TUHH
Technische Universität Hamburg-Harburg

TUHH STS
Software
Technology
Systems

Diplomarbeit

Entwicklung, Evaluierung und Optimierung einer Einfeldsuche für das Fahrplanauskunftssystem des Hamburger Verkehrsverbunds

Thomas Dziemian

thomas.dziemian@tu-harburg.de
Studiengang Informatik Ingenieurwesen
Matr.-Nr. 20523909

Erstgutachter: Prof. Dr. rer. nat. habil. Ralf Möller
Zweitgutachter: Prof. Dr. Karl-Heinz Zimmermann

Der Erfolgreichste im Leben ist der, der am besten informiert wird.
– Benjamin Disraeli (1804-81), brit. Politiker u. Schriftsteller, 1868 u. 1874-80 Premiermin.

1 Eidesstattliche Erklärung

Ich versichere, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und mich anderer als der im beigefügten Verzeichnis angegebenen Hilfsmittel nicht bedient habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

Hamburg, den _____ Unterschrift: _____

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

2 Einleitung

2.1 Motivation

Seit Mitte der 1990er Jahre gelang es den Informationstechnologien und -diensten, allen voran dem World Wide Web, sich in nahezu allen öffentlichen und privaten Bereichen des modernen Lebens zu etablieren. Der Trend zur elektronischen Informationsverarbeitung und Informationsspeicherung ist deutlich erkennbar. Waren früher Informationssysteme isolierte Lösungen für eine begrenzte Anwenderschar mit genau definierbaren und ebenso begrenzten Aufgaben, so müssen heutige Systeme eine weitaus breitere Masse von Benutzern zufriedenstellen. Mit den Möglichkeiten des Internets ist gleichzeitig auch der Bedarf nach Informationen gestiegen und damit auch eine Nachfrage nach intuitiv bedienbaren und überall zugänglichen Informationssystemen, die den Nutzern komplexe Informationen aus den unterschiedlichsten Themengebieten in verschiedensten Ausprägungen präsentieren. Bei derartigen Aufgaben gelangen erprobte Verfahren und Modelle der Datenverarbeitung, wie z.B. die relationale oder objektorientierte Datenmodellierung schnell an ihre Grenzen. Semistrukturierte[1] Datenmodelle und Datenbanken stellen einen anderen Ansatz dar. Sie bieten Mechanismen, um komplexe Daten flexibel zu verwalten und zu pflegen und eignen sich daher für eine breite Palette von Internetportalen, die Informationsdienste anbieten. Geoinformationssysteme sind eine von vielen Arten von Informationssystemen, die mit semistrukturierten Informationen umgehen müssen. Sie unterliegen heutzutage einer schnellen Verbreitung und spielen eine große Rolle in der Logistik und Handel, jedoch vor allem in der Freizeitindustrie und im Tourismus. Die Letztgenannten machen einen nicht zu unterschätzenden Anteil aus, laut dem Verband Bitkom e.V. haben in Deutschland mehr als 31 Millionen Menschen ihren Urlaub mit Hilfe des Internets geplant und gebucht. Es kann in dem Zusammenhang davon ausgegangen werden, dass ein erheblicher Anteil dieser Nutzer auf Geoinformationssysteme zugegriffen hat, beispielsweise auf Stadtpläne oder Internetauskünfte der Öffentlichen Verkehrsmittelbetreiber. Um hier ein Beispiel zu nennen, erhält der Hamburger Verkehrsbund pro Tag rund zweithunderttausend Anfragen bezüglich einer Bahn- bzw. Busverbindung[2]. Der steigende Trend zum Umsteigen auf die öffentlichen Verkehrsmittel wird die Anfragelast sicherlich noch erhöhen.

[1] http:// de.wikipedia.org/wiki/Semistrukturierte_Daten
[2] Quelle: HBT GmbH

2.2 Problemstellung

Die Firma HBT GmbH entwickelte in den neunziger Jahren das GEOFOX, ein Fahrplanauskunftssystem für den Hamburger Verkehrsbund. Seit diesem Zeitpunkt ist das GEOFOX-Portal kräftig gewachsen und ist heutzutage kaum wegzudenken.
Der Kern des GEOFOX ist die Suche von Verbindungen zwischen zwei Orten zu einer gegebenen Uhrzeit. Die Anfrage erfolgt dabei strukturiert in einem Menü bestehend aus mehreren Textfeldern und Auswahloptionen. Es ist jedoch für die nahe Zukunft geplant das Suchsystem auf die Verarbeitung von unstrukturierten Anfragen umzurüsten und gleichzeitig die Qualität der Suchergebnisse zu steigern.
Im Rahmen der vorliegenden Arbeit wurde ein Softwaresystem zur rangorientierten Suche von Objekten innerhalb der Fahrplanauskunft des Hamburger öffentlichen Personennahverkehr (GEOFOX) entwickelt. Der Schwerpunkt lag dabei auf Entwicklung einer Einfeldsuche und Untersuchung diverser Methoden zur Optimierung und Adaption.

2.3 Überblick

Das Kapitel 2 umfasst das zum Verständnis dieser Arbeit notwendige Wissen. Im ersten Abschnitt wird das GEOFOX vorgestellt, wobei der Fokus auf die bestehende Objektsuche gelegt wird. Der Gegenstand des zweiten und dritten Abschnitts ist das Information Retrieval und seine programmtechnische Umsetzung in dem Java-Framework Lucene. Kapitel 3 beschreibt kurz und formal die Anforderungen an das zu entwickelte System. Im Kapitel 4 wird das Konzept, im Kapitel 5 Entwicklung des Prototyps vorgestellt. Im Kapitel 6 werden Möglichkeiten zur Qualitätsoptimierung mit Hilfe evolutionärer Algorithmen untersucht und ausgewertet. Kapitel 7 enthält eine Zusammenfassung der Ergebnisse und zeigt Fragen auf, die sich im Laufe der Arbeit ergaben und in weiteren Forschungsarbeiten erörtert werden können.

3 Grundlagen

In diesem Kapitel werden die für diese Arbeit relevanten Grundlagen erläutert. Hierzu zählt zunächst das im Abschnitt 3.1 beschriebene GEOFOX, das Fahrplanauskunftssystem des Hamburger Verkehrverbunds. Der Abschnitt 3.2 beschäftigt sich mit der Funktionsweise und den Techniken des Information Retrieval, einer Methodik der Computerlinguistik, deren Zweck das Auffinden von Informationen in semistrukturierten Datenbeständen ist. Das Thema des Abschnitts 3.3 ist das Framework *Lucene*, eine speziell für den Einsatz von Information Retrieval ausgelegte Ansammlung von Klassen und Funktionen.

3.1 GEOFOX

Das Fahrgastinformationssystem GEOFOX wurde vom Unternehmen HBT GmbH Anfang der neunziger Jahre entwickelt und ist ein eingetragenes Warenzeichen. Im Jahre 1998 wurde GEOFOX als Internetdienst bereitgestellt und wird vom Hamburger Verkehrsverbund unter einer Lizenz verwendet. Die am meisten benutze Funktion ist die Zusammenstellung einer Verbindung zu einer bestimmten Zeit (persönlicher Fahrplan). Es werden weitere Informationen angeboten, beispielsweise die Aushänge der Haltestellen sowie die Linienfahrpläne. Die Fahrplanauskunft enthält verschiedene Optionen, mit denen der Benutzer seine persönlichen Präferenzen angeben kann. Die wahrscheinlich wichtigsten Optionen sind die zur Mobilität und zur Suche behindertengerechter Verbindungen, sowie die Wahl des Verkehrsmittels. Die Anzahl der Ergebnisse kann ebenfalls eingestellt werden. Bei der Eingabe des Start- und Zielpunktes wird der Benutzer mit Hilfe einer dynamisch kreierten Vorschlagsliste unterstützt. Ein intelligenter Suchalgorithmus minimiert die Antwortzeiten, die sich im Bereich von mehreren Hundert Millisekunden bewegen. Generell ist das Ergebnis unterhalb einer Sekunde auf dem Bildschirm. Zu erwähnen ist, dass der Algorithmus nicht auf vorgerechnete Ergebnisse zurückgreift, sondern die Strecken samt den Optionen jedes Mal neu berechnet. GEOFOX ist zum größten Teil in Java geschrieben.
Weitere Informationen zu GEOFOX sind unter *http://de.wikipedia.org/wiki/GEOFOX* zu finden, die Fahrplanauskunft erreicht man entweder über *www.hvv.de* oder direkt über *http://www.geofox.de*.

