_n$ in überlappende Teilsequenzen (Schrittweite: 1) der Länge n aufgeteilt. Der nachfolgende Operator $\mathcal{E}_C$ wird auf die soeben ermittelten Teilsequenzen angewendet, wobei $C \subseteq 0,1^n$ ein „Codebuch“ darstellt. $\mathcal{E}_C$ weisst jeder Teilsequenz das bzgl. des Hammingabstandes nächste Codewort zu. Überschreitet dabei der Hammingabstand einen Schwellwert ϵ, so wird der Teilsequenz das Codewort 0^n zugewiesen. Insgesamt ergibt sich folgender Merkmalsextraktor:

$$F_C[x] := \mathcal{E}_C \circ \mathcal{C}_n \circ \Delta \circ V_s \circ C_f[x].$$

Dieser Merkmalsextraktor weist also Positionen im Signal x Merkmalsklassen aus dem Bereich $[0 : 2^n - 1]$ zu.
Um eine bessere Verteilung über die extrahierten Codeworte zu erreichen, wird die Binärsequenz in M Sequenzen aufgeteilt, die jeweils einer Phase entsprechen und mittels des Operators $(\quad M)$ „downgesampled“ worden sind.

5.2.1.5 Merkmale basierend auf MP3 Datenströmen

In der Diplomarbeit von Wagener [Wag03] wurde versucht, einen Merkmalsextraktor zu entwickeln, der als Eingabe ein MP3-Datenstrom erhält. Die Idee dabei ist, dass die Merkmalsextraktion auf dem MP3-Datenstrom durchgeführt wird, um auf diese Weise die aufwändige Dekodierung des Signals zu vermeiden.
Wie bereits in 4.3.3 beschrieben, erfolgt die Dekodierung eines MP3-Datenstroms in mehreren Stufen. In [Wag03] wurde daher untersucht, welche Stufe der Dekodierung es erlaubt, robuste Merkmale aus MP3-Datenströmen zu extrahieren und diese zur Audioidentifikation zu nutzen. Die Herausforderung besteht in diesem Fall darin, dass die für ein Signal von unterschiedlichen Enkodern berechneten MP3-Datenströme sehr unterschiedlich sein können, je nachdem welche Strategie ein MP3-Enkoder anwendet.
Wagener schlägt vor, die Merkmalsextraktion auf den Subbandsamples im Zeitbereich durchzuführen. Bis diese Information aus dem MP3-Datenstrom gewonnen werden kann, sind zunächst die Huffmancodeworte zu dekodieren, die De-Quantisierung der MDCT-Spektren durchzuführen und anschliessend die inverse MDCT zu berechnen. Bei der Merkmalsextraktion auf der Stufe der Subbandsamples wird also die Berechnung der inversen Filterbank im Vergleich zur Extraktion von Merkmalen im vollständig dekodierten Signal überflüssig.
Ähnlich wie bei 5.2.1.4 wird auch hier die Signalenergie als zentrale Eigenschaft betrachtet, auf deren Grundlage die Merkmale berechnet werden. Dazu wird getrennt für eine festgelegte Zahl von

Subbändern die gefensterte Signalenergie bei einer Schrittweite von 64 Subbandsamples und einer Fensterlänge von 2048 Subbandsamples berechnet. Auf den Energiesignalen der ausgewählten Subbändern wird im nachfolgenden Schritt nach lokalen Extremstellen gesucht, deren Zeitpositionen Merkmalspositionen entsprechen. Nach [Wag03] setzt sich ein Merkmal aus fünf Komponenten durch Aneinanderfügen einzelner Dezimalwerte zusammen:

1. Die Art der Extremstelle: 0 für Minimum, 1 für ein Maximum,
2. die Nummer des Subbandes, in dessen Signal die Extremstelle gefunden worden ist,
3. der quantisierte Amplitundenwert an der Stelle des betrachteten Extremums,
4. der quantisierte Amplitundenwert an der Stelle des vorhergehenden Maximums,
5. der quantisierte Amplitundenwert an der Stelle des vorhergehenden Minimums.

Nach Wagener liegt die Anzahl von Merkmalen bei der Betrachtung von fünf Subbandsignalen bei etwa 7 Merkmalen pro Sekunde. Somit sollte in der Regel eine genügende Anzahl von Merkmalen auch aus kurzen Signalausschnitten extrahiert werden können.
Dieser Merkmalsextraktor wurde benutzt, um den in Abschnitt 3.5 zur Evaluation benutzten Datenbestand zu generieren. Wie Abbildung 3.4 zeigt, werden manche Merkmalsklassen deutlich häufiger extrahiert als andere. Insgesamt werden lediglich 4.096 unterschiedliche Merkmalsklassen überhaupt extrahiert. Beide Gründe zusammen sind mit ein Grund für die in [Wag03] angegeben Zeiten zur Beantwortung einer Suchanfrage. Dort benötigte eine Anfrage bestehend aus zehn Merkmale bei 30 Prozent Fehlerrate z.T. über eine Sekunde. Um die Anzahl von „False Positive“-Treffern zu reduzieren, wurden mehrere Segmente angefragt und die Suchergebnisse dieser Einzelanfrage zusammengefasst, so dass eine Audioidentifikation im Falle von [Wag03] mehrere Sekunden in Anspruch genommen hat. Vergleicht man diese Zeiten mit den in Abbildung 3.5, so ist dies ebenfalls ein Indiz für die Leistungsstärke der in Kapitel 3 neu vorgestellten Algorithmen.

5.2.1.6 Lokale Maxima im Spektralbereich

Dieser Merkmalsextraktor ist eng verwandt mit dem, der von der Firma Shazam in Abschnitt 5.1.1.1 vorgestellt worden ist. Allerdings benötigt der Ansatz basieriend auf G-invertierten Listen keine aufwändige Kombination von einzelnen im Spektrogramm lokalisierten Maxima zu Hashwerten. Die Merkmale bestehend aus der Zeitposition und dem Frequenzindex beschreibenden Positionen im Spektrogramm eindeutig und eignen sich ebenfalls zum Aufbau eines Suchindexes zur Audioidentifikation. Die Anzahl der unterschiedlichen Merkmalsklassen ist jedoch beschränkt durch die Auflösung des Spektrogramms im Frequenzbereich. Eine geringe Auflösung im Frequenzbereich führt zwar zu robusteren Merkmals, jedoch reduziert dies die Anzahl der G-invertierten Listen im Suchindex, was direkt zu einer großen mittleren Anzahl von Listeneinträgen führt. Durch die Kombination von Merkmalen ähnlich wie bei den Ansätzen von Shazam bzw. Wagener könnte zu einer größeren Anzahl von G-invertierten Listen führen.
Wählt man z.B. eine FFT der Länge 2048, so bleiben maximal 1024 unterschiedliche Merkmalsklassen. Wenn man nun davon ausgeht, dass vor einer MP3-Kodierung eine Tiefpassfilterung durchgeführt wird, bei der Frequenzen oberhalb von 16 kHz aus dem Signal entfernt werden (vgl. 5.4), reduziert sich die Zahl möglicher Merkmalsklassen weiter. Außerdem konzentriert sich die Signalenergie meist im unteren Bereich des Spektrums, so dass die Wahrscheinlichkeit, dort lokale Maxima zu finden bedeutend höher ist.

5.2.1.6.1 Weiteres Merkmale im Spektralbereich In [RK02] wird ein weiterer Merkmalsextraktor F_{wft} vorgeschlagen, der im Spektralbereich arbeitet. Nach der Berechnung eines Spektrogramms mittels einer WFT, wird für jedes Spektrum des Spektrogramms durch den Operator S die jeweilige Centroide bestimmt:

$$S[y](n) := \frac{1}{y(.,n)_{\ 1}} \sum_{k=0}^{N\ 2-1} (k+1)|y(k,n)|,$$

wobei $S[y](k,n)$ die k-te Frequenzkomponente des n-ten Spektrums bezeichnet. Auf diese Weise wird aus dem Spektrogramm ein normalisiertes Zeitsignal gewonnen, in welchem nach einer Tiefpassfilterung zur Glättung nach lokalen Maxima gesucht wird. Die quantisierten Abstände zwischen den Maxima werden dann wie in 5.2.1.2 als Merkmalsklasse aufgefasst. Wie in [RK02] angegeben, extrahiert dieser Merkmalsextraktor im Durchschnitt rund 30 Merkmale pro zehn Sekunden Audio. Damit lässt sich ein Suchindex aufbauen, der nur wenig Speicherplatz benötigt. Allerdings sinkt die Robustheit der Merkmale bei einer MP3-Kodierung mit 96 kbps (stereo) bereits deutlich, wie auch in Abbildung 5.7 zu erkennen ist.

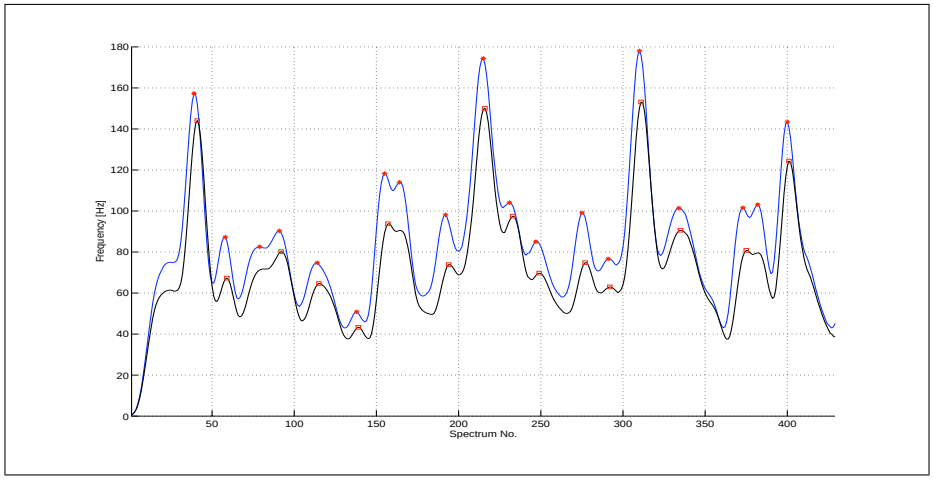


Abbildung 5.7: Die Abbildung zeigt das mittels des Operators S aus dem Spectrogramm eines Signals extrahierte Zeitsignal, welches mit einem Tiefpassfilter geglättet worden ist. Die von dem Merkmalsextraktor lokalisierten Maxima sind hervorgehoben. Die blaue Kurve zeigt das Ergebnis für ein mit 128 kbps codiertes MP3-Signal, hingegen wurde die schwarze Kurve aus dem gleichen jedoch mit 96 kbps codierten Signal gewonnen. Die Zeitpunkte der Maxima stimmen relativ gut überein, wohingegen die Anzahl der Maxima sich deutlich unterscheidet. Dies führt zur Berechnung von unterschiedlichen Merkmalen, da ja die Abstände zwischen den Maxima als Basis für die Merkmalsgewinnung dienen.

5.2.2 Web-basierter Audioidenti kationsdienst

In diesem Abschnitt wird der prinzipielle Aufbau des Audioidentifikationssystems beschrieben, welcher auch innerhalb der danach beschriebenen „SyncPlayer“-Anwendung Verwendung findet. Eine detaillierte Beschreibung findest sich in [Rib06]. Die Abbildung 5.8 zeigt die beteiligten Komponenten. Dem Aufbau liegt die Idee zugrunde, dass das System gut skalierbar in Bezug auf die

Menge an Audiodateien sein soll. Aus diesem Grund wurde die Anwendung mittels des Java-RMI-Frameworks als verteilte Applikation entworfen. Das gleiche Framework kommt auch in einer anderen web-basierten Anwendung der Arbeitsgruppe von Prof. Clausen zum Einsatz, bei der es um die Suche in Melodien geht.

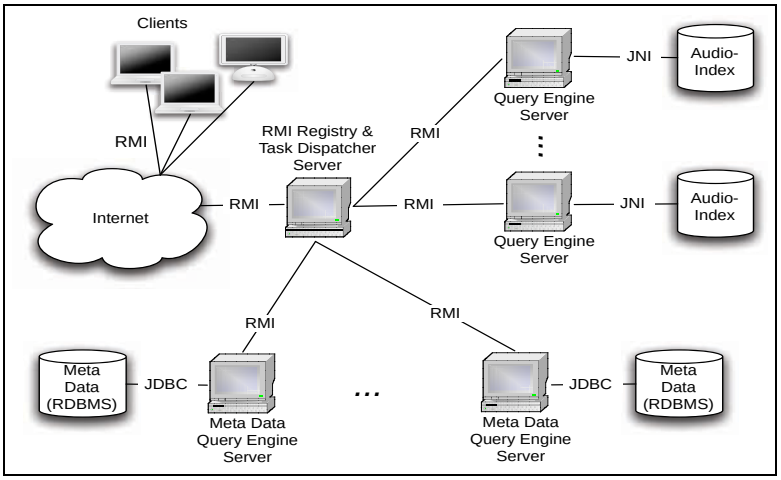


Abbildung 5.8: Die Abbildung zeigt den prinzipiellen Aufbau der Audioidentifikationskomponente im Rahmen des SyncPlayer Frameworks. Dabei wurde Wert auf die Skalierbarkeit der Anwendung gelegt.

Der zentrale RMI-Server, der auch über das Internet erreichbar ist, koordiniert alle ankommenden Anfragen. Jeder Client verbindet sich zu diesem Server über die Java-RMI Technologie. Gleiches gilt für die Komponenten, die die eigentliche Audioidentifikation implementieren und jene Komponenten, die die Metadaten zu den Dokumenten in der Dokumentenkollektion verwalten. Durch Hinzufügen weiterer Rechner, skaliert somit das Gesamtsystem mit der Anzahl der Dokumente in der Dokumentenkollektion.

Szenario 1: Bei diesem Szenario besitzt jeder der „Query Engine"-Server eine Kopie des gesamten Audioindex. Somit genügt es dem zentralen Server, das zu einer Anfrage generierte Task-Objekt an einen beim zentralen Server registrierten „Query Engine"-Server zur Verarbeitung weiterzureichen. Bei einer nicht zu großen Datenmenge garantiert dieser Aufbau schnelle Antwortzeiten, da in der Regel eine Anfrage sofort an einen Server weitergereicht werden kann, der gerade keine Anfrage bearbeitet. Dies setzt natürlich eine entsprechende Anzahl von Rechnern voraus, die die inhaltsbasierte Suche durchführen.
Ähnlich verhält es sich hier mit den Metadaten zu den Audiodateien im Audio-Index. Auch hier gewährleistet eine gewisse Anzahl von Rechnern die schnelle Bearbeitung von Metadatenanfragen.

Szenario 2: Bei größeren Datenmengen ist es sinnvoll, den Datenbestand des Audio-Index auf mehrere Rechner zu verteilen. Da jedem Eintrag im Audio-Index eine eindeutige Dokumenten-ID zugeordnet ist und da Treffer nur innerhalb des gleichen Dokuments auftreten können, bietet sich eine Aufteilung nach der Dokumenten-ID an. Der zentrale Server muss in diesem Szenario das aus der Anfrage generierte Task-Objekt an so viele „Query Engine"-Server weiterleiten, so dass der gesamte Datenbestand abgedeckt ist. Ausserdem müssen die einzelnen Sucher-

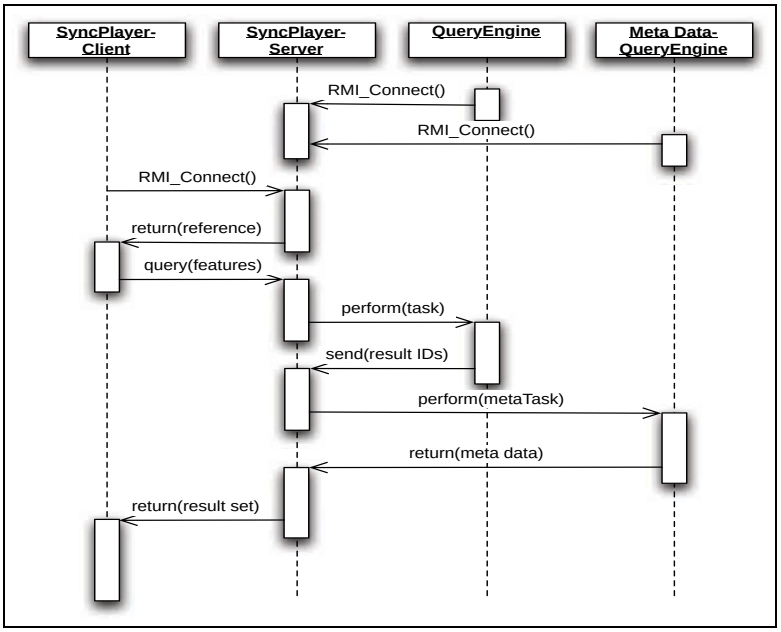


Abbildung 5.9: Kommunikation der einzelnen Komponenten bei der Identifizierung eines angefragten Audiosignals in Form einer Merkmalsfolge.

gebnisse von dem zentralen Server zu einem Ergebnis zusammengefasst werden, bevor den Ergebnisdokumenten-IDs die jeweiligen Metadaten zugeordnet werden können.

Eine Erweiterung umfasst die Zusammenfassung von Servern zu *Servergruppen*. In Anlehnung an das erste Szenario wählt der zentrale Server aus jeder Gruppe einen Server zur Bearbeitung einer Anfrage aus. Eine hinreichend große Anzahl von Rechnern vorausgesetzt, führt dies zu einer schnelleren Bearbeitung einzelner Anfragen, da in der Regel aus jeder Gruppe ein Server direkt zur Bearbeitung der Anfrage benutzt werden kann.

Weisst man z.B. allen Audosignalen in der Dokumentenkollektion, die zu einem Interpreten gehören, fortlaufende Dokumenten-IDs zu, führt die Aufteilung des Datenbestandes unter Berücksichtigung der Eigenschaft „Interpret“ dazu, dass die zusätzlich zu der Anfrage spezifizierten Metadaten genutzt werden können, die Anfrage gezielt an nur einen Server zur eigentlichen Audioidentifikation zu senden. Dies reduziert ebenfalls die Last auf dem Gesamtsystem in solchen Fällen, wo korrekte Metadaten zu einer Anfrage vom der anfragenden Person spezifiziert worden sind, da die Anfrage nicht an alle Server gesendet werden muss. Zusammen mit der Gruppierung von Servern kann auf diese Weise eine Reduzierung der Antwortzeit des Systems erwartet werden.

5.2.3 Der „SyncPlayer“

In der Arbeitsgruppe von Prof. Clausen an der Universität Bonn wurde eine Software *SyncPlayer* [KMD+05, KMR+04] entwickelt, die als eine Kernkomponente die Audioidentifikation beinhaltet. Die Abbildung 5.10 zeigt das Hauptfenster und das sog. *Playlist-Plugin*. Die zentrale Komponente des SyncPlayers lässt sich durch *Plugins* leicht um neue Funktionalitäten erweitern. Diese sind es auch, die die besonderen Eigenschaften der Software darstellen. Die Client-Software ist für die

gängigsten Betriebssysteme erhältlich über die Webseite des Projekts [Syn04]. Daneben wird dort auch Demomaterial angeboten.

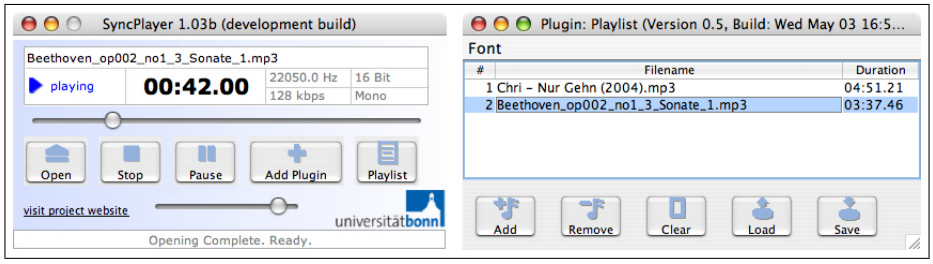


Abbildung 5.10: Das Hauptfenster des *SyncPlayer* zusammen mit dem *Playlist*-Plugin .

5.2.3.1 Synchrone Darstellung unterschiedlicher Medien

Der Name des SyncPlayers ist durch die Fähigkeit der Darstellung unterschiedlicher Medien *synchron* zu dem gerade wiedergegebenen Audiostück. Da z.B. Liedtexte von der Server-Komponente des SyncPlayers an den Client gesendet werden, muss zunächst festgestellt werden, welches Stück gerade abgespielt wird. Hierzu wird eine Audioidentifikationskomponente basierend auf dem in den voran gegangenen Kapiteln vorgestellten Ansatzes verwendet. Zunächst werden daher die sog. Merkmale aus dem aktuellen (d.h. gerade spielenden) Audiosignal extrahiert. Und nur diese werden an die Serverkomponente über das Netzwerk mittels der Java RMI-Technologie übermittelt. Dort wird geprüft, ob das Signal dem Server bekannt ist, d.h. ob es in der Dokumentenkollektion vorhanden ist (vgl. Abbildung 5.9). Ist dies der Fall, werden dem Client z.B. der Liedtext gesendet. Da zu jedem darin enthaltenen Wort auch die Zeitpositionen im Audiosignal abgelegt ist, ist es dem Client möglich, die Textstelle hervorzuheben, die gerade abgespielt wird.
Es ist derzeit ein aktuelles Forschungsgebiet, die Noten eines Musikstücks *automatisch* Zeitpositionen im einem Audiosignal zuzuordnen (siehe z.B. [MKR04]). Da derzeit keine allgemein verwendbaren Verfahren existieren, die zur automatischen Annotation von beliebigen Audiosignalen z.B. mit Noten oder gar Text genutzt werden können, wurde für den SyncPlayer ein entsprechendes Plugin erstellt, welches eine manuelle Annotation von Audiosignalen ermöglicht.

5.2.3.1.1 Piano-Roll Darstellung Liegen zu einem Musikstück die Noten in Form von MIDI-Dateien zusätzlich zur Audiodatei vor, so stellt dieses Plugin die Noten dar, welche gerade gespielt werden. Die Noten werden dabei in der Piano-Roll Darstellung angezeigt. Rot dargestellte Noten sind bereits gespielt worden, blau angezeigt werden die Noten, die gerade gespielt werden und grün die Noten, welche noch nicht gespielt worden sind. Die Abbildung 5.11 gibt dazu ein Beispiel.

5.2.3.1.2 Liedtexte Das zuvor bereits erwähnte Plugin erlaubt auch den Text z.B. zu einem Lied synchron mit der gerade spielenden Audiosignal anzuzeigen, wie man es z.B. von Karaoke Geräten her kennt (Abbildung 5.12). In einem Vorverarbeitungsschritt müssen dazu die exakten Wortpositionen eines jeden Wortes des Liedtextes in dem Audiosignal ermittelt werden. Es ist anzumerken, dass die automatische Synchronisation von Text und Gesang derzeit nicht automatisch erfolgt, sondern gänzlich per Hand durchgeführt werden muss. Ein entsprechendes Plugin stell das Projekt ebenfalls zur Verfügung.

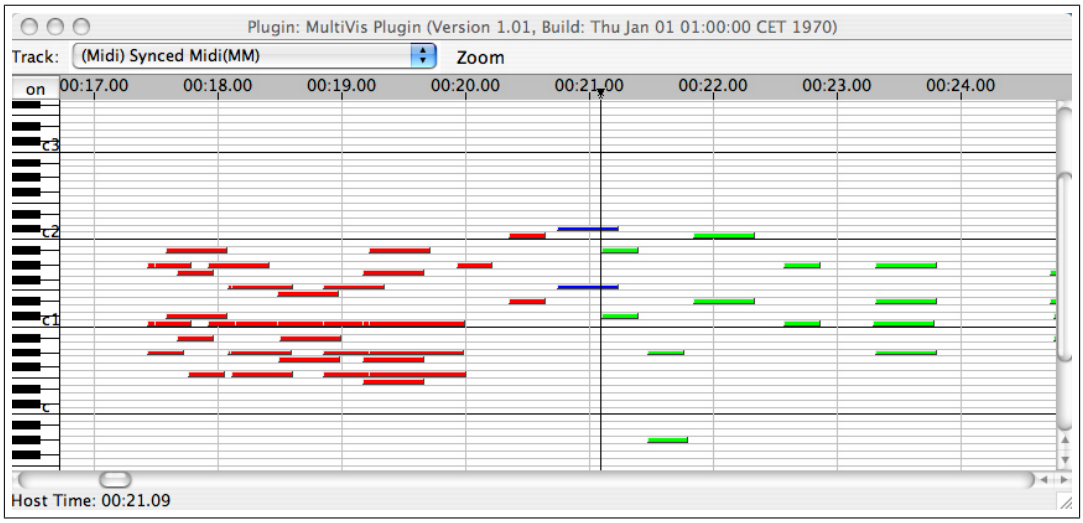


Abbildung 5.11: Visualisierung einer Partitur in der Piano-Roll Darstellung. Die gerade erklingenden Noten sind blau dargestellt, rot die bereits gespielten Noten und grün die noch zu spielenden Noten.

5.3 Zusammenfassung

Dieses Kapitel hat gezeigt, dass die inhaltsbasierte Suche in Audiodaten (insbesondere Musik) viel Beachtung gefunden hat. Viele Forschergruppen haben entsprechende Systeme entwickelt, die z.T. auch kommerziell über unterschiedliche Mobilfunkanbieter dem Kunden zugänglich sind. Die Ansätze lassen sich grob in zwei Kategorien unterteilen. „Lernverfahren“ werden in [AHH+01, BMG02] verwendet, wohingegen in [HKO01, KC01, Wan03] versucht wird, charakteristische Eigenschaften des Audiosignals zu extrahieren und diese in Indexdatenstrukturen einzufügen. Anzumerken ist, dass der in [Wan03] vorgestellte Ansatz mehr oder weniger ein Spezialfall von [CEMS00] und auch von [WR97] darstellt.

Neben der Identifikation von Audiosignalen sehr schlechter Qualität über das Mobilfunknetz, bietet www.MusicDNS.org [mus06a] einen Audioidentifikationsdienst an, der weniger auf schlechte Signalqualität hin optimiert ist, sondern darauf ausgelegt ist, qualitativ hochwertige Metadaten zu Audiostücken zu liefern, um auf diese Weise z.B. ein großes Archiv von Musikstücken nach einem einheitlichen Schema zu verschlagworten. Der Rückgriff auf eine Audioidentifikationstechnik ist dabei unbedingt notwendig, da z.T. die Metadaten zu digital gespeicherten Musikstücken nur in der Ordnerstruktur auf einem Datenträger enthalten sind und gar nicht mit dem Stück selber verknüpft sind.

Die in der Arbeitsgruppe von Prof. Clausen entwickelte Applikation „SyncPlayer“ [Syn04, KMR+04, KMD+05] verfolgt ebenfalls das Konzept der Metadatenbereitstellung. Allerdings wird hierbei mehr Wert auf Metadaten gelegt, die synchron mit einer Audiodatei dem Benutzer angezeigt werden können. Zusammen mit dem vorgestellten RMI-Framework [Rib06] wäre die in dem Projekt verwendete Audioidentifikationskomponente auch für große Datenmengen (mehrere Millionen Audiostücke als Dokumentenkollektion) gewachsen.

In dem nächsten Kapitel wenden wir uns von Audiosignalen als Dokumentenbasis ab, um uns der inhaltsbasierten Suche in Proteinstrukturdaten zu widmen. Dieser extreme Schwenk zu einem ganz anderen Dokumentenformat (PDBXML) soll zweierlei zeigen. Zum einen die Variabilität des Ansat-

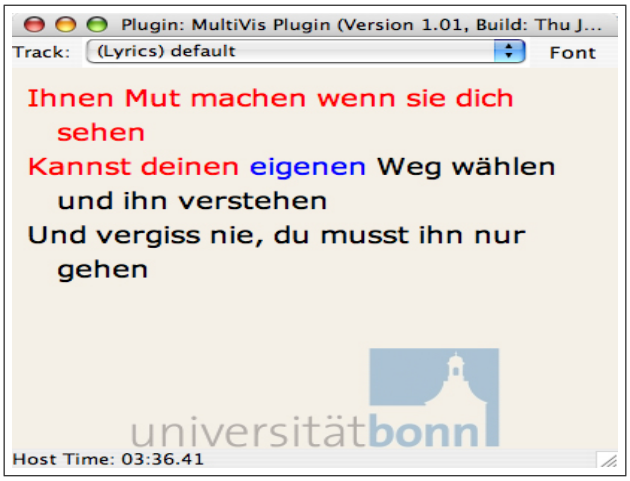


Abbildung 5.12: Liedtext zu einem gerade wiedergegebenen Musikstück. Das Wort, welches gerade gesungen wird, ist blau hervorgehoben.

zes der inhaltsbasierten Suche basierend auf G-invertierten Listen. Zum anderen soll deutlich werden, wie das Grundkonzept an diese neue Aufgabenstellung anzupassen ist.

Kapitel 6

Inhaltsbasierte Suche in chemischen Molekülen

6.1 Einleitung

Im vorangegangenen Kapitel wurde die robuste Audioidentifikation beschrieben. Bei dieser Anwendung operiert die additive Gruppe $\mathbb{Z}$ als Gruppe von ganzzahligen Translationen auf einer Menge $M = [0 : N] \times \mathbb{Z}$, wobei gewissen Zeitpunkten $t \quad \mathbb{Z}$ eine lokale Eigenschaft des Signals in Form eines Wertes $c \quad [0 : N]$ zugeordnet wird. Beispiele hierfür sind die unterschiedlichen Merkmalsextraktoren, wie sie in Kapitel 5 vorgestellt worden sind.

Die Operation der Gruppe $\mathbb{Z}$ auf der eben beschriebenen Menge M ist sicherlich eine der „einfacheren“ Anwendungsfälle für die in Kapitel 2 abstrakt beschriebene Technik zur inhaltsbasierten Suche in großen Dokumentenkollektionen. Um zu zeigen, dass diese Suchtechnik auch für kompliziertere Gruppenoperationen anwendbar ist, wird in diesem Kapitel vorgestellt, wie man damit die effiziente Suche nach Atomkonstellationen in einer Datenbank von Proteinmolekülen gestalten kann. Die Rolle von $\mathbb{Z}$ übernimmt hier die spezielle Euklidische Gruppe $SE(3)$, die von den Translationen und Drehungen im $\mathbb{R}^3$ erzeugt wird.

Da dieser Anwendung die gleiche Technik zu Suche verwendet wird, die in den vorherigen Kapiteln zum einen abstrakt vorgestellt wurde, und die im Kapitel 5 zur Suche in großen Audiodatenbanken verwendet wurde, wird in diesem Kapitel der Schwerpunkt auf die zeit- und platzeffiziente Implementierung der $SE(3)$-Operationen gelegt. Dabei werden auch numerische Gesichtspunkte eine Rolle spielen.

Zunächst wird im folgenden Abschnitt die „Protein Data Bank“ [webc, BWF$^+$00] und die dort öffentlich zugänglichen Daten vorgestellt, sowie die verfügbaren Suchmöglichkeiten diskutiert. Die hier vorgestellte Anwendung versteht sich als Machbarkeitsstudie, die einen ersten Schritt in Richtung der inhaltsbasierten Suche in Proteinmolekülen darstellt. Es folgt die Beschreibung der Gruppe $SE(3)$ der euklidschen Bewegungen im dreidimensionalen Raum, welche in dieser Anwendung auf den Moleküldaten in natürlicher Weise operiert. Danach wird auf die wichtige Frage nach der Art der Modellierung der Daten in der Anwendung eingegangen, um in Anschluss daran interessante Trefferbegriffe zu definieren. Der letzte Abschnitt dieses Kapitels befasst sich mit den Details der Implementierung des Systems zur inhaltsbasierten Suche in 3D-Strukturdaten.

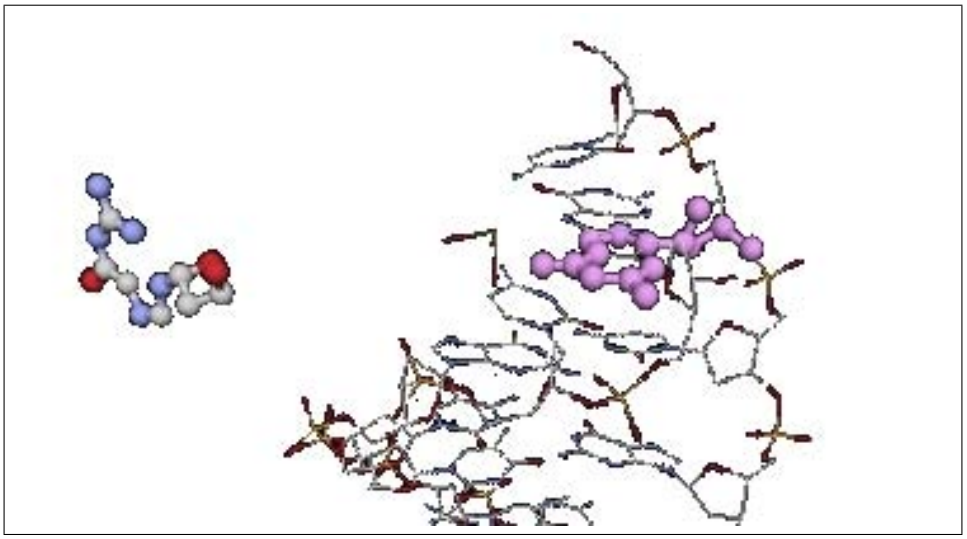


Abbildung 6.1: Die Abbildung verdeutlicht die Zielsetzung der vorgestellten Anwendung: Ein Ausschnitt aus einem Protein wurde in einem Protein in der Dantenbank lokalisiert. Dabei stellt der hervorgehobene Treffer eine rotierte und verschobene Version der Anfrage dar.