Die Fahrplanauskunft ist die am Meisten benutzte Funktion des GEOFOX. Der Benutzer gibt seinen Start- und Zielort ein und bekommt mehrere Vorschläge. Dabei kann er

optional einige Präferenzen angeben, beispielsweise wenn er gerne über eine bestimmte Zwischenhaltestelle fahren oder ein bestimmtes Verkehrsmittel benutzen möchte.
Im Folgenden wird anhand eines Beispiels der Suchvorgang in der Einzelplatzrechner[1]-Version näher erklärt. Als Erstes werden der Start und das Ziel eingegeben (Abb. 3.1).

Abbildung 3.1: Eingabemaske für die Verbindungssuche

Nach dem Drücken des Buttons *Suche starten* erscheinen die möglichen Verbindungen in tabellarischer Form (Abb. 3.2)

Abbildung 3.2: Verbindungsvorschläge

[1]Personal Computer, Laptop, Notebook, KEIN mobiles Endgerät

3.1.1 Ortssuche

Im vorangegangenen Abschnitt wurden sowohl der Start- als auch der Zielort korrekt eingegeben, das System konnte somit die Eingaben den jeweiligen Orten problemlos zuordnen. Es ist jedoch auch eine unvollständige oder fehlerhafte Eingabe von Orten möglich. Das System unterstützt dabei den Benutzer, indem es ihm eine Vorschlagsliste mit allen ähnlichen Orten präsentiert.
Ein Ort kann eine Haltestelle, eine Adresse bestehend aus Straßenname und Hausnummer sowie eine besondere Stätte (POI[2]) sein. Als Erstes wird das Menü vorgestellt und erklärt, im späteren Teil wird der Algorithmus grob umrissen.
Das Suchmenü (Abb. 3.3) ist überschaubar und intuitiv zu bedienen. In das erste Feld wird der Name der Stadt eingetragen, in das zweite der Name. Über die Auswahlbuttons über den Textfeldern wird der Typ des Ortes ausgewählt, es kann eine Haltestelle, eine Adresse oder eine besondere Stätte sein (POI). Sowohl der Start- als auch der Zielort werden auf diese Weise bestimmt.

Abbildung 3.3: Eingabefelder für die Ortssuche

Als Erstes wählt der Benutzer den Typ des Ortes aus, dann trägt er den Namen und die Stadt ein. Nachdem der Benutzer mindestens drei Buchstaben eingegeben hat, wird eine Liste mit Vorschlägen angezeigt (Abb. 3.4).

Entweder wählt der Benutzer einen Vorschlag aus, oder er tippt den Namen vollständig ein und drückt auf den Button Suche starten. Im folgenden Beispiel wurden die ersten vier Buchstaben der Station „Borgweg“ eingegeben. Die Anzeige einer Vorschlagsliste hat mehrere Vorteile:

- Der Ort ist vollständig beschrieben
- Der Ort ist korrekt und eindeutig
- Es ist keine weitere Tipparbeit nötig
- Es sind ähnliche Orte aufgelistet

Wenn der Benutzer keinen korrekten Namen eingibt und auf Suche starten drückt, wird ihm die gleiche Vorschlagsliste angeboten, diesmal jedoch als Auswahlliste (Abb. 3.5).

Abbildung 3.4: Vorschlagsliste

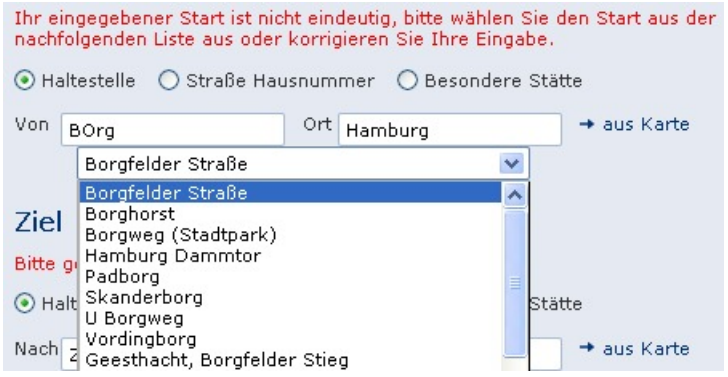

Abbildung 3.5: Auswahlliste

Die erzeugte Vorschlagsliste enthält zwei beachtliche Qualitätsmängel:

- Einige Vorschläge sind intuitiv nicht nachvollziehbar (z.B. Hamburg Dammtor)
- Die Relevanz der Orte wird nicht berücksichtigt (z.B. Borgweg Stadtpark)

Die Schwachpunkte des bestehenden Systems kommen noch deutlicher zum Vorschein, wenn eine inkorrekte Eingabe vorgenommen wird. Es ist genau dann der Fall, wenn sich Rechtschreibfehler oder Tippfehler einschleichen. Das vorangehende Beispiel aufgreifend geben wir jetzt „Botgwe“ein, womit die U-Bahn Station „Borgweg“ gemeint ist. Auf der Tastatur befindet sich neben dem Buchstaben „r“ der Buchstabe „t“, es ist also möglich, dass aus Versehen „t“ statt „r“ eingegeben wird. Das Fehlen des letzten Buchstabens soll in diesem Fall keinen Tippfehler darstellen, sondern die Angewohnheit des Benutzers simulieren Präfixe statt ganzer Namen einzugeben. Das Ergebnis ist in der folgenden Abbildung dargestellt (Abb. 3.6)

Abbildung 3.6: Vorschlagsliste bei inkorrekten Eingaben

3.1.2 Mobile Fahrplanauskunft

Die GEOFOX-Fahrplanauskunft ist nicht nur auf PCs beschränkt, sondern auch für die Benutzer von Smartphones zugängig. Bei den so genannten „mobilen“ Versionen der Fahrplanauskunft handelt es sich nicht um eigenständige Applikationen, sondern lediglich um vom Design her angepasste Internetseiten.

[2]Point of Interest

3.1.3 Fazit

Die Verbindungssuche des GEOFOX weist ein klar strukturiertes Menü auf und ist relativ schnell erlernbar. Die Vorschlagfunktion unterstützt den Benutzer bei der Eingabe, weist jedoch in der Qualität der Ergebnisse einige Mängel vor. Des Weiteren ist eine Suchmaske mit mehreren Feldern und einer Auswahlbox nicht mehr zeitgemäß.

3.2 Information Retrieval

3.2.1 Einleitung

Für die Lösung der vorgegebenen Aufgabenstellung wurden einige Methoden und Techniken des *Information Retrieval* verwendet, welche in diesem Abschnitt vorgestellt werden.
Information Retrieval bzw. Informationsgewinnung, ist ein Bereich der Informatik und Computerlinguistik, das sich mit rechnergestützten Suchen nach Textinhalten beschäftigt. Das Wort „retrieval“ (Wiederherstellung, Suche, Abfrage) bedeutet in dem Zusammenhang nicht die Strukturierung, Klassifikation oder Extraktion sondern das reine Auffinden von Informationen in großen Mengen von Dokumenten. Es wird vor allem bei der Suche von Webinhalten und in Bibliothekensystemen eingesetzt.
Grob gesehen werden für die praktische Umsetzung eines Information Retrieval Systems zwei Mechanismen benötigt, das *Indizieren* und das *Abfragen*. Beim Indizieren werden alle relevanten Dokumente samt ihrer Struktur in einem Index abgelegt. Der Index dient dem schnellen Auffinden von Dokumenten, enthält jedoch auch weitere Informationen, beispielsweise die Häufigkeit eines Terms (Wortes) innerhalb des Dokuments oder die Anzahl der Dokumente, in denen ein Term vorkommt. Je nach Implementierung werden weitere Informationen im Index gespeichert.
Beim Abfragen werden die gesuchten Terme im Index gesucht und die jeweiligen Dokumente bestimmt.
Im Bereich des Information Retrievals sind zahlreiche Modelle vorhanden, zu den bekanntesten zählen das *Boolesche Modell*, das *Vektorraummodell*, das *probabilistische Modell* und die *Language Modelle*. In den folgenden Abschnitten werden das Boolesche Modell und das Vektorraummodell vorgestellt. Für weiterführende Informationen zum Information Retrieval sei an dieser Stelle auf die im Rahmen dieser Arbeit benutzten Quellen verwiesen [Got09] und [MRS08].

3.2.2 Die Wahl des IR-Modells

Das im Rahmen dieser Arbeit verwendete Modell ist das **Erweiterte Boolesche Modell**, welches das Boolesche Modell um neue Komponenten zum Dokumentenvergleich erweitert. Das reine boolesche Modell ist ein auf der booleschen Logik basierendes Modell,

welches die Anfragen als logische Ausdrücke interpretiert und die Dokumente auf den Wahrheitsgehalt der Existenz der darin vorkommende Terme überprüft. Die Erweiterung dieses Modells stellt hauptsächlich die *Ähnlichkeitsfunktion* dar, die Anfragen mit Dokumenten vergleicht und das Maß der *Ähnlichkeit* auf einer Skala von *0* bis *1* ausdrückt. Es gibt zahlreiche Ansätze für die Definition der Ähnlichkeit, der in der vorliegenden Arbeit verwendete Ansatz beruht auf dem Vektorraum Modell. Das Vektorraum Modell betrachtet sowohl die Anfrage als auch die Dokumente als multidimensionale Vektoren und macht sich für den Vergleich Methoden der Geometrie zu Nutze. Im verwendeten Erweiterten Booleschen Modell werden beide Modelle kombiniert und ergänzen sich somit gegenseitig.
Die Wahl des Erweiterten Booleschen Modells hatte folgende Gründe:

- **Einfachheit**- sowohl das Boolesche als auch das Vektorraummodell sind klare und einfache Modelle für die Erstellung, Darstellung und den Vergleich von Anfragen und Dokumenten.
- **Flexibilität**- die im Modell enthaltenen Strukturen (z.B. Dokumente) sowie die repräsentierten Inhalte sind problemlos änderbar und erweiterbar.
- **Erweiterbarkeit**- das Modell kann problemlos um neue Elemente (z.B. Ontologien, Thesauren) erweitert werden.
- **Tools**- mit dem Framework *Lucene* [Fou] liegt bereits eine mächtige und performante Java-Bibliothek vor, die das Modell vollständig implementiert.
- **Kompatibilität mit anderen Modellen des Information Retrieval** - mit Hilfe hybrider Modelle ([Cai02], [VA10]) kann auf kundenspezifische Anforderungen besser eingegangen werden, wodurch die qualitative Kompetenz des Unternehmens gesteigert wird

Dokumentenvorverarbeitung

Im Vorfeld wird die Dokumentenvorverarbeitung vorgestellt, da sie unabhängig von einem Modell durchgeführt wird.
Bei der Vorverarbeitung der Dokumente werden zunächst alle Wörter *tokenisiert* (bzw. segmentiert), *normalisiert* und *gefiltert*. Beim **Tokenisieren** werden aus dem Volltext die einzelnen Wörter herausgenommen. Es klingt zwar nach einer einfachen Aufgabe, hat aber beim genauen Hinschauen seine Tücken. Hierzu betrachtet man den Satz „In den U.S.A wurde ein Hamburger mit einem Durchmesser von 1,5 Meter zubereitet“. Man könnte alle Wörter an den Leerzeichen und an Interpunktionen trennen, würde dann jedoch das Akronym „U.S.A“ als drei Terme speichern, das gleiche gilt für „1,5“. Es haben sich hierfür keine Standardtechniken durchgesetzt, vielmehr muss situations- und domänenabhängig geprüft werden, wie mit den Trennzeichen umgegangen wird.

Die **Normalisierung** beschreibt einen Vorgang, bei dem die Tokens auf eine Grundform gebracht werden und damit für den späteren Vergleich besser geeignet sind. Die einfachste Technik ist das *Case Folding*. Damit werden alle Wörter klein- bzw. großgeschrieben. Eine weitere, jedoch etwas komplexere Technik ist die *Lemmatisierung*, deren Zweck die grammatikalische Stammformreduktion ist. So wird beispielsweise aus dem Wort „gehst“ seine Stammform gebildet, und zwar „gehen“. Da die Lemmatisierung komplex und schwierig bei der Umsetzung ist, wird stattdessen das *Stemming* verwendet. Beim Stemming wird das Wort mit Hilfe einfacher Regeln auf eine „künstliche“ Grundform gebracht. Das Wort „gehst“ würde dann beispielsweise auf das Wort „geh“ reduziert werden, indem die Endung „st“ abgeschnitten wird. Die Wahl der Regeln muss gründlich überdacht werden, um die Reduktion zwei verschiedener Wörter auf den gleichen Wortstamm zu vermeiden. Im folgenden Beispiel werden zwei simple Reduktionsregeln aufgeführt, die diese Gefahr bergen:

$$\text{Regel 1: st} \rightarrow \text{„ “}$$

$$\text{Regel 2: t} \rightarrow \text{„ “}$$

Mit den obigen Reduktionsregeln werden die Wörter „gehst“ und „geht“ auf die Stammform „geh“ gebracht. Wählt man jedoch zwei Substantive, beispielsweise „Rast“ und „Rat“, so erhält man in beiden Fällen das Wort „Ra“. Es sollte lieber auf die Erstellung eines eigenen Stemmers verzichtet werden und auf fertige Stemmer zurückgegriffen werden. Bewährte Stemmer für viele Sprachen findet man unter *http://snowball.tartarus.org/*.
In der deutschen Sprache gibt es ein weiteres Problem, und zwar die **zusammengesetzten Substantive** (z.B. Versicherungsfachangestellter). Es wäre sinnvoll diese Substantive in ihre Bestandteile aufzuteilen und dann zu speichern. Hierfür werden so genante *compound splitter* eingesetzt.
Der Zweck der **Filterung** ist die Entfernung der *Stoppwörter* (irrelevanter Wörter), die nicht in den Index aufgenommen werden. Hierzu zählen vor allem Wörter wie Präpositionen (in, am, um), Konjunktionen (und, mit, oder) und sonstige grammatikalische Hilfswörter (z.B. Artikel). Der Grund ist, dass sie eine geringe bis keine semantische Bedeutung haben.

Boolesches Modell

Das Boolesche Modell ist einst der einfachsten und ältesten Modelle des Information Retrieval und wird bis heute in vielen Systemen eingesetzt.
Das Boolesche Modell verfolgt den Ansatz, Dokumente nur nach dem Vorkommen der in der Anfrage angegebenen Termen zu durchsuchen. Die Anfrage wird dabei als ein logischer Ausdruck formuliert. Die boolesche Logik in dieser Art der Suche ist darin enthalten, dass die Anfrage als logischer Ausdruck interpretiert wird und geprüft wird, ob ein Dokument diese erfüllt wird oder nicht. Wenn mehrere Wörter in der Suchanfrage auftauchen werden sie gemäß der logischen Operatoren ausgewertet:

- Lautet die Verknüpfung *UND*, so müssen alle gesuchten Begriffe in einem Dokument enthalten sein, damit es in die Ergebnisse aufgenommen wird
- Lautet die Verknüpfung *ODER*, so ist das Vorkommen eines einzigen Suchbegriffs in einem Dokument ausreichend
- Sind die Suchbegriffe durch Leerzeichen getrennt, so wird die komplette Phrase gesucht. In der Praxis wird meistens jedoch ein *UND* oder ein *ODER* implizit eingefügt (siehe GOOGLE).