6.2 Die Protein Data Bank

Die „Protein Data Bank“ (PDB) [webc, BWF+00] ist die Ressource von dreidimensionalen Strukturdaten von großen biologischen Molekülen. Dies sind zum einen Proteine, welche sich aus langen Ketten von Aminosäuren ergeben. Zum anderen enthält die Datenbank DNA-Sequenzen. Sogenannte Sequenzierverfahren liefern als Ergebnis die Reihenfolge der Aminosäuren, aus denen z.B. ein Polypeptid (Protein) besteht. Jedoch genügt es nicht die Reihenfolge der einzelnen Aminosäuren eines Proteins zu kennen, um dessen Eigenschaften zu verstehen. Aufgrund der sehr komplexen Wechselwirkungen der Atome eines Proteins untereinander, falten sich Aminosäuren im dreidimensionalen Raum zu komplexen Atomkonstellationen. Erst die jedem Protein ureigene Art der Faltung ist entscheidend für die Funktion des Proteins. Z.B. können durch geeignete Faltungen reaktive Zentren an einem Protein entstehen, welche bei Reaktionen in Pflanzen und Tieren benötigt werden. Auf diese Weise unterstützen (katalysieren) bestimmte Proteine Reaktionen, die ohne die Anwesenheit der jeweiligen Proteine nicht ablaufen würden.
Die Vorhersage der Struktur eines Proteins aus einer Sequenz von Aminosäuren ist Gegenstand aktueller Forschung. Aufgrund der notwendigen Rechenleistungen gibt es unterschiedliche Projekte, die ähnlich zu dem bekannten SETI@home-Projekt auf die Vernetzung von vielen tausend privaten Computern über das Internet setzten [webd, webe], um die notwendige Rechenkapazitäten zu erhalten. Neben der Vorhersage gibt es chemisch-physikalische Verfahren, um die Struktur eines Proteins aufzuklären. Dies bedeutet, dass durch Analysemethoden versucht wird, Art und Position eines jeden Atoms in einem Protein in allen drei Koordinaten zu bestimmen. Ein gängiges Verfahren ist die Röntgenstrukturanalyse, welche anhand der Beugung von Röntgenstrahlen eine Karte der Elektronendichte ergibt, aufgrund derer die Positionen der Atome berechnet werden können. Allerdings muss hierzu das Protein als Kristall vorliegen, was an sich ein nicht immer lösbares Problem darstellt. Selbst wenn dies möglich ist, bleibt zu bedenken, dass man nur eine Momentaufnahme des Moleküls analysieren kann. Weiterhin muss die kristalline Struktur des Proteins nicht zwingend der physiologischen Faltung des

Proteins entsprechen. Für eine Vielzahl von Proteinen ist bis heute keine Kristallisation gelungen.
Eine andere genutzte Technik ist die „Kernspinresonanz-Spektroskopie“ (nuclear magnetic resonance spectroscopy, NMR). Diese erlaubt die Strukturaufklärung von Proteinen in einem Lösungsmittel, also in einer beinahen physiologischen Umgebung. Bei diesem Verfahren werden Wechselwirkungen von Elektronen mit einem Magnetfeld gemessen, wenn diese durch einen elektromagnetischen Impuls angeregt werden. Die so gewonnene Änderung des Magnetfelds über die Zeit kann mittels Fourier-Analyse in den Frequenzraum transformiert werden, wo sich deutliche „Peaks“ zeigen. Anhand dieser Peaks kann auf die Elektronenverteilung und damit auf Atome und deren Bindungen geschlossen werden. Es liegt auf der Hand, dass diese Analyse bei sehr großen Molekülen äußerst schwierig ist.

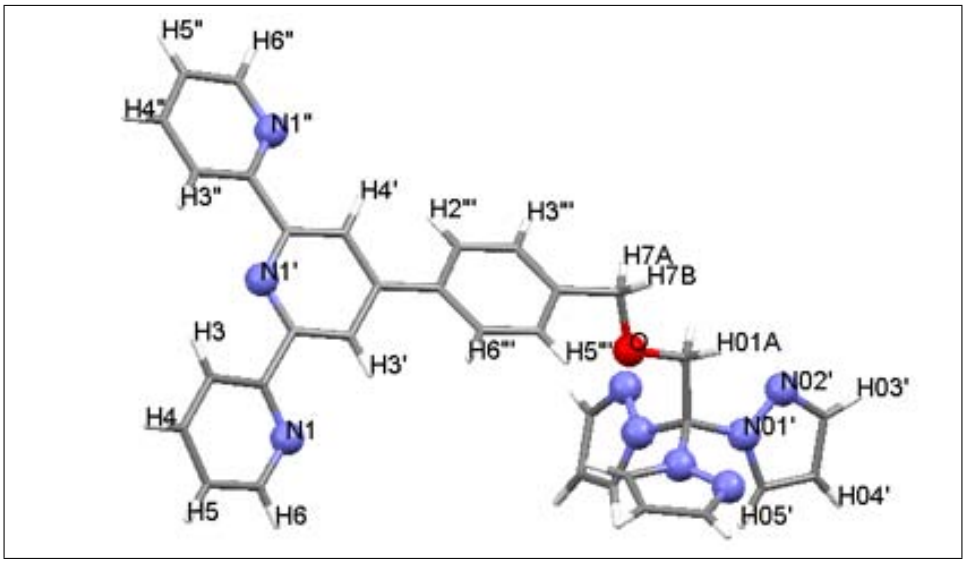


Abbildung 6.2: Beispiel eines Moleküls, entnommen aus [ZH05].

Neben den Atomkoordinaten enthält die PDB auch Metainformationen zu jedem Molekül, so auch meist die Sequenz, eine Beschreibung der Funktion, Bedingungen zum Zeitpunkt der Strukturanalyse etc. Momentan enthält die Datenbank mehr als 35.000 biologische Makromoleküle. Das Webportal [webc] bietet mehrere Suchfunktionalitäten an, auf die in Abschnitt 6.3 näher eingegangen wird.

6.2.1 Inhalt und Dateiformat

Die Ursprünge der PDB liegen in den frühen 1970 er Jahren. Aufgrund der damals vorhandenen Rechenkapazitäten wurde das PDB Dateiformat als Textdatei konzipiert, wobei die Daten in einer Tabelle mit festen Spaltenbreiten abgelegt wurden. Über die Jahre haben unterschiedliche Forschergruppen das Dateiformat für ihre spezifischen Anforderungen mit speziellen Kodierungen versehen, die die Bearbeitung solcher Dokumente erschwert. So haben manche Forschergruppen Doppelbindungen dadurch kodiert, dass sie zwei gleiche Einträge in das Dokument geschrieben haben. In [Say01] werden einige der zu beachtenden Ausnahmen erläutert. Allerdings stehen seit einiger Zeit die PDB-Informationen auch in Form von XML-Dateien zur Verfügung. Aufgrund der einfacheren Bearbeitung von XML-Dateien mit entsprechenden verfügbaren Parsern, wurde dieses Dateiformat als Datenbasis für die in dieser Arbeit vorgestellte Applikation gewählt. Nicht verschwiegen werden sollte dabei einer der Hauptnachteile von XML-Dokumenten: Die unkomprimierten XML-Dateien belegen meist

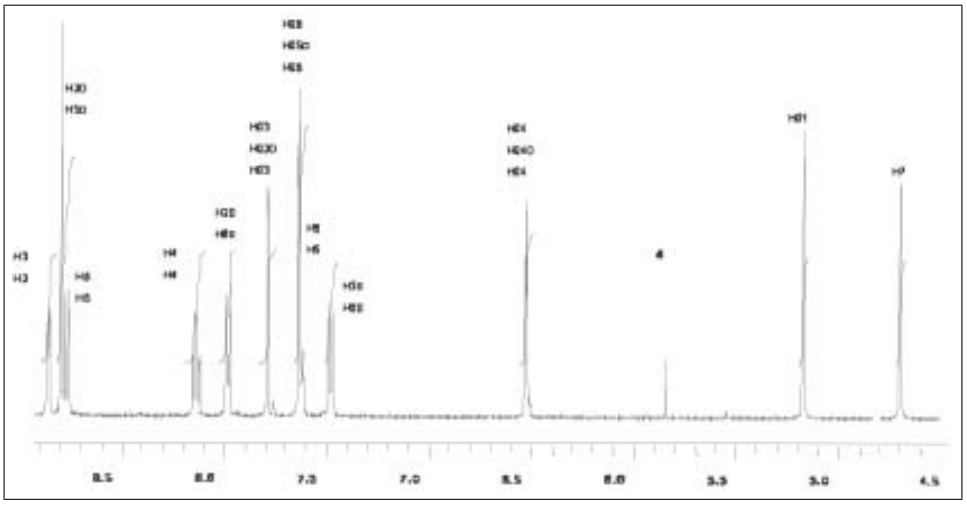


Abbildung 6.3: Ausschnitt aus dem NMR-Spektrum des in 6.2 gezeigten Moleküls, wobei die x-Achse der Frequenzachse entspricht. Die Peaks im Spektrum sind einzelnen Atomen im Molekül zugeordnet worden, was die Grundlage für die Darstellung in 6.2 ist [ZH05].

mehrere Megabyte Speicherplatz, wohingegen eine komprimierte Version nur wenige hundert Kilobyte Speicherplatz belegt. Um die Bearbeitung von mehr als 35.000 Dokumenten zu beschleunigen, wurden die für die Applikation wichtigen Informationen aus den XML-Dateien extrahiert und in eine Binärdatei geschrieben, welche eine schnellere Bearbeitung der Daten ermöglicht. Im wesentlichen wurden bei der Konversion die Atomkoordinaten und die vierstellige PDB-ID übernommen. Angaben zu Autoren, Beschreibungen des Proteins etc. können im Rahmen der Metadatensuche, wie sie in Abschnitt 3.6 zuvor beschrieben worden ist, bei Bedarf berücksichtigt werden.

6.2.1.1 Ausgewählte Inhalte von PDBML-Dateien

Wie bereits kurz erwähnt, ist das ursprüngliche PDB-Dateiformat an seine Grenzen gestoßen, da z.B. die Atomnummern nur maximal 5-stellig sein können, heute jedoch Moleküle mit deutlich mehr Atomen in der Praxis eine Rolle spielen. Aufgrund der starren Struktur des Dateiformats wurde ein XML-Schema [WIN+05, MEPR] entworfen. In diesem Abschnitt werden einige Teile dieses XML-Schemas kurz vorgestellt, um z.B. auf die Kombination von inhaltsbasierter Suche in den Daten der PDB und Suche in Metadaten hinzuweisen. Ausgehend von den XML-Dateien ist die Überführung von Metadaten wie Autorennamen etc. in ein geeignetes RDBMS sehr leicht möglich.

PDBx:audit_authorCategory: Auf diese Weise werden die Autoren einer PDBML-Datei gespeichert. Diese auszulesen und in einem RDBMS abzulegen wäre leicht möglich.

```
<PDBx:audit_authorCategory>
      <PDBx:audit_author name="Brzozowski, A.M."></PDBx:audit_author>
      <PDBx:audit_author name="Lawson, D.M."></PDBx:audit_author>
      <PDBx:audit_author name="Turkenburg, J.P."></PDBx:audit_author>
      <PDBx:audit_author name="Bisgaard-Frantzen, H."></PDBx:audit_author>
      <PDBx:audit_author name="Svendsen, A."></PDBx:audit_author>
      <PDBx:audit_author name="Borchert, T.V."></PDBx:audit_author>
      <PDBx:audit_author name="Dauter, Z."></PDBx:audit_author>
      <PDBx:audit_author name="Wilson, K.S."></PDBx:audit_author>
```

```
      <PDBx:audit_author name="Davies, G.J."></PDBx:audit_author>
   </PDBx:audit_authorCategory>
```

PDBx:database_PDB_remark: Zu einem PDBML-Dokument werden häufig Anmerkungen gespeichert, die z.B. die genaueren Umstände beschreiben, die bei der Strukturaufklärung geherrscht haben. Durch die „remark_id“ ist es möglich auf weitere Bemerkungen zu verweisen.

```
<PDBx:database_PDB_remarkCategory>
      <PDBx:database_PDB_remark id="2">
          <PDBx:text>
             RESOLUTION. 1.93 ANGSTROMS.
          </PDBx:text>
      </PDBx:database_PDB_remark>
      ...
```

PDBx:atom_site: Dieser Teil eines PDBML-Dokuments beschreibt ein Atom Nummer 42 und dessen Koordinaten im Raum, den Elementtyp (hier Schwefel, S), ein „Label“ zum Atom und dass es zum ersten Modell in der Datei gehört.

```
<PDBx:atom_site id="42">
         <PDBx:group_PDB>ATOM</PDBx:group_PDB>
         <PDBx:type_symbol>S</PDBx:type_symbol>
         <PDBx:label_atom_id>SD</PDBx:label_atom_id>
         <PDBx:label_alt_id>B</PDBx:label_alt_id>
         <PDBx:label_comp_id>MET</PDBx:label_comp_id>
         <PDBx:label_asym_id>A</PDBx:label_asym_id>
         <PDBx:label_entity_id>1</PDBx:label_entity_id>
         <PDBx:label_seq_id>6</PDBx:label_seq_id>
         <PDBx:Cartn_x>12.921</PDBx:Cartn_x>
         <PDBx:Cartn_y>34.151</PDBx:Cartn_y>
         <PDBx:Cartn_z>19.897</PDBx:Cartn_z>
         <PDBx:occupancy>0.50</PDBx:occupancy>
         <PDBx:pdbx_PDB_model_num>1</PDBx:pdbx_PDB_model_num>
      </PDBx:atom_site>
```

PDBx:entity_poly_seqCategory: In dieser Kategorie wird die Sequenz eines Proteins, d.h. die Reihenfolge der Aminosäuren, beschrieben. Das Attribut „num“ wird dabei in einem `PDBx:atom_site` Datensatz referenziert (`PDBx:label_seq_id`), um auf diese Weise einer Vielzahl von Atomen einzelne Aminosäuren zuzuordnen. Die Sequenzinformation wird meist zusätzlich auch in Form einer Zeichenkette angegeben, wie sie in der Einleitung zu diesem Kapitel beschrieben wurde. In weiteren Datensätzen werden den Aminosäuren Ketten zugeordnet, welche bei der Beschreibung eines Atoms durch das Attribut `PDBx:label_asym_id` angegeben werden.

```
<PDBx:entity_poly_seqCategory>
      <PDBx:entity_poly_seq entity_id="1" num="1" mon_id="VAL">
      </PDBx:entity_poly_seq>
      <PDBx:entity_poly_seq entity_id="1" num="2" mon_id="ASN">
      </PDBx:entity_poly_seq>
      <PDBx:entity_poly_seq entity_id="1" num="3" mon_id="GLY">
      </PDBx:entity_poly_seq>
      ...
      <PDBx:entity_poly_seq entity_id="1" num="6" mon_id="MET">
      </PDBx:entity_poly_seq>
      ...
```

Neben den hier vorgestellten Auszügen aus einer PDBML-Datei werden noch viel mehr Informationen gespeichert. Allerdings ist deren Verwendung in dem in dieser Arbeit entworfenen Prototypen nicht von Belang, weshalb auch auf Umrechnungen der Atomkoordinaten zwischen verschiedenen

Experimenten etc. verzichtet wird, da diese auf die hier diskutierte Anwendung keinen direkten Einfluss haben. Vielmehr wurden lediglich die Koordinaten von Atomen und deren Elementnamen aus Einträgen vom Typ `PDBx:atom_site_id` verwendet. Das Wissen zu welcher Aminosäure ein Atom gehört und zu welcher Kette wiederum die Aminosäure gezählt wird, wurde bei der hier vorgestellten Applikation nicht benutzt. Allerdings ließe sich diese Information sehr leicht einem Repräsentantensystem hinzufügen.

6.3 Suchmöglichkeiten in der PDB

Über die Portal-Webseite [webc] werden unterschiedliche Suchmöglichkeiten dem Benutzer geboten, die sich jedoch alle im wesentlichen auf den Metadaten zu den Molekülen beziehen. So können Moleküle nach deren PDB-IDs gesucht werden, Name eines Moleküls, Datum der Eintragung, Verfahren der Sturkturaufklärung etc. Statistische Eigenschaften der Sekundärstruktur eines Proteins sind ebenfalls in einer Abfrage zu formulieren, d.h. die Anzahl von Windungen, Anteil von α-Helices oder der Anteil von β-Faltblattstrukturen. Die Suche nach Molekülen mit gewissen Liganden ist ebenso möglich. Weitere Suchbereiche umfassen Informationen wie Autoren, Journale, Funktionen von Proteinen, die Lebewesen, bei denen das Protein gefunden wurde etc.
Neben der Information über die Anordnung der Atome eines Moleküls im dreidimensionalen Raum, enthält ein Eintrag der PDB in der Regel auch die *Proteinsequenz*

$$\ldots \text{TFGSGEADCGLRPLFEKKSLEDKTERELLESYIDGR} \ldots,$$

die einem Protein zugrunde liegt. Diese Sequenz beschreibt die Reihenfolge der Aminosäuren in dem jeweiligen Protein:

$$\ldots, \text{Threonin, Phenylalanin, Glycin, Serin, Glycin, } \ldots$$

Solche Sequenzen stellen einen möglichen Zugang zur Suche nach ähnlichen Proteinen in einer Datenbank dar. In [Pea95] werden unterschiedliche Algorithmen und von denen genutzte Bewertungsmatrizen miteinander verglichen, die zum Vergleich solcher biologischer Sequenzen benutzt werden (z.B. [AGM$^+$90, PL88, SW88]). Demnach werden neu gewonnene Sequenzen routinemäßig mit einem vorhandenen Sequenzdatenbestand verglichen, da ähnliche Sequenzen häufig Rückschlüsse auf ähnliche Struktur bzw. Funktion der aktuell bestimmten Sequenz ermöglichen können. Eine Sequenz von Aminosäuren dient als Eingabe und als Suchergebnis ausgegeben werden „entfernt verwandte“ Sequenzen. Dabei hat sich herausgestellt, dass für unterschiedliche Klassen von Proteinen unterschiedliche Bewertungsmatrizen zu besseren Ergebnissen führen.

6.4 Modellierung und Notation

Im folgenden geht es idealisiert um starre Konstellationen von Atomen. Dabei spielt die Lage dieser Konstellation im umgebenden dreidimensionalen Raum keine Rolle. Daher modelliert die Gruppe $SE(3)$ der speziellen Euklidschen Bewegungen genau die gewünschte Lageinvarianz. Im folgenden werden einige Notationen eingeführt, um das Problem der Suche in Proteinstrukturdaten mathematisch zu beschreiben.

6.4.1 Die Dokumentenmenge

Die Menge der Atomnamen wird mit $\mathcal{A}$ bezeichnet. Sie enthält alle Paare bestehend aus Elementnummer und -symbol, wie sie im Periodensystem der Elemente (PSE, z.B. [web05]) Verwendung finden:

$$\mathcal{A} := \{(1,\text{H}), (2,\text{He}) , (3,\text{Li}), \dots\}.$$

Wie bereits zuvor angemerkt, enthält ein PDB-Dokument weitere Informationen zu einem Atom. Dies sind z.B. eine genauere Bezeichnung des Atoms im Kontext der jeweiligen Aminosäure (z.B. C-α-Atom). Außerdem ist der jeweilige Aminosäurerest (engl. residue), zu dem das jeweilige Atom gehört, dort hinterlegt. Diese Informationen können ebenfalls bei Bedarf zur inhaltsbasierten Suche benutzt werden. Bei der hier vorgestellten Applikation wird jedoch nur die Position eines Atoms im Raum sowie dessen Art benutzt. An geeigneter Stelle wird auf Möglichkeiten der Integration von weiteren Informationen hingewiesen. Die Modellierung von PDB-Dokumenten basiert daher auf der Menge

$$M := \mathbb{R}^3 \times \mathcal{A}.$$

M besteht also aus Paaren. π_i bezeichne die Projektion auf die i-te Komponente.

De nition 6.1 *Ein* Molekül D *ist eine endliche Teilmenge von* M*, die (abgesehen von weiteren physikalisch-chemischen Bedingungen)* $|\pi_1(D)| = |D|$ *erfüllt.*

Die Dokumente der PDB-Datenbank führen zu einer Dokumentenkollektion $\mathcal{D} := (D_1, D_2, \dots, D_N)$, $D_i \subset M$. Der Index i wird als *Dokumenten-ID* bezeichnet. Die Anzahl von Atomen in einem Dokument D_i schwankt zwischen wenigen Hundert und mehreren Zehntausend.

6.4.2 Die Gruppe der euklidschen Bewegungen $SE(3)$

Die Abbildung 6.1 zeigt eine Anfrage und einen Treffer im Kontext einer DNA-Sequenz. Der Treffer ist gegenüber der Anfrage verschoben und rotiert.
Die Gruppe $SE(3)$ ist ein semi-direktes Produkt des Translationennormalteilers $(\mathbb{R}^3, +)$ und der speziellen orthogonalen Gruppe $SO(3)$:

$$G = SE(3) = SO(3) \ltimes \mathbb{R}^3.$$

Sind (R, T) und (R', T') zwei Elemente aus $SE(3)$ mit Rotations- bzw. Translationsanteilen R, R' bzw. T, T', so ist deren Produkt gegeben durch $(R, T)(R', T') = (RR', RT' + T)$. Dies wird deutlich durch die Realisation von $SE(3)$ als Matrixgruppe, indem man dem Paar (R, T) die 4×4-Matrix

$$\begin{array}{c|c} R & T \\ \hline 0 & 1 \end{array} = \begin{array}{ccc|c} r_1 & r_2 & r_3 & t_1 \\ r_4 & r_5 & r_6 & t_2 \\ r_7 & r_8 & r_9 & t_3 \\ \hline 0 & 0 & 0 & 1 \end{array} \qquad SE(3), \tag{6.1}$$

zuordnet. Diese Zuordnung ist ein Gruppenisomorphismus; wir identifizieren im folgenden $SE(3)$ oft mit dieser Matrixgruppe.

Satz 6.1 *Für* $g = (R, T) \quad SE(3)$ *gilt:*

$$g^{-1} = \begin{array}{c|c} R^\top & -R^\top T \\ \hline 0 & 1 \end{array} . \tag{6.2}$$

Beweis: Eine Matrix $R \quad SO(3)$ ist eine *orthonormale* Matrix, d.h. die inverse Matrix lässt sich durch Transposition bestimmen: $R^{-1} = R^{\top}$. Eine leichte Rechnung zusammen mit der Orthonormalität von R zeigt, dass die Gleichung (6.2) korrekt ist. □

Somit lässt sich ein Gruppenelement $g \quad SE(3)$ als ein 12-stelliger Vektor über $\mathbb{R}$ auffassen. Dies ist im Hinblick auf die Implementierung von Interesse, da die Anzahl der zu speichernden Werte pro Gruppenelement direkten Einfluss auf die zu speichernde und zu verarbeitende Datenmenge hat.

6.4.3 k-Fehlstellentreffer

Rufen wir uns an dieser Stelle die Definition der Treffermenge der k-Fehlstellentreffer ins Gedächtnis:

$$G^k_{\mathcal{D}}(Q) := \{(g, i) \quad G \times [1 : N] \mid |(gQ) \quad D_i| \leq k\}. \tag{6.3}$$

Die Gleichung beschreibt die Menge der sog. $(G, \mathcal{D}, k)$-*Treffer*, d.h. die Menge aller Paare (g, i), wobei die elementweise Operation von g auf einem Anfragedokument Q dazu führt, dass $gQ \subset D_i$ bis auf höchstens k Ausnahmen gilt.
Die Berechnung der Menge $G^k_{\mathcal{D}}(Q)$ bezüglich eines Anfragedokumentes Q erfolgt durch folgende Schnittmengenberechnung:

$$G^k_{\mathcal{D}}(Q) = \bigcap_{q \in Q}^{k} G_{\mathcal{D}}(r_q) g_q^{-1}.$$

Die Elemente der G-invertierten Liste zu einem Repräsentanten r_q werden mit dem inversen Gruppenelement g_q^{-1} multipliziert und auf diese Weise an das jeweilige Element des Dokumentes Q angepasst. Damit ein Gruppenelement in der Treffermenge $G^k_{\mathcal{D}}(Q)$ enthalten ist, muss es in mindestens $|Q| - k$ vielen, durch g_q^{-1} veränderten G-invertierten Listen vorhanden sein.

6.4.3.1 Berechnungsaufwand

Der Berechnungsaufwand für die Berechnung der obigen Schnittmenge durch den vorgestellten Algorithmus basierend auf Indikatorfunktionen, bestimmt sich aus den Längen der einzelnen G-invertierten Listen, die zur Berechnung betrachtet werden müssen:

$$\text{Anzahl Gruppenoperationen} = \sum_{q \in Q} |G_{\mathcal{D}}(r_q)|.$$

Zusätzlich müssen noch $|Q|$ viele Invertierungen von Gruppenelementen berechnet werden, mit denen die Elemente der G-invertierten Listen multipliziert werden, jedoch sollten diese im Gegensatz zu den zu berechnenden Gruppenoperationen nur wenig Einfluss auf die Laufzeit eines Suchalgorithmus haben. Der Aufwand zur Berechnung eines Produktes gh, $g, h \quad SE(3)$, ist aufgrund der Besonderheit der letzten Zeile der Matrizen etwas geringer als der Aufwand zur Berechnung eines 4×4-Matrixprodukts. Im einzelnen sind für eine Gruppenoperation (vgl. (6.1)) *36 Multiplikationen* und *45 Additionen* zu berechnen.
Neben dem Aufwand für die Berechnung der Gruppenoperationen und der Inversion der Gruppenelemente in der Anfrage, ist die Verwaltung der Ergebnisse der Gruppenoperation von entscheidender Bedeutung. Wie im Kapitel 3 gezeigt wurde, ist die explizite Berechnung von Schnittmengen in der Praxis bei großen Datenmengen ineffizient, obwohl die Berechnung einer Schnittmenge von zwei sortierten Folgen eine Komplexität von $O(n)$ bei gleichzeitigem linearen Durchlaufen beider Listen

aufweist. Soll eine G-invertierte Liste der Länge n mit einer Liste der Länge m, $m \ll n$ geschnitten werden, bietet sich auch die Verwendung der Binärsuche mit einem Aufwand von $O(n \log m)$ an. Allerdings muss hierzu die verwendete Datenstruktur der G-invertierten Liste den wahlfreien Zugriff auf die Elemente erlauben. Wird die Datenstruktur eines Vektors zur Speicherung einer G-invertierten Liste benutzt, so sind beide Arten der Schnittmengenberechnung möglich. Ist jedoch die Anzahl der erlaubten Fehlstellen $k > 0$, so sind bei der Berechnung von $G^k_{\mathcal{D}}(Q)$ nicht nur Schnittmengen zu berechnen, sondern auch *Vereinigungen* von G-invertierten Listen. Das Einfügen eines Elements in einen Vektor ist eine vergleichsweise aufwändige Operation, da ggf. große Teile eines Vektors bei jeder Einfügeoperation kopiert werden müssen, um an geeigneter Stelle Platz für das einzufügende Element zu schaffen. Diese Problematik ist sicherlich der Hauptgrund für die in [Wag03] gemessenen Zeiten pro Anfrage.

6.4.4 Simulation von $SE(3)$ mittels Quaternionen

Mit Hilfe von Quaternionen werden wir auf der 3-Sphäre $\mathbb{S}^3$ eine Gruppenstruktur einführen. Ein surjektiver Gruppenmorphismus $\rho\colon \mathbb{S}^3 \quad SO(3)$ mit Kern$\rho = \{1, -1\}$ ermöglicht es, die Gruppe $SE(3) = SO(3) \ltimes \mathbb{R}^3$ über das semidirekte Produkt $\mathbb{S}^3 \ltimes \mathbb{R}^3$, definiert durch

$$(q, t)(q', t') := (qq', \rho(q)(t') + t),$$

zu simulieren. Zunächst erinnern wir kurz an Quaternionen, geben dann den Morphismus ρ an und diskutieren schließlich Aufwandsfragen.
Im Jahre 1843 entdeckte Hamilton [Ham47] den Schiefkörper[1] $\mathbb{H}$, der ein 4-dimensionaler reeller Vektorraum mit Basis $1, i, j, k$ ist. Die multiplikative Struktur wird beschrieben durch die Relationen:

$$i^2 = j^2 = k^2 = ijk = -1.$$

Daraus folgt sofort

$$ij = k = -ji, \quad jk = i = -kj \quad \text{und} \quad ki = j = -ik.$$

Durch $q \ _{\mathbb{H}} := \sqrt{s^2 + x^2 + y^2 + z^2}$ für $q = s + xi + yj + zk \quad \mathbb{H}$ wird $\mathbb{H}$ zum normierten Vektorraum. Im Fall $s = 0$ heißt q reines Quaternion. Mit der sog. *Konjugierten* $\overline{q} = s - xi - yj - zk \quad \mathbb{H}$ von q ergibt sich $q \ _{\mathbb{H}} = \quad \overline{q}\overline{q} = \quad \overline{\overline{q}q}$. Für ein $q = 0$ gilt $q^{-1} = \overline{q} \ (q\overline{q})$. Quaternionen mit $q \ _{\mathbb{H}} = 1$ werden *Einheitsquaternionen* genannt. Für derartige Quaternionen gilt $q^{-1} = \overline{q}$, da $1 \ (q\overline{q}) = 1 \quad q \ ^2_{\mathbb{H}} = 1$. In diesen Fällen ist die Berechnung des inversen Quaternions sehr einfach durch Vorzeichenumkehr möglich. Die Quaternionenmultiplikation ist assoziativ, jedoch nicht kommutativ. Zur Vereinfachung der Notation wird ein Quaternion $q \quad \mathbb{H}$ geschrieben als $(x \quad y \quad z \quad s)$. (Man beachte die Reihenfolge der Koeffizienten. Reine Quaternionen sind also gekennzeichnet durch das Verschwinden der letzten Komponente, also $s = 0$.) Die Einheitsquaternionen bilden bzgl. der Multiplikation in $\mathbb{H}$ eine Gruppe, die mit $\mathbb{S}^3$ bezeichnet wird.
Sind $p = (x_1 \quad y_1 \quad z_1 \quad s_1) = x_1 i + y_1 j + z_1 k + s_1$ und $q = (x_2 \quad y_2 \quad z_2 \quad s_2) = x_2 i + y_2 j + z_2 k + s_2$ Quaternionen, so ist ihr Produkt gegeben durch

$$\begin{aligned} pq = \quad & (s_1 x_2 + x_1 s_2 + y_1 z_2 - z_1 y_2)\, i \\ + \quad & (s_1 y_2 - x_1 z_2 + y_1 s_2 + z_1 x_2)\, j \\ + \quad & (s_1 z_2 + x_1 y_2 - y_1 x_2 + z_1 s_2)\, k \\ + \quad & (s_1 s_2 - x_1 x_2 - y_1 y_2 - z_1 z_2). \end{aligned}$$

[1] Ein *Schiefkörper* ist ein assoziativer Ring mit 1, der bis auf die Kommutativität der Multiplikation dieselben Gesetze wie ein Körper erfüllt. Insbesondere sind alle von Null verschiedenen Elemente invertierbar.

Zur Multiplikation zweier Quaternionen werden bei direkter Umsetzung obiger Formel *16 Multiplikationen* und *12 Additionen* benötigt. Im Vergleich dazu benötigt die Multiplikation zweier 3×3-Rotationsmatrizen 27 Multiplikationen und 18 Additionen. Besonders die geringere Anzahl von Multiplikationen ist oftmals entscheidend für die Verwendung von Quaternionen, da sich der Rechenaufwand bei der Verkettung von Rotationen reduziert und sich gleichzeitig die numerische Stabilität erhöht. Dieser Vorteil muss allerdings später bei Berücksichtigung der Translationskomponenten zum Teil relativiert werden.
Für ein Einheitsquaternion $q \quad \mathbb{S}^3$ und $p \quad \mathbb{H}$ setzen wir

$$\rho(q)(p) := qpq^{-1} = qp\overline{q}.$$

Eine leichte Rechnung zeigt, dass jedes $\rho(q) \quad O(4)$ eine orthogonale Abbildung auf $\mathbb{H} \quad \mathbb{R}^4$ ist. Darüberhinaus definiert $q \quad \rho(q)$ einen Gruppenmorphismus $\mathbb{S}^3 \quad O(4)$, denn für $q_1, q_2 \quad \mathbb{S}^3$ und $p \quad \mathbb{H}$ gilt

$$\rho(q_1q_2)(p) = q_1q_2p\overline{q_1q_2} = q_1(q_2p\overline{q_2})\overline{q_1} = \rho(q_1)\rho(q_2)(p).$$

Wir behaupten weiter, dass der Kern von ρ nur aus den Elementen 1 und -1 besteht: dass diese beiden Elemente zum Kern gehören ist trivial. Sei umgekehrt $q = (x \quad y \quad z \quad s)$ aus dem Kern. Dann liefert $\rho(q)(i) = i$ sofort $(x \quad y \quad z \quad s)(i)(-x \quad -y \quad -z \quad s)$, woraus sich $s^2 + x^2 - y^2 - z^2 = 1$ ergibt. Zusammen mit $s^2 + x^2 + y^2 + z^2 = 1$ folgt $y = z = 0$. Aus $\rho(q)(j) = j$ folgt analog $x = 0$. Somit ist $s^2 = 1$, also $q = \pm 1$. Da die Elemente aus $\mathbb{R}$ mit allen Elementen aus $\mathbb{H}$ vertauschen folgt $\rho(q)(s) = s$, für alle $s \quad \mathbb{R} \subset \mathbb{H}$. Da $\rho(q)$ orthogonal ist, ist mit $\mathbb{R}$ auch das orthogonale Komplement, das von i, j, k aufgespannt wird, ebenfalls invariant, d.h. die Einschränkung $\rho(q)$ von jedem $\rho(q)$ auf $\mathbb{R}^3 \equiv \mathbb{R}i + \mathbb{R}j + \mathbb{R}k$ ist eine orthogonale Abbildung aus $O(3)$. Damit ist folgender Satz vorbereitet:

Satz 6.2 *Für $q \quad \mathbb{S}^3$ und einem reinen Quaternion p wird durch $\rho(q)(p) := qp\overline{q}$ ein surjektiver Gruppenhomomorphismus $\rho\colon \mathbb{S}^3 \quad SO(3)$ de niert, dessen Kern aus 1 und -1 besteht.*

Beweis: Siehe [Cur79]. □

Ein eigenständiger Beweis wird weiter unten gegeben.

6.4.4.1 Matrixdarstellung der Rotationsquaternionen

Der Satz 6.2 beschreibt abstrakt einen surjektiven Gruppenhomomorphismus $\rho\colon \mathbb{S}^3 \quad SO(3)$. Der folgende Satz konkretisiert ρ.

Satz 6.3 *Für $q = (x \quad y \quad z \quad s) \quad \mathbb{S}^3$ ist*

$$\rho(q) = \begin{matrix} s^2 + x^2 - y^2 - z^2 & 2(xy + sz) & 2(xz - sy) \\ 2(xy - sz) & s^2 - x^2 + y^2 - z^2 & 2(yz + sx) \\ 2(xz + sy) & 2(yz - sx) & s^2 - x^2 - y^2 + z^2 \end{matrix}.$$

Ferner liegt $\rho(q)$ in $SO(3)$.

Beweis:
Gegeben seien zwei Einheitsquaternionen $q_1 = (x_1 \quad y_1 \quad z_1 \quad w_1)$ und $q_2 = (x_2 \quad y_2 \quad z_2 \quad w_2)$.

Die Linksmultiplikation mit einem festen q_1 ist eine $\mathbb{R}$-lineare Abbildung $L_{q_1} : \mathbb{H} \quad \mathbb{H} \quad \mathbb{R}^4$, die angewandt auf ein beliebiges $q_2 \quad \mathbb{H}$ sich wie folgt als Matrix-Vektor-Multiplikation schreiben lässt:

$$q_1 q_2 = q_2 L_{q_1} = (x_2 \quad y_2 \quad z_2 \quad s_2) \begin{pmatrix} s_1 & z_1 & -y_1 & -x_1 \\ -z_1 & s_1 & x_1 & -y_1 \\ y_1 & -x_1 & s_1 & -z_1 \\ x_1 & y_1 & z_1 & s_1 \end{pmatrix} . \tag{6.4}$$

Analog dazu ergibt sich bei festem $q_2 \quad \mathbb{H}$ für die Rechtsmultiplikation $R_{q_2} : \mathbb{H} \quad \mathbb{H}$ von q_1 mit q_2:

$$q_1 q_2 = q_1 R_{q_2} = (x_1 \quad y_1 \quad z_1 \quad s_1) \begin{pmatrix} s_2 & -z_2 & y_2 & -x_2 \\ z_2 & s_2 & -x_2 & -y_2 \\ -y_2 & x_2 & s_2 & -z_2 \\ x_2 & y_2 & z_2 & s_2 \end{pmatrix} . \tag{6.5}$$

Zusammen mit (6.4) und (6.5) und einem reinen Quaternion P, welches dem zu rotierenden Punkt $p \quad \mathbb{R}^3$ entspricht, gilt die Gleichungskette

$$qP\overline{q} = q(P\overline{q}) = q(PR_{\overline{q}}) = (PR_{\overline{q}})L_q = P(R_{\overline{q}}L_q).$$

Aus $(R_{\overline{q}}L_q)$ ergibt sich die in Satz 6.3 angegebene Matrixdarstellung.
Dass $\rho(q)$ ein Element der Gruppe $SO(3)$ beschreibt ergibt sich aus der Tatsache, dass $\rho(q)$ eben gerade die allgemeine Darstellung einer Rotationsmatrix ergibt, die durch v und ϕ parametrisiert wird. □

Satz 6.4 *Kern*$(\rho) = \{\pm 1\}$. *Insbesondere gilt für ein Rotationsquaternion* $q \quad \mathbb{H}$*:* $\rho(q) = \rho(-q)$*, d.h. die Quaternionen* q *und* $-q$ *beschreiben* das gleiche *Element in* $SO(3)$*. Insgesamt ist* $\mathbb{H}$ *eine zweifache Überlagerung von* $SO(3)$*.*

Beweis: Setzt man in Satz 6.3 $\rho(q) = 1$, so erhält man die Bedingungen $xz = yz = xy = 0$ sowie $x^2 = y^2 = z^2$, was zusammen mit $s^2 + x^2 + y^2 + z^2 = 1$ zu $x = y = z = 0$ und $s^2 = 1$, also $s = \pm 1$ führt. □

Die Berechnung der zu einem Rotationsquaternion q gehörigen Rotationsmatrix $\rho(q)$ benötigt höchstens *16 Multiplikationen* und *15 Additionen*, wobei Zwischenergebnisse der Art xy zwischengespeichert werden. Sechs der 16 Multiplikationen sind Multiplikationen mit Zwei, so dass sich diese Multiplikationen auch durch Additionen ersetzen lassen.

6.4.4.2 Beschreibung von Rotationen mittels Quaternionen

In diesem Abschnitt studieren wir in gewisser Weise die Umkehrung zu ρ. Wir gehen aus von einer Rotation, beschrieben durch eine Drehachse $\mathbb{R}v$, zum normierten Vektor $v = (v_1, v_2, v_3)$ aus $\mathbb{R}^3$, und einen Drehwinkel ϕ. Dies führt zu folgendem Rotationsquaternion:

$$q = (v_1 \sin\frac{\phi}{2} \quad v_2 \sin\frac{\phi}{2} \quad v_3 \sin\frac{\phi}{2} \quad \cos\frac{\phi}{2}). \tag{6.6}$$

Satz 6.5 *Ein auf diese Weise gewonnenes Quaternion* q *liegt in* $\mathbb{S}^3$*. Weiterhin ist* $\rho(q)(v) = v$ *und* $\rho(q)$ *beschreibt eine Drehung um den Winkel* ϕ*. Insbesondere ist damit* ρ*:* $\mathbb{S}^3 \quad SO(3)$ *surjektiv.*

Beweis: Sei $v = (v_1, v_2, v_3)$ ein Vektor der Länge 1, $s = \sin\frac{\phi}{2}$, $c = \cos\frac{\phi}{2}$ und

$$q = (v_1 s \quad v_2 s \quad v_3 s \quad c) \quad \mathbb{H}.$$

Wegen $q_{\mathbb{H}} = \overline{q}\overline{q}$ ist

$$q_{\mathbb{H}}2 = c^2 + (v_1 s)^2 + (v_2 s)^2 + (v_3 s)^2 = c^2 + (v_1^2 + v_2^2 + v_3^2)s^2 = 1,$$

denn v hat die Länge 1 und $\cos^2\frac{\phi}{2} + \sin^2\frac{\phi}{2} = 1$. Die Argumentation lässt sich umkehren.
Zu zeigen ist weiterhin, dass $qP\overline{q} = Mp$ gilt, d.h. das „Sandwichprodukt“ $qP\overline{q}$ entspricht tatsächlich einem Element der $SO(3)$. Dabei ist P das reine Quaternion zum Punkt p $\mathbb{R}^3$ und M $SO(3)$ die aus v und ϕ abgeleitete Rotationsmatrix. Zur Vereinfachung der Notation schreiben wir das Einheitsquaternion q $\mathbb{H}$ als ein Paar bestehend aus Vektor- und Skalarkomponente, also $q = (sv, c)$.

$$\begin{aligned} q(p,0)\overline{q} &= (sv, c)(p, 0)(-sv, c) \\ &= (cp + s(v \times p), -sv \cdot p)(-sv, c) \\ &= (s^2 v \cdot (v \cdot p) + c^2 p + cs(v \times p) - cs(p \times v) - s^2(v \times p) \times v, \\ &\quad -csv \cdot p + csv \cdot p + s^2(v \times p) \cdot v). \end{aligned}$$

Die ersten beiden Terme der Skalarkomponente heben sich auf und zusammen mit den Rechenregeln für das Kreuz- bzw. Skalarprodukt von Vektoren ergibt sich

$$q(p,0)\overline{q} = (2s^2(p \cdot v)v + (c^2 - s^2)p + 2cs(n \times p), 0).$$

In einem letzten Schritt setzen wir $2s^2 = 1 - \cos\phi$, $c^2 - s^2 = \cos\phi$ und $2cs = \sin\phi$. Damit erhalten wir die Behauptung:

$$q(p,0)\overline{q} = (v(v \cdot p)(1 - \cos\phi) + p\cos\phi + (v \times p)\sin\phi, 0).$$

Der imaginäre Teil des Quaternions $qP\overline{q}$ entspricht gerade dem Ergebnis der Multiplikation Mp, d.h. dem Ergebnis der Anwendung der Rotationsmatrix M auf den Punkt p. Demnach beschreibt das „Sandwichprodukt“ $q(p,0)\overline{q}$ das reine Quaternion $(Mp, 0)$, welches wiederum der durch M rotierten Version des Punktes p entspricht.
Dies beweist auch die in Satz 6.2 gemachte Behauptung, $\rho\colon \mathbb{S}^3$ $SO(3)$ sei ein surjektiver Gruppenhomorphismus, dessen Kern aus -1 und 1 besteht. □

Mit Hilfe von Satz 6.5 ist eine einfache Berechnung gegeben, um aus der Kombination von Rotationsachse und einem Drehwinkel ein zugehöriges Quaternion zu bestimmen. Diese Berechnung ist umkehrbar, so dass man aus der Darstellung des Quaternions den Drehwinkel und darauf aufbauend die Rotationsachse schnell bestimmen kann. Letzteres wird z.B. bei der Suche in Moleküldaten benutzt, um die Ähnlichkeit von Quaternionen zu bestimmen.

6.5 Vergleich verschiedener $SE(3)$-Realisierungen

Wie im vorherigen Abschnitt gezeigt, lassen sich die multiplikative Verknüpfung zweier Quaternionen, die jeweils eine 3D-Rotation beschreiben, mit weniger Multiplikationen berechnen, als wenn dazu zwei 3×3-Rotationsmatrizen miteinander multipliziert würden. Auch die kompakte Darstellung einer Rotation als ein Vektor aus $\mathbb{R}^4$ anstelle eines Vektors aus $\mathbb{R}^9$ lässt die Verwendung von Quaternionen im Hinblick auf Speicherplatzfragen sinnvoll erscheinen.

Nun besitzt ein beliebiges Element g aus $SE(3)$ neben einem Rotationsanteil $R \quad SO(3)$ auch einen Translationsanteil $T \quad \mathbb{R}^3$. Somit ist g durch das Paar $g = (R, T)$ spezifiziert. Wie wir gesehen haben ist die Gruppe der Einheitsquaternionen eine zweifache Überlagerung der $SO(3)$. Also kann man insbesondere die Multiplikation in $SO(3)$ durch Einheitsquaternionen mit den oben genannten Vorteilen simulieren. In diesem Abschnitt geht es darum, die $SE(3)$ mittels Quaternionen zu realisieren und den resultierenden Rechen- und Speicheraufwand für Multiplikation und Inversion zu vergleichen mit dem Aufwand, der sich aus der direkten Umsetzung der Formeln

$$g^{-1} = (R, T)^{-1} = (R^\top, -R^\top T) \quad \text{und} \quad (R, T)(R', T') = (RR', RT' + T)$$

ergibt.
Zur Invertierung mittels $(R, T)^{-1} = (R^\top, -R^\top T)$ werden neun Multiplikationen und sechs Additionen benötigt, die sich aus der Multiplikation $R^\top T$ ergeben. Zusätzlich sind bei der Inversion der Rotationsmatrix R sechs Einträge der Matrix zu vertauschen. Die Multiplikation zweier Gruppenelemente benötigt 36 Multiplikationen und 27 Additionen.
Als nächstes interessiert die Frage, wie das inverse Element eines Gruppenelements von $\mathbb{S}^3 \ltimes \mathbb{R}^3$ berechnet werden kann. Sei dazu $q = (x \quad y \quad z \quad s)$ ein Einheitsquaternion, welches eine Rotation um eine Achse und Winkel im $\mathbb{R}^3$ beschreibt. Wie zuvor angesprochen, bietet es sich an, die Rotation des Translationsvektors t *nicht* durch ein „Sandwichprodukt“ zu berechnen, sondern zunächst das Quaternion q in die korrespondierende Rotationsmatrix $\rho(q)$ gemäß Satz 6.3 zu überführen. Die so gewonnene Matrix wird nun auf einen Translationsvektor $t = (a, b, c)^\top \quad \mathbb{R}^3$ angewendet. Es gilt $-\rho(q)^\top t = (a', b', c')^\top$ mit

$$\begin{aligned} a' &= a(s^2 + x^2 - y^2 - z^2) + 2b(xy + sz) + 2c(xz - sy) \\ b' &= 2a(xy - sz) + b(s^2 - x^2 - y^2 - z^2) + 2c(yz + sx) \\ c' &= 2a(xz + sy) + 2b(yz - sx) + c(s^2 - x^2 - y^2 + z^2). \end{aligned}$$

6.5.1 Aufwandsanalyse

Abschließend werden der Speicherplatzbedarf und die benötigten Rechenschritte für beide Realisierungen der $SE(3)$ gegenübergestellt. Dabei spielen die Aufwände für die multiplikative Verknüpfung zweier Elemente der $SE(3)$ sowie die Aufwände zur Berechnung des inversen Gruppenelements eine wichtige Rolle. Letztere Operation muss für jedes Element einer Anfrage einmal durchgeführt werden, wohingegen die multiplikative Verknüpfung zweier Gruppenelemente ggf. mehrere Millionen mal bei einer Anfrage berechnet werden muss. Demnach ist diese Operation maßgebend für die Zeit, die zur Bearbeitung einer Anfrage benötigt wird.
Wie in Abbildung 6.4 zu sehen ist, werden bei der Darstellung der $SE(3)$ mittels Quaternionen mehr Operationen benötigt als bei der Realisierung der Rotationskomponenten durch 3×3-Matrizen. Dies geht zurück auf die Berechnung von $\rho(q)$ aus dem Quaternion, welches einem Gruppenelement g $SE(3)$ zugrunde liegt. Die dritte Spalte der Tabelle listet die Aufwände für den Fall, dass anstelle der Berechnung der Rotationsmatrix, die „Sandwichprodukte“ berechnet werden.
In der Praxis sollte der Speicherplatzbedarf nicht vernachlässigt werden, da mehrere Millionen Einträge in einem Suchindex eher die Regel sein werden. Bei Speicherplatzbedarf von 8 Byte pro Gleitkommawert und rund 4,3 Millionen Indexeinträgen, belegt ein Suchindex basierend auf Rotationsmatrizen rund 413 MB und basierend auf Quaternionen 250 MB Speicherplatz. Es lässt sich feststellen, dass die Verwendung von Quaternionen trotz der höheren Anzahl von Rechenoperationen kürzere Reaktionszeiten auf Anfragen bewirkt.

	$SE(3)$-**Matrix**	$SE(3)$-**Quaternion**	$SE(3)$-**Quaternion***
Speicherplatzbedarf: g	12 Gleitkommawerte	7 Gleitkommawerte	7 Gleitkommawerte
Berechnung: g^{-1}	9 Multiplikationen 6 Additionen 6 Vertauschungen	22 Multiplikationen 21 Additionen	32 Multiplikationen 24 Additionen
Berechnung: gh	36 Multiplikationen 27 Additionen	38 Multiplikationen 36 Additionen	48 Multiplikationen 39 Additionen

Abbildung 6.4: Speicherplatzbedarf und Aufwände für die wichtigsten Gruppenoperationen. (*) In diesem Fall werden die „Sandwichprodukte“ berechnet, anstelle $\rho(q)t$ zu berechnen.

Eigenschaft	**Rotationsmatrizen**	**Quaternionen**
Anzahl Moleküle	1.000	
Anzahl Atome	3.098.283	
Anzahl Bindungen	6.058.640	
Anzahl Indexeinträge	4.260.593	
Anzahl Listen	142.870	
Zeit Indexaufbau	726 [s]	683 [s]
Dateigröße	413 MB	250 MB
Anfragedauer, $\|Q\| = 16$	30 [ms]	20 [ms]

Abbildung 6.5: Einige statistische Daten zu zwei Indexdateien. Die unterschiedliche Zeit, die für den Aufbau des Suchindexes aus den Rohdaten benötigt wird, spiegelt sehr gut die Bedeutung des Speicherplatzes wider, den jedes Gruppenelement benötigt

6.6 Modellierung der Elementarobjekte

Wie in dem Abschnitt über das Datenformat der PDB erwähnt wurde, interessieren bei der hier diskutierten Anwendung hauptsächlich die Koordinaten der Atome in den einzelnen Molekülen. Wie bereits in dem Grundlagenkapitel zur Suche mittels G-invertierter Listen angedeutet, sind einzelne Atome eines Moleküls als Elementarobjekte für die Indexierung nicht geeignet, da die Kardinalität der Stabilisatoren unendlich wäre. Schliesslich gibt es unendlich viele Gruppenelemente $g \quad SE(3)$, die ein Atom um eine Achse durch den Atommittelpunkt rotieren und dabei die Ortskoordinate des Atoms unverändert lassen.
Die in dem Grundlagenkapitel angesprochene Strategie, um diesem Problem entgegenzuwirken, legt nahe, jeweils zwei Atome, die untereinander eine Bindung eingehen, als Elementarobjekte für eine Indexierung zu verwenden. Allerdings löst der Übergang zu Paaren von Atomen das Problem der unendlich großen Stabilisatoren ebenfalls noch nicht. Denn es existieren auch in diesem Fall unendlich viele Gruppenelemente g, für die $gm = m$ gilt. Dies sind eben gerade die Gruppenelemente, die die beiden Atome eines Paares um die Bindungsachse drehen und deren Translationskomponente die Ortskoordinaten der Atome unberührt lässt. Damit ist die Ordnung eines Stabilisators G_m äquivalent zur Kardinalität des Intervalls $[0 : 2\pi)$. Ausserdem zeigt sich in der Praxis, dass die Anzahl der *unterschiedlichen* Bindungen im Vergleich zur Gesamtanzahl aller Bindungen sehr klein ist. Dies würde nur zu einer geringen Anzahl von G-invertierten Listen führen, die typischerweise eine große Anzahl von Einträgen aufwiesen.
Aus diesem Grund werden wir im folgenden Abschnitt zu Dreiermengen von Atomen übergehen, wobei an die Wahl der Dreiermengen zusätzliche Bedingungen geknüpft sind, um zu garantieren, dass

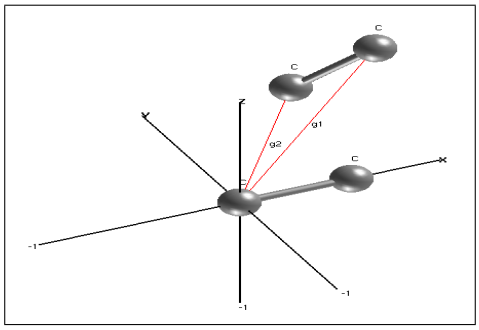


Abbildung 6.6: Paar von gebundenen C-Atomen und der zugehörige Repräsentant lokalisiert im Ursprung. Die beiden Linien deuten die Mehrdeutigkeiten bei der Bestimmung des Gruppenelements g an.

die Stabilisatoren trivial werden. Wir werden sehen, dass die Zusammenfassung von drei Atomen zu einem Elementarobjekt zu einem Repräsentantensystem führen wird, dessen Kardinalität in der Praxis groß genug ist, um eine ausreichend große Zahl von G-invertierten Listen zu erhalten. Dies bedeutet, dass im Mittel die Listen nicht zu viele Elemente enthalten, was wiederum die Antwortzeit des Retrievalsystems positiv beeinflusst.
Im Gegensatz zu den zuvor genannten Vorteilen, führt die Zusammenfassung von drei Atomen zu einem Elementarobjekt auch zu Nachteilen für den Benutzer. Zum Beispiel bezieht sich der Parameter k bei einer fehlertoleranten Suche nun nicht mehr auf Atome, sondern auf Dreiermengen von Atomen, was dazu führt, dass bei einer Anfrage der Parameter k nicht mehr direkt angegeben werden kann, sondern lediglich ein Prozentwert, der besagt, dass $k\%$ der angefragten Elementarobjekte mit dem Dokument in der Dokumentenkollektion übereinstimmen müssen.
Angenommen eine Anfrage Q ist ein Fragment des Moleküls D_i, jedoch stimme ein Atom $m \quad Q$ nicht mit einem Atom in D_i überein. In dem Grundlagenkapitel wurde für solche Anfragen das Prinzip von Fehlstellen eingeführt. Geht man nun zu Dreiermengen von Atomen über, so muss die Zahl der Fehlstellen der Anzahl der Dreiermengen entsprechen, in denen m enthalten ist. Auf diese Weise kann es sein, dass eine Vielzahl von Fehlstellen zugelassen werden muss.

6.6.1 Dreiermengen von Atomen als Elementarobjekte

Aufgrund dieser Vorüberlegungen werden im folgenden *drei* Atome eines Moleküls D zu einem Elementarobjekt zusammengefasst, sofern sie gewisse geometrische sowie chemische Nebenbedingungen erfüllen. Wir beginnen mit den geometrischen Bedingungen.

De nition 6.2 *Das Universum U von Elementarobjekten sei die Menge aller 3-elementigen Teilmengen $u = \{a, b, c\}$ von $\mathbb{R}^3 \times \mathcal{A}$, die die folgenden Bedingungen erfüllen:*

1. $\pi_1(a), \pi_1(b), \pi_1(c)$ *sind* nicht kollinear.
2. *Beschreiben $\pi_1(a), \pi_1(b), \pi_1(c)$ ein gleichseitiges Dreieck, so müssen die zugehörigen Elementnamen $\pi_2(a), \pi_2(b), \pi_2(c)$ paarweise unterschiedlich sein.*
3. *Beschreiben $\pi_1(a), \pi_1(b), \pi_1(c)$ ein gleichschenkliges Dreieck, mit o.B.d.A. $\pi_1(a) - \pi_1(b) = \pi_1(a) - \pi_1(c) = \pi_1(b) - \pi_1(c)$, müssen b und c über unterschiedliche Elementnamen verfügen.*

Im folgenden sprechen wir statt von 3-elementigen Teilmengen auch (etwas ungenau) von *Atomtripeln*. Auf der Menge U' aller Atomtripel aus $\mathbb{R}^3 \times \mathcal{A}$ operiert die Gruppe $SE(3)$ kanonisch auf den ersten Komponenten. Das obige Universum U ist eine $SE(3)$-invariante Teilmenge, die gerade aus allen Elementen von U besteht, die triviale Stabilisatoren besitzen.
Wir kommen nun zu den chemischen Bedingungen an die Elemente von U. Dabei benutzen wir die Sprechweisen von Definition 6.1.

De nition 6.3 *Zwei verschiedene Atome a, b D gelten hier als* chemisch gebunden*, wenn deren Euklidscher Abstand kleiner oder gleich der Summe der kovalenten Bindungsradii der beiden Atome und eines Toleranzwertes $\epsilon > 0$ ist:*

$$\pi_1(a) - \pi_1(b) \ \leq \mathrm{Rad}_{cov}(\pi_2(a)) + \mathrm{Rad}_{cov}(\pi_2(b)) + \epsilon.$$

Sind Atome i und j miteinander verbunden, so schreiben wir kurz symbolisch: $i \bowtie j$. Mit $B_D = (b_{ij})$ wird die Adjazenzmatrix des Bindungsgraphen bezeichnet. Es gilt also $b_{ij} = 1$ gdw. $i \bowtie j$. (Nach De nition ist stets $b_{ii} = 0$.) Die Anzahl $_D(a)$ der Bindungspartner von a in D heißt der Grad *von a in D.*

De nition 6.4 *Die Menge $D_\triangle$ der Atomtripel zu einem Molekül D ist gegeben durch*

$$D_\triangle := \{\{i,j,k\} \quad \binom{D}{3} \setminus U \mid (i \bowtie j \bowtie k) \quad (i \bowtie k \bowtie j) \quad (k \bowtie i \bowtie j)\},$$

wobei für eine Menge M und eine Zahl k mit $\binom{M}{k}$ die Menge aller k-elementigen Teilmengen von M bezeichnet wird. Zur Molekülkollektion $\mathcal{D} = (D_1, \ldots, D_N)$ gehört entsprechend die Kollektion $\mathcal{D}_\triangle := ((D_1)_\triangle, \ldots, (D_N)_\triangle)$. Ein Molekül D (analog eine Anfrage Q) heißt -überdeckbar*, wenn $D = \ _{X \in D_\triangle} X$, d.h. jedes Atom in D kommt in mindestens einem Atomtripel aus $D_\triangle$ vor. $D_\triangle$ heißt dann auch* -Überdeckung *von D.*

In der Praxis kann es aufgrund unserer geometrischen und chemischen Nebenbedingungen in einem Molekül separate Atome und Atompaare geben. Diese bleiben bei der Indexierung unberücksichtigt.

Die Definition von $D_\triangle$ hat eine wichtige Monotonie- und Invarianzeigenschaft, die im folgenden Satz genauer erläutert wird.

Satz 6.6 *Es sei D' eine Teilmenge des Moleküls D und g $SE(3)$. Dann gilt: $g^{-1}(gD')_\triangle \subseteq D_\triangle$. Das heißt, beim Übergang zu einem Fragment D' von D und Abänderung aller Ortskoordinaten des Fragments durch eine Euklidsche Bewegung, kommen keine neuen Atomtripel hinzu.*

Beweis: In die Definition der Bindungsrelation $\bowtie$ geht wesentlich der Euklidsche Abstand ein. Dieser ist unter starren Bewegungen invariant. Zusammen mit $\binom{D'}{3} \subseteq \binom{D}{3}$ folgt die Behauptung. □

Der folgende Satz beschreibt die Größe von $D_\triangle$.

Satz 6.7 *Für ein Molekül D mit d_i Knoten vom Grad i besitzt $D_\triangle$ genau $\ _i d_i \binom{i}{2}$ viele Elemente.*

Beweis: Ein Knoten a vom Grad i hat genau i Bindungspartner. Je zwei von ihnen, etwa b, c, bilden zusammen mit a ein Atomtripel mit a im Zentrum: $b \bowtie a \bowtie c$. □

Die Abbildung 6.7 illustriert den obigen Satz.

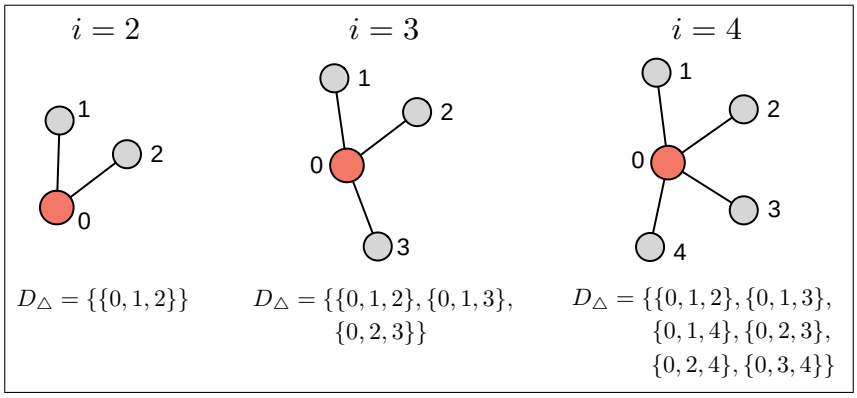


Abbildung 6.7: Illustration der Mengen $D_\triangle$ für die Fälle $2 \leq i \leq 4$.

Algorithmus 6.6.1: CALCULATETRIPLES(Molecule D)

B_D CALCULATEBONDS(D);
$D_\triangle$;
for each Atom a D
for each Atom b D $a \bowtie b$
for each Atom c D $a = c$ $b \bowtie c$
if not $\{a,b,c\}$ $D_\triangle$ **and** VALID($\{a,b,c\}$)
then $D_\triangle$ $D_\triangle$ $\{a,b,c\}$;
return ($D_\triangle$);

Algorithmus 6.6.1: Algorithmus zur Berechnung der Menge $D_\triangle$ von Atom-Tripeln zu einem Molekül D. Wichtig ist, dass eine Dreiermenge von Atomen nur einmal in $D_\triangle$ eingefügt wird.

De nition 6.5 *Sei $D_\triangle$ eine* Überdeckung *von D. Zwei Elemente a, b $D_\triangle$ heißen* unabhängig*, wenn $a \setminus b =$ gilt. Allgemeiner heißt $|a \setminus b|$ $[0:3]$ der* Abhängigkeitsgrad *zwischen a und b.*

Bisher haben wir ein Molekül D aus Indexierungssicht durch alle chemisch sinnvollen Atomtripel überdeckt. Diese Darstellung ist zwar sehr redundant, erlaubt aber eine Beantwortung aller angefragten Molekülfragmente, die sich durch eine beliebige Atomtripelmenge überdecken lassen. Während man zum Zwecke der Indexierung an einer möglichst umfassenden Atomtripelüberdeckung interessiert ist, wird man aus Effizienzgründen zu einer Anfrage $Q\colon X$ $\mathcal{A}$, X eine endliche Teilmenge von $\mathbb{R}^3$, eine möglichst sparsame Überdeckung von Q wählen. Zunächst interessiert das folgende fundamentale Lemma.

Lemma 6.1 *Ist die Anfrage Q -überdeckbar, so gilt:*

$$G_{\mathcal{D}}(Q) = G_{\mathcal{D}_\triangle}(Q_\triangle).$$

Beweis:

(g,i) $G_{\mathcal{D}}(Q)$ $gQ \subseteq D_i$
(wg. Q ist -überdeckbar) q $Q_\triangle : gq$ $(D_i)_\triangle$
(g,i) $G_{\mathcal{D}_\triangle}(Q_\triangle)$.

□

Der Übergang von Q zu $Q_\triangle$ macht suchtechnisch also keinen Unterschied, wenn Q bzw. D -überdeckbar sind.

De nition 6.6 *Eine Teilmenge $T_\triangle \subseteq Q_\triangle$* überdeckt *$Q$, wenn jedes Element von Q in mindestens einem Element von $T_\triangle$ vorkommt: q Q : t $T_\triangle : q$ t.*

Satz 6.8 *Ist eine Anfrage Q -überdeckbar, so gilt für jede Teilmenge $T_\triangle \subseteq Q_\triangle$, die Q überdeckt*

$$G_{\mathcal{D}}(Q) = G_{\mathcal{D}_\triangle}(T_\triangle).$$

Beweis: Da $G_{\mathcal{D}_\triangle}(T_\triangle) \supseteq G_{\mathcal{D}_\triangle}(Q_\triangle) = G_{\mathcal{D}}(Q)$ gilt, genügt es $G_{\mathcal{D}_\triangle}(T_\triangle) \subseteq G_{\mathcal{D}}(Q)$ zu zeigen. Nun gilt (g, i) $G_{\mathcal{D}_\triangle}(T_\triangle)$ $\{a, b, c\}$ $T_\triangle : \{ga, gb, gc\}$ $(D_i)_\triangle$, also ga, gb, gc D_i. Da Q von $T_\triangle$ überdeckt wird, gilt gq D_i für alle q Q. Dies bedeutet $gQ \subset D_i$ und somit auch (g, i) $G_{\mathcal{D}}(Q)$. □

Aus Gründen der Effizienz sind wir gerade an solchen Mengen $T_\triangle$ interessiert, die eine möglichst geringe Kardinalität aufweisen. Wir bezeichnen im folgenden zu einem Molekül(-fragment) Q eine solche Menge mit möglichst geringer Kardinalität mit $Q_\triangle$.
Zur näheren Untersuchung nehmen wir der Einfachheit einen allgemeineren Standpunkt ein. Die Atomtripel kann man ansehen als Pfade der Länge 2 in dem zu Q gehörigen Verbindungsgraphen. Aus technischen Gründen brauchen wir auch Pfade der Länge 1, d.h. Kanten. Weiter muss gewährleistet werden, dass jeder Knoten an mindestens einer Kante beteiligt ist, also nicht isoliert ist. Solche Graphen wollen wir *unisoliert* nennen.
Dies motiviert folgende Definition.