Neben den booleschen Operatoren UND und ODER erlauben boolesche IR Systeme auch noch den Einsatz der Negation. Natürlich ist es möglich komplexe Ausdrücke zu definieren, die aus der klassischen Logik bekannt sind.
Nachdem das Prinzip des booleschen Retrieval erklärt wurde, wird anhand mehrerer Beispiele die Funktionsweise genauer erklärt. Auf eine formale Beschreibung wird verzichtet auf externe Literatur verwiesen [MRS08].

Beispiel 1
Gegeben seien vier Dokumente mit folgendem Inhalt:

- D1 = Lothar Matheäus war ein Fußballspieler
- D2 = Fußballspieler verdienen Geld
- D3 = Geld regiert die Welt

Es werden zu jedem Term seine Vorkommnisse in den Dokumenten gespeichert, in dem eine Liste mit den betroffenen Dokumentennummern angehängt wird. Diese Listen werden *Postings* genannt und bilden den invertierten Index:

- Lothar - 1, 4
- Matheäus - 1
- Fußballspieler - 1, 2
- Verdienen - 2
- Geld - 2, 3
- Regiert - 3
- Welt - 3

Es wurde in dem Beispiel bewusst darauf verzichtet, die Terme „war“, „ein“ und „die“ in den Index aufzunehmen, weil es sich um Stoppwörter handelt.

Beispiel 2

Es wird der Term „Fußballspieler“ gesucht. Das System schaut in den Postings nach und findet raus, dass der Term in den Dokumenten 1 und 2 vorhanden ist und liefert somit das Ergebnis:

- D1 = Lothar Matheäus war ein Fußballspieler
- D2 = Fußballspieler verdienen Geld

Im zweiten Beispiel wird die Anfrage „Fußballspieler *UND* Geld“ gestellt. Folgerichtig werden nur die Dokumente gefunden, in denen beide Stichwörter vorkommen und das Ergebnis lautet:

- D2 = Fußballspieler verdienen Geld

Beispiel 3
Im dritten Beispiel werden die Suchterme mit einem ODER verknüpft, die Anfrage lautet demnach „Fußballspieler ODER Geld“. Mindestens eines dieser Worte kommt in jedem Dokument vor, so dass die Ergebnismenge alle Dokumente enthält.

Beurteilung
Das boolesche Modell ist einfach zu implementieren, weist jedoch einige Nachteile auf:

1. Alle gefundenen Dokumente werden als gleichwertig betrachtet
2. Alle Terme (Wörter) werden als gleichwertig betrachtet
3. Jeder Suchbegriff wird nach dem strengen *Ja/Nein* Prinzip ausgewertet.

Vektorraum Modell

Den Problemen des booleschen Modells wird dadurch begegnet, dass die Terme gewichtet werden und eine Ähnlichkeit zwischen der Anfrage und den Dokumenten definiert wird.
Für die **Gewichtung** der Terme hat sich im laufe der Jahre die idf-tf Formel bewährt. Als Parameter werden einerseits die *Dokumentfrequenz* (Häufigkeit der Dokumente), in denen der Term vorkommt und andererseits die *Termfrequenz* Häufigkeit des Terms innerhalb des Dokuments) herangezogen und damit seine *Diskriminationskraft* (Aussagekraft, Relevanz) bestimmt. So wird ein Term, der in dem betreffenden Dokument selten, in den übrigen Dokumenten jedoch oft vorkommt als aussageschwach bewertet und seine Diskriminationskraft ist gering. Ein Term, der in wenigen Dokumenten präsent ist, dafür jedoch in dem betroffenen ganz oft, hat eine größere Diskriminationskraft und wird höher gewichtet. Der mathematische Ausdruck für die Formal variiert je nach Implementierung und wird im Zusammenhang mit Lucene im Kapitel 3.3 näher erläutert.
Die **Ähnlichkeit** zwischen einer Anfrage und einem Dokument wird geometrisch interpretiert und berechnet. Hierfür werden zunächst alle Dokumente als Vektoren eines

multidimensionalen Raumes dargestellt. Die Anzahl der Dimensionen des Vektorraumes entspricht dabei der Anzahl der im Gesamtvokabular vorkommenden Terme, jeder Term bildet nämlich eine Dimension des Vektorraums. Jedes Dokument d und jede Anfrage q werden somit als Tupeln betrachtet und haben dementsprechend folgende Form:

$$d = (g_1, g_2, ..., g_n)$$

$$q = (t_1, t_2, ..., t_m)$$

Die Ähnlichkeit eines Dokumentes und einer Anfrage wird mit einer Funktion berechnet, die den Anfrage- und Dokumentenvektor auf einen skalaren Wert abbildet:

$$sim(d, q) = c$$

Auf der Abb. 3.7 sind beispielhaft drei Dokumentvektoren *d1*, *d2* und *d3* sowie der Anfragevektor q grafisch dargestellt. Sowohl die Dokumentvektoren als auch der Anfragevektor sind auf einem Korpus von genau zwei Termen aufgebaut, weshalb der Vektorraum zweidimensional ist. Für die Berechnung der geometrischen Ähnlichkeit werden verschiedene Maße verwendet, in den meisten Modellen jedoch das Kosinus-Maß, das dem Skalarprodukt sehr ähnlich ist.

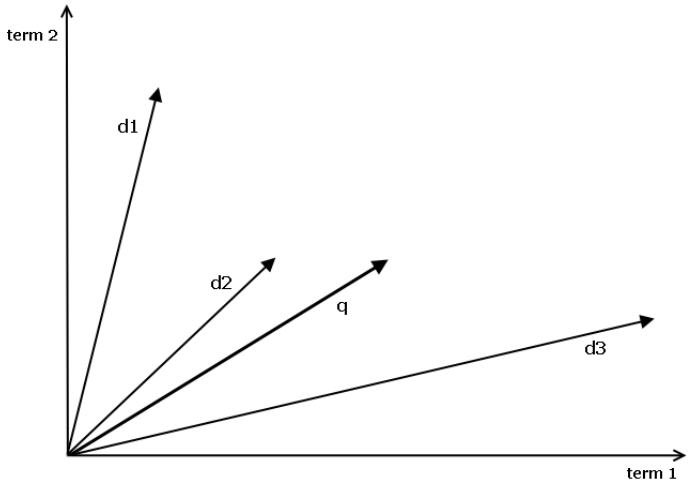

Abbildung 3.7: Vektorraummodell

Bei dem dem Kosinus-Maß wird das Ergebnis aus dem Sklarprodukt durch die Multiplikation der Längen der beiden Vektoren geteilt. Für die Wahl der Ähnlichkeitsfunktion sind auch andere Maße möglich, beispielsweise das Overlap-Maß oder Jaccard-Maß.

Beispiel

Das Vokabular bestehe aus den Termen „Suchmaschinenoptimierung“ und „Frage“, der Vektorraum hat also genau zwei Dimensionen. Es gebe zwei Dokumente $d_1 = (1, 0.5)$ und $d_2 = (0, 0.8)$, wobei der erste Tupeleintrag die Gewichtung des Terms „Suchmaschinenoptimierung“ und der zweite die Gewichtung des Terms „Frage“ ausdrücken.
Es seien ferner zwei Beispielsanfragen $q_1 = (1, 0)$ und $q_2 = (0.5, 0.5)$. Wenn für die Ähnlichkeitsfunktion das Skalaprodukt gewählt wird, ergeben sich folgende Ergebnisse:

$$d_1 * q_1 = (1, 0.5) * (1, 0) = 1$$

$$d_1 * q_2 = (1, 0.5) * (0.5, 0.5) = 0.5 + 0.25 = 0.75$$

$$d_2 * q_1 = (0, 0.8) * (1, 0) = 0$$

$$d_2 * q_2 = (0, 0.8) * (0.5, 0.5) = 0.4$$

3.3 Lucene

Lucene[Fou] ist ein in der Programmiersprache Java geschriebenes Framework, das verschiedene Klassen und Funktionen für eine leistungsstarke und effiziente Textsuche anbietet. Es wurde im Rahmen des Jakarta Projekts von der Apache Software Foundation entwickelt und findet weite Verbreitung[3]. Die Leistung, Skalierbarkeit und Portabilität machen Lucene zu einem ernstzunehmenden Konkurrenten kommerzieller Produkte.

3.3.1 Überblick

In Lucene ist das bereits erwähnte Erweiterte Boolesche Retrieval Modell mit Gewichtungen , welches die Charakteristika des klassischen booleschen Modells mit Anfragegewichtungen und des Vektorraum Modells mit Termgewichtungen vereinigt. In diesem Modell ist es möglich bestimmten Dokumentenbereiche und Teilen einer Anfrage eine höhere Gewichtung zuzuordnen. Des Weiteren beinhaltet Lucene ein Bewertungssystem, welches eine rangorientierte Suche ermöglicht. Für weitere Details zu Lucene sei an dieser Stelle auf weitere Quellen verwiesen [Fou] [McC10].
Im Folgenden werden einige Features von Lucene vorgestellt:

- Geringe RAM Anforderung, nur 1 MB Heapspeicher
- Indexgröße beträgt ca. 20-30% des indizierten Textes
- Rangorientierte Suche
- Verschiedene Anfragen möglich (z.B. fuzzy, boolean)
- Berücksichtigung der Dokumentenstruktur

[3] http://www.hebis.de/de/1ueber_uns/projekte/portal2/glossar.php#suchmaschtech

- Open Source unter der Apache Lizenz
- Komplett in Java geschrieben

Für die Wahl von Lucene haben folgende Punkte gesprochen:

- Die Suchobjekte im GEOFOX bestehen aus Attributen und haben damit eine Struktur
- Die einzelnen Attribute der Suchobjekte besitzen unterschiedliche Relevanz und können im Lucene unterschiedlich gewichtet werden
- Die Benutzeranfrage soll ein einem Textfeld erfolgen und keine Schlüsselwörter enthalten, mit Lucene ist die Transformation der Anfrage in eine komplexere Anfrage problemlos möglich
- Die rangorientierte Suche in Lucene erleichtert die Erstellung der Vorschlagslisten
- Lucene ist in Java implementiert, was eine einfache Integration in das GEOFOX ermöglicht.

In folgenden Abschnitten wird die Umsetzung der wichtigsten Funktionen erläutert.

3.3.2 Index

Die Erstellung eines Index ist der erste Schritt und wird von Lucene vollständig unterstützt. Für die Tokenisierung stehen diverse Methoden zur Verfügung, die Filterung und Normalisierung ist aufgrund der angelsächsischen Herkunft für die deutsche Sprache jedoch teils eingeschränkt. Die Indexstruktur ist in drei Schichten gegliedert (Abb. 3.8). dargestellt ist. Er besteht aus einer beliebigen Menge an Dokumenten, die eins oder mehrere Felder beinhalten. Sowohl die Namen der Felder als auch ihre Anzahl sind variabel und können für jedes Dokument individuell gestaltet werden. Jedes Feld stellt für die Terme einen Container dar, der eine beliebige Anzahl an Termen aufnehmen kann. Die Terme selber entsprechen meistens den Tokens, können jedoch auch aus Wortphrasen bestehen.

Das in Lucene umgesetzte Konzept eines Dokumentes wird im Folgenden anhand eines Beispiels für eine einfache Buchbeschreibung in einem Bibliothekensystem näher erläutert. Ein Buch wird dabei durch drei Angaben beschrieben, den Titel, die Schlagwörter und den Inhalt. Der Titel des Buches lautet *UML Einfach*, die Schlagwörter *UML, Modellierung, Softwareentwicklung* und der Inhalt *Unified Modelling Language, Klassendiagramm*. Die Angaben zu dem Buch werden in Lucene in einem Dokument beschrieben, das folgendermaßen strukturiert ist:

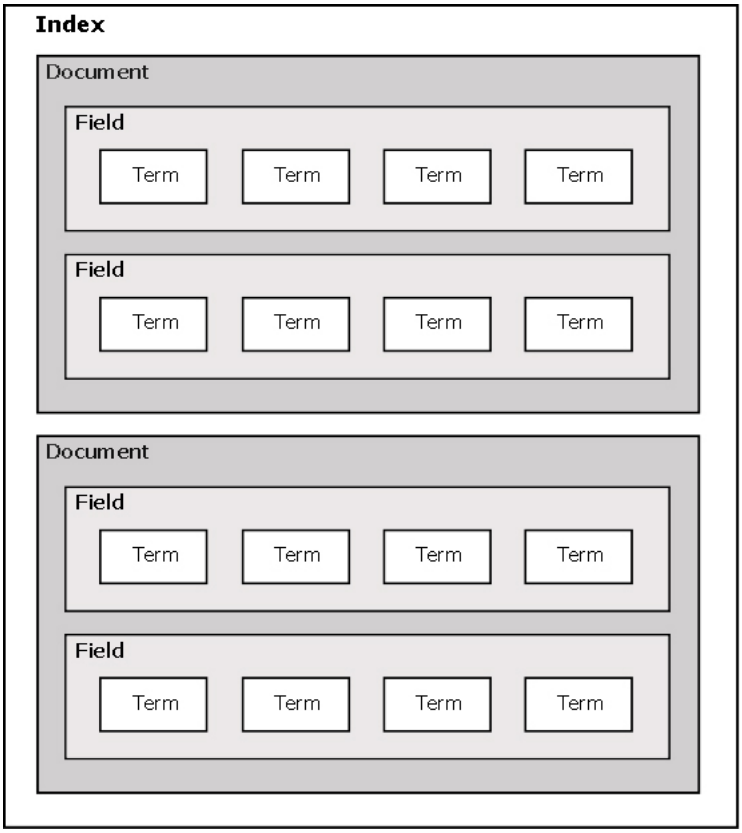

Abbildung 3.8: Indexstruktur in Lucene

Feldname	Term1	Term2	Term3	Term4
Titel:	UML	Einfach		
Schlagwörter:	UML	Modellierung	Softwareentwicklung	
Inhalt:	Unified	Modelling	Language	Klassendiagramm

3.3.3 Gewichtung

Beim Erstellen des Index können Dokumente und Felder gewichtet werden. Die Gewichtung beeinflusst die Relevanz der Dokumente und wirkt sich somit zum Teil auf die Rangordnung der Suchergebnisse aus. In dem bereits angeführten Beispiel ist es beispielsweise sinnvoll das Feld „Schlagwörter“ mehr zu gewichten als das Feld „Inhalt“. Des Weiteren können diejenigen Dokumente höher gewichtet werden, die später erschienene Bücher oder Bücher bekannterer Autoren repräsentieren.

3.3.4 Anfrage

Der erste Schritt des Suchvorgangs, nämlich die Erstellung einer Anfrage, ist für die Qualität des Systems besonders wichtig. Hiefür bietet Lucene vielfältige Möglichkeiten und