De nition 6.7 *Es sei $\Gamma = (V, E)$ ein unisolierter Graph mit $n := |V| \geq 2$ Knoten. Die zu einem Pfad $a - b - c$ der Länge 2 in Γ gehörige Menge $\{a, b, c\}$ wollen wir kurz als einen* 2-Pfad *bezeichnen. Entsprechend bezeichnen wir eine Kante $\{a, b\}$ in Γ als* 1*-Pfad. Indem wir einen* 1*-Pfad künstlich durch $a-b-a$ zu einem* 2*-Pfad erweitern, wollen wir im folgenden einheitlich von* 2*-Pfaden sprechen. Es sei $\Pi_2(\Gamma)$ die Menge aller* 2*-Pfade in Γ. Eine Teilmenge C von $\Pi_2(\Gamma)$ mit $V = \gamma \in C \gamma$ heißt* 2-Pfad-Überdeckung *von Γ. Ein solches C von minimaler Kardinalität $pc(\Gamma)$ heißt eine* minimale 2-Pfad-Überdeckung.

Satz 6.9 *Für einen unisolierten Graphen Γ mit $n \geq 2$ Knoten gilt:*

$$\frac{n}{3} \leq pc(\Gamma) \leq \frac{n}{2}.$$

Die angegebenen Schranken sind scharf: Zu unendlich vielen Knotenanzahlen n gibt es unisolierte Graphen Γ_{0n} und Γ_{1n} mit jeweils n Knoten so dass $pc(\Gamma_{0n}) = n$ 3 und $pc(\Gamma_{1n}) = n$ 2.

Beweis: Pro 2-Pfad kann man maximal 3 Elemente von V überdecken. Damit gilt die untere Schranke.

Wir machen uns zunächst klar, dass $pc(\Gamma) = \Theta(n)$ ist: Da Γ unisoliert ist, ist jeder Knoten an mindestens einem 2-Pfad beteiligt. Mit jedem neuen 2-Pfad kann man also mindestens einen weiteren Punkt von V überdecken.

Die behauptete obere Schranke beweisen wir nun durch Induktion nach n. Der Start $n = 2$ ist trivial. Damit kommen wir zum Induktionsschritt. Entsteht der unisolierte Graph Γ' aus Γ durch Weglassen

einiger Kanten aus E, so ist $pc(\Gamma) \leq pc(\Gamma')$. Daher genügt es für einen beliebigen, unisolierten Spannwald Γ' von Γ zu zeigen, dass $pc(\Gamma') \leq n$ 2 gilt. Im folgenden sei also Γ selbst ein unisolierter Spannwald mit $n \geq 2$ Knoten. Zu einem Knoten v V sei B_v die Menge der mit diesem Knoten verbundenen Blätter und R_v die Menge der restlichen Knoten, die mit v verbunden sind. Weiterhin sei $b_v := |B_v|$ und $r_v := |R_v|$. Man beachte, dass alle w R_v mindestens den Grad 2 haben.

Es genügt, die obere Schranke für jede Zusammenhangskomponente von Γ zu zeigen. Zusammenhangskomponenten mit nur zwei Knoten bereiten keine Schwierigkeiten, da beide Knoten mit einem 1-Pfad überdeckbar sind. Sei also o.E. Γ ein Baum mit $n \geq 3$ Knoten. Hier ist die Menge V' der inneren Knoten in Γ nichtleer. Enthält V' genau ein Element v, so ist Γ ein Stern mit Zentrum v und $n-1 \geq 2$ Blättern $w_1, \ldots, w_{n-1}$. Im Fall $n-1 = 2m$ ist $\{\{w_{2i-1}, v, w_{2i}\} \mid i \quad [1:m]\}$ eine 2-Pfad-Überdeckung von Γ, während $\{\{w_{2i-1}, v, w_{2i}\} \mid i \quad [1:m-1]\} \quad \{\{v, w_{n-1}\}\}$ im Fall $n = 2m$ den Baum Γ überdeckt. Damit ist $pc(\Gamma) \leq m = (n-1)$ 2, falls n ungerade ist, und $pc(\Gamma) \leq m = n$ 2, falls n gerade ist. Man zeigt sofort, das sogar in beiden Fällen $pc(\Gamma) = m$ gilt.

Nun betrachten wir den Fall, dass es mehr als ein Element in V' gibt. Sei v V' ein Knoten mit $b_v > 0$ und $r_v = 1$, etwa $R_v = \{z\}$. Da Γ eingeschränkt auf V' selbst wieder ein Baum ist, existiert stets ein solches v. Wir unterscheiden zwei Fälle.

Fall 1: $b_v = 2m > 0$ ist gerade. Sei $B_v = \{w_1, \ldots, w_{2m}\}$. Hier kann man $\{v\}$ B_v mit den 2-Pfaden $\{w_{2i-1}, v, w_{2i}\}$, $i \quad [1:m]$, überdecken. Nun entferne man $B_v \quad \{v\}$ und alle v enthaltenden Kanten

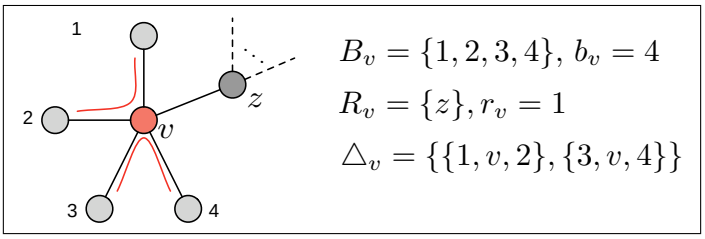


Abbildung 6.8: Fall 1: Bei einer geraden Anzahl $b_v = 2m$ von Blättern genügen genau m viele 2-Pfade für eine vollständige Überdeckung der Knotenmenge $B_v \quad \{v\}$. Im nächsten Schritt bleiben die Knoten aus $B_v \quad \{v\}$ sowie die mit v verbundenen Kanten unberücksichtigt.

aus Γ. Wegen $r_v = 1$ erhalten wir wieder einen Baum $\Gamma' = (V', E')$ mit genau $n - b_v - 1$ Knoten. Weiterhin besitzt V' mindestens zwei Knoten, da V' nach Konstruktion ein Element aus R_v enthält und alle Elemente aus R_v mindestens den Grad 2 haben. Nach Induktionsannahme erhalten wir also

$$pc(\Gamma) \leq \frac{b_v}{2} + pc(\Gamma') \leq \frac{b_v}{2} + \frac{|V'|}{2} \leq \frac{b_v}{2} + \frac{n-b_v-1}{2} \leq \frac{n}{2}.$$

Fall 2: $b_v = 2m+1$ ist ungerade. Sei $B_v = \{w_0, w_1, \ldots, w_{2m}\}$.

Fall 2.1: $b_z > 0$, d.h. z ist durch eine Kante mit einem Blatt verbunden. Hier kann die Knotenmenge $\{v\} \quad B_v$ durch $C_v := \{\{w_{2i-1}, v, w_{2i}\} \mid i \quad [1:m]\} \quad \{\{w_0, v\}\}$ überdeckt werden. Dies sind (b_v+1) 2 $= m+1$ viele 2-Pfade. Indem wir mit dem zur Knotenmenge $V \quad (\{v\} \quad B_v)$ gehörenden Teilbaum von Γ induktiv fortfahren, erhalten wir ähnlich wie im ersten Fall $pc(\Gamma) \leq \frac{n}{2}$.

Fall 2.2: $b_z = 0$. In diesem Fall kann die Knotenmenge $\{v, z\} \quad B_v$ durch $C_v := \{\{w_{2i-1}, v, w_{2i}\} \mid i \quad [1:m]\} \quad \{\{w_0, v, z\}\}$ überdeckt werden. Dies sind (b_v+1) 2 $= m+1$ viele 2-Pfade. Indem wir Γ auf die Knotenmenge $V \quad (\{v, z\} \quad B_v)$ einschränken, erhalten wir einen Wald von genau r_z Zusammenhangshangskomponenten mit jeweils mindestens zwei Knoten. Mit Hilfe der Induktionsannahme erhalten wir wieder $pc(\Gamma) \leq \frac{n}{2}$. Damit ist die obere Schranke bewiesen.

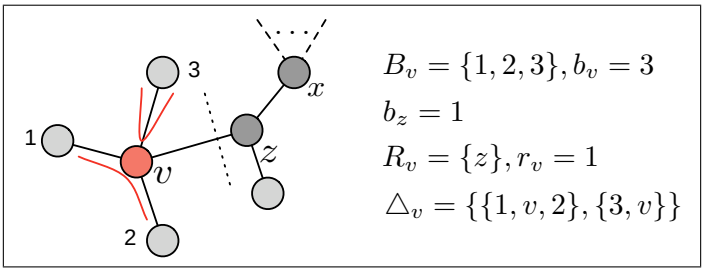


Abbildung 6.9: Fall 2.1: Bei einer ungeraden Anzahl $b_v = 2m+1$ von mit v verbundenen Blättern und $R_v = \{z\}$ mit $b_z > 0$ genügen genau $m+1$ viele 2-Pfade für eine vollständige Überdeckung der Knotenmenge $B_v \quad \{v\}$. Im nächsten Schritt bleiben die Knoten aus $B_v \quad \{v\}$ sowie die mit v verbundenen Kanten unberücksichtigt.

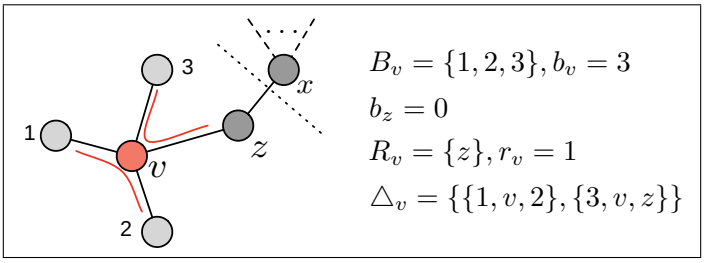


Abbildung 6.10: Fall 2.2: Bei einer ungeraden Anzahl $b_v = 2m+1$ von mit v verbundenen Blättern und $R_v = \{z\}$ mit $b_z = 0$ genügen genau $m+1$ viele 2-Pfade für eine vollständige Überdeckung der Knotenmenge $B_v \quad \{v,z\}$. Im nächsten Schritt bleiben die Knoten aus $B_v \quad \{v,z\}$ sowie die mit v und z verbundenen Kanten unberücksichtigt.

Zum Beweis der Schärfe der angegebenen Schranken sei $n = 3(2m+1)$. Bezeichnet Γ_{0n} eine Kette und Γ_{1n} einen Stern mit jeweils n Knoten, dann ist $pc(\Gamma_{0n}) = n \;\; 3$ und $pc(\Gamma_{1n}) = n \;\; 2$. □

Daraus erhalten wir folgendes

Korollar 6.1 $\frac{n}{3} \leq |Q_{\triangle}| \leq \frac{n}{2}$.

6.6.1.1 Berechnung einer minimalen Überdeckung

Der obige Beweis hat gezeigt, dass wir ausgehend von einem beliebigen unisolierten Graphen uns auf den Spezialfall von Spannwäldern beschränken können. Nun haben typische Anfragen an Molekülda-tenbanken fast Baumcharakter. Das folgende Resultat zeigt, dass das im obigen Beweis beschriebene Vorgehen für Spannwälder optimal ist.

Satz 6.10 *Es sei Γ ein Baum mit $n \geq 2$ Knoten. Wendet man das im Beweis von Satz 6.9 beschriebene Verfahren auf Γ an, so erhält man eine optimale 2-Pfad-Überdeckung von Γ.*

Beweis: Falls Γ ein Stern ist, liefert das obige Verfahren offensichtlich eine optimale 2-Pfad-Über-deckung. Sei nun $v \quad V$ mit $b_v > 0$ und $r_v = 1$, etwa $R_v = \{z\}$. Weiterhin sei C eine beliebige

optimale 2-Pfad-Überdeckung von Γ. Wir betrachten die Menge C_v aller 2-Pfade in C, die mit B_v einen nichtleeren Schnitt haben:

$$C_v := \{\gamma \quad C \mid \gamma \setminus B_v = \ \}.$$

Jedes γ C_v enthält mindestens ein und höchstens zwei Elemente aus B_v: $1 \leq |\gamma \setminus B_v| \leq 2$. Weiterhin gilt für die Menge $\overline{C}_v := \ _{\gamma \in C_v} \gamma$ der von B_v aus mittels 2-Pfaden aus C überdeckten Knoten in V die Abschätzung

$$B_v \quad \{v\} \subseteq \overline{C}_v \subseteq B_v \quad \{v\} \quad R_v = B_v \quad \{v, z\}.$$

Sei Γ_v die Einschränkung von Γ auf die Knotenmenge $B_v \quad \{v\}$ und Γ_v^z die Anreicherung von Γ_v um die Kante $\{v, z\}$. Dann gilt im Fall $b_v = 2m - 1$:

$$pc(\Gamma_v) = pc(\Gamma_v^z) = m$$

und im Fall $b_v = 2m$:

$$pc(\Gamma_v) = m < m + 1 = pc(\Gamma_v^z).$$

Im Fall $b_v = 2m$ können wir C_v ohne Optimalitätsverlust ersetzen durch die im obigen Beweis konstruierte 2-Pfad-Teilüberdeckung $\{w_{2i-1}, v, w_{2i}\}$, $i \quad [1 : m]$. Man beachte, dass hier unsere Konstruktion mit $m + 1$ vielen 2-Pfaden nicht nur $B_v \quad \{v, z\}$ sondern sogar $B_v \quad \{v, z, x, y\}$ überdecken kann, wobei $z - x - y$ ein beliebiger 2-Pfad in Γ ist. Im Fall $b_v = 2m - 1$ und $b_z > 0$ lohnt es nicht, dass man von B_v mit einem 2-Pfad den Knoten z überdeckt, da dieser wegen $b_z > 0$ gezwungenermaßen über jedes seiner Blattkinder an mindestens einem 2-Pfad beteiligt ist. Also reicht es aus, Γ_v mit m vielen 2-Pfaden zu überdecken. Ist jedoch (Fall 2.2) $b_z = 0$, so lohnt sehr wohl die Überdeckung von $B_v \quad \{v, z\}$ mit m vielen 2-Pfaden. Hier ist der Anteil C_v bis auf Umnummerierung sogar eindeutig. Damit ist nichts abzuändern. Dies beweist die Behauptung. □

In der Praxis werden wir jedoch von Bäumen mit mindestens drei Knoten ausgehen, da ansonsten kein 2-Pfad gebildet werden kann.
Der Algorithmus 6.6.2 berechnet eine minimale Überdeckung einer Anfrage Q, nachdem diese ggf. durch Vorverarbeitung, den Anforderungen an die Eingabe Γ genügt. Zunächst werden für jeden Knoten $v \quad V$ die Menge der Blätter B_v und deren Anzahl b_v bestimmt. Ausserdem interessieren die verbundenen inneren Knoten R_v und ebenfalls deren Anzahl r_v. Die Knoten, welche Verbindungen zu Blättern haben, werden in der Menge V' zusammengefasst. Da wir fordern, dass Γ ein Baum mit mindestens drei Knoten ist, ist V' in keinem Fall leer. Ausgehend von Knoten $v \quad V'$, die mit nur einem inneren Knoten z verbunden sind, werden nun gemäß dem Satz 6.9 v und die damit verbundenen Blätter durch 2-Pfade überdeckt.
Ist die Anzahl der mit v verbundenen Blätter b_v gerade, so wird die Knotenmenge $\{v\} \quad B_v$ mit 2-Pfaden überdeckt. Mit $\Gamma^{z \setminus v}$ bezeichnen wir den Teilbaum der z enthält und aus dem die Kante $\{v, z\} \quad E$ entnommen wird. Ist die Anzahl der Knoten, $|\Gamma^{z \setminus v}|$, in diesem Teilbaum gleich zwei, so wird auch das mit z verbundene Blatt mit einem 2-Pfad der Art $\{v, z, \ \}$ überdeckt. Knoten, die an einem 2-Pfad beteiligt sind, werden aus V' entfernt und ebenfalls alle Kanten E_v, die v mit anderen Knoten verbinden.
Im Unterschied zu dem Beweis von Satz 6.9 überdecken wir im Falle einer ungeraden Anzahl von Blättern ebenfalls den Knoten z durch einen 2-Pfad. Auch hier wird im Fall $|\Gamma^{z \setminus v}| = 2$ das mit z verbundene Blatt überdeckt, womit v, z aus V' und die Kantenmengen E_v, E_z aus E entfernt werden können.

Im Fall b_v ungerade gilt es noch den Fall 2.2 aus dem Beweis zu berücksichtigen. Ist z mit keinem Blatt verbunden und haben alle Teilbäume $|\Gamma^{x\setminus z}|, x \quad R_z$, mindestens 3 Knoten, so können wir z und die Kante E_z aus V' bzw. E entfernen, da der Knoten z bereits überdeckt worden ist. Durch die Wegnahme der Kante E_z aus Γ zerfällt der Baum in die zuvor angegebenen Teilbäume mit mindestens drei Knoten. Aufgrund diese neuen Situation müssen für alle Knoten $x \quad (R_z \quad _{y\in R_z:b_y=0\wedge r_y=1} R_y)$ die Werte b_x, r_x und die Mengen B_X, R_x aktualisiert werden. In diesem Zuge können auch neue Knoten zu der Menge V' hinzukommen.
Existieren in V' keine Knoten mehr, welche mit Blättern *und* genau einem weiteren inneren Knoten verbunden sind, so beschreiben die in V' verbliebenen Knoten v Sterne mit b_v vielen Blättern. Diese Fälle werden in der zweiten **while**-Schleife des Algorithmus behandelt.
In Satz 6.10 wurde von Bäumen als Eingabe ausgegangen, da dies der hier diskutierten Anwendung auf Molekülдaten entspricht. Für allgemeine Graphen gilt folgender Satz in Bezug auf die Berechenbarkeit einer optimalen 2-Pfad Überdeckung.

Satz 6.11 *Das Entscheidungsproblem, ob ein beliebiger 2-Pfad überdeckbarer Graph Γ mit n Knoten durch disjunkte 2-Pfade überdeckt werden kann ist NP-vollständig.*

Beweisidee: Der Beweis der NP-Vollständigkeit des Entscheidungsproblems basiert auf einem Satz von Kirkpatrik und Hell [KH78]. Dies besagt, dass für einen Graphen G das Entscheidungsproblem, ob ein gegebener Graph H eine G-Partition ermöglicht, NP-vollständig ist, wenn G über mindestens eine Zusammenhangskomponente mit mindestens 3 Knoten verfügt. Weitere Details dazu finden sich in [BCR06].

6.6.1.2 Repräsentantenberechnung

Um die allgemeine Suchtechnik aus Kapitel 2 anwenden zu können, sind nun zwei Probleme zu lösen:

1. Bestimmung eines Repräsentantensystems der $SE(3)$-Bahnen von U.
2. Beschreibung eines Normalformenalgorithmus, der jedem Element u von U seinen Repräsentanten r_u sowie das eindeutig bestimmte Gruppenelement $g_u \quad SE(3)$ mit $g_u r_u = u$ zuordnet.

Das durch ein $u \quad U$ beschriebene Dreieck wird im folgenden mit $\quad _u$ bezeichnet. Zunächst ein grober Überblick über das weitere Vorgehen: In einem ersten Schritt wird zu $u = \{a, b, c\} \quad U$ der Schwerpunkt des Dreiecks $\quad _u$ auf den Ursprung verschoben. In einem zweiten Schritt wird das Dreieck in die xy-Ebene rotiert. Es folgt die Bestimmung der Hypothenuse und (ggf. nach Umbenennung) die Rotation des Punktes $\pi_1(a)$, welcher nicht auf der Hypothenuse liegt, auf den positiven Bereich der x-Achse. Im letzten Schritt wird geprüft, welcher der beiden Hypothenusenpunkte $\pi_1(b), \pi(c)$ den kleineren x-Wert aufweist. Damit dieser Punkt im positiven Bereich der y-Achse liegt, müssen alle Punkte aus u ggf. um 180 Grad um die x-Achse rotiert werden. Für den Fall, dass ein gleichseitiges oder gleichschenkliges Dreieck vorliegt, erfolgt die Ermittlung der Reihenfolge von a, b, c gemäß der lexikographischen Ordnung der Elementnamen. Es schließt sich die genauere Beschreibung für den Fall an, dass das zugrunde liegende Dreieck keine Symmetrien aufweist.

1. Bestimmung des Schwerpunktes: Die drei Ortsvektoren des Atomtripels $u = \{a, b, c\}$ beschreiben ein Dreieck im $\mathbb{R}^3$. Der Schwerpunkt dieses Dreiecks berechnet sich durch Durchschnittsberechnung in jeder Komponente und wird mit $s_\triangle \quad \mathbb{R}^3$ bezeichnet.

2. Translation von u um $s_\triangle$: Durch die Verschiebung der Ortskomponenten von u mit dem Translationsvektor $s_\triangle$ wird der Schwerpunkt des Dreiecks auf den Ursprung verschoben: $u' := u - s_\triangle$.

Algorithmus 6.6.2: CALCULATEMINCOVERING ($\Gamma = (V, E)$)

Input: a tree Γ with at least 3 vertices
Output: a minimal 2-path cover $\Gamma_\triangle$ of Γ
Global variables: sets: V, E, V', arrays: B, R, b, r

procedure UPDATE(Vertex v)
  CALCULATE B_v, R_v, b_v, r_v;
  if $b_v > 0$ // is v connected to a leaf?
    then $V' \leftarrow V' \cup \{v\}$;

main
  $\Gamma_\triangle \leftarrow \emptyset$; $V' \leftarrow \emptyset$;
  for each $v \in V$
    do UPDATE(v);
  while $\exists v \in V'$ **and** $R_v = \{z\}$
    do
      if $b_v = 2m$ // Case 1 in Theorem 6.9
        then
          for $i \leftarrow 1$ **to** m
            do $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{w_{2i-1}, v, w_{2i}\}\}$; // $B_v = \{w_1, \ldots, w_{2m}\}$
          $V' \leftarrow V' \setminus \{v\}$; $E \leftarrow E \setminus E_v$;
          if $|\Gamma^{z \setminus v}| = 2$
            then $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{v, z\} \cup B_z\}$; $V' \leftarrow V' \setminus \{z\}$; $E \leftarrow E \setminus E_z$;
      if $b_v = 2m + 1$
        then
          for $i \leftarrow 1$ **to** m
            do $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{w_{2i-1}, v, w_{2i}\}\}$; // $B_v = \{w_0, w_1, \ldots, w_{2m}\}$
          $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{w_0, v, z\}\}$; $V' \leftarrow V' \setminus \{v\}$; $E \leftarrow E \setminus E_v$;
          if $|\Gamma^{z \setminus v}| = 2$ // $b_z = r_z = 1$
            then $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{v, z\} \cup B_z\}$; $V' \leftarrow V' \setminus \{z\}$; $E \leftarrow E \setminus E_z$;
          if $b_z = 0$ **and** $\forall x \in R_z : |\Gamma^{x \setminus z}| \geq 3$ // based on Case 2.2 in Theorem 6.9
            then
              $E \leftarrow E \setminus E_z$; // also remove E_z. Note: $z \notin V'$
              for each $x \in (R_z \cup \bigcup_{y \in R_z : b_y = 0 \wedge r_y = 1} R_y)$
                do UPDATE(x);
            else UPDATE(z);
  while $\exists v \in V'$ // is v s connected component in the modified Γ a star with center v?
    do
      if $b_v = 2m$
        then
          for $i \leftarrow 1$ **to** m
            do $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{w_{2i-1}, v, w_{2i}\}\}$; // $B_v = \{w_1, \ldots, w_{2m}\}$
      if $b_v = 2m + 1$
        then
          for $i \leftarrow 1$ **to** m
            do $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{w_{2i-1}, v, w_{2i}\}\}$; // $B_v = \{w_1, \ldots, w_{2m+1}\}$
          $\Gamma_\triangle \leftarrow \Gamma_\triangle \cup \{\{w_{2m}, v, w_{2m+1}\}\}$
      $V' \leftarrow V' \setminus \{v\}$;
  return ($\Gamma_\triangle$);

Algorithmus 6.6.2: Die Menge V' enthält alle inneren Knoten, die mit Blättern verbunden sind. Zunächst werden die Elemente v aus V' mit $r_v = 1$ abgearbeitet. Im Verlaufe der Abarbeitung derartiger Elemente von V' können Teilbäume entstehen, die Sterne mit Zentren v' aus V' darstellen. Diese Anteile werden wegen $r_{v'} = 0$ am Ende gesondert überdeckt. Im Unterschied zum Beweis zu Satz 6.9 werden hier 1-Pfade durch die Kante zum eindeutig bestimmten Knoten aus R_v zu 2-Pfaden erweitert

3. Rotation in die xy-Ebene: Die Fläche, welche durch die drei Ortskoordinaten bestimmt ist, wird in diesem Schritt derart rotiert, dass sie in der xy-Ebene des Koordinatensystems zu liegen kommt, d.h. die z-Komponente der Ortsvektoren in u' werden zu Null. Die Schritte im einzelnen:

- Berechnung eines normierten Normalenvektors zur Ebene des Dreiecks:
  $n := (\pi_1(a') \times \pi_1(b'))$ $\pi_1(a') \times \pi_1(b')$
- Um die Rotationsrichtungen einheitlich festzulegen, muss für den Normalenvektor die z-Komponente $n_3 \geq 0$ gelten. Ggf. muss daher die Richtung des Vektors umgekehrt werden.
- Die gesuchte Rotationsachse berechnet sich als Kreuzprodukt des Normalenvektors und der z-Achse: $r := n \times (0, 0, 1)$. Der Drehwinkel α ist das Skalarpodukt zwischen den beiden Vektoren: $\alpha :=$ $n, (0, 0, 1)$. Dies zusammen ergibt eine Rotationsmatrix R_1, deren Anwendung die Punkte $\pi_1(a'), \pi_1(b'), \pi_1(c')$ in die xy-Ebene rotiert.

4. Rotation eines Punktes auf die x-Achse: Um eine eindeutige Lage des Dreiecks in der XY-Ebene zu erhalten, wird gefordert, dass das Dreieck derart zu rotieren ist, dass (o.E.) $\pi_1(a')$ auf dem positiven Teil der x-Achse zu liegen kommt. Um einen der drei Eckpunkte von $_{u'}$ derart auszuzeichnen, wird der Eckpunkt $\pi_1(a')$ ausgewählt, welcher der Hypothenuse von $_{u'}$ gegenüber liegt. Hierzu wird der Winkel β zwischen dem Vektor $\pi_1(a')$ und der x-Achse bestimmt: $\beta :=$ $\pi_1(a')$ $\pi_1(a')$ $, (1, 0, 0)$. Das Ergebnis dieser Berechnungen ist eine Rotationsmatrix R_2, welche auf alle Ortsvektoren von u' angewendet wird.

5. Rotation um die x-Achse: Um die Position des Dreiecks in der Ebene eindeutig festzulegen, muss ein weiterer Punkt von $_{u'}$ gewählt werden, o.E. b'. Hierzu werden die x-Komponenten der Vektoren $\pi_1(b'), \pi_1(c')$, die auf der Hypothenuse liegen, verglichen:

$$b' := \begin{cases} b' & \pi_1(b')_x \leq \pi_1(c')_x \\ c' & \text{sonst} \end{cases}$$

Es wird weiterhin gefordert, dass der Punkt b' einen positiven y-Wert aufweisen soll. Aus diesem Grund kann ggf. eine Rotation des Dreiecks um 180 Grad um die x-Achse notwendig werden, was zur Rotationsmatrix R_3 führt. Dies ist die Identität für den Fall, dass die Rotation um die x-Achse nicht nötig ist. O.B.d.A. wird c' gleich dem anderem Punkt auf der Hypothenuse gesetzt.

6. Bestimmung des Gruppenelements: Das Gruppenelement g $SE(3)$, welches in einer G-invertierten Liste abgelegt werden wird, ergibt sich aus dem Produkt der Rotationsmatrizen $R := R_3 R_2 R_1$ und der Translation an den Ursprung, welche durch $s_\triangle$ beschrieben wird:

$$g = (R^\top, s_\triangle).$$

Die Transposition der Rotationsmatrix R ist notwendig, da in den Berechnungen zuvor die Rotation berechnet worden ist, welche auf ein Atomtripel angewendet werden muss, um das Tripel in den jeweiligen Repräsentanten zu überführen. In der invertierten Liste wird jedoch das Gruppenelement abgelegt, welches den Repräsentanten in ein Tripel transformiert, wie es in einem Molekül vorkommt.

7. Bestimmung des Repräsentanten: Nachdem das durch die drei Atomkoordinaten definierte Dreieck $_u$ durch die zuvor beschriebenen Schritte in die xy-Ebene rotiert worden und dessen

Schwerpunkt auf den Ursprung verschoben worden ist, muss das derart berechnete Gruppenelement g einem Repräsentanten und damit einer G-invertierten Liste zugeordnet werden. Der Repräsentant bestimmt sich zum einen aus den Abkürzungen der drei Elementnamen der beteiligten Atome und den auf eine festgelegte Anzahl von Nachkommastellen gerundeten Koordinaten der Atome in der Ebene. In der Praxis hat sich die Verwendung von zwei Nachkommastellen als guter Kompromiss zwischen der Anzahl von G-invertierten Listen und deren Länge herausgestellt. Eine genauere Darstellung der Koordinaten würde zu einer derart großen Zahl von invertierten Listen führen, dass deren Verwaltung die eigentliche Suche verlangsamen würde.

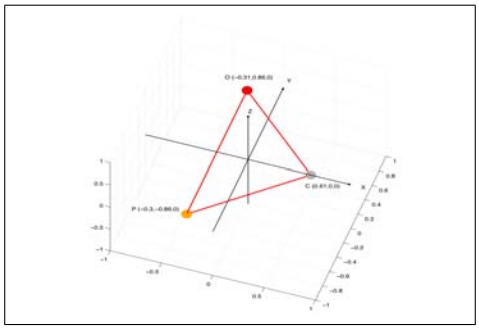


Abbildung 6.11: Atomtripel in der xy-Ebene. Hieraus ergibt sich der Repräsentant COP_$|x_C|$_$|x_O|$_$|y_O|$_$|x_P|$_$|y_P|$ = COP_0.61_0.31_0.66_0.3_0.66.

Entscheidend für die Bestimmung eines Repräsentanten ist die Reihenfolge, in der die zu einem Tripel gehörenden Atome gesehen werden. Die zuvor beschriebenen Schritte zur Berechnung des Gruppenelements liefern hierzu die notwendigen Informationen durch die Reihenfolge der Atome. In Abbildung 6.11 wird die Bestimmung des Repräsentanten veranschaulicht. Das gewählte Repräsentantensystem R ergibt sich formal als Teilmenge von $\mathcal{A} \times \mathcal{A} \times \mathcal{A} \times \mathbb{R} \times \mathbb{R}^2 \times \mathbb{R}^2$.

6.7 Fehlertoleranz

Da im Allgemeinen nicht ausgeschlossen werden kann, dass eine Anfrage in sich fehlerhaft ist, sind Fehlertoleranzmechanismen zur inhaltsbasierten Suche von fundamentaler Bedeutung. In diesem Abschnitt werden auf die Suche in Moleküldaten zugeschnittene Fehlertoleranzmechanismen vorgestellt.

6.7.1 Fehlstellensuche

Hinsichtlich der in Abschnitt 2.2.3 diskutierten Suche mit Erlauben von Fehlstellen ergibt sich bei der Suche in Moleküldaten die Schwierigkeit, dass *ein* Atom in *mehreren* Atomtripel vorhanden sein kann. Folglich wirkt sich ein Atom, welches fehlerhaft in einer Anfrage spezifiziert wird, auf alle Atomtripeln aus, an denen es beteiligt ist. Somit ist die intuitive Bedeutung des Parameters k für die Anzahl der erlaubten Fehlstellen bei einer Anfrage nicht mehr gegeben.
Zunächst kann festgehalten werden, dass der Parameter $k \geq 0$ unterschiedliche Interpretationen zulässt. Am Beispiel von $k = 2$ soll dies verdeutlicht werden. Wie wir zuvor gesehen haben, wird aus der Atomkonstellation der Anfrage Q die Menge $Q_\triangle$ berechnet, die Dreiermengen von Atomen enthält, wobei jedes Atom aus Q in mindestens einer Menge in $Q_\triangle$ enthalten ist. Erlaubt ein

Benutzer zwei Fehlstellen bzgl. Q, so kann sich dies auf $Q_\triangle$ ganz unterschiedlich auswirken: im günstigsten Fall befinden sich die beiden Fehlstellen in genau einem Atomtripel. Dies entspricht einer 1-Fehlersuche in der -Welt. Im ungünstigsten Fall, wenn Q ein Stern mit n Knoten und eine der beiden Fehlstellen gerade das Zentrum des Stern ist, so enthält die n 2 -elementige Menge $Q_\triangle$ ausschließlich fehlerhafte Atomtripel. Umgekehrt können k erlaubte Fehlstellen in der -Welt bis zu $3k$ falsche Atome in Q bedeuten. Typischerweise gleichen daher erlaubte Fehlstellen in Q vorzugsweise fehlerhaft spezifizierte Atome aus, welche nur mit sehr wenigen anderen Atomen (meist nur einem Atom) verbunden sind, da diese aufgrund der Art der Berechnung von $Q_\triangle$ an nur einem oder wenigen Atomtripeln beteiligt sind.