Alternativen, die über die Umsetzung des bereits im Abschnitt 3.2.2 vorgestellten Booleschen Modell weit hinausgehen. Das wohl wichtigste Konstrukt ist die boolesche Anfrage, mit der man jeden logischen Ausdruck mit den Operatoren UND, ODER und NICHT darstellen kann. Die booleschen Anfragen können aus mehreren Unteranfragen zusammengesetzt werden, so dass eine hierarchische Struktur erzeugt werden kann. Die atomaren Einheiten einer booleschen Anfrage sind meistens die Termanfragen, die die in das Suchfeld eingegeben Begriffe beinhalten.
Die eben vorgestellten Konstrukten decken zwar das Boolesche Modell vollständig ab, sind jedoch für ein modernes Information Retrieval ungenügend. Daher bietet Lucene weitere nützliche Anfragekonstrukte. Zu den wohl wichtigsten zählen Fuzzy-, Phrasen-, und Präfixanfragen. Bei den Fuzzyanfragen werden auch ähnliche Terme als Treffer gewertet, bei den Präfixanfragen die Präfixe. Die Phrasenanfrage ist ein spezieller Fall von Anfragen, weil hier die Reihenfolge der gesuchten Begriffe innerhalb der Phrase ausschlaggebend ist. Des Weiteren kann in der Phrasenanfrage der maximale Abstand der gesuchten Begriffe im Text eingestellt werden, damit der Kontext berücksichtigt wird. Sie spielen eine wichtige Rolle, wenn der Text viele Homonyme (Wörter mit mehreren Bedeutungen) im Suchtext enthält.
Das Konzept der Gewichtung findet in Lucene auch bei der Erstellung von Anfragen Anwendung. Auf diese Weise wird die höhere Relevanz bestimmter Anfrageteile ausgedrückt. Dieses Konzept eignet sich hervorragend für die Mehrfeldsuche, d.h. wenn die Suchbegriffe in mehrere Textfelder eingegeben werden und jedem Textfeld eine andere Relevanz zugesprochen wird.
Aus der theoretischen Sicht spielt die Gewichtung der Terme eine weitere Rolle, sie bildet eine Verbindung zum Vektorraum Modell, wo die Terme ebenfalls gewichtet werden. Durch die Gewichte wird der Anfragevektor modifiziert und somit auch die geometrische Ähnlichkeit mit dem Dokumentenvektor beeinflusst (siehe 3.2.2).

3.3.5 Suche

Nach dem der Index ordnungsgemäß erstellt wurde, wird auf diesem die Suche nach Dokumenten ausgeführt. Hiefür wird zunächst eine Anfrage formuliert, die dem gesuchten Dokument möglichst ähnlich ist.
Generell besteht der Suchprozess standardmäßig aus drei Schritten:

1. Transformation des vom Benutzer eingegebenen Textes in eine formale und wohlgeformte Anfrage
2. Ausführen der Suche
3. Auswertung und Aufbereitung der Ergebnisdokumente

Es sind prinzipiell auch andere Suchprozesse möglich, beispielsweise ist es in bestimmten Fällen sinnvoll die Suche unter der Berücksichtigung der Suchergebnisse mehrmals

auszuführen (*Relevance Feedback*).

3.3.6 Bewertung der Ergebnisdokumente

Jedes zu der Anfrage passendes Dokument wird bewertet. Hierfür wird das im Abschnitt 3.2.2 vorgestellte Kosinus-Maß verwendet. Auf diese Weise wird die Ähnlichkeit ausgedrückt und das Erstellen einer Rangliste ermöglicht.

3.3.7 Kernklassen

Die im vorigen Abschnitt beschriebenen Funktionen werden von speziellen Klassen umgesetzt. In diesem Abschnitt werden die wohl am meist benutzten und damit auch wichtigsten Klassen der Lucene Bibliothek vorgestellt und erläutert:

- IndexWriter - enthält alle Funktionen für Erstellung des Indizes und Einfügen von Dokumenten
- IndexReader - bietet Methoden für das Auslesen des Indizies
- Query - Oberklasse aller Query-Klassen, wie z.B. BooleanQuery und FuzzyQuery
- IndexSearcher - dient der Suche auf dem Index mit der gegebenen Query
- Similarity - berechnet den Score

Der ***IndexWriter*** ist für die Erstellung des Indizes zuständig. Der Index kann dabei sowohl in einer Datei als auch im RAM-Speicher hinterlegt werden. Die Klassen *Document* und *Field* sind separate Klassen und müssen im Indexierungsprozess für jedes Dokument instanziert werden. Eine Instanz des *IndexWriters* fügt dann das erstellte Dokument in den Index. Einem bereits erstellten Index können nachträglich sowohl neue Dokumente hinzugefügt als auch vorhandene gelöscht werden. Es ist ebenfalls möglich zwei oder mehrere Indizes zu einem Index zu vereinen. Der ***IndexReader*** ist für den Zugriff auf den Index zuständig. Eine Instanz des *IndexReaders* bietet diverse Funktionen für das Auslesen der im Index untergebrachten Daten, beispielsweise der Postings und der Termfrequenzen. Die Klasse ***Query*** repräsentiert eine Anfrage und ist die Oberklasse aller Query-Klassen. Für komplexere Anfragen steht die Klasse *BooleanQuery* zur Verfügung, in der jede von der Klasse *Query* erbende Klasse eingebunden werden kann. Die Hauptfunktion der Klasse ***IndexSearcher*** ist das Suchen nach Dokumenten zu einer gegebenen Anfrage. Beim Suchen wird die Klasse **Similarity** verwendet, in der die Ähnlichkeit zwischen der Anfrage und einem Dokument beschrieben ist.

3.3.8 Implementierung des Erweiterten Booleschen Modells

Das im Abschnitt 3.2.2 vorgestellte Information Retrieval Modell wird in Lucene erweitert. Die Erweiterungen betreffen hauptsächlich die Retrieval-Funktion und werden im

Laufe des Abschnitts näher erläutert.
In den vorigen Abschnitten wurde bereits gezeigt, wie in Lucene das boolesche Modell umgesetzt wurde, nämlich mit der Familie der Query-Klassen, die komplexe boolesche Ausdrücke erlauben und für die Vorauswahl der Dokumente eingesetzt werden. Mit Hilfe der formulierten Anfrage wird zunächst eine Menge der Dokumente erzeugt, die die in der Anfrage enthaltenen oder ähnliche Terme (siehe *FuzzyQuery*) enthalten. Die Dokumente dieser Menge und die Anfrage werden dann mit der Retrieval Funktion des Vektorraum Modells auf Ähnlichkeit untersucht und in einer Rangliste präsentiert.
Der zentrale Punkt Aspekt des Vektorraum Modells ist die allgemeine Problemstellung der Ähnlichkeit zweier Dokumentvektoren. Die mathematische Definition der Gleichheit ist einfach zu erklären, es brauchen die beiden Vektoren die gleiche Länge und die gleiche Richtung zu haben. Weil jedoch die Ähnlichkeit definiert werden muss, also ein Maß benötigt wird, müssen die beiden Merkmale Richtung und Länge im Kontext des praktischen Information Retrievals herangezogen werden.

Retrieval Funktion
Im folgenden Abschnitt wird die Retrieval Funktion von Lucene erklärt. Die Retrieval Funktion ist eine Abbildung eines Vektorpaares auf einen skalaren Wert (*Score*):

$$sim(v_1, v_2) = score$$

Den Grundbaustein der Retrieval Funktion bildet die schon aus dem theoretischen IR bekannte Formel für die Winkelbestimmung zweier Vektoren:

$$f(v_1, v_2) = \frac{v_1 * v_2}{|v_1| * |v_2|}$$

bzw.

$$sim(d, q) = \frac{d * q}{|d| * |q|}$$

mit d = Dokumentenvektor und q = Anfragevektor

Im theoretischen IR werden die Terme nur global über die *idf-tf* Formel gewichtet (siehe Abschnitt 3.2.2). Das bedeutet jedoch, dass die Position der Terme innerhalb der Dokumentenstruktur nicht berücksichtigt wird. Zur Lösung dieses Problems, wurde das bereits erwähnte Konzept der Felder eingeführt, wodurch eine exaktere Repräsentation eines Dokumentes erreicht wird. Als Beispiel sei in dem Zusammenhang die Struktur von HTML-Dokumenten, Buchbeschreibungen oder Gerichtsurteilen erwähnt. Sie sind alle in Textabschnitte gegliedert. Diese enthalten nur bestimmte Informationen und sind von unterschiedlicher Relevanz. Diese Tatsache wird in Lucene aufgegriffen, so dass eine Gewichtung der Terme über ihre Felder möglich ist.
Die in Lucene implementierte Formel für die Berechnung der Ähnlichkeit enthält weitere

Elemente, die nicht im klassischen Vektorraum Modell vorkommen. Sie wird im Folgenden ausführlich in mehreren Schritten hergeleitet:

- Die Normalisierung des Dokumentenvektors V(d) erweist sich als problematisch, weil damit die Dokumentenlänge nicht berücksichtigt wird. Das mag zwar in einigen Fällen gewollt sein, in anderen jedoch nicht, vor allem, wenn das gesuchte Wort dort selten vorkommt. In Lucene wird die Längennormierung entfernt und durch die Dokumentlängennormierung ersetzt (*docLenNorm(d)*). Der Wert dieser Normierung ist größer gleich dem Wert der Längennormierung, d.h. es gilt:

$$docLenNorm(d) >= |V(d)|$$

- Um bestimmten Dokumenten eine höhere Relevanz zuzuweisen werden diese gewichtet. Das bedeutet, dass jeder Score mit einem Boostwert (Gewichtungswert) multipliziert wird (*docBoost(d)*).

- Die Termgewichtung wird dadurch realisiert, dass die Felder gewichtet werden. Jeder Term innerhalb des entsprechenden Feldes bekommt seine Gewichtung.

- Es können ganze Anfragen oder nur Teile davon gewichtet werden, und zwar mit dem *queryBoost(q)*.

- Eine Anfrage besteht möglicherweise aus mehreren Subanfragen, die unterschiedliche Suchbegriffe enthalten. Je mehr Terme einer Anfrage im Dokument auftauchen, desto höher fällt die Gewichtung des Dokuments aus. Diese Tatsache wird mit dem Wert *coordFactor(q,d)* ausgedrückt.

Aus den eben beschriebenen Größen wird die Retrieval Funktion theoretisch auf folgende Weise beschrieben:

$$sim(q,d) = coordFactor(q,d) * queryBoost(q) * \frac{d * q}{|d| * |q|} * docLenNorm(d) * docBoost(d)$$

Aus der theoretischen Formel wird eine Implementierung hergeleitet, die zu Zwecken der Optimierung anders strukturiert ist. Vor allem wurde darauf geachtet, möglichst viele Werte bei der Indizierung und am Anfang einer Suche im Vorfeld zu berechnen und wiederzuverwenden. Die implementierte Formel lautet:

$$score(q,d) = coord(q,d) * queryNorm(q) * \sum(tf(t_d) * idf(t)2 * t.getBoost() * norm(t,d))$$

Die neuen Faktoren haben folgende Bedeutung:

- queryNorm(q): beschreibt die Längennormierung des Anfragevektors $\frac{1}{|q|}$

- tf: Anzahl der Vorkommnisse des Terms im Dokument (*term frequency*)

- idf: Das Inverse der Anzahl der Dokumente, in denen der Term vorkommt (*inverse document frequency*)
- norm(t,d): fasst die docLenNorm(d) und den docBoost(d) zusammen
- t.getBoost(): Gewichtung des Terms innerhalb der Anfrage

3.3.9 Zusammenfassung

Lucene überzeugt durch ein durchdachtes Konzept und eignet sich hervorragend für die Implementierung von IR-Systemen. Der intuitive Zugang zu den Konzepten des booleschen Modells und die vielfältigen Gestaltungsmöglichkeiten beim Einsatz des Vektorraum Modells machen Lucene zu einem mächtigen und dennoch benutzerfreundlichen Framework, das ohne große theoretische Kenntnisse effektiv eingesetzt werden kann.

4 Anforderungen

Das Ziel der vorliegenden Arbeit ist die Implementierung einer Suchfunktion, die über ein einziges Textfeld bedient wird. Die zu verarbeitenden Anfragen sollen vom Benutzer intuitiv gestellt werden, eine vordefinierte Struktur ist nicht vorgesehen. Das bedeutet vor allem, dass im Unterschied zu dem bestehenden Suchsystem keine explizite Angabe des Typs, des Ortes und des Namens erfolgt.
Die Anforderungen und Vorgaben wurden von der Projektleitung formuliert und werden im Folgenden aufgelistet:

- **Keine Schlüsselwörter** - zu dem gesuchten Objekt soll es nicht notwendig sein, eine Angabe über seinen Typ (Station, Adresse oder Point of Interest) zu machen. Das System soll in der Lage sein, den Typ des gesuchtes Objektes automatisch zu erkennen. Ähnliche Suchobjekte eines anderen Typs sollen jedoch nicht ignoriert werden.

- **Berücksichtigung von Rechtschreibfehlern** - die Wörter der Anfragen müssen nicht zwingend korrekt geschrieben sein, um erkannt zu werden. Für die Ähnlichkeit soll ein Maß definiert werden, der sich auf das Ranking auswirkt.

- **Erkennung von Präfixen** - alle Wörter sollen ebenfalls in der Präfixschreibweise (z.B. „jungf“ für Jungfernstieg) erkannt werden können, und zwar auch, wenn sie Rechtschreibfehler enthalten (z.B. „junf“ statt „jungf“).

- **Akzeptanz von Kürzeln** - die bisherige Unterstützung für Kürzel (z.B. hbf) soll erhalten bleiben.

- **Bevorzugung von Dokumenten mit gleicher bzw. ähnlicher Länge** - bei mehreren, ähnlichen Ergebnisdokumenten sollen diejenigen bevorzugt werden,

- **Vermeidung leerer Ergebnislisten** - das System soll die Anfragen nicht zu „streng“ auswerten und auch bei sehr ungenauen Anfragen Ergebnisse liefern.

- **Berücksichtigung der Suchhäufigkeit** - öfters angefragte Suchobjekte[1] sollen als „relevanter“ gewertet werden und bei der Erstellung der Rangliste den weniger relevanten Dokumenten bevorzugt werden.

[1] laut GEOFOX-Statistik

- **Erfüllung der Benutzererwartung** - die vom System gefundenen Dokumente sollen die Benutzererwartung erfüllen, bzw. diese approximieren. Dieser Aspekt wird in dem Zusammenhang als „Systemqualität“ bezeichnet.
- **Flexibilität** - das System soll flexibel für neue Dokumentenstrukturen sein (z.B. neue Felder in den Suchobjekten).
- **Kurze Antwortzeit** - die Antwortzeit von des Systems muss für mindestens 90% der Anfragen unter 100 ms liegen (Intel(R) Xeon(R) CPU X5365 @ 3.00GHz)
- **Einfache Integration** - diese Forderung wird dadurch erfüllt, dass das System in der Programmiersprache Java implementiert wird und somit die Integration in das GEOFOX-System problemlos ist.

Die meisten Anforderungen ergeben sich zum größten Teil aus den Features des aktuell eingesetzten Systems und aus den Wunschvorstellungen der GEOFOX-Projektleitung. Die gewünschte Toleranz des Systems gegenüber Rechtschreibfehlern und Präfixen wurde aus der strategischen Überlegung abgeleitet, den Suchdienst zukünftig für Nutzer von mobilen Geräten attraktiv zu machen. Die Flexibilität bezüglich neuer Dokumentenstrukturen soll das Hinzufügen neuer Informationen ermöglichen, beispielsweise Angaben zum Vorhandensein von Fahrstühlen an Stationen oder Textbeschreibungen zu den POIs.

5 Konzept

Ausgehend von den Anforderungen wurde zunächst ein Konzept für die Implementierung des Prototyps erstellt. Im Laufe der Implementierung wurden zwar einige Punkte geändert, das Kernkonzept blieb jedoch bestehen und wird in diesem Kapitel ausführlich erläutert.

5.1 Grobkonzept

Im Folgenden Kapitel werden konzeptuelle Überlegungen zur Umsetzung der im Kapitel 4 definierten Anforderungen vorgestellt.

Schlüsselwörter: Wie bereits erwähnt, beinhalten die Eingabedaten keine expliziten Schlüsselwörter und keine semantische Struktur (*bag of words*). Es scheint auf den ersten Blick unproblematisch, weil das verwendete IR-System damit umgehen kann und gerade aus diesem Grund gewählte wurde. Es sind jedoch viele Suchobjekte vorhanden, die einen ähnlichen oder gleichen Namen haben und trotzdem unterschiedlichen Typs sind. Das System muss also in der Menge der eingegeben Begriffe ein **Muster** erkennen und die Anfrage erweitern (*query expansion*).