Neben der Tatsache, dass die deutlich geringere Anzahl von Elementen in $Q_\triangle$ gegenüber $Q_\triangle$ die Anfragebearbeitung beschleunigt, spricht für die Verwendung von $Q_\triangle$ im Bezug auf Fehlstellen außerdem, dass die Elemente der Menge unabhängiger voneinander sind. Da es mehrere Überdeckungen gleicher Kardinalität von $Q_\triangle$ zu einer Anfrage Q gibt, gehen wir in der Praxis davon aus, dass die berechnete Menge $Q_\triangle$ über möglichst wenige Elemente verfügt und dass die Anzahl von unabhängigen Elementen in $Q_\triangle$ möglichst groß sein soll.

Mit Hilfe des Konzepts der Fuzzy-Suche lässt sich dieses Prinzip noch derart erweitern, dass unterschiedliche Überdeckungen von Q als Fuzzy-Anfrage formuliert werden, um auf diese Weise zu erreichen, dass möglichst viele Atome in Q in nur einem Atomtripel enthalten sind. Das folgende Beispiel soll diese Vorgehensweise verdeutlichen:

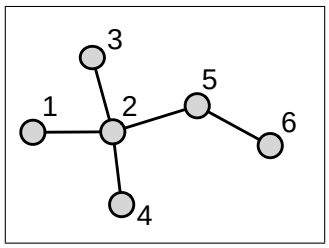


Abbildung 6.12: Zur Überdeckung sind drei 2-Pfade optimal; davon gibt es insgesamt fünf.

Beispiel 6.1 Abbildung 6.12 zeigt einen Baum mit 6 Knoten. Folgende minimale Überdeckungen mit 2-Pfaden sind möglich: $Q_{1\triangle} = \{\{1,2,3\},\{1,2,4\},\{2,5,6\}\}$, $Q_{2\triangle} = \{\{1,2,5\},\{2,3,4\},\{2,5,6\}\}$, $Q_{3\triangle} = \{\{1,2,3\},\{2,4,5\},\{2,5,6\}\}$, $Q_{4\triangle} = \{\{1,2,4\},\{2,3,5\},\{2,5,6\}\}$, $Q_{5\triangle} = \{\{1,2,3\},\{2,3,4\},\{2,5,6\}\}$.

Es fällt auf, dass der Knoten 6 in allen Überdeckungen nur in einem Atomtripel vorkommt wohingegen der Knoten 2 in immer genau zwei Atomtripeln zu finden ist. Der Hauptunterschied zwischen $Q_{1\triangle}$ und den anderen ist, dass in $Q_{1\triangle}$ der Knoten 5 in nur einem Tripel vorkommt. In allen anderen Überdeckungen ist dieser Knoten immer an zweien beteiligt. Somit würde es Sinn machen, aus z.B. $Q_{1\triangle}$ und $Q_{2\triangle}$ eine Fuzzy-Anfrage zu generieren, damit eine maximale Anzahl von Knoten in genau einem Atomtripel vorkommt. Dies erhöht die Flexibilität im Fall der Fehlstellensuche.

In Analogie zum vorherigen Beispiel können auch beliebige Elemente von $Q_\triangle$ genutzt werden, um eine Anfrage $Q_{1\triangle}$ derart zu ergänzen, dass möglichst viele unterschiedliche Atome in möglichst unabhängigen Tripeln enthalten sind.

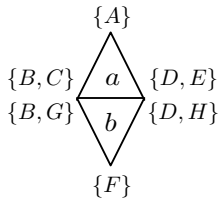
6.7.2 Fuzzy-Anfragen

Im Kontext der allgemeinen Vorgehensweise besteht eine Fuzzy-Anfrage $\mathbf{F}$ hier aus einer Folge $(F_1, \ldots, F_n)$ von endlichen, nichtleeren Teilmengen F_i unseres Universums U bestehend aus gewissen Dreiermengen aus $\mathbb{R}^3 \times \mathcal{A}$. Die zu $\mathbf{F}$ gehörige Treffermenge $G_{\mathcal{D}_\triangle}(\mathbf{F})$ besteht aus allen Paaren (g, i) $SE(3) \times [1 : N]$, zu denen es Atomtripel a_1 $F_1, \ldots, a_n$ F_n gibt, so dass $g\{a_1, \ldots, a_n\} \subseteq (D_i)_\triangle$ gilt. Die Berechnung kann wieder über die Formel

$$G_{\mathcal{D}_\triangle}(\mathbf{F}) = \bigcap_{i=1}^{n} \bigcup_{a_i \in F_i} G_{\mathcal{D}_\triangle}(a_i)$$

erfolgen. Das folgende Beispiel soll illustrieren, warum hier letztendlich doch kein Problem auftritt, obwohl es anfangs so aussieht.

Beispiel 6.2 Die Anfrage $Q_\triangle$ beinhalte zwei Atomtripel a, b, wobei jedem Atom eine Menge von Atomnamen zugewiesen ist.



Dies führt zu folgenden Mengen von Alternativen:
$F_1 = \{(A, B, D), (A, B, E), (A, C, D), (A, C, E)\}$ und
$F_2 = \{(F, B, D), (F, B, H), (F, G, D), (F, G, H)\}$.

Betrachtet man nun die Berechnung der Treffermenge $G_{\mathcal{D}_\triangle}(\mathbf{F})$, so stellt man fest, dass es möglich wäre, dass ein Element (g, i) $G_{\mathcal{D}_\triangle}(\mathbf{F})$ das erste Element aus F_1 und z.B. das letzte Element von F_2 in ein Dokument $(D_i)_\triangle$ transportiert. Rein theoretisch wäre dies ein gültiger Fuzzy-Treffer, obwohl sich z.B. an einer Position im Raum sowohl ein Atom des Elements B als auch ein Atom des Elements G befinden müsste. Da jedoch in einem Molekül D_i keine zwei Atome am selben Ort im Raum vorkommen, tritt dieser Fall in der Praxis nicht auf. Somit bleiben hier die jeweils ersten Elemente die einzig sinnvolle Wahl für Elemente aus F_1 und F_2. Um die Anfrage zu beschleunigen, bietet es sich daher an, die Mengen von Elementnamen, die sich auf ein und den selben Ort im Raum beziehen, durch die Berechnung der Schnittmenge vor der Anfrage zu verkleinern.

Aus Effizienzgründen verfolgen wir in diesem Abschnitt eine zum gerade beschriebenen allgemeinen Vorgehen andere Strategie, indem wir die Fuzzifizierung in drei Etappen aufteilen:

1. Gegenüber einer gewöhnlichen (scharfen) Anfrage Q, die man als Abbildung $Q\colon X$ $\mathcal{A}$ auffassen kann, wobei X eine endliche Teilmenge von $\mathbb{R}^3$ ist, wird die $\mathcal{A}$-Information flexibler gestaltet, indem man einer Ortskoordinate eine endliche, nichtleere Teilmenge von $\mathcal{A}$ zuordnen darf. Statt Q betrachtet man also zunächst $\mathcal{A}$-Fuzzy-Anfragen, die modelliert werden können als Abbildungen X $2^{\mathcal{A}}$ $\{\ \}$.

2. Zusätzliche „Fuzzyness“ wird dadurch erreicht, dass die Anfrage implizit um Tripel erweitert wird, die jeweils Elementen aus Q_D ähnlich sind. Auf diese Weise wird der Fall modelliert, dass die Atome um einen Wert $\epsilon > 0$ von der angefragten Position abweichen dürfen. Prinzipiell ist dies zusammen mit dem vorherigen Punkt die klassische Fuzzysuche. Im Detail befasst sich der nachfolgende Abschnitt mit dieser Thematik.

3. Trefferverschmelzung durch Clusterbildung in der Gruppe $SE(3)$. Diese wird notwendig, da durch die Erweiterung der Anfrage um ähnliche Repräsentanten der Gleichheitsbegriff in der Gruppe $SE(3)$ dem Ähnlichkeitsbegriff auf Repräsentanten angepasst werden muss.

6.7.2.1 Bestimmung des Schwellwerts bei Fuzzyanfragen

Die Fuzzysuche verläuft prinzipiell sehr ähnlich zu der Suche ohne Fuzzyinformation in der Anfrage. Berechnet man die Treffermenge mittels Summation von charakteristischen Funktionen, so muss zuvor ein Schwellwert bestimmt werden, ab dem ein Tupel (g, i) als Treffer charakterisiert wird.
Sei nun zunächst $\mathbf{F}\colon X \quad 2^{\mathcal{A}} \quad \{\ \}$ eine sog. *$\mathcal{A}$-Fuzzy-Anfrage* wie in dem Beispiel zuvor beschrieben. Dieses $\mathbf{F}$ kann als Fuzzy-Molekül, also als Menge $Q(\mathbf{F})$ aller Moleküle aufgefasst werden, wobei jedes Molekül in der Menge genau einer Wahlmöglichkeit über alle Fuzzy-Atome entspricht:

$$Q(\mathbf{F}) := \{Q\colon X \quad \mathcal{A} \mid \ x \quad X : Q(x) \quad \mathbf{F}(x)\}.$$

Um nun den Schwellwert für eine Fuzzysuche mit oder ohne Fehlstellen zu bestimmen, suchen wir nach dem Molekül Q aus $Q(\mathbf{F})$, für das $|Q_{\triangle}|$ minimal wird.
Aufgrund der Tatsache, dass die Atome in einem Molekül einen kovalenten Bindungsradius aufweisen, der von dem Elementtyp abhängig ist, können unterschiedliche Elemente in $Q(\mathbf{F})$ zu unterschiedlichen Überdeckungen führen. Dieses Molekül ist dann zu benutzen, um den Schwellwert für die Fuzzysuche zu bestimmen.
Um nun das Molekül $Q = \mathbf{F}_{\text{min}}$ zu bestimmen, bemerken wir, dass durch die Wahl von Atomen mit einem minimalen Bindungsradius die Möglichkeit gegeben ist, dass ein Molekül in mehrere Teilbäume zerfällt. Zusammen mit der Tatsache, dass sich in einem Molekül an einer Position im Raum keine zwei Atome gleichzeitig befinden können, ist die Bestimmung von $\mathbf{F}_{\text{min}}$ durch Wahl der Atome mit jeweils kleinstem Bindungsradius möglich. Hier hilft die gemachte Beobachtung, dass $pc(\Gamma') \geq pc(\Gamma)$ ist, wenn Γ' aus Γ durch Streichen von Kanten hervorgeht:

$$\mathbf{F}_{\text{min}} = \{q_1, \ldots, q_n\}, q_i \quad F_i \text{ und } \mathrm{Rad}_{\text{cov}}(q_i) \text{ ist minimal bzgl. } F_i.$$

Weder $\mathbf{F}_{\text{min}}$ noch der zugehörige Graph sind im allgemeinen eindeutig bestimmt. Auf die sehr technischen Details sei hier verzichtet.

6.7.3 Automatische Anfrageerweiterung

Im vorhergehenden Abschnitt wurde auf die Erweiterung der $\mathcal{A}$-Fuzzy-Anfragen durch die Verwendung von ähnlichen Atomtripeln hingewiesen. Die Treffermenge $G_{\mathcal{D}_{\triangle}}(\mathbf{F})$ zu einer klassischen Fuzzyanfrage $\mathbf{F}$ besteht bekanntlich aus allen Paaren $(g, i) \quad SE(3) \times [1 : N]$, zu denen es Atomtripel $a_1 \quad F_1, \ldots, a_n \quad F_n$ gibt, so dass $g\{a_1, \ldots, a_n\} \subseteq (D_i)_{\triangle}$ gilt. Bei $\mathcal{A}$-Fuzzyanfragen enthalten die Mengen F_i immer die gleichen Atomkoordinaten, jedoch mit unterschiedlichen Atomnamen.
Bei der Suche in Moleküldaten liegt es aber in der Natur der Sache, dass die Atomkoordinaten fehlerbehaftet sind. Daher macht es Sinn nach Treffern zu fragen, bei denen die Atomkoordinaten zwischen Anfrage und Dokument nicht genau übereinstimmen müssen, sondern bis zu einem Wert ϵ abweichen dürfen. Dies führt prinzipiell zu unendlichen Fuzzymengen F_i, die neben Wahlmöglichkeiten für die Atomnamen auch Fehlerkugeln vom Radius ϵ um jede Atomkoordinate beinhalten. Derartige Anfragen bezeichnen wir als *ϵ-Fuzzyanfragen*.

De nition 6.8 *Zwei 2-Pfade $r = \{r_0, r_1, r_2\}, s = \{s_0, s_1, s_2\} \subset \mathbb{R}^3 \times \mathcal{A}$ heißen* ϵ-ähnlich*, wenn es eine Permutation σ gibt, so dass für alle $i \quad [0 : 2]$ gilt:*

$$\left\|\pi_1(r_i) - \pi_1(s_{\sigma(i)})\right\| \leq \epsilon \qquad \pi_2(r_i) = \pi_2(s_{\sigma(i)}).$$

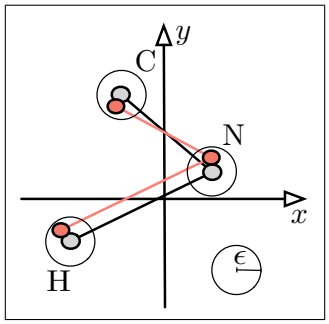


Abbildung 6.13: Die Abbildung zeigt zwei ϵ-ähnliche 2-Pfade r, s in der Ebene: Es gibt eine Korrespondenz zwischen den Punkten von r und denen von s, so dass jeder Punkt von s in der ϵ-Umgebung des korrespondierenden Punktes von r liegt und die Elementtypen korrespondierender Punkte übereinstimmen.

Dies bedeutet, dass die Ortskoordinaten von zwei korrespondierenden Atomen in r und s maximal einen Abstand von ϵ aufweisen dürfen bei gleichen Atomnamen. In Abbildung 6.13 sind zwei ϵ-ähnliche 2-Pfade dargestellt.
Erinnern wir uns zunächst an die Berechnung der Treffermenge zu Fuzzyanfragen, wie sie zuvor beschrieben worden ist. Dabei werden zunächst die Treffermengen für alle Elemente aus einer Menge F_i vereinigt, um allen durch F_i beschriebenen Möglichkeiten Rechnung zu tragen. Im zweiten Schritt wird die Schnittmenge aus allen Zwischenergebnissen berechnet, welche dann die Treffer in Form von Paaren (g, i) enthält.
Nun sind im Fall der Molekülsuche die Mengen F_i unendlich groß, da die Menge zu einem 2-Pfad $t_\triangle$ alle 2-Pfade enthält, deren Knoten über die gleichen Atomnamen verfügen und bei denen der Abstand der jeweils korrespondierenden Knoten kleiner oder gleich ϵ ist. Allerdings ist die Zahl der ϵ-ähnlichen 2-Pfade, für die $G_{\mathcal{D}_\triangle}(t_\triangle) =$ endlich, da die Dokumentenkollektion $\mathcal{D}_\triangle$ nur endlich viele Elemente beinhaltet. Dazu kommt noch, dass bei der Repräsentantenberechnung die Ortskoordinaten gerundet werden, so dass auch dadurch die Zahl der bei einer Anfrage zu betrachtenden G-invertierten Listen ebenfalls reduziert wird.
Aus diesem Grund beschäftigen wir uns jetzt mit folgender Frage: Gegeben sei ein 2-Pfad $t_\triangle$. Mit $T^\epsilon_{t_\triangle}$ bezeichnen wir die Menge aller zu $t_\triangle$ ϵ-ähnlichen 2-Pfade. Es sei $R^\epsilon_{t_\triangle}$ die Menge der zu den Elementen aus $T^\epsilon_{t_\triangle}$ gehörigen Repräsentanten. Diese Menge von Repräsentanten gilt es zu charakterisieren, denn für die Beantwortung einer ϵ-Fuzzyanfrage müssen alle Listen betrachtet werden, deren Repräsentanten sich aus der Menge $T^\epsilon_{t_\triangle}$ ergeben und im Suchindex vorhanden sind.

Sei nun $t_\triangle$ $Q_\triangle$ ein 2-Pfad mit den Knoten t_0, t_1, t_2 $\mathbb{R}^3$, $t_i = (t_i^x, t_i^y, t_i^z)$. Wie in dem Abschnitt über die Berechnung von Repräsentanten geschrieben, werden die Knoten in $t_\triangle$ gemäß der geometrischen Beschaffenheit des 2-Pfades sortiert. Der erste Knoten liegt der Hypothenuse des durch $t_\triangle$ beschriebenen Dreiecks gegenüber und ist auf dem positiven Bereich der x-Achse lokalisiert. Der zweite Knoten hat den zweitgrößten x-Wert und hat einen positiven y-Wert, der größer ist als der des dritten Knotens. O.B.d.A. seien hier nun t_0, t_1, t_2 in dieser Reihenfolge. Die Wahl von ϵ und die geometrischen Eigenschaften von $t_\triangle$ haben entscheidenden Einfluss darauf, wie aufwändig die Berechnung der Menge $R^\epsilon_{t_\triangle}$ ist.
Im einfachsten Fall hat die gewährte Fehlertoleranz keinen Einfluss auf die Reihenfolge der Atome im Repräsentanten. Dies bedeutet, dass alle Repräsentanten in $R^\epsilon_{t_\triangle}$ die gleiche Namenskomponente (vgl. Abbildung 6.11) aufweisen und sich lediglich in den gerundeten Koordinaten unterscheiden. Da der

erste Knoten eines Repräsentanten eines zu $t_\triangle$ ϵ-ähnlichen 2-Pfades ebenfalls auf der x-Achse liegt und dieser höchstens einen Abstand von ϵ von t_0 hat, liegt dieser in dem Intervall $[t_0^x - \epsilon, t_0^x + \epsilon]$. Die Abbildung 6.14 verdeutlicht dies.

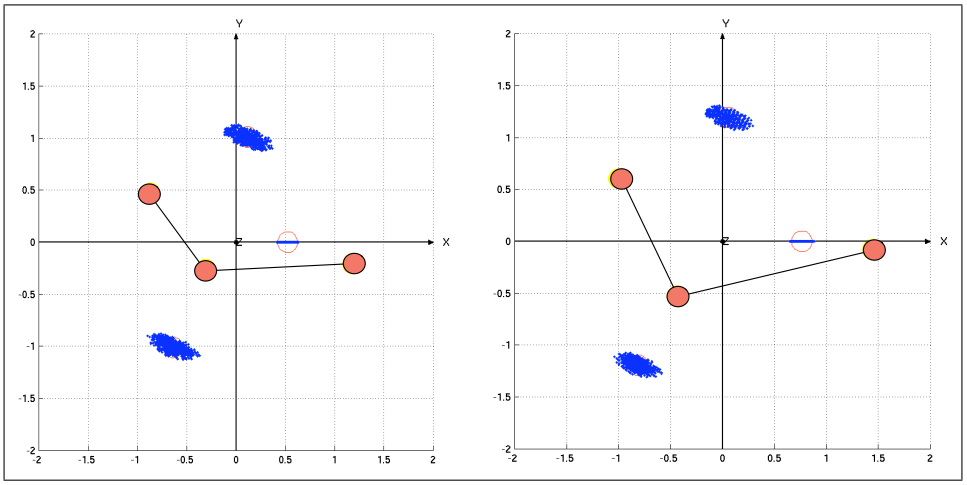


Abbildung 6.14: Ein Beispiel für zwei unterschiedliche 2-Pfade und die daraus resultierenden Möglichkeiten für Repräsentanten. Deutlich zu erkennen ist der Wertebereich auf der x-Achse für den ersten Knoten im Repräsentanten, wohingegen die Möglichkeiten für die Ortskoordinaten für die beiden verbleibenden Atome im Tripel deutlich variabler sind.

In dem jetzt folgenden Beispiel soll die Hypothenuse des durch $t_\triangle$ beschriebenen Dreiecks eindeutig bestimmt sein, auch wenn sich die Ortskoordinaten der Knoten in dem 2-Pfad um ϵ variieren dürfen. Dies bedeutet, dass der Punkt gegenüber der Hypothenuse fest bleibt, jedoch die beiden anderen Punkte des Repräsentanten nicht festgelegt sind, wie sich aus Schritt 5 der Repräsentantenberechnung ergibt. Somit kann es eine Vertauschung der letzten beiden Atome im Repräsentanten geben, die bei der Bestimmung der Menge $R^\epsilon_{t_\triangle}$ berücksichtigt werden müssen.
Im nachfolgenden Fall gibt es mindestens zwei Kanten in dem Dreieck zu $t_\triangle$, deren Längen sich um weniger als ϵ unterscheiden. In diesem Fall ist es möglich, dass sich die Reihenfolge der Atome, auf der die Repräsentantenberechnung beruht, verändert, da die Hypothenuse des Dreiecks z.B. durch (t_0, t_1) oder aber auch durch (t_0, t_2) gebildet wird. Dies hätte Einfluss auf die Namenskomponente des Repräsentanten als auch auf die Ortskoordinaten und tritt häufig bei nahezu gleichseitigen Dreiecken auf. Hier unterliegt die Namenskomponente des Repräsentanten häufigen Änderungen je nachdem, welcher zu $t_\triangle$ ϵ-ähnliche 2-Pfad betrachtet wird. Die Abbildung 6.15 zeigt dazu ein Beispiel.

6.7.3.1 ϵ-Clustering der Zwischenergebnisse

Wir starten mit einem Gedankenexperiment: Unsere Kollektion besteht nur aus einem -überdeckbaren Moleküldokument D_0. Ferner sei Q eine -überdeckbare Anfrage, die in D_0 vorkommt: für geeignetes g $SE(3)$ sei $gQ \subseteq D_0$. Also ist auch $gQ_\triangle \subseteq (D_0)_\triangle$. Desweiteren zerfalle Q disjunkt in -überdeckbare Teile P und R so dass $Q_\triangle = P_\triangle$ $R_\triangle$. Nun translatieren wir alle Elemente aus P und damit aus $P_\triangle$ um den Ortsvektor v der Länge $\leq \epsilon$. So entsteht die neue Anfrage $Q^\epsilon_\triangle := (P_\triangle + v)$ $R_\triangle$. Die neue Anfrage kommt unter allen ϵ-Anfragen zu $Q_\triangle$ vor. Allerdings wird die zuvor mehrfach beschriebene Berechnung der Treffermenge zu keinem Ergebnis führen, denn es

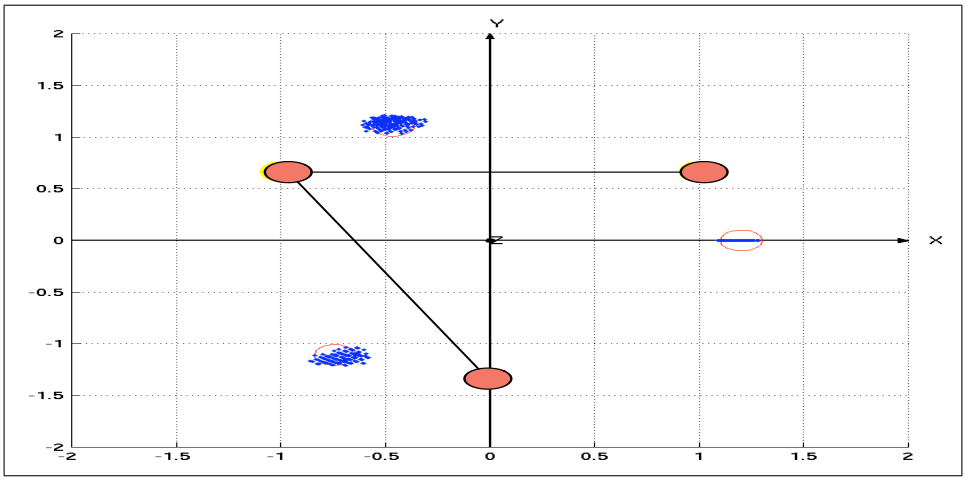


Abbildung 6.15: Beispiel für einen 2-Pfad, der beinahe ein gleichseitiges Dreieck beschreibt. Je nach Wahl einer Ortskoordinate für die drei Knoten des 2-Pfades kann die Hypothenuse des Dreiecks durch unterschiedliche Knoten gebildet werden, was wiederum zu einer anderen Reihenfolge der Atome im Repräsentanten führt.

würden sich bei der Berechnung zwei Trefferkandidaten $(g, 0)$ und $(h, 0)$ herauskristallisieren, für die $gR_{\triangle} \subseteq (D_0)_{\triangle}$ und $hP_{\triangle} \subseteq (D_0)_{\triangle}$ gilt. Beide Gruppenelemente g, h transportieren jeweils nur einen Teil der Anfrage $Q^{\epsilon}_{\triangle}$ in das Dokument $(D_0)_{\triangle}$. Das Gruppenelement h berücksichtigt dabei die zuvor durchgeführte Verschiebung der Elemente von $P_{\triangle}$ um v. Da jedoch keines der beiden Gruppenelemente für sich die gesamte Anfrage $Q^{\epsilon}_{\triangle}$ in das Dokument transportiert, ist die Treffermenge zu dieser ϵ-Fuzzyanfrage leer.
Folglich müssen wir uns nun der Frage widmen, was die beiden Gruppenelemente g, h miteinander verbindet, um eine derartige ϵ-Fuzzyanfrage sinnvoll beantworten zu können. Dazu übertragen wir den Begriff der ϵ-Ähnlichkeit auch auf Gruppenelemente $g, h \quad SE(3)$.

De nition 6.9 *Zwei Elemente* $g = (R_g, T_g), h = (R_h, T_h) \quad SE(3)$ *heißen* ϵ-ähnlich *bzgl. Translationen, wenn die Rotationsanteile übereinstimmen und die Translationsanteile höchstens den Abstand* ϵ *haben, d.h.* $R_g = R_h$ *und* $T_g - T_h \leq \epsilon$.

Im vorhergehenden Beispiel sind die Gruppenelemente g, h ϵ-ähnlich, da sich deren Translationsanteile um weniger als ϵ voneinander unterscheiden, denn der Unterschied in der Translationskomponente ist gerade v .
Allgemeines Vorgehen:

1. Für jede Liste zur Anfrage wird zunächst die Nachjustierung vorgenommen, um vom Repräsentanten zur eigentlichen Liste zu kommen, danach wird für jedes Listenelement geprüft, ob es zu einem bereits bestehenden ϵ-Cluster in der Gruppe gehört. Falls dies zutrifft, wird das Listenelement dem Cluster hinzugefügt; falls nicht, wird mit diesem Listenelement ein neues Cluster eröffnet.

2. Enthält ein Cluster schließlich mehr Einträge als ein vorgegebener Schwellwert, so stellt dieses Cluster ein Treffer dar. Dieser Treffer wird repräsentiert durch das arithmetische Mittel der Translationskomponenten der Clusterelemente.

6.7.3.2 Erweiterung des Prinzips auf die Rotationskomponente

Das zuvor vorgestellte Prinzip lässt sich auch auf die Rotationskomponente eines Elements aus $SE(3)$ erweitern. Hier erweist sich die Realisierung der Elemente der $SE(3)$ durch Quaternionen als vorteilhaft, da aus der Quaternionendarstellung schnell die Drehachse und auch der Drehwinkel bestimmen lassen. Bei dem Vergleich von zwei Quaternionen werden also zum einen die Translationskomponenten miteinander verglichen und zum zweiten die Rotationsachse und der Rotationswinkel. Für die beiden letzte Parameter können vor der Suche Schwellwerte angegeben werden. Ein Clusterzentrum ist ein Gruppenelement aus $SE(3)$, welches die mittlere Translation aller Gruppenelemente, die dem Cluster angehören, enthält. Weiterhin beinhaltet das Clusterzentrum eine mittlere Rotationsachse und einen mittleren Drehwinkel.
Notwendig wird dies, da wir ja zuvor ϵ-Fehlerkugeln um die Koordinaten der angefragten Atome in Q zugelassen haben. Folglich werden auch Atomtripel betrachtet, deren induziertes Dreieck eine andere Ebene beschreibt als das Dreieck, welches durch das ursprüngliche Atomtripel beschrieben wird. Um die daraus resultierenden Unterschiede in der Rotationsachse und auch -winkel zu kompensieren, muss bei der Clusterung von Gruppenelementen auch die Rotationskomponente mitbeachtet werden.

6.8 Aminosäuren als Indexobjekte

Nachdem wir nun Elementarobjekte zur Indexierung aus einzelnen Atomen betrachtet haben, soll es in diesem Abschnitt um Aminosäuren als Elementarobjekte gehen. Die Grundbausteine der Proteine sind die sog. *proteinogenen* Aminosäuren (Auflistung z.B. in [BJ05, web06a]), die allesamt α-Aminosäuren sind (vgl. Abbildung 6.16). Es liegt also nahe eine Suchindex auf der semantischen Ebene der Aminosäuren eine inhaltsbasierte Suche in den Molekül̈daten z.B. der PDB ermöglicht. Wie wir sehen werden, ändern sich die Elementarobjekte, jedoch können die Überlegungen aus dem vorherigen Abschnitt bzgl. der Gruppe $SE(3)$ direkt übertragen werden.

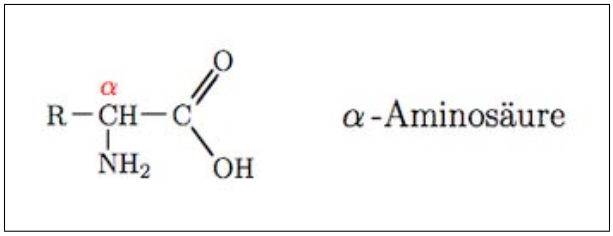


Abbildung 6.16: Die *proteinogenen* Aminosäuren sind α-Aminosäuren (Abbildung aus [web06a]), d.h. die Amino-Gruppe ($-NH_2$) befindet sich am α-C-Atom, welches das erste, mit der Carboxylgruppe ($-COOH$) verbundene C-Atom ist. Aminosäuren dieser Art bilden die Grundbausteine der Proteine. Die Unterscheidung der Aminosäuren erfolgt durch den sog. *Rest* (engl. residue) $-R$. Siehe auch [BJ05, MM03, HCH02].

Ein Vorteil dieser Vorgehensweise besteht in der deutlichen Reduktion der Einträge in den Suchindex, da pro Aminosäure mindestens 9 Atome bzw. 5 daraus folgenden 2-Pfade zu einem Indexeintrag zusammengefasst werden. Je nachdem welcher Rest $-R$ mit dem α-C-Atom verbunden ist, können dies auch deutlich mehr Atome sein.
Wie wir sehen werden, geht mit der Betrachtung von Aminosäuren eine Vergröberung des Trefferbegriffes im Vergleich zum vorherigen Abschnitt einher, da die räumliche Struktur einer jeden Ami-

nosäure bei der Indexierung im wesentlichen durch einen Vektor zwischen den Ortskoordinaten des α-C-Atoms und dem Schwerpunkt der Aminosäure repräsentiert wird. Diese grobe Darstellung, wie sie auch in Abbildung 6.18 dargestellt ist, bietet dem Benutzer eher die Möglichkeit, eine Anfrage manuell zu bearbeiten, z.B. durch das Hinzufügen von Fuzzy-Elementen.
Man kann die inhaltsbasierte Suche auf der Ebene der Aminosäuren auch als eine Art Vorverarbeitungsschritt für andere Algorithmen sehen. So kann nachfolgend eine Suche auf atomarer Ebene durchgeführt werden, wobei die Menge der möglichen Trefferdokumente in Anlehnung an Abschnitt 3.6 geeignet eingeschränkt wird. Eine weitere Anwendungsmöglichkeit bestünde auch darin, die Trefferstellen aus der inhaltsbasierten Suche als Eingabe für Algorithmen zu verwenden, die genauere, aufwändig zu berechnende Vergleiche durchführten.

6.8.1 Die Peptidbindung

Über die *Peptidbindung* verbinden sich Aminosäuren zu Ketten und bilden sog. *Polypeptide*. Besteht ein solches Polypeptid aus mehr als 100 Aminosäuren, wird von Proteinen gesprochen. Die Peptidbindung wird zwischen der Carboxylgruppe $(-COOH)$ einer Aminosäure und der Aminogruppe $(-NH_2)$ einer anderen Aminosäure unter Abspaltung eines Wassermoleküls etabliert.

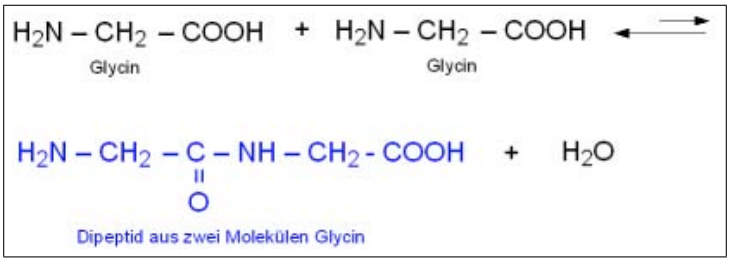


Abbildung 6.17: Die Peptidbindung zwischen zwei Aminosäuren (hier *Glycin*). Formal kondensieren zwei Aminosäuren unter Abgabe eines Wassermoleküls zu einem *Dipeptid* (Abbildung aus [web06b]).

6.8.2 Bestimmung der Elementarobjekte

Abbildung 6.18 zeigt eine Sequenz von Aminosäuren, die durch die Peptidbindung eine Kette bilden. Die in rot dargestellten Punkte entsprechen den Ortskoordinaten der α-C-Atome einer jeder Aminosäure. Blau dargestellt sind die Schwerpunkte einer jeden Aminosäure. Auf diese Wiese erhalten wir eine Ortskomponente und dazu eine Raumrichtungskomponente, die grob die räumliche Struktur der jeweiligen Aminosäure im Kontext des Polypeptids bzw. Proteins widerspiegelt.
Wie bereits in den Abschnitten zuvor diskutiert worden ist, führt die Wahl eines Repräsentantensystems R bestehend aus Repräsentanten r, welche sich aus dem Namen der jeweiligen Aminosäure (z.B. GLY, VAL...) und dem euklidschen Abstand zwischen dem α-C-Atom und dem Schwerpunkt der zugehörigen Aminosäure zusammensetzen, i.A. zu nicht-trivialen Stabilisatoren. Denn wenn die Drehachse der Drehkomponente von g $SE(3)$ mit der Richtung des Vektors vom α-C-Atom zum Schwerpunkt übereinstimmt, gibt es unendlich viele Drehwinkel, um die ein Repräsentant r R gedreht werden kann ohne den Repräsentanten zu verändern. Hier wurde vorausgesetzt, dass die Translationskomponente von g trivial ist.
Um diese Mehrdeutigkeiten in nahezu allen Fällen zu verhindern, wird analog zur Suche auf atomarer Ebene auch hier ein dritter Punkt im Raum benötigt. O.B.d.A. wählen wir dazu die Ortskoordinate des

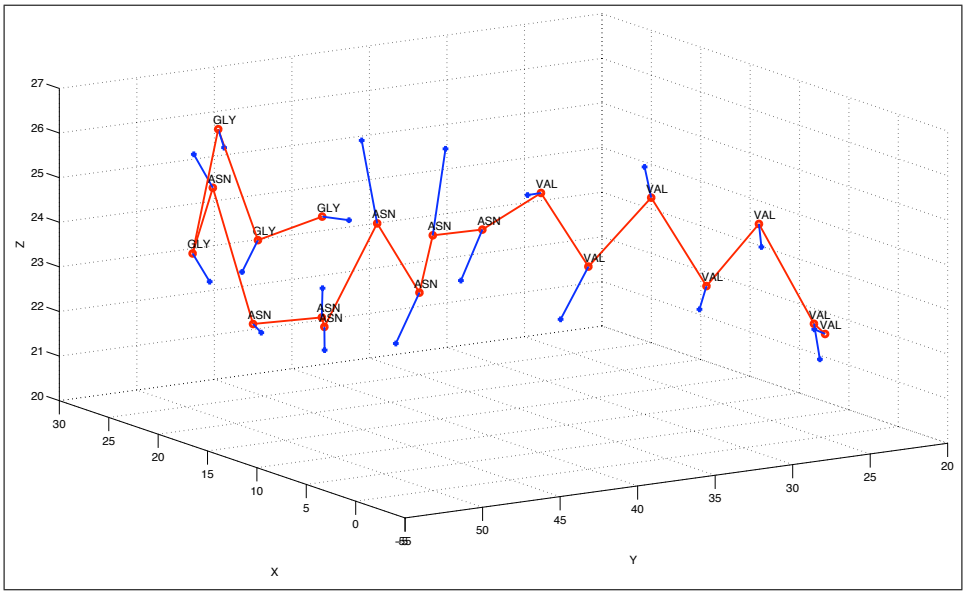


Abbildung 6.18: Darstellung eines Polypeptids, reduziert auf die Ortskoordinaten der α-C-Atome der Aminosäure in Relation zum Schwerpunkt der jeweiligen Aminosäure.

Stickstoffatoms (N), welches zur Peptidbindung beiträgt. Auf diese Weise erhalten wir triviale Stabilisatoren, es sei denn, die drei Raumkoordinaten liegen kolinear zueinander, was jedoch aufgrund der Struktur der Peptidbindung nahezu ausgeschlossen werden kann. Ein Repräsentant setzt sich daher zusammen aus dem Namen der jeweiligen Aminosäure (vgl. Abbildung 6.19), dem euklidschen Abstand zwischen dem α-C-Atom und dem Schwerpunkt der Aminosäure und als letzte Komponente dem euklidschen Abstand zwischen dem α-C-Atom und dem N-Atom der Aminosäure, welches zur Peptidbindung beiträgt. Wie wir sehen werden, lässt sich die Anzahl der G-invertierten Listen dadurch steuern, dass die Abstände auf eine zuvor festgelegte Anzahl von Nachkommastellen gerundet werden. Verzichtete man ganz auf die Verwendung der Abstände, so ergäben sich die in Abbildung 6.19 aufgezeigten Listenlängen. Allerdings stellt dies sicherlich einen Extremfall von Fehlertoleranz dar, ist doch die Zahl der bei einer Anfrage zu betrachtenden Listeneinträge ein Indiz für die Zeit, die zur Beantwortung der Anfrage benötigt wird.

6.8.2.1 Beispieldatensatz

Zur Illustration der Datenmengen, werden im folgenden drei Szenarien vorgestellt. Die Dokumentenkollektion wird durch die ersten (lexikographische Ordnung auf den PDB-IDs) 10.000 PDBML-Dokumente dargestellt. Insgesamt enthalten diese 5,7 Millionen Aminosäuren, was der Anzahl der zu speichernden Einträge in den G-invertierten Listen entspricht (vgl. Abbildung 6.19).

6.8.2.1.1 Rundung auf eine Nachkommastelle Die Abbildung 6.20 zeigt die Verteilung der 5,7 Millionen Einträge in die G-invertierten Listen, wobei die Abstände, die bei der Bestimmung eines Repräsentanten berechnet wurden, auf eine Nachkommastelle gerundet worden sind.

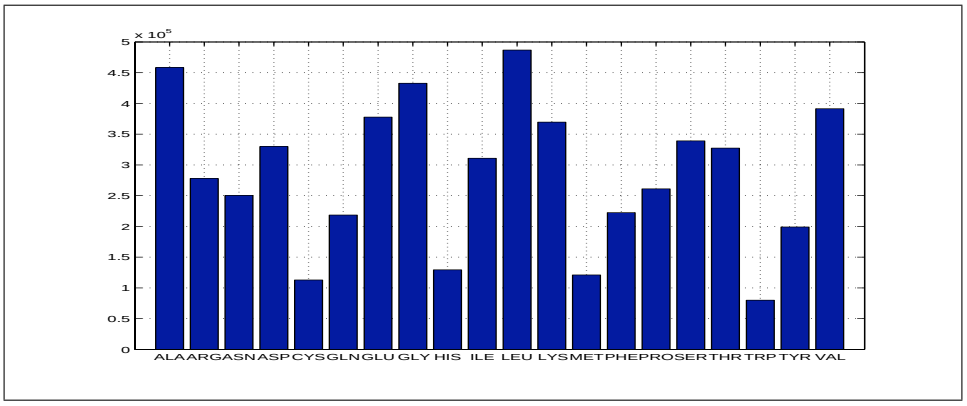


Abbildung 6.19: Häufigkeiten der proteinogenen Aminosäuren bezogen auf die ersten 10.000 Dokumente der PDB.

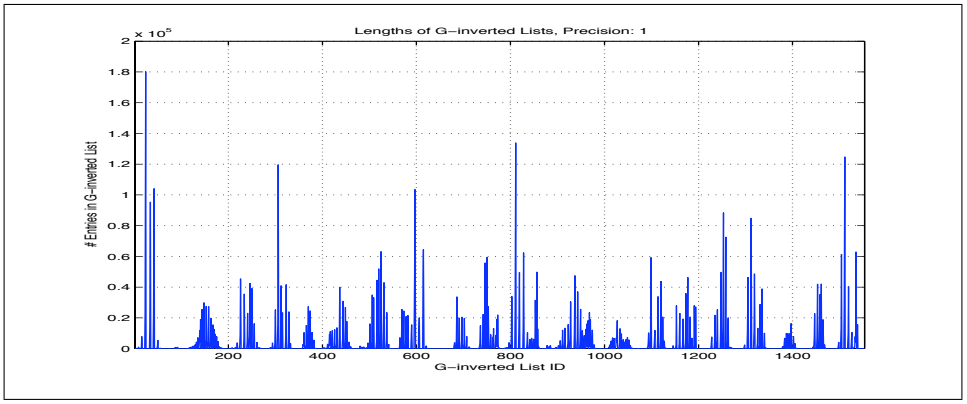


Abbildung 6.20: Verteilung der 5,7 Millionen Indexeinträge auf die 1.553 G-invertierten Listen. Im Mittel enthält eine Liste rund 3.700 Einträge.

6.8.2.1.2 Rundung auf drei Nachkommastellen Die Abbildung 6.21 zeigt die Verteilung der 5,7 Millionen Einträge in die G-invertierten Listen, wobei die Abstände, die bei der Bestimmung eines Repräsentanten berechnet wurden, auf drei Nachkommastellen gerundet worden sind.

6.8.3 Ausnutzung der Sequenzinformation

Wie zu Anfang dieses Kapitels beschrieben, enthalten die PDB-Dokumente, die Polypetide und Proteine beschreiben, die Reihenfolge der Aminosäuren als Sequenz von einbuchstabigen Abkürzungen der Namen der proteinogenen Aminosäuren. Diese Information liegt auch zu jeder Anfrage Q vor, wenn diese ein Polypeptid (z.B. ein Teil eines Proteins) ist.
Diese Information kann zunächst genutzt werden, um die inhaltsbasierte Suche auf Dokumente einzuschränken, die eine entsprechende Teilsequenz beinhalten. In einem Vorverarbeitungsschritt wird eine Menge von Dokumenten-IDs bestimmt, die die IDs möglicher Trefferdokumente beinhaltet. Dabei sind Fehlstellen und Fuzzyanfragen ebenfalls zu berücksichtigen. Dazu böte sich ebenfalls die Verwendung eines Suchindexes basierend auf G-invertierten Listen an, wobei das Repräsentantensystem durch die einbuchstabigen Abkürzungen gebildet wird und die Gruppe $G = (\mathbb{Z}, +)$ auf Elementen von

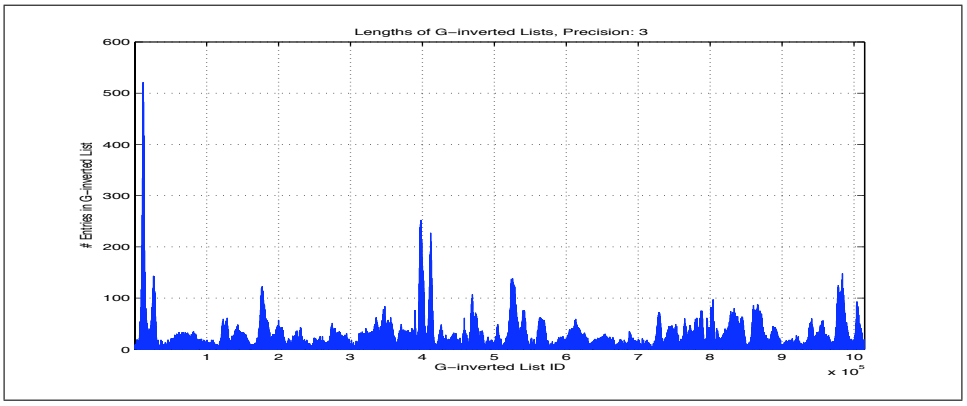


Abbildung 6.21: Verteilung der 5,7 Millionen Indexeinträge auf die 1.014.449 G-invertierten Listen. Im Mittel enthält eine Liste rund 6 Einträge.

$\mathbb{Z}$ operiert. Die Abbildung 6.19 zeigt für diesen Fall beispielhaft die Verteilung von Indexeinträgen auf die G-invertierten Listen. Der Einsatz von Stringmatchingverfahren, die auch z.T. biologisch motiviert sein können [Pea95], ist sicherlich ebenfalls möglich.
Dieser Ansatz lässt sich noch weiter führen. In dem in Kapitel 2 vorgestellten Ansatz wurde von Einträgen in den G-invertierten Listen der Form (g, i) $G \times [0 : N-1]$ ausgegangen, d.h. zu jedem Gruppenelement g wird die zugehörige ID des Dokuments in der Dokumentenkollektion gespeichert. Im dem hier diskutierten Fall sind die Dokumente Polypeptide bzw. Proteine, zu denen eine Sequenzinformation S und damit eine Reihenfolge vorliegt. Daher können wir die Listeneinträge erweitern zu

$$(g, i, s) \quad G \times [0 : N-1] \times [0 : |S|-1].$$

Die Komponente s $[0, \ldots, |S|-1]$ besagt, dass das Gruppenelement g den zugehörigen Repräsentanten r in das Dokument i überführt, wobei dies eben gerade der s-ten Aminosäure in der Sequenz S_i zum i-ten Dokument entspricht. Zusammen mit dieser Information kann das Ergebnis des zuvor beschriebenen Vorverarbeitungsschritts genutzt werden, um die inhaltsbasierte Suche nicht nur auf Dokumente einzuschränken, sondern zusätzlich *auf bestimmte Teile* eines Dokuments.

Kapitel 7

Implementierung einer verteilten Applikation zur Suche in 3D-Molekülдaten

Die in dem vorangehenden Kapitel vorgestellte Applikation zur Suche in Proteinmolekülen wurde als Client-Server Applikation realisiert. Der Client ist ein Java-Programm, welches die Visualisierung von Proteinmolekülen mit OpenGL-Unterstützung auf unterschiedlichen Betriebssystemen ermöglicht. Der Quelltext des an der Universität Osaka entwickelten „jV3"-Programms ist unter [KN] frei erhältlich. Die Abbildung 7.3 zeigt schematisch die Client-Applikation. In diesem Fall ist eine Anfrage geladen worden und wird als Ball&Stick-Modell dargestellt. Das Textfeld unterhalb des Darstellungsbereichs für Moleküle dient der Interaktion mit dem Programm mittels des aus dem „Urvater" dieser Art von Programmen „RasMol" [Say98] bekannten Befehlssatzes. So können einzelne Atome selektiert werden, das Molekül im Raum gedreht werden usw. Letztere Art der Interaktion ist natürlich auch über die Maus als Eingabegerät möglich. Die vor dem Start der Suche selektierten Atome bilden das Anfragedokument Q.
In diesem Kapitel soll die verteilte Architektur und die generische C++ Bibliothek, die die eigentliche Suchfunktionalität bereitstellt, vorgestellt werden. Die Anwendung bearbeitet Anfragen auf atomarer Ebene, allerdings lässt sich ein System, welches auf Ebene der Aminosäuren arbeitet, leicht realisieren, zumal die Gruppe G identisch ist und auch die Modellierung der Elementarobjekte letztendlich auch keinen Unterschied aufweist, denn bei beide Applikationen ergeben sich die Elementarobjekte aus der Verknüpfung von drei Raumkoordinaten. Lediglich das Repräsentantensystem unterscheidet beide Anwendungen.

7.1 Client-Server Architektur

Die Abbildung 7.1 zeigt in einer Übersicht die Architektur der Client-Server Applikation. Auf Seiten des Clients läuft das schon zuvor beschriebene Java-Programm „jV3" erweitert um die Funktionalität, mittels RMI eine Anfrage über ein Netzwerk an die Server-Komponente zu senden. Zusätzlich dazu wurde das Visualisierungstool um die Darstellung von Treffern im Kontext eines Proteins erweitert. Ein Treffer, der dem Client nach einer Suche übermittel wird, besteht aus dem PDB-Namen des jeweiligen Proteins und einen Gruppenelement $g \quad SE(3)$, welches die Anfrage in das jeweilige Protein verschiebt und rotiert. Die eigentlichen Daten des Proteins werden hier nicht übermittelt. Vielmehr wird davon ausgegangen, dass das Visualisierungstool die entsprechende XML-Datei in einem zuvor

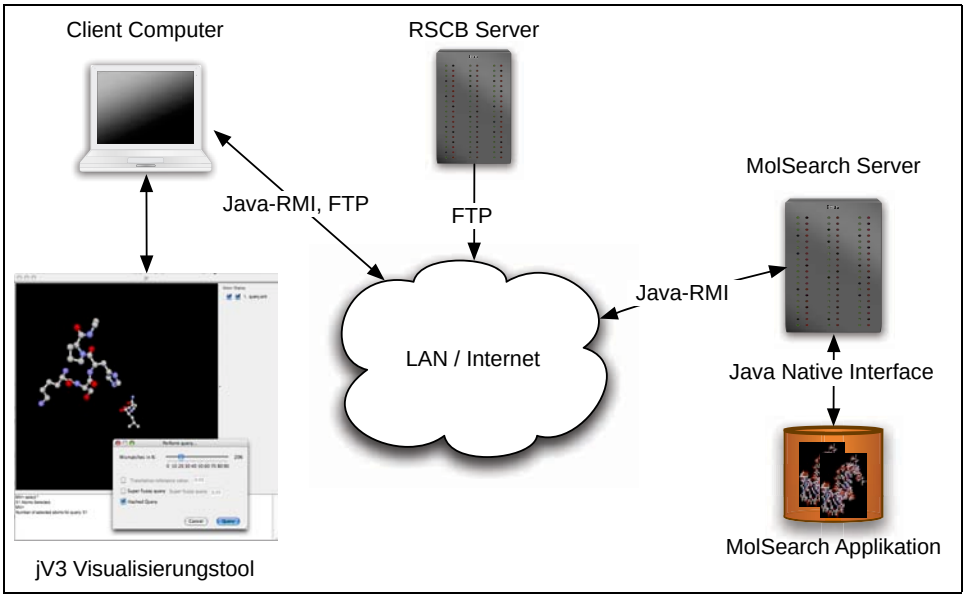


Abbildung 7.1: Aufbau der gesamten Applikation zur Suche in Proteindaten. Als Protokoll zur Verbindung von Client und Server kommt das Java „Remote Method Invocation“-Protokoll (RMI) [Micb] zum Einsatz. Daneben können auch Proteine per FT-Protokoll vom Server der PDB [webc] geladen werden, sofern diese das Ergebnis einer Suchanfrage sind und noch nicht lokal vorliegen.

festgelegten Verzeichnis vorfindet. Ist dies nicht der Fall, besteht die Möglichkeit, die PDB-XML-Datei zu einem Protein automatisch vom FTP-Server der Protein Data Bank in komprimierter Form zu laden und anzuzeigen.
Die sog. „MolSearch“ Server-Komponente ist ebenfalls ein Java-Programm, welche die per RMI von Clients übermittelten Anfragen an eine Instanz des zuvor beschriebenen Suchindexes weiterleitet. Dieser Suchindex ist eine C++ „Dynamic Link Library“, welche ein Interface zur Verfügung stellt, das eine Anbindung an ein Java-Programm über das sog. „Java Native Interface“ [Mica] ermöglicht. Auf diese Weise können die Stärken beider Programmiersprachen, Java und C++, optimal kombiniert werden. Java ist sehr gut geeignet, um ein plattformunabhängige Benutzerschnittstelle in Form des Visualisierungstools zu implementieren. Ausserdem verfügt Java über Technologien, die die Integration von Services über ein Netzwerk ermöglichen. Die Vorteile von C++ liegen eindeutig im Bereich der Ausführungsgeschwindigkeit, weshalb die eigentliche Suche in Form von generischen C++ Algorithmen vorliegen.

7.1.1 Die Client-Server Kommunikation

Das Java-RMI Framework bietet die Möglichkeit bei verteilten Applikationen Objektreferenzen über ein Netzwerk hinweg zu benutzen. Hat das Clientprogramm eine Referenz auf ein Objekt auf einem anderen Rechner erhalten, so können Methoden dieses Objektes aufgerufen werden, als bezöge sich die Referenz auf ein lokales Objekt. Allerdings muss sichergestellt sein, dass die Datentypen, die an einem RMI-Methodenaufruf beteiligt sind, *serialisierbar* sind. Diese Eigenschaft bieten die elementaren Datentypen und auch Java-Strings können bei RMI-Methodenaufrufen benutzt werden.
In dem Absatz zuvor wurde bereits angedeutet, dass ein Clientprogramm eine Referenz auf ein sog. „Remote“-Objekt benötigt, um Methoden des entfernten Objektes aufrufen zu können. Um eine solche

Referenz zu erhalten, existiert die sog. „RMIRegistry“, ein Programm, bei dem „Remote“-Objekte angemeldet werden können. Durch Ansprechen dieses Dienstes erhält ein Clientprogramm die benötigte Referenz auf das „Remote“-Objekt.

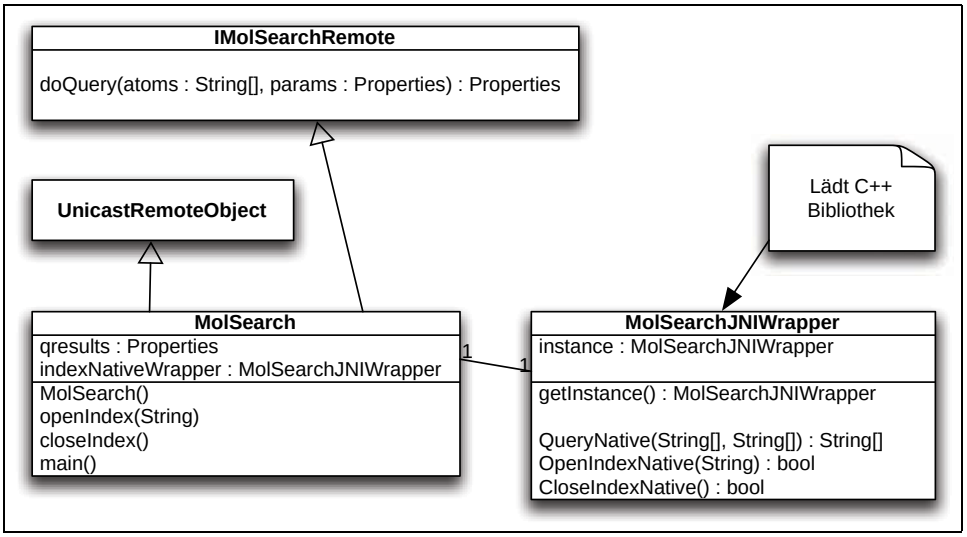


Abbildung 7.2: Übersicht über die serverseitig verwendeten Java-Objekte.

Bei der hier vorgestellten Applikation wird ein Java-Objekt auf diesem Wege zugreifbar gemacht, welches die Anbindung an die C++ Bibliothek, in der die Indextechnologie bereit gestellt wird, steuert. In Abbildung 7.2 sind die Java-basierten Komponenten dargestellt, die serverseitig verwendet werden. Das „Remote-Interface“ `IRemoteMolSearch` enthält nur eine Funktion, die über das Netzwerk aufgerufen werden kann. Es handelt sich dabei um die Methode, die die Suche nach einem Molekülfragment startet. Der Methode werden zwei Parameter übergeben. Zum ersten ein Array von Strings, welche die serialisierte Informationen zu den Atomen des Anfragedokumentes beinhalten. Als zweiten Parameter wird ein Java-„Properties“-Objekt übergeben, welches die Parameter enthält, die vom Benutzer eingestellt worden sind. Ein „Properties“-Objekt verwaltet Schlüssel/Werte-Paare und bietet entsprechende Zugriffsfunktionen an. Ein solches Objekt wird auch als Ergebnis der Suche von der Methode an das aufrufende Clientprogramm übergeben. Details zu dem Inhalt des „Properties“-Objekts werden in Abschnitt 7.1.3 beschrieben. Die Verwendung dieses Objekttyps lässt es zu, neue Parameter oder neue Informationen zu dem Suchergebnis zu übertragen, ohne dass das „Interface“ geändert werden muss.
Die Hauptklasse auf Seiten des Servers ist die Klasse „MolSearch“, welche das „Remote“-Interface implementiert und sich von dem RMI-Objekt „UnicastRemoteObject“ ableitet. Diese Klasse enthält die `main`-Methode der Serverkomponente, deren Aufgabe es ist, Parameter aus einer Konfigurationsdatei zu lesen, eine Referenz auf eine Instanz der Klasse `MolSearchJNIWrapper` zu erlangen und eine Indexdatei zu öffnen. Waren diese Schritte erfolgreich, wird das „MolSearch“-Objekt bei der „RMIRegistry“ registriert und damit im jeweiligen Netzwerk verfügbar gemacht. Die Klasse `MolSearchJNIWrapper` ist als sog. „Singleton“ (siehe [GHRV94]) implementiert, d.h. der Konstruktor der Klasse ist als „private“ deklariert, und eine Referenz ist lediglich durch die Methode `getInstance()` zu erhalten. Zusammen mit der Implementierung von `getInstance()` garantiert dies, dass es in dem jeweiligen Kontext nur eine Instanz der Klasse `MolSearchJNIWrapper` geben kann. Auf diese Weise soll verhindert werden, dass zwei Prozesse gleichzeitig Zugriff auf die

C++ Bibliothek haben. Die „Wrapper“-Klasse kapselt den Java-seitigen Zugriff auf die nativen Funktionen der C++ Bibliothek, d.h. Zugriffe auf die Bibliotheksfunktionen werden synchronisiert und die Parameter in dem „Properties“-Objekt werden ausgelesen, um diese an die C++ Bibliothek übergeben zu können. Nach der Bearbeitung einer Suchanfrage werden die Ergebnisse in Einträge in einem „Properties“-Objekt konvertiert, um sie einem Clientprogramm zu übermitteln.

7.1.2 Parameter zur Suche

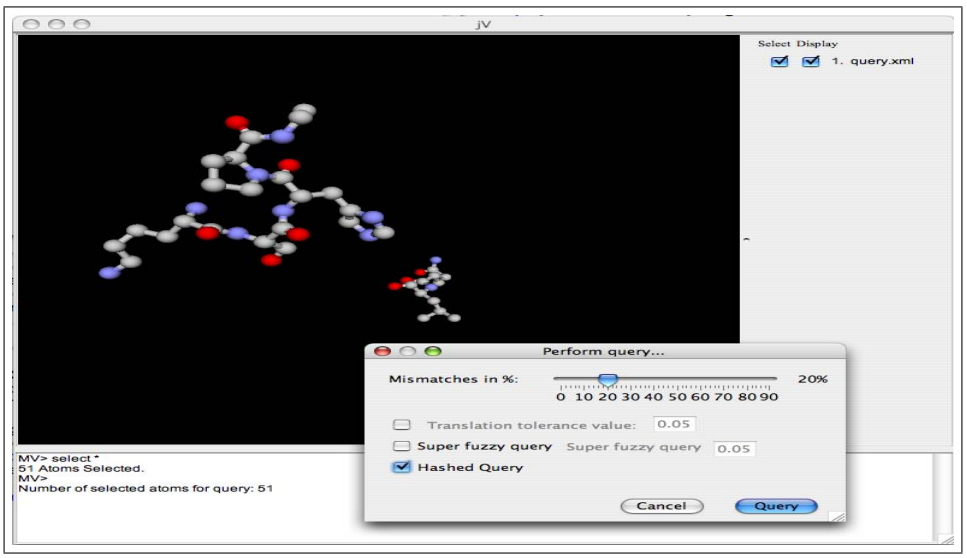


Abbildung 7.3: Screenshot der Client-Applikation basierend auf der „jV3“-Applikation [KN]. Der gezeigte Dialog wird benutzt, um einzelne Parameter der Suchfunktion zu konfigurieren.

Die Parameter, die von dem Nutzer vor dem Start einer Suche ausgewählt werden können, sind im einzelnen:

1. **Prozentuale Anzahl von Fehlstellen:** Mit dem Schieberegler im oberen Bereich des Dialogs kann die Anzahl von erlaubten Fehlstellen bei einer Anfrage im Bereich von 0% bis 90% gewählt werden. Anzumerken ist, dass dieser Prozentwert sich auf die Anzahl von Atom-Tripeln bezieht, in denen sich Anfrage und Dokument unterscheiden dürfen. Dies meint ausdrücklich nicht die prozentuale Anzahl von Atomen, die zwischen Anfrage und Dokument unterschiedlich sein dürfen. Dennoch erlaubt dieser Wert eine intuitive Wahl des Maßes an Fehlertoleranz.

2. **Toleranzwert für die Ähnlichkeit von Gruppenelementen:** Aufgrund der enormen Freiheitsgrade bei der Suche nach Objekten im dreidimensionalen Raum, wird dem Nutzer angeboten, einen Schwellwert für die Ähnlichkeit von Gruppenelementen anzugeben. Auf diese Weise werden zwei Gruppenelemente, die während der Suche berechnet werden, auch dann noch als gleich angesehen, wenn sich z.B. die Translationskomponenten um weniger als dem gewählten Schwellwert unterscheiden. Diese Art der Fehlertoleranz benötigt einen erheblichen Mehraufwand an Rechenzeit im Verglich zu der Standardvariante, bei der die berechneten Gruppenelemente genau gleich sein müssen.

3. Automatische Anfrageerweiterung: Wird diese Option gewählt, wird die zuvor in Abschnitt 6.7.3 beschriebene Erweiterung der Anfrage um den Elementarobjekten der Anfrage ähnlichen Objekten aktiviert. Dazu kann der Nutzer einen Schwellwert angeben, der den maximalen Unterschied zweier Repräsentanten festlegt.

4. Zusätzliches Hashing: Bei Anfragedokumenten mit vielen Atomen und nur mäßiger Anzahl zugelassener Fehlstellen, kann die Wahl dieser Option die Suche deutlich beschleunigen, indem die berechneten Zwischenergebnisse mit geringem Aufwand in eine Hashtabelle eingetragen werden, anstatt diese mit höherem Aufwand z.B. in einen R*-Baum einzufügen. In der Regel erreichen nur wenige Listen der Hashtabelle die notwendige Anzahl von Gruppenelementen, so dass die noch durchzuführende detaillierte Untersuchung der berechneten Gruppenelemente nur für einen Bruchteil der zuvor berechneten Elemente durchgeführt werden muss. Der Geschwindigkeitsvorteil kann mehrere Sekunden betragen. Allerdings ist eine Kombination mit der in 3. angegeben Fehlertoleranz nicht möglich, da diese Art der Fehlertoleranz nicht mit dem Prinzip der Hashtabelle vereinbar ist.

7.1.3 Übermittlung der Suchergebnisse

Nach der Bearbeitung einer Suchanfrage wird das Ergebnis der Anfrage in Form eines „Properties"-Objekts an das Clientprogramm übermittelt. Die folgenden Schlüssel sind unabhängig vom Ergebnis der Suche in der Kollektion von Schlüssel/Werte-Paaren vorhanden:

NumResults Dies ist eine ganze Zahl ≥ 0, die die Anzahl der gefundenen Treffer angibt.

queryTime Die Zeit in Millisekunden, die die Suchanfrage benötigt hat.

GroupOpsCnt Die Anzahl der während der Suche berechneten Gruppenoperationen.

GroupOpsCntPercent Die Anzahl der während der Suche berechneten Gruppenoperationen in Relation zur Gesamtanzahl von Indexeinträgen.

numDocumentsInIndex Die Anzahl der Dokumente im Suchindex.

Ist die Anzahl der gefundenen Treffer größer als Null, so werden für jeden Treffer die folgenden Informationen an den Client übermittelt ($i \quad [0 : \text{NumResults} - 1]$):

score_i Ein Wert, der die Güte eines Treffers beschreibt. Enthält eine Anfrage $|Q_\triangle|$ viele Atomtripel, so ist dies der Maximalwert.

scorepercent_i Der Punktewert des Treffers in Relation zur Länge der Anfrage $|Q_\triangle|$.

docID_i Die ID des Dokuments, in dem der Treffer zu finden ist.

groupelem_i Das serialisierte Gruppenelement, welches Q in das Dokument mit der ID „docID" transformiert.

docName_i Der Name des Dokuments, in dem sich der Treffer befindet.

7.1.3.1 Visualisierung von Treffern

Der erste Schritt ist die Bestimmung der Häufigkeiten der in $Q \subset \mathbb{R}^3 \times \mathcal{A}$ enthaltenen Elementtypen. Wir erhalten so eine Abbildung $H_Q \colon \mathcal{A} \quad \mathbb{Z}_{\geq 0}$, die Elementtypen aus $\mathcal{A}$ deren Häufigkeit in Q zuweist. Für jeden Treffer (g_j, i_j) werden nun folgenden Schritte für jedes anzuzeigende Dokument durchlaufen.
Zunächst wird das Gruppenelement g_j des j-ten Treffers (g_j, i_j) auf die Atome in der Anfrage Q angewendet: $Q' := g_j Q$. Die Elemente von Q' werden in einen R^*-Baum eingefügt, um auf diese Weise effizient NN-Anfragen beantworten zu können. Dies ist notwendig, da keine Information darüber vorliegt, welches Atom der Anfrage mit welchem Atom im Trefferdokument D_{i_j} korrespondiert. Um dieses Problem zu lösen, definieren wir zunächst eine Teilmenge $D_{i_j}^{Q'} \subseteq D_{i_j}$, die D_{i_j} auf die Elementtypen einschränkt, die in Q' vorhanden sind, denn es mach keinen Sinn für ein Atom a aus D_{i_j} nach einem korrespondierendem Atom in Q' zu suchen, wenn $H_Q(\pi_2(a)) = 0$ gilt.
Sei nun $E := \{e \quad \mathcal{A} \mid \quad a \quad D_{i_j} : e = \pi_2(a) \quad H_Q(\pi_2(a)) > 0\}$ die Menge aller Elementtypen, die sowohl in D_{i_j} als auch in Q mindestens einmal enthalten sind. Zugleich zerfällt D_{i_j} in Teildokumente $D_{i_j}^e, e \quad E$, die jeweils nur Atome vom Element e beinhalten. In diesem Schritt wird nun für alle Atome $a \quad D_{i_j}^e$ mittels des zuvor aus Q' gewonnenen R^*-Baums eine NN-Suche durchgeführt, deren Ergebnis wir mit q_a bezeichnen. Ist $\pi_1(q_a) - \pi_1(a) \leq \epsilon$ und gilt $\pi_2(q_a) = \pi_2(a)$, so wird das Atom $a \quad D_{i_j}$ als zum Treffer gehörend markiert. Aufgrund von begrenzter Rechnergenauigkeit und Fehlertoleranz muss der Schwellwert ϵ entsprechend gewählt werden.
Die markierten Atome werden in dem Visualisierungstool farblich hervorgehoben und selektiert, was den Anwender in die Lage versetzt, diese Atome gezielt zu manipulieren, z.B. in der Form der Darstellung etc.

7.2 Die generische C++ Komponente

Das Herzstück der Client/Server-Applikation ist die Bibliothek, welche über das Java Native Interface angesprochen wird und die eigentliche Suchfunktionalität bereitstellt. In diesem Abschnitt wird der Aufbau der Bibliothek und der daran beteiligten Komponenten vorgestellt. Die zentrale C++ Klasse `GIndex` ist als generische Klasse wie folgt deklariert:

```
template<class Group, class Representative, class ListManager> GIndex
```

Die Template-Parameter `Group`, `Representative` und `ListManager` sind die zentralen Begriffe im Kontext der Klasse `GIndex`. Durch die Verwendung von Templates lassen sich z.B. mit unterschiedlichen Gruppen und Repräsentanten leicht konkrete Ausprägungen eines `GIndex`-Objekts erstellen, ohne dass an dem Quelltext der Klasse `GIndex` etwas geändert werden muss. Die Template-Parameter werden nun im Einzelnen vorgestellt.

7.2.1 Template-Parameter `Group`

Der Template-Parameter `Group` ist die abstrakte Bezeichnung für eine Klasse, die die Gruppe implementiert, die dem jeweiligen `GIndex` zugrundeliegt. Auf diese Weise können unterschiedliche Indexe aufgebaut werden, aber auch unterschiedliche Implementierungen von ein und derselben Gruppe miteinander verglichen werden. Im Rahmen dieser Arbeit wurde zur Suche in Proteinmolekülen die Gruppe $SE(3)$ mit Rotationsmatrizen implementiert, aber auch mit Quaternionen, die die jeweilige Rotation beschreiben.

Um sicherzustellen, dass eine Implementierung einer Gruppenklasse alle von der Klasse `GIndex` benötigten Operatoren und Methoden zur Verfügung stellt, wurde eine abstrakte Basisklasse `Group` definiert, von der sich z.B. die Implementierungen der Gruppe $SE(3)$ ableiten.

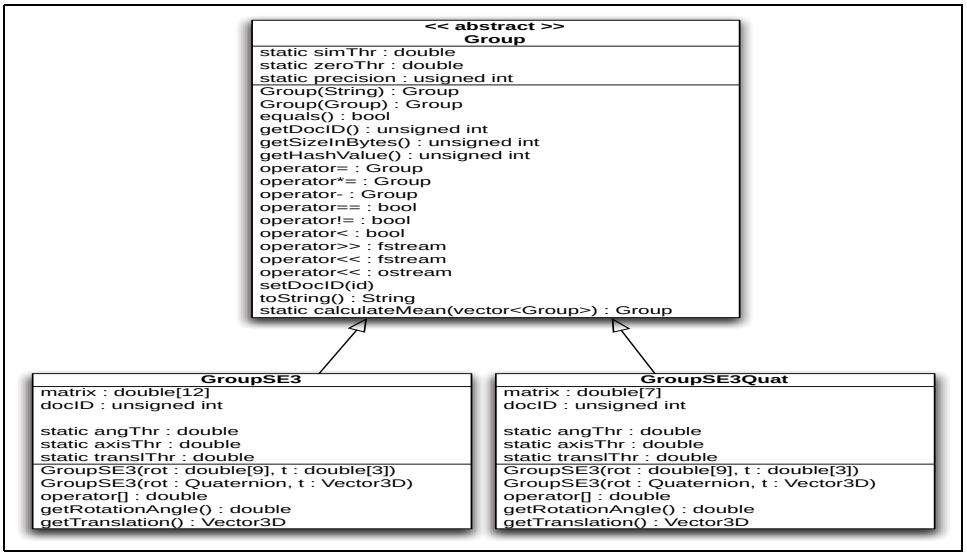


Abbildung 7.4: Klassendiagramm zur Veranschaulichung der Ableitung von der abstrakten Basisklasse `Group`.

Einige der Methoden der abstrakten Basisklasse `Group`, wie in Abbildung 7.4 dargestellt, bedürfen einiger Erläuterungen.

Statische Attribute: Die statischen Klassenvariablen `simThr`, `zeroThr` und `precision` werden benutzt, um Operatoren wie z.B. == zu parametrisieren. Der erste Wert beschränkt den Unterschied zwischen zwei Werten. Unterschreitet die Differenz diesen Wert, gelten beide Werte als gleich. Der `zeroThr` gibt an, ab welchem Wert ein Wert als Null anzusehen ist. Die `precision` bezeichnet die Anzahl von Nachkommastellen, auf die z.B. Werte bei der Serialisierung eines Gruppenelements in eine Zeichenfolge gerundet werden.

`equals()` und `operator==`: Der Operator == prüft elementweise, ob die Werte, die ein Gruppenelement beschreiben (z.B. `GroupSE3::matrix`), bis auf den Wert `simThr` gleich sind. Hingegen vergleicht `equals()` zwei Gruppenelemente auf einer anderen semantischen Ebene. Bei der Klasse `GroupSE3Quat` wird der Abstand der Translationsvektoren zweier Gruppenelemente verglichen, der Rotationswinkel und auch die Rotationsachse. Für jeden dieser Vergleiche existieren die entsprechenden Schwellwerte: `translThr`, `angThr` und `axisThr`. Diese sind mit dem Schlüsselwort statisch deklariert, damit diese für alle Instanzen einer Klasse verfügbar sind. Vor einer Suche werden alle Schwellwerte gemäß den übergebenen Suchparametern verändert bzw. auf den Standardwerten belassen, wenn vom Benutzer keine anderen Werte vorgegeben werden. Die Funktionalität von `equals()` wird z.B. bei der auf Cluster basierenden Suchstrategie eingesetzt, um die Zugehörigkeit von Gruppenelementen zu einem Cluster zu bestimmen.

`getHashValue()`: Diese Methode bezieht sich auf die in Abschnitt 3.3.2 vorgestellte Suchstrategie. Hier werden die Ergebnisse der Listenjustierungen in einem zusätzlichen Zwischenschritt

in einer Hashtabelle abgelegt. Die Idee dabei ist, dass ein Gruppenelement g nur dann zu einem Treffer gehören kann, wenn in der Hashtabelle zu dem Hashwert von g mindestens so viele Einträge zu finden sind, wie der Trefferschwellwert fordert. Da aber die Abbildung von Gruppenelementen auf ihren jeweiligen Hashwert durch die zugrundeliegende Hashfunktion nicht eindeutig ist, müssen nur die Gruppenelemente in den Listen zu Hashwerten mit genügend Einträgen miteinander verglichen werden, um zu entscheiden, ob mindestens thr viele Gruppenelemente überstimmen bzgl. des zuvor angesprochenen Operators ==. Dieses Vorgehen macht allerdings nur Sinn, wenn Hashwerte zu Gruppenelementen stets schnell berechnet werden können.

`getSizeInBytes()`: Um ein Gruppenelement in eine Binärdatei zu schreiben oder aus einer solchen Datei zu lesen, wird der tatsächliche Speicherplatzbedarf eines Gruppenelements benötigt. Dies ist notwendig, da ggf. Attribute einer Gruppenklasse bei der Konstruktion eines Gruppenelementes aus anderen Attributen berechnen lassen. Diese Attribute müssen dann z.B. nicht mit auf den externen Datenträger geschrieben werden.

`operator-`: Dieser einstellige Operator bezeichnet die Berechnung des inversen Gruppenelements: $g \quad g^{-1}$.

`toString()`: Diese Methode stellt ein Gruppenelement als Zeichenkette dar. Auf diese Weise lassen sich Objekte „als Text“ über ein Netzwerk versenden. Das verwendete Java-RMI erlaubt es zwar, ohne grossen Aufwand serialisierte Objekte über das ein Netzwerk zu senden, jedoch ist ein „String“-Objekt in diesem Kontext einfacher zu handhaben.

7.2.2 Template-Parameter `Representative`

In der abstrakten Basisklasse `Representative` sind die notwendigen Methoden zusammengefasst, die eine Repräsentanten-Klasse implementieren muss. Da die Implementierung der Klasse `GIndex` von der genauen Beschaffenheit eines Repräsentanten abstrahiert, können auf einfache Weise Suchindexe mit unterschiedlichen Repräsentanten implementiert werden, ohne dafür die Implementierung der Klasse `GIndex` anzupassen. Die Abbildung 7.5 zeigt die Elemente der Basisklasse.

`toString()`: Wie zuvor bereits erläutert, werden Strings benutzt, um ein solches Objekt zu serialisieren und so über ein Netzwerk zu senden. Der Konstruktor, der einen String als Argument annimmt, dient der Deserialisierung.

Die Operatoren == und <: Besonders der Operator < ist wichtig für Repräsentanten, da dieser benötigt wird, um zum Beispiel Repräsentanten in R*-Bäumen zu verwalten. Dabei ist es unwichtig, ob die durch < induzierte Ordnung auf den Repräsentanten eine semantisch sinnvolle Ordnung beschreibt.

`operator-`: Dieser Operator berechnet einen Abstand $d \quad \mathbb{R}$ zwischen zwei Repräsentanten. Mit diesem Abstandsbegriff können z.B. zu einem Repräsentanten ähnliche Repräsentanten berechnet werden, die automatisch einer Anfrage hinzugerechnet werden.

7.2.2.1 Implementierung von `PDBRepresentative`

In der von der abstrakten Basisklasse `Representative` abgeleiteten Klasse `PDBRepresentative` ist in dem Konstruktor

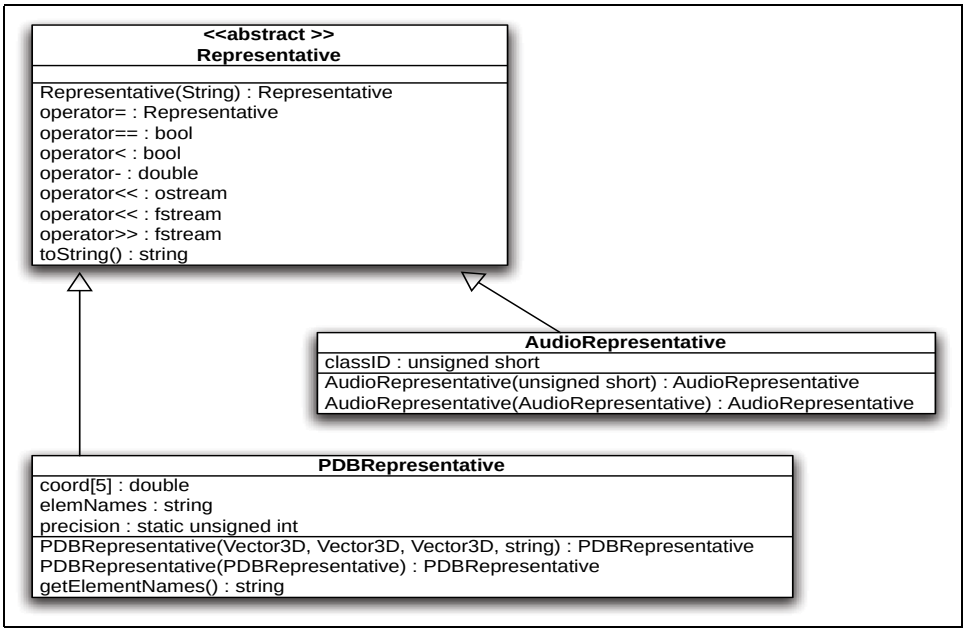


Abbildung 7.5: Klassendiagramm zur abstrakten Basisklasse `Representative` und den davon abgeleiteten Klassen `PDBRepresentative` und `AudioRepresentative`. Letzterer wird z.B. in der Audioidentifikation benutzt.

```
PDBRepresentative(Vector3D, Vector3D, Vector3D, string)
```

der in Abschnitt 6.6.1.2, Schritt 7, beschriebene Prozess implementiert. Zunächst wird die Reihenfolge der in dem Tripel vorkommenden Atome bestimmt und anschliessend die jeweils auf `PDBRepresentative::precision` viele Nachkommastellen gerundeten Werte in dem Array `coords[5]` gespeichert. Diese Werte werden von den Operatoren „==“, „<“ und „−“ verwendet. Letzterer berechnet den maximalen Euklidschen Abstand zwischen den Punkten zweier Repräsentanten r, s:

$$\begin{aligned} d_0 &= |r.\texttt{coord[0]} - s.\texttt{coord[0]}|, \\ d_1 &= \sqrt{(r.\texttt{coord[1]} - s.\texttt{coord[1]})^2 + (r.\texttt{coord[2]} - s.\texttt{coord[2]})^2}, \\ d_2 &= \sqrt{(r.\texttt{coord[3]} - s.\texttt{coord[3]})^2 + (r.\texttt{coord[4]} - s.\texttt{coord[4]})^2}, \\ r - s &:= \max\{d_0, d_1, d_2\}. \end{aligned}$$

Dabei ist zu beachten, dass in diese Berechnung der Vergleich der Elementnamen der zu einem Repräsentanten gehörenden Atome nicht mit einbezogen wird. Dieser Operator setzt implizit voraus, dass sich die gleichen Atome in beiden Repräsentanten an den gleichen Ecken des Dreiecks befinden. Dementsprechend sind zwei Repräsentanten gleich, wenn die Atomnamen übereinstimmen und die Werte in dem Array `coords` elementweise gleich sind. Bei der Frage, ob ein Repräsentant kleiner ist als der andere, werden zunächst die Namen der Atome verglichen und dann die ersten Koordinaten, die nicht gleich sind.

7.2.2.2 Implementierung von `AudioRepresentative`

Im Vergleich zu dem zuvor diskutierten Repräsentanten für Tripel von Atomen, ist der Repräsentant, wie er bei dem System „audentify!“ benutzt wird, deutlich einfacher. In diesem Fall besteht der

Repräsentant lediglich aus einer ganzzahligen Identifikationsnummer. Damit sind die Implementierungen der Operatoren mehr oder minder trivial. Allerdings bleibt die Frage, wie der Abstand zweier Repräsentanten in diesem Fall zu werten ist. In der Praxis ist es so, dass die Merkmale, die aus Audiosignalen berechnet werden, einem Frequenzband zugeordnet werden. Diesem Frequenzband wiederum werden IDs zugewiesen. Somit kann der Abstand von zwei Repräsentanten $|r - s| = 1$ auf Grenzfälle bei der Merkmalsberechnung hindeuten. Folglich kann es bei Anfragen durchaus sinnvoll sein, auch „benachbarte“ G-invertierte Listen zu betrachten.

7.2.3 Template-Parameter `ListManager`

Die Aufgabe eines `ListManager`-Objektes innerhalb eines `GIndex`-Objektes besteht darin, (Such-)Algorithmen mittels eines „Iterators“ (siehe [GHRV94]) Zugriff auf die Elemente einer G-invertierten Liste zu gewähren. Dieses Konzept folgt dem „Strategy“ Design Pattern aus [GHRV94]. Durch die Verwendung von Iteratoren wird der Zugriff auf die Elemente der G-invertierten Listen standardisiert. Durch die Verwendung von unterschiedlichen `ListManager`-Objekten bei der Instanziierung des `GIndex`-Objektes können die G-invertierten Listen je nach Anforderung unterschiedlich verwaltet werden. Im Falle eines Suchindexes mit wenigen Daten kann es z.B. Sinn machen, dass alle G-invertierten Listen beim Start des Programms in den Hauptspeicher geladen werden, um Suchalgorithmen schnellstmöglichen Zugriff auf die Listenelemente zu gewähren. Ist der Suchindex größer und kann nicht mehr ganz in den Hauptspeicher geladen werden, kann ein entsprechender „ListManager“ eine Art Cache-Funktion übernehmen und einmal geladene Listen im Hauptspeicher zu halten, bis ein bestimmter Hauptspeicheranteil durch G-invertierte Listen belegt ist. Bevor nun eine neue Liste von der Festplatte geladen wird, ist es Aufgabe des „ListManager“s durch Löschen von z.B. längere Zeit nicht mehr genutzten Listen, wieder freien Speicherplatz zu schaffen. Auf diese Weise können bei Implementierungen für Mehrprozessorsysteme „ListManager“ eingesetzt werden, die die Arbeit auf mehrere Threads verteilen.
Welche konkrete Implementierung eines „ListManagers“ von einem `GIndex`-Objekt verwendet wird, wird durch den Template-Parameter der „GIndex“-Klasse festgelegt. Um zu gewährleisten, dass eine Implementierung eines „ListManagers“ alle benötigten Funktionen bereitstellt, wurde eine abstrakte Basisklasse `ListManager` definiert, von der die eigentlich verwendeten „ListManager“-Implementierungen abgeleitet sind.

7.3 Der Aufbau eines Suchindexes

Im Gegensatz zur inhaltsbasierten Suche in MIDI- oder PCM-Audiodaten ist der Speicheraufwand im Falle von Moelküldaten deutlich höher. Werden Quaternionen zur Modellierung der $SE(3)$ verwendet, müssen zu jedem Gruppenelement sieben Gleitkommawerte und eine Dokumenten-ID gespeichert werden. Bei der Suche in PCM-Audiodaten sind nur die Dokumenten-ID und ein ganzzahliger Wert, der der Verschiebung entspricht, zu speichern. In diesen Fällen konnte ein Suchindex für z.B. 10.000 Musikstücke im Hauptspeicher abgelegt werden. Bei der Indexierung von Proteinstrukturdaten ist dies nicht mehr möglich, so dass eine andere Strategie bei der Erstellung eines Suchindexes verwendet werden muss.
Aus diesem Grund wurde eine von der zuvor beschriebenen „GroupIndex“-Klasse unabhängige Softwarekomponente entwickelt, die dazu benutzt wird, Suchindexe mit mehreren Gigabyte Speicherplatzbedarf zu erstellen. Die sog. `GIndexBuilder` -Klasse ist ebenfalls eine generische Templateklasse, die von der einem Index zugrundeliegenden Gruppe und des Repräsentantensystems abstra-

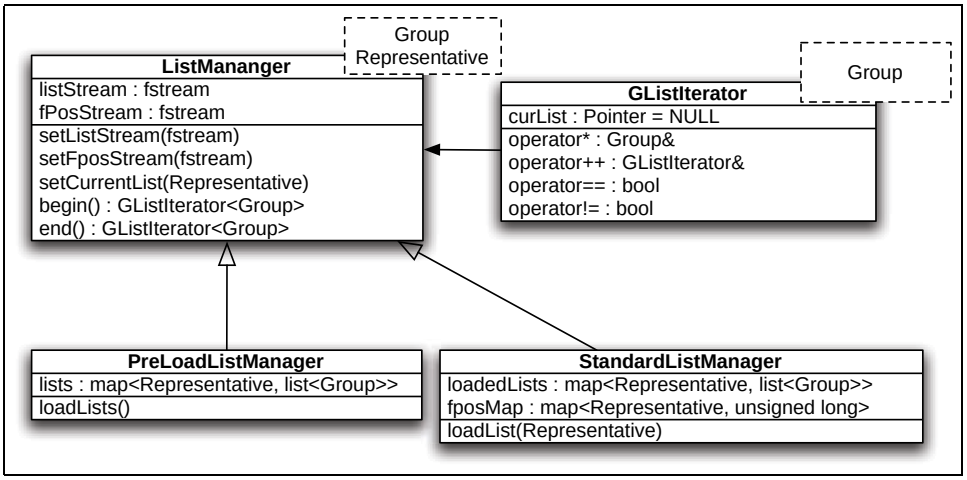


Abbildung 7.6: Klassendiagramm für die im Rahmen dieser Arbeit implementierten ListManager. Zur Suche in Moleküldaten, wird lediglich der „StandardListManager“ genutzt.

hiert:

```
template<class Group, class Representative> GIndexBuilder
```

7.3.1 Merkmalsextraktion

Wie zuvor bereits beschrieben und auch in dem Kapitel über die Suche in PCM-Audiodaten, ist ein Vorverarbeitungsschritt die sog. *Merkmalsextraktion*. In diesem Schritt werden die Rohdaten (hier z.B. Proteinmoleküle als XML Datei) so verarbeitet, dass die Elementarobjekte, die letztendlich zur Indexierung genutzt werden, berechnet werden. Bei den Proteinmoleküldaten sind dies eben gerade die Dreiergruppen von Atomen, Tripel genannt. Zu jedem Tripel eines Proteins werden ein Gruppenelement g_i und der zugehörige Repräsentant r_i bestimmt. In Abbildung 7.7 ist dieser Prozess

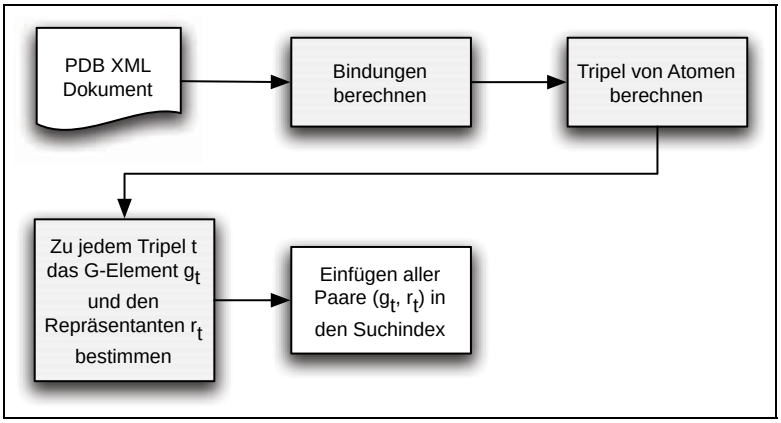


Abbildung 7.7: Prozess zur Extraktion von „Features“ aus PDB-XML Dateien. Dieser Prozess muss für jedes XML-Dokument, welches dem Suchindex hinzugefügt werden soll, durchlaufen werden. Auch ein Anfragedokument Q durchläuft diesen Prozess vor der eigentlichen Suche.

graphisch dargestellt. Für jedes Dokument, welches dem Suchindex hinzugefügt werden soll und für jede Anfrage Q an den Suchindex, ist dieser Prozess zu durchlaufen. Die Folge von Paaren (g_i, r_i) ist die Grundlage für die Indexerstellung.

7.3.2 Hinzufügen eines Dokuments zu einem Suchindex

Die zentrale Methode der Klasse `GIndexBuilder` ist die zum Hinzufügen von neuen Daten:

```
unsigned int add(vector<Group>& g, vector<Representative>& r)
```

In zwei Vektoren werden der Methode die während der Extraktion von Merkmalen aus den Rohdaten bestimmten Paare (g_i, r_i) als Parameter übergeben. Der Rückgabewert der Methode ist eine ganzzahlige Identifikationsnummer, welche in den Ergebnissen zu einer Suchanfrage das jeweilige Dokument bezeichnet. Diese IDs können z.B. in einem relationalen Datenbankmanagementsystem (RDBMS) mit Masterdaten zu jenem Dokument verknüpft werden.

Algorithmus 7.3.1: ADD$((g_i)_{i\in[0:N-1]}, (r_i)_{i\in[0:N-1]})$

for i 0 **to** $(N-1)$
  $docID$ $(docID+1)$;
  if not $gListMap$.CONTAINS_KEY(r_i)
    then $gListMap[r_i]$ NEW$(list < Group >)$;
  g_i.SET_DOC_ID$(docID)$;
  $gListMap[r_i]$.PUSH_BACK(g_i);
  if $gListMap[r_i]$.SIZE$() > thr$
    $fpos$ FLUSH_TO_DISK$(gListMap[r_i])$;
    l.CLEAR$()$;
    then **if not** $fposMap$.CONTAINS_KEY(r_i)
      then $fposMap[r_i]$ NEW$(list < usigned\,long >)$;
    $fposMap[r_i]$.PUSH_BACK$(fpos)$;
return $(docID-1)$;

Algorithmus 7.3.1: Pseudocode-Algorithmus zum Hinzufügen von Paaren (g_i, r_i), die zuvor aus einem Dokument gewonnen wurden. Die Variable *docID* ist vor dem ersten Aufruf mit 0 initialisiert worden. Die beiden Mapping-Tabellen sind vor dem ersten Einfügen von Daten leer.

Der Algorithmus 7.3.2 beschreibt in Pseudocode das Vorgehen, um Daten dem Suchindex hinzuzufügen. Beide Vektoren, die als Parameter der Methode übergeben worden sind, werden sequentiell durchlaufen. Für jedes sich aus den Listen ergebende Paar (g_i, r_i) wird zunächst geprüft, ob zu dem Repräsentanten r_i bereits eine Liste von Gruppenelementen geführt wird. Ist dies nicht der Fall, wird eine neue Liste zu dem Repräsentanten angelegt und das Element g_i dieser Liste hinzugefügt. Existiert zu dem Repräsentanten r_i bereits eine Liste, so wird lediglich das Element g_i der Liste hinzugefügt. Überschreitet eine Liste eine bestimmte Länge, so wird diese in eine Datei auf der Festplatte gespeichert und der Inhalt der Liste im Speicher wird gelöscht. Gleichzeitig wird die Dateiposition vermerkt, an der die Daten zu einer Liste abgelegt worden sind.

Algorithmus 7.3.2: CLOSE()

```
representatives ← gListMap.GET_KEYS()
vector < Group > v;
for each r ∈ representatives
    v.CLEAR();
    for each g ∈ gListMap[r]
      do v.PUSH_BACK(g);
    for each fpos ∈ fposMap[r]
      do tmpFile.SET_FILE_READ_POS(fpos);
         for i ← 0 to (thr − 1)
           do { g ← tmpFile.READ();
                v.PUSH_BACK(g);
    v.SORT();
    newFposMap[r] ← indexFile.GET_CURRENT_FILE_POS();
    indexFile.WRITE(r);
    for i ← 0 to (|v| − 1)
      do {indexFile.WRITE(v[i]);
for each key ∈ newFposMap
  do { indexPropertyFile.WRITE(key);
       indexPropertyFile.WRITE(newFposMap[key]);
```

Algorithmus 7.3.2: Nach dem Einfügen aller Daten in den Suchindex, wird dieser durch Aufruf von `close()` in die Form überführt, die später von den Suchalgorithmen benötigt wird. Der Suchindex teilt sich in zwei Dateien auf. Die erste enthält lediglich die G-invertierten Listen, die zweite Datei enthält eine Mappingtabelle, in der einem jedem im Index vorhandenen Repräsentanten eine Position in der ersten Datei zugeordnet ist, an der die jeweilige G-invertierte Liste zu dem Repräsentanten gespeichert ist. Durch entsprechendes Setzen des Lesezeigers kann die gewünschte G-invertierte Liste geladen werden. Die Variable $tmpFile$ bezeichnet die Datei, in welche die Daten geschrieben werden. $indexFile$ und $indexPropertyFile$ stehen für die beiden Dateien, welche am Ende des Gesamtprozesses den Suchindex enthalten.

Sind alle Dokumente dem Suchindex hinzugefügt worden, folgt die zweite Phase des Indexaufbaus, die in Algorithmus 7.3.2 dargestellt ist. Für jeden Repräsentanten r, der während der ersten Phase mindestens einmal dem Index hinzugefügt worden ist, werden die Gruppenelemente, die ggf. noch im Speicher sind, mit denen zuvor auf der Festplatte ausgelagerten Gruppenelementen in einen Vektor geladen, sortiert und zusammen mit dem jeweiligen Repräsentanten in eine neue Datei auf der Festplatte abgelegt. Erneut wird die Dateiposition, an der der Repräsentant zu einer Liste zu finden ist, zu dem jeweiligen Repräsentanten gespeichert. Diese Information wird in einer separaten Datei auf der Festplatte gespeichert. Es ist diese Datei, welche von einem `GIndex`-Objekt beim Öffnen eines Suchindex ganz von der Festplatte in den Hauptspeicher geladen wird, um schnellstmöglich die Dateiposition in der eigentlichen Suchindexdatei, an der eine Liste zu einem Repräsentanten r abgelegt ist, zu finden. Je nach Anzahl der Dokumente in einem Suchindex und der damit proportinonalen Anzahl von Repräsentanten, umfasst eine solche Datei mehrere zehn Megabyte. Im Vergleich dazu belegt die Datei mit den G-invertierten Listen mehrere Gigabyte Speicherplatz auf der Festplatte.
Die maximale Größe, die eine Suchindexdatei erreichen kann, ist lediglich durch die maximale Dateigröße limitiert, die wiederum von dem jeweiligen Betriebssystem abhängt. Aber auch diese Limitierung liesse sich umgehen, wenn die G-invertierten Listen auf mehrere Dateien verteilt und diese Information beim Öffnen eines Suchindex ebenfalls in den Hauptspeicher geladen wird.

Kapitel 8

Ausblick und Diskussion

In der vorliegenden Arbeit ging es um die inhaltsbasierte Suche in sehr unterschiedlichen Dokumenten: Zum einen Audiodaten und zum anderen um biologische Makromoleküle. Dazu wurde in Kapitel 2 basierend auf [CEMS00] und weiteren Veröffentlichungen der gleichen Arbeitsgruppe das Prinzip der inhaltsbasierten Suche basierend auf dem algebraischen Prinzip der Operation einer Gruppe auf einer Menge vorgestellt. Aufgrund der Allgemeinheit dieses Prinzips konnten in den nachfolgenden Kapiteln Systeme vorgestellt und diskutiert werden, die eine inhaltsbasierte Suche in den unterschiedlichen Dokumenten ermöglichen. Einen Schwerpunkt dieser Arbeit bilden die in Kapitel 3 vorgestellten Algorithmen zur Berechnung der Treffermenge $G_{\mathcal{D}\,k}(Q)$ zu einer Anfrage Q.

8.1 Zusammenfassung der Vorgehensweise

Ein zentraler Grundsatz bei der inhaltsbasierten Suche mittels G-invertierter Listen in einem Suchindex ist die Trennung von Modellierung der Dokumenteninformation und der je nach Anwendung operierenden Gruppe. Die Suchalgorithmen arbeiten in der Regel auf Tupeln $(g, i) \quad G \times [0, \ldots, |\mathcal{D}| - 1]$, die in den G-invertierten Listen abgelegt sind.
Bei einer Anwendung dieser Indexierungstechnik z.B. auf eine neue Art von Dokumenten, sind drei Gruppen von Fragestellungen bei der technischen Umsetzung von besonderem Interesse:

1. Welche Gruppe G operiert auf der Menge M in diesem Anwendungsfall? Wie kann diese effizient implementiert werden (vgl. $SE(3)$ mittels Rotationsmatrizen oder Quaternionen)?

2. Ein Merkmalsextraktor muss definiert werden. Wie robust sind die Merkmale? Wie können möglichst viele unterschiedliche Merkmale berechnet werden, um ein Repräsentantensystem großer Kardinalität zu erreichen? Inwieweit kann durch Merkmalsextraktion eine sinnvolle Datenreduktion erreicht werden? Wie kann dieser Merkmalsextraktor effizient implementiert werden?

3. Welche Suchalgorithmus soll verwendet werden? Sind die in dieser Arbeit diskutierten Fehlertoleranzmechanismen ausreichend oder müssen die Fehlertoleranzkonzepte ggf. erweitert werden?

Die Wahl der richtigen Gruppe und deren Implementierung gehört neben der Wahl eines Suchalgorithmus sicherlich zu den einfacheren Fragestellungen. Erfordert die Anwendung einen Trefferbegriff, der nicht über Fehlstellen, Fuzzy-Anfragen etc. modelliert werden kann, so müssen ggf. die verwendeten Suchalgorithmen angepasst werden.

Die Entwicklung eines geeigneten Merkmalsextraktors ist sicherlich die schwierigste Aufgabe, da hier Kompromisse zwischen sich widersprechenden Anforderungen gefunden werden müssen, sofern dies überhaupt im Hinblick auf die Anwendung möglich ist. So kann die Robustheit der extrahierten Merkmale erhöht werden, indem die Zahl der unterschiedlichen Merkmalsklassen von vornherein reduziert wird. Dies kann jedoch dazu führen, dass die Zahl der „False-Positive"-Treffer ansteigt und dass die G-invertierten Listen viele Einträge haben, die bei einer Suchanfrage zu betrachten sind.

8.2 Audio-Identi kation

Bei dieser Aufgabenstellung steht der Merkmalsextraktor im Vordergrund. Je nach Qualität des angefragten Audiosignals sind die Anforderungen an die Robustheit der extrahierten Merkmale sehr hoch. Da in diesem Fall die Gruppe $G = (\mathbb{Z}, +)$ operiert, spielt die Anzahl von zu berechnenden Gruppenoperationen im Hinblick auf die benötigte Zeit zur Bestimmung der Treffermenge eine untergeordnete Rolle, so dass auch G-invertierte Listen mit vielen Einträgen verwendet werden können. Da hier die Gruppe G angeordnet ist, bieten sich Suchverfahren an, die die Sortierung der G-invertierten Listen nutzen können, um z.B. Elemente in den Listen zu überspringen, d.h. weniger Gruppenverknüpfungen zu berechnen.
Das im Rahmen des SyncPlayer-Projekts verwendete Modul zur Audioidentifikation nutzt ausserdem das in [Rib06] erläuterte Framework, welches mittels verteiltem Rechnen die Audioidentifikation auch bei großen Datenmengen ermöglicht. Durch die Anlehnung an eine „Shared-Nothing"-Architektur konnte der Suchindex auf mehrere Rechner verteilt werden bzw. durch die Verwendung von mehreren Rechner mit ein und dem selben Suchindex ein Lastverteilung erreicht werden.

8.3 Suche in 3D-Molekülen

Die Suche in großen Datenbeständen von biologischen Molekülen wie Proteinen mit mehreren tausend Atomen hingegen verlangte nach der Verwendung der Gruppe gebildet durch die euklidschen Bewegungen im dreidimensionalen Raum. Die Gruppe $SE(3)$ wurde auf unterschiedliche Weise realisiert. Zum einen durch die Verwendung von Rotationsmatrizen zusammen mit einem Translationsvektor. Dies führt zu einem Speicherplatzbedarf von zwölf Gleitkommazahlen. Durch die Realisierung der Rotationskomponente durch Quaternionen konnte dieser Speicherplatzbedarf auf Kosten einer ein wenig aufwändigeren Gruppenverknüpfung auf sieben Gleitkommawerte reduziert werden.
Neben einer effizienten Implementierung der Gruppe $SE(3)$, wurde ein Merkmalsextraktor entwickelt, der zu einem Moleküldokument eine Folge von Atomtripeln berechnet, welche die Elementarobjekte bei dem Aufbau eines Suchindex darstellen. Dabei sind zwei Zielsetzungen von Interesse. Da eine möglichst schnelle Bearbeitung einer Suchanfrage gefordert ist, soll dieser Merkmalsextraktor aus einem Anfragemolekül Q eine Folge von Tripeln berechnen, die minimale Länge hat. Dabei sollen alle Atome des Moleküls mindestens in einem der berechneten Atomtripel enthalten sein. Die Annahme, dass es keine Atome in einem Molekül gibt, die entweder mit keinem oder nur mit einem Atom verbunden sind, erlaubte die Angabe eines effizienten Algorithmus, der eine solche Optimale Überdeckung eines Moleküls mit Atomtripeln berechnet. Hingegen hat Bardeli in [BCR06] zeigen können, das dieses Problem bei allgemeinen Graphen NP-vollständig ist.
Da es eine Vielzahl von minimalen Überdeckungen des Anfragemoleküls Q existieren, sind bei dem Aufbau eines Suchindex *alle* möglichen Überdeckungen eines Moleküls zu berücksichtigen, um keine Treffer zu verlieren. Aus diesem Grund werden bei der Indexierung eines Moleküls deutlich mehr Merkmale berechnet, die alle in den Suchindex eingefügt werden.

Eine weitere allgemeine Art von Fehlertoleranz wurde durch die Verwendung eines Ähnlichkeitsmaßes auf den Elementen des Repräsentantensystems eingeführt. Sei r_q der aus $q \quad Q$ folgende Repräsentant. Durch die Angabe eines Toleranzwertes $\epsilon > 0$ werden automatisch alle G-invertierten Listen bei der Suche betrachtet, deren Repräsentanten einen Abstand kleiner gleich ϵ zu r_q aufweisen. Dadurch können kleinere Unterschiede, die zu unterschiedlichen Repräsentanten führen können, zwischen Anfrage und Dokument ausgeglichen werden, ohne dass dazu das Konzept von Fehlstellen bemüht werden muss.
Zu Demonstrationszwecken wurde das in Java implementierte Darstellungsprogramm [KN] derart erweitert, dass über die RMI-Technologie von Java ein Anfragemolekül über ein Netzwerk an eine Serverkomponente zusammen mit Parametern zur Suche gesendet werden kann. Diese Serverkomponente wiederum griff auf eine in C++ implementiere Bibliothek zurück, in der die eigentliche inhaltsbasierte Suche realisiert wurde.

8.4 Ausblick

Wie bereits an den unterschiedlichen Ansätzen zur Audioidentifikation zu erkennen war, existieren verschiedene Systeme, die zum Teil auch kommerziell vermarktet werden. Die eigentlich Aufgabe der Identifikation eines unbekannten Audiosignals in guter Qualität ist sicherlich mit allen Systemen möglich. Dabei kommt dem Ansatz basierend auf G-invertierten Listen zugute, dass er flexibel auf unterschiedliche Anwendungen angepasst werden kann. Zusammen mit dem vorgestellten Framework und einem entsprechenden Merkmalsextraktor liessen sich auch große Datenbestände von mehreren Millionen Musiktiteln verwalten - entsprechend schnelle und fehlertolerante Suchalgorithmen wurden entwickelt und in dieser Arbeit detailliert vorgestellt.
Herausfordernder ist diese Aufgabe sicherlich in Fällen, bei denen das Anfragesignal Q in nur sehr schlechter Qualität vorliegt. Bei der dabei notwendigen Fehlertoleranz sind fehlerhafte Identifizierungen zu vermeiden, was hauptsächlich durch die Wahl der Merkmale erreicht werden kann. Allerdings scheinen hier die in [Wan03] vorgeschlagenen lokalen Maxima im Spektralbereich durchaus geeignet zu sein, denn schliesslich wird ein Identifikationsdienst per Mobilfunktelefon basierend auf eben gerade diesen Merkmalen, in einigen Ländern kommerziell angeboten. Mit dem wachsenden Markt des Musikvertriebes über das Internet oder über das schnelle UMTS-Mobilfunktnetz kann dieser Dienst für den Mobilfunkbetreiber sicherlich interessanter werden, da hier neben der Identifikation auch ggf. der Verkauf von Musik an den Kunden möglich wird. Ob die Benutzer diesen Weg des Musikkaufs annehmen sei dahingestellt.
Neben der eigentlichen Audioidentifikation ist die nur schwer formal zu fassende Fragestellung der „Ähnlichkeit“ von Musikstücken derzeit im Fokus der Forschung. Es gibt eine Vielzahl von Systemen (z.B. [web06c]) , die ausgehend von einem vom Benutzer ausgewählten Musikstücks „ähnliche“ Musikstücke vorschlagen. Dabei könnte man die Audioidentifikation als einen Einstieg sehen: Der Benutzer hört ein Stück im Radio, überträgt das Signal über das Mobilfunknetz an einen Server, dieser identifiziert das Stück und stellt ausgehend von diesem eine Liste von Stücken zusammen, die dann z.B. über das Mobilfunknetz an den Kunden übertragen werden. Kommerziellen Erfolg birgt auch der Ansatz von [mus06a] in sich. Die automatische Zuordnung von qualitativ hochwertigen Metadaten zu Musikstücken kann der Schlüssel sein, um große Bestände digitaler Musik über deren Metadaten sinnvoll zu ordnen. Auch beinhaltet die Grundidee des SyncPlayer-Projekts [Syn04] das Potential, um den Benutzer mit zusätzlichen Informationen zu versorgen, die synchron zum Musikstück wiedergegeben werden.
Im Vergleich zu dem zuvor angesprochenen kommerziellen Einsatz der Audioidentifikation, ist die

im Rahmen dieser Arbeit entwickelte Applikation zur inhaltsbasierten Suche in dreidimensionalen biologischen Molekülen eine Machbarkeitsstudie. Ziel war es die Flexibilität des vorgestellten Ansatzes des Suchindexes basierend auf G-invertierten Listen an einem Beispiel zu demonstrieren, welches im Vergleich zur Audioidentifikation auf eine Gruppe aufbaut, deren Verknüpfung aufwändiger zu berechnen ist. Dabei stand jedoch die erste Umsetzung eines solchen Systems im Vordergrund, was sich besonders an den verwendeten Trefferbegriffen festmachen lässt. Sicherlich führen konkrete biologische Fragestellungen zu weiteren Trefferbegriffen, die entsprechende spezialisierte Versionen der vorgestellten Suchalgorithmen benötigen. Diese Applikation ist eine Machbarkeitsstudie und daher nur ein erster Schritt in Richtung der inhaltsbasierten Suche in dreidimensionalen Strukturdaten. Vielmehr ging es um Realisierung einer inhaltsbasierten Suche, der eine Gruppe G zugrunde liegt, deren Struktur deutlich komplexer ist, als z.B. der ganzzahligen Translation.
In Abschnitt 6.8 werden die Ergebnisse der vorangegangenen Abschnitte genutzt, um ein System zu skizzieren, welches nicht mehr Dreiergruppen von Atomen als Elementarobjekte verwendet, sondern im Falle von Proteinen die räumliche Ausrichtung der Aminosäuren als Elementarobjekte verwendet. Durch die Verwendung von semantisch gehaltvolleren Elementarobjekten erweitern sich automatisch auch die Möglichkeiten der Anfrageformulierung. Ausserdem kann die inhaltsbasierte Suche in dem Fall mit Algorithmen kombiniert werden, die nach homologen Sequenzen von Proteinen suchen. Diese Art der Suche kann auch ein Vorverarbeitungsschritt für aufwändigere Algorithmen sein, die aufgrund des Zeitbedarfs nicht für alle Proteine einer Dokumentenkollektion berechnet werden können.
Wie in dem einführenden Abschnitt zur PDB in Abschnitt 6.2 erwähnt wurde, bietet das Webportal der PDB eine Fülle von Suchmöglichkeiten auf Basis von Metadaten an. Diese Metainformationen können auch bei der inhaltsbasierten Suche sehr hilfreich sein, indem eine Metadatenanfrage zusammen mit der inhaltsbasierten Anfrage gestellt wird. In Abschnitt 3.6 wurde beschrieben, wie eine aufgrund von Metadaten gewonnene Listen von Dokumenten-IDs in den Prozess der inhaltsbasierten Suche eingebunden werden kann. Durch die Reduktion der Anzahl der möglichen Trefferdokumente kann die inhaltsbasierte Suche deutlich beschleunigt werden, da sicherlich eine Vielzahl von Einträgen in den G-invertierten Listen übersprungen werden können, *ohne* dass für diese Elemente eine Gruppenverknüpfung berechnet werden müsste.
Die Parallelisierung der in Kapitel 3 diskutierten Suchalgorithmen im Hinblick auf Multiprozessor-Systeme bietet ebenfalls noch ein großes Potential, wenn es um die Reduzierung der Suchzeiten geht. Besonders bei dem Verfahren basierend auf Hypercubes können mehrere Prozesse parallel abgearbeitet werden, ohne dass dabei viel Aufwand für die Synchronisierung der Prozesse untereinander aufgewendet werden müsste.

Literaturverzeichnis

[AGM+90] ALTSCHUL, S.F., W. GISH, W. MILLER, E.G. MYERS und D.J. LIPMAN: *Basic Local Alignment Search Tool*. Journal on Molecular Biology, Seiten 403–410, 1990.

[AHH+01] ALLAMANCHE, E., J. HERRE, O. HELMUTH, B. FRBA, T. KASTEN und M. CREMER: *Content-Based Identi cation of Audio Material Using MPEG-7 Low Level Description*, 2001.

[Bar03] BARDELI, R.: *Ef ziente Algorithmen zur deformationstoleranten Suche in Audiodaten*. Diplomarbeit, Universität Bonn, 2003.

[BC01] BÖHM C., BERCHTOLD S., KEIM D.: *Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases*. In: *ACM Computing Surveys*, 2001.

[BCR06] BARDELI, R., M. CLAUSEN und A. RIBBROCK: *A Cover Problem which is Easy for Trees, but NP-Complete for Trivalent Graphs*. Submitted to Journal of Discrete Algorithms, 2006.

[BJ05] BREITMAIER, E. und G. JUNG: *Organische Chemie. Grundlagen, Stoffklassen, Reaktionen, Konzepte, Molekülstruktur*. Thieme, April 2005.

[BMG02] BATTLE, E., J. MASIP und E. GUAUS: *Automatic Song Identi cation in Noisy Broadcast Audio*. 2002.

[BuDK] BÖHM, C. und S. BERCHTOLD UND D.A. KEIM: *Searching in High-dimensional Spaces – Index Structures for Improving the Performance of Multimedia Databases*. Technischer Bericht, Universitäten München, Halle und STB-GmbH.

[BWF+00] BERMAN, H.M., J. WESTBROOK, Z. FENG, G. GILLILAND, T.N. BHAT, H. WEISSIG, I.N. SHINDYALOV und P.E. BOURNE: *The Protein Data Bank*. Nucleic Acids Research, (28):235–242, 2000.

[CB93] CLAUSEN, M. und U. BAUM: *Fast Fourier Transforms*. 1993.

[CBG+02] CANO, P., E. BATLLE, E. GOMEZ, L. DE, C. GOMES und M. BONNET: *Audio nger-printing: concepts and applications*. In: *1st International Conference on Fuzzy Systems*, Singapore, November 2002.

[CBKH02] CANO, P., E. BATLLE, T. KLAKER und J. HAITSMA: *A Review of Algorithms for Audio Fingerprinting*. In: *International Workshop on Multimedia Signal Processing*, US Virgin Islands, December 2002.

[CBMN02] CANO, P., E. BATLLE, H. MAYER und H. NEUSCHMIED: *Robust Sound Modeling for Song Detection in Broadcast Audio*. In: *Proceedings of the 112th AES Convention*, Munich, Germany, May 2002.

[CEMS00] CLAUSEN, M., R. ENGELBRECHT, D. MEYER und J. SCHMITZ: *PROMS: A Web-based Tool for Searching in Polyphonic Music*. In: *Proceedings of the First International Conference on Music Information Retrieval (ISMIR)*, Plymouth, MA, October 2000.

[CK02] CLAUSEN, M. und F. KURTH: *A Uni ed Approach to Content-Based and Fault Tolerant Music Identi cation*. In: *Proceedings of the International Conference On Web Delivering of Music*, Darmstadt, Germany, December 2002.

[CK04a] CLAUSEN, M. und F. KURTH: *Content-based Information Retrieval by Group Theoretical Methods*, Band 136, Kapitel Computational Noncommutative Algebra and Applications, NATO Science Series, II. Mathematics, Physics and Chemistry, Seiten 29–55. Kluwer Academic Publishers, 2004.

[CK04b] CLAUSEN, M. und F. KURTH: *A Uni ed Approach to Content-Based and Fault Tolerant Music Recognition*. IEEE Transactions on Multimedia, 6(5):717–731, October 2004.

[CKK03] CLAUSEN, M., H. KÖRNER und F. KURTH: *An Ef cient Indexing and Search Technique for Multimedia Databases*. In: *Proceedings of the SIGIR 2003 Workshop on Multimedia Retrieval*, Toronto, Canada, 2003.

[CKMR04] CLAUSEN, M., F. KURTH, M. MÜLLER und A. RIBBROCK: *Content-based Retrieval in Digital Music Libraries*. In: *8th European Conference on Digital Libraries*, Band Springer LNCS 3232, Seiten 292–303, 2004.

[Cur79] CURTIS, M.L.: *Matrix Groups*. Springer Verlag, 1979.

[DG02] DAVY, M. und S.J. GODSILL: *Audio Information Retrieval: A Bibliographical Study*. Technischer Bericht, Cambridge University Engineering Department, citeseer.nj.nec.com/davy02audio.html, February 2002.

[DH73] DUDA, R.O. und P.E. HART: *Pattern Classi cation and Scene Analysis*. John Wiley & Sons, New York, 1973.

[Fas99] FASULO, D.: *An Analysis of Recent Work on Clustering Algorithms*. Technischer Bericht, Department of Computer Science and Engineering, University of Washington, Seattle, April 1999.

[FB74] FINKEL, R.A. und J.L. BENTLEY: *Quad trees: A Data Structure for Retrieval on Composite Keys*. In: *ACTA Informatica*, 1974.

[FRM94] FALOUTSOS, C., M. RANGANATHAN und Y. MANOLOPOULOS: *Fast subsequence matching in time-series databases*. In: *Proceedings of the International Conference on Management of Data*, Seiten 419–429, 1994.

[GHRV94] GAMMA, E., R. HELM, R.JOHNSON und J. VLISSIDES: *Design Patterns : elements of reusable object-oriented software*. Addison-Wesley Professional Computing Series, 1994.

[Gut84] GUTTMAN, A.: *R-trees: A Dynamic Index Structure for Spatial Searching*. In: *ACM SIGMOD Int. Conf. on Management of Data*, Seiten 47–57, 1984.

[Ham47] HAMILTON, W.R.: *On Quaternions*. In: *Proceedings of the Royal Irish Academy*, Band 3, Seiten 1–16, 1847.

[HCH02] HART, H., L. E. CRAINE und D. J. HART: *Organische Chemie. Ein kurzes Lehruch*. Wiley-VCH, 2002.

[HK02] HAITSMA, J. und T. KALKER: *A Highly Robust Audio Fingerprinting System*. In: *Proceedings of the ISMIR Conference*, Paris, France, October 2002.

[HKO01] HAITSMA, J., T. KALKER und J. OOSTVEEN: *Robust Audio Hashing for Content Identi cation*. In: *Content Based Multimedia Indexing*, Brescia, Italy, September 2001.

[HS95] HJALTASON, G.R. und H. SAMET: *Ranking in Spatial Databases*. In: *Symposium on Large Spatial Databases*, Seiten 83–95, 1995.

[ISO93] ISO/IEC, 11172-3: *Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s - Part 3: Audio*. International Standrad, August 1993.

[ISO00] ISO/IEC: *ISO-IEC/JTC1 SC92 WG11 Moving Pictures Expert Group, Information technology - multimedia content description interface - part 4: Audio. Comittee Draft 15938-4*, 2000.

[JD88] JAIN, A. und R. DUBES: *Algorithms for Clustering Data*. Prentice Hall Advanced Reference Series, 1988.

[JMF99] JAIN, A.K., M.N. MURTY und P.J. FLYNN: *Data clustering: a review*. ACM Computing Surveys, 31(3):264–323, 1999.

[KC01] KURTH, F. und M. CLAUSEN: *Full-Text Indexing of Very Large Audio Data Bases*. In: *Proceedings of the 110th AES Convention*, Amsterdam, The Netherlands, May 2001.

[KC03] KURTH, F. und M. CLAUSEN: *Unterlagen zur Vorlesung Multimedia Information Retrieval an der Universität Bonn*, 2003.

[KH78] KIRKPATRICK, D.G. und P. HELL: *On The Complexity of a Generalized Matching Problem*. In: *Proceedings of the 10th Ann. ACM Symp. on Theory of Computing*, New York, 1978. ACM.

[KMD+05] KURTH, F., M. MÜLLER, D. DAMM, C. FREMEREY, A. RIBBROCK und M. CLAUSEN: *Syncplayer - An Advanced System for Multimodal Music Access*. In: *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR)*, London, UK, September 2005.

[KMR+04] KURTH, F., M. MÜLLER, A. RIBBROCK, T. RÖDER, D. DAMM und C. FREMEREY: *A Prototypical Service for Real-Time Access to Local Context-Based Music Information*. In: *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, Barcelona, Spain, October 2004.

[KN] KINOSHITA, K. und H. NAKAMURA: *Webseite des Visualisierungstools jV3*. Website. http://pdbj.protein.osaka-u.ac.jp/PDBjViewer/jV3.html.

[KR04] KURTH, F. und A. RIBBROCK: *MultiMedia Information Retrieval*, Kapitel Chapter 5: Recent Advances in Audio Retrieval, Seiten 191–207. Associazione Italiana per la Documentazione Avanzata (AIDA), 2004.

[KRC02a] KURTH, F., A. RIBBROCK und M. CLAUSEN: *Ef cient Fault Tolerant Search Techniques for Full-Text Audio Retrieval*. In: *Proceedings of the 112th AES Convention*, Munich, Germany, May 2002.

[KRC02b] KURTH, F., A. RIBBROCK und M. CLAUSEN: *Identi cation of Highly Distorted Audio Material for Querying Large Scale Data Bases*. In: *Proceedings of the 112th AES Convention*, Munich, Germany, May 2002.

[Kru56] KRUSKAL, J.B.: *On the shortest spanning subtree and the traveling salesman problem*. In: *Proceedings of the American Mathematical Society*, Nummer 7, Seiten 48–50. American Mathematical Society, 1956.

[Krü02] KRÜMMEL, A.: *Datenstrukturen und Algorithmen zur Rechnergestützten Fugenanalyse*. Diplomarbeit, Universität Bonn, 2002.

[KS] KREHER, D.L. und D.R. STINSON: *LaTeX-Style le zur Darstellung von Algorithmen*. Website. http://www.math.mtu.edu/ kreher/cages/pseudocode.html.

[Kur99] KURTH, F.: *Vermeidung von Generationseffekten in der Audiokodierung*. Doktorarbeit, Universität Bonn, 1999.

[Kur02] KURTH, F.: *A Ranking Technique for Fast Audio Identi cation*. In: *International Workshop on Multimedia Signal Processing*, St. Thomas, US Virgin Islands, December 2002.

[Lam] LAM, T. Y.: *Hamilton s Quaternions*. University of California, Berkekey, Ca 94720.

[LNT00] LU, H., Y.Y. NG und Z. TIAN: *T-Tree or B-Tree: Main Memory Database Index Structure Revisited*. In: *Australian Database Conference*, Seiten 65–73, 2000.

[Log00] LOGAN, B.: *Mel Frequency Cepstral Coef cients for music modeling*. In: *Proceedings of the First International Conference on Music Information Retrieval (ISMIR)*, Plymouth, MA, October 2000.

[MEPR] MSD-EBI, PDBJ und RCSB: *PDBML-Schema*. Website. http://pdbml.pdb.org/schema/pdbx.xsd.

[Mica] MICROSYSTEMS, SUN: *Einführung und Tutorial zu Java Native Interface*. Website. http://java.sun.com/docs/books/tutorial/native1.1/index.html.

[Micb] MICROSYSTEMS, SUN: *Portalseite zu Java-RMI*. Website. http://java.sun.com/products/jdk/rmi/index.jsp.

[MKR04] MÜLLER, M., F. KURTH und T. RÖDER: *Towards an Ef cient Algorithm for Automatic Score-to-Audio Synchronization*. In: *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR)*, Barcelona, Spain, October 2004.

[MM03] MORTIMER, C. E. und U. MÜLLER: *Chemie. Das Basiswissen der Chemie*. Thieme, 2003.

[MNPT05] MANOLOPOULOS, Y., A. NANOPOULOS, A.N. PAPADOPOULOS und Y. THEODORIDIS: *Rtrees: Theory and Applications*. Series in Advanced Information and Knowledge Processing. Springer, 2005.

[MPSN01] MOLAU, S., M. PITZ, R. SCHLUTER und H. NEY: *Computing MelFrequency Cepstral Coef cients on the Power Spectrum*. In: *Proceedings of the International Conference on Acoustic, Speech and Signal Processing*, Salt Lake City, UT, June 2001.

[MRC05] MÜLLER, M., T. RÖDER und M. CLAUSEN: *Ef cient Content-Based Retrieval of Motion Capture Data*. ACM Transactions on Graphics, 24(3):677–685, 2005.

[Mue05] MUELLER, M.J.: *Indexstrukturen und Algorithmen zur ef zienten tonhöheninvarianten Musiksuche*. Diplomarbeit, Universität Bonn, Institut für Informatik, February 2005.

[mus06a] MUSICIP.COM: *MusicDNS Service Whitepaper*, 1. Auflage, March 2006.

[mus06b] MUSICIP.COM: *Open Fingerprint Architecture, Whitepaper*, 1. Auflage, March 2006.

[NMB01] NEUSCHMIED, H., H. MAYER und E. BATLLE: *Identi cation of Audio Titles on the Internet*. In: *Proceedings of International Conference on Web Delivering of Music*, Florence, Italy, November 2001.

[Nus90] NUSSBAUMER, H.J.: *Fast Fourier Transform and Convolution Algorithms*, Seite 84 ff. Springer Verlag, 2. Auflage, 1990.

[OKH02] OOSTVEEN, J., T. KALKER und J. HAITSMA: *Feature Extraction and a Database Strategy for Video Fingerprinting*. In: *LNCS 2314*, Seiten 117–128. Springer Berlin, März 2002.

[PB86] PRINCEN, J. P. und A. B. BRADLEY: *Analysis/Synthesis Filter Bank Design Based on Time Domain Aliasing Cancellation*. In: *IEEE Transactions on Acoustic, Speech and Signal Processing*, Seiten 1153–1161. IEEE, October 1986.

[Pea95] PEARSON, W.R.: *Comparison of Methods for Searching Protein Sequence Databases*. Protein Science, Seiten 1145–1160, 1995.

[Pen05] PENZ, M.: *Spaß mit Quaternionen*. 2005.

[PL88] PERASON, W.R. und D.J. LIPMAN: *Improved tools for biological sequence comparision*. In: *Proc. Natl. Academy Science, Vol. 85*, Seiten 2444–2448, 1988.

[PM96] PROAKIS, J.G. und D.G. MANOLAKIS: *Digital Signal Processing: Principles, Algorithms and Applications*. Prentice Hall, 3. Auflage, 1996.

[Pop03] POPCATCHER: *Webseite der Firma Popcatcher*. Website, 2003. http://www.popcatcher.com.

[Rab89] RABINER, L.R.: *A tutorial on hidden Markov models*. In: *Proceedings of the IEEE*, Seiten 257–286. Vol. 77, 1989.

[RC06] RIBBROCK, A. und M. CLAUSEN: *Content-based Search in Large Protein Databases*. in preparation, 2006.

[Rib00] RIBBROCK, A.: *Ein Audiocodierer zur Vermeidung von Generationseffekten aufbauend auf dem MPEG-1 Stardard*. Diplomarbeit, Universität Bonn, November 2000.

[Rib06] RIBBROCK, A.: *Ein universelles Java-RMI Framework zur verteilten inhaltsbasierten Suche*. Technischer Bericht, Universität Bonn, 2006.

[RK01] RIBBROCK, A. und F. KURTH: *An Embedding Codec for Multiple Generations Compression based on MPEG-1, Layer III*. In: *Proceedings of the 110th AES Convention*, 2001.

[RK02] RIBBROCK, A. und F. KURTH: *A Full-Text Approach to Content-Based Audio Identi - cation*. In: *International Workshop on Multimedia Signal Processing*, St. Thomas, US Virgin Islands, December 2002.

[RKV95] ROUSSOPOULOS, N., S. KELLEY und F. VINCENT: *Nearest Neighbor Queries*. In: *ACM-Sigmod*, Seiten 71–79, 1995.

[Röd02] RÖDER, T.: *A Group Theoretical Approach to Content-Based Image Retrieval*. Diplomarbeit, Universität Bonn, Institut für Informatik, October 2002.

[RWO95] REDL, S., M. WEBER und M. OLIPHANT: *An Introduction to GSM*. ISBN 0-89006-785-6. Artech House, Inc., Norwood, MA, USA, 1995.

[SASZ+97] S.F. ALTSCHUL, T.L. MADDEN, A.A SCHAEFFER, J. ZHANG, Z. ZHANG, W. MILLER und D.J. LIPMAN: *Gapped BLAST and PSI-BLAST: A New Generation of Protein Database Search Programs*. Nucleid Acids Research, Seiten 3389–3402, 1997.

[Say98] SAYLE, R.: *Webseite des Molecular Graphics Visualisation Tool (RasMol V2.6)*. Website, December 1998. http://www.rasmol.org/.

[Say01] SAYLE, R.: *PDB: Cruft to Content*. http://www.daylight.com/meetings/-mug01/Sayle/m4xbondage.html, 2001.

[Sha48] SHANNON, C. E.: *A mathematical theory of communication*. In: *Bell System Technical Journal*, Seiten 379–423 and 623–656, July and October 1948.

[Sha03] SHAZAM, FA.: *Webseite der Firma Shazam*. Website, 2003. http://www.popcatcher.com.

[Sho85] SHOEMAKE, K.: *Animating Rotation with Quaternion Curves*. In: *Siggraph*, Band 19, Seiten 245–254. ACM, 1985.

[SKS97] SILBERSCHATZ, A., H.F. KORTH und S. SUDARSHAN: *Database System Concepts*. McGraw-Hill, 3rd Auflage, 1997.

[SW88] SMITH, T.F. und M.S. WATERMAN: *Identi cation of Common Molecular Subsequences*. Journal on Molecular Biology, Seiten 195–197, 1988.

[SW93] SCHABACK, R. und H. WERNER: *Numerische Mathematik*. Springer-Verlag, 1993.

[Syn04] SYNCPLAYER: *Webseite des SyncPlayer Projekts*. Website, 2004. http://www-mmdb.iai.uni-bonn.de/projects/syncplayer/index.php.

[TGS03] TZANETAKIS, G., J. GAO und P. STEENKISTE: *A Scaleable Peer-to-Peer System for Music Content and Information Retrieval*. In: *Proceedings of the Fourth International Conference on Music Information Retrieval (ISMIR)*, Baltimore, Maryland, USA, October 2003.

[Tre95] TREBBE, H.: *The Münster Tagging Project - Errata in Rabinser s HMM-Tutorial*. Technischer Bericht, Arbeitsbereich Linguistik, Universität Münster, May 1995.

[Wag03] WAGENER, H.: *Merkmalsextraktion aus MP3-Datenströmen zur Audioindexierung und zum Audioretrieval*. Diplomarbeit, Universität Bonn, August 2003.

[Wan03] WANG, A.: *An Industrial-Strength Audio Search Algorithm*. In: *Proceedings of the Fourth International Conference on Music Information Retrieval (ISMIR)*, Baltimore, Maryland, USA, October 2003.

[WBKW96] WOLD, E., T. BLUM, D. KEISLAR und J. WHEATON: *Content-Based Classi cation, Search, and Retrieval of Audio*. In: *IEEE Multimedia*, Seiten 27–36, October 1996.

[weba] *Source code of R-trees and variations*. Website. http://www.rtreeportal.org.

[webb] *Webseite der Firma Gracenote*. Website. http://www.gracenote.com/.

[webc] *Webseite der Protein Data Bank (PDB)*. Website. http://www.rcsb.org/pdb/.

[webd] *Webseite des Folding at Home Projekts*. Website. http://folding.stanford.edu/.

[webe] *Webseite des Rosetta at Home Projekts*. Website. http://boinc.bakerlab.org/rosetta/.

[web05] *Periodensystem der Elemente*. http://de.wikipedia.org/wiki/Periodensystem, November 2005.

[web06a] *Aminosäuren*. http://de.wikipedia.org/wiki/Aminos

[web06b] *Aminosäuren*. http://de.wikipedia.org/wiki/Peptidbindung, Mai 2006.

[web06c] *Webseite des Pandora Projekts*. Website, 2006. http://www.pandora.com.

[WIN+05] WESBROOK, J., N. ITO, H. NAKAMURA, K. HENRICK und H.M. BERMAN: *PDBML: the representation of archival macromolecular structure data in XML.* Bioinformatics, 21:988–992, 2005.

[WMB99] WITTEN, I. H., A. MOFFAT und T. C. BELL: *Managing Gigabytes*. Van Nostrand Reinhold, 2 Auflage, 1999.

[WR97] WOLFSON, H.J. und I. RIGOUTSOS: *Geometric Hashing: An Overview*. IEEE Computational Science & Engineering, 4(4):10–21, /1997.

[WR01] WARD, S. und I. RICHARDS: *WIPO publication WO 02/073520A1*, 19 September 2002, (Priority 13 March 2001).

[WS00] WANG, A. und J. SMITH: *WIPO publication WO 02/11123A2*, 7 February 2002, (Priority 31 July 2000).

[Yan01] YANG, C.: *MACS: Music Audio Characteristic Sequence Indexing For Similarity Retrieval*. Technischer Bericht, Department of Computer Science, Stanford University, 2001.

[ZF90] ZWICKER, E. und H. FASTL: *Psychoacoustics: Facts and Models*. Springer-Verlag, Heidelberg, Germany, 1990.

[ZH05] ZIBASERESHT, R. und R. M. HARTSHORN: *Coordination chemistry of a terpyridine-tris(pyrazolyl) ditopic ligand*. In: *Dalton Transactions*, Seiten 3898–3908. The Royal Society of Chemistry, 2005.