Rechtschreibfehler, Präfixe und Kürzel: Dem Problem unvollständiger, fehlerhafter Eingabedaten wird damit begegnet, dass die Anfrageterme sowohl einem Fuzzy- als auch einem Präfixvergleich unterzogen werden. Ein weiteres Problem sind die zusammengesetzten Wörter und getrennt geschriebene Wörter, die auch zusammengeschrieben werden können. Für die Trennung wird ein *Compound Splitter* verwendet. Getrennte geschriebene Wörter werden zu einem Wort zusammengefasst. Es ist nicht möglich festzustellen, ob eine getrennte oder zusammengeschriebene Variante richtig ist, weil es sich bei fast allen Termen um Namen handelt. Deshalb sollen alle Varianten als Alternativanfragen formuliert werden.

Vermeidung leerer Ergebnislisten: Ein weiterer Aspekt bei der Verarbeitung von Anfragen ist die richtige Wahl der booleschen Operatoren, um alle gebildeten Erweiterungen und Alternativen in ein ganzheitliches logisches Anfragekonstrukt zu vereinen. Die gestellte Forderung nach einer gewissen Toleranz gegenüber falschen Anfragen wird dadurch erfüllt, dass die Anfrage aus mehreren Teilen besteht, die unterschiedlich „streng“ sind.

Erstellung einer Rangliste: Die Implementierung des Vektorraum Modells in Lucene beinhaltet ebenfalls das Scoring der Dokumente, so dass eine Rangliste implizit erzeugt wird. Die Länge der Ergebnisliste wird auf einen maximalen Wert gesetzt, der jedoch von der tatsächlichen Anwendung abhängen wird, um dem vorgegebenen Design der graphischen Oberfläche gerecht zu werden.

Kurze Antwortzeit: Die für die Ausführung einer Suche benötigte Zeit kann im Vorfeld nicht abgeschätzt werden und wurde in einem Test gemessen. Hierfür wurde ein Intel Core Duo E6850 Rechner verwendet, welcher mit dem Produktivrechner vergleichbar ist und ein Index mit ca. 58.000 Dokumenten angelegt (ca. 150.000 Wörter). Für eine reine Termsuche benötigte das Testprogramm eine Zeit von knapp unter einer Millisekunde, was für die Anforderungen mehr als ausreichend war. Es ist jedoch zu beachten, dass eine Fuzzy- und Präfixsuche erheblich mehr Zeit in Anspruch nimmt, unter anderem weil die Levensteindistanz zu allen Termen des Index berechnet wird. Da die Kennwerte jedoch nicht bekannt sind, kann eine endgültige Aussage über die Performance erst im Laufe der Implementierung getroffen werden.

Flexibilität: Die Erweiterung der Dokumentenstruktur ist allgemein unproblematisch. Das in Lucene implementierte Konzept des Dokumentes mit mehreren Feldern erlaubt eine flexible Handhabung der Dokumentenstrukturen und Feldinhalte.

Berücksichtigung der Relevanz: Aus den Benutzerdaten wird für jedes Suchobjekt die Häufigkeit seiner Anfrage ermittelt. Suchobjekte mit größerer Häufigkeit werden höher gewichtet, dadurch erhalten sie beim Scoring einen höheren Wert und werden beim Ranking den anderen Suchobjekten bevorzugt.

Bevorzugung kurzer Dokumenteninhalte: Die Forderung nach der Bevorzugung kurzer Dokumenteninhalten wurde von der Projektleitung gestellt und soll die „Genauigkeit“ erhöhen. Eine Anfrage trifft ein Dokument genau dann „genau“, wenn alle Terme und ihre Anzahl exakt übereinstimmen. Die beiden Überlegungen werden an einem Beispiel verdeutlicht:

Beispiel

Der Benutzer gibt in das System den Term „Alster“ ein. Sein Zielort ist tatsächlich der See mit dem Namen „Alster“. Der Term „Alster“ ist jedoch sowohl direkt als auch als Präfix in vielen Dokumenten enthalten. Hierzu zählen beispielsweise „Alster Canoe Club“, „An der Alster“ sowie Älsterdorfer Straßeünd Älsterschwimmhalle". Einige von ihnen haben aufgrund ihrer Relevanz eine höhere Gewichtung und werden als Erste präsentiert. Diesem unerwünschten Effekt wird entgegengewirkt, indem eine „Bestrafung“ längerer Dokumente eingeführt wird.

Erfüllung der Benutzererwartung - Systemqualität: Die Benutzererwartung ist ein zentraler Aspekt der Arbeit und wird als Maß für die Bewertung der Systemqualität

herangezogen. Der Grad der Erfüllung der „Erwartung“ ist gleich dem Grad der Übereinstimmung der erwarteten mit den gelieferten Dokumenten. Da die Befragung einer genügend großer Anzahl an Benutzern sowohl den finanziellen als auch den zeitlichen Rahmen dieser Arbeit sprengen würde, wird für die Beschaffung der Benutzerdaten die Vorschlager-Funktion(siehe 3.1) des GEOFOX-Systems benutzt. Ausgehend vom Cranfield Paradigma [MRS08] für Evaluation von IR-Systemen werden einzelnen Log-Einträge als Jurorbewertungen aufgefasst. Aus diesen werden zu jeder Anfrage die relevanten Dokumente bestimmt.
Im nächsten Unterabschnitt (5.1.1) wird der Begriff der Systemqualität näher erläutert, die Verwendung der Benutzerdaten und Evaluierung des Systems sind Gegenstand des Abschnitts 5.2.

5.1.1 Definition der Systemqualität

Für eine Bewertung der Systemqualität basierend auf der Benutzererwartung ist zunächst eine Definition der Qualität notwendig, die im Folgenden erläutert wird.
Die Dokumentensuche kann als eine Funktion aufgefasst werden, die eine Anfrage auf eine Teilmenge aller Dokumente abbildet. Die Funktion arbeitet dabei optimal, wenn die Benutzererwartung vollständig erfüllt wird. Zunächst wird eine mengentheoretische Definition aller betroffenen Elemente vorgestellt:

- Q: Menge aller Anfragen
- D: Menge aller Dokumente (Stationen, Adressen und POIs)
- R: Menge relevanter (erwarteter) Objekte zu einer Anfrage q
- E: Menge der Ergebnisobjekte einer Anfrage q
- $f(q) = E$: die Suchfunktion
- $T = E \cap R$: die Menge der Treffer (Abb. 5.1).

Die Beziehung zwischen der Suchfunktion und den jeweiligen Mengen ist in der Abbildung 5.1 schematisch dargestellt.
Die nun gesuchte Funktion soll mehrere Ziele gleichzeitig erfüllen:

1. Max |T| - die Anzahl der Treffer ist zu maximieren.
2. Min |E/T| - die Anzahl der Ergebnisdokumente, die keine Treffer sind, ist zu minimieren.

Für die Umsetzung einer Funktion zur Erfüllung der oben genannten Ziele sind messbare Maße notwendig.

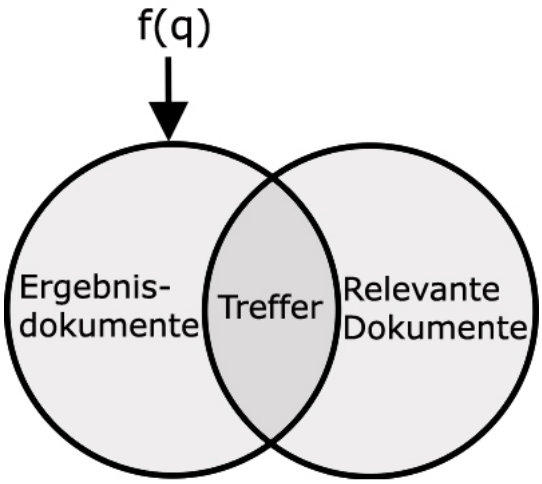

Abbildung 5.1: Treffermenge

Zunächst sei der Parameter ***Recall* (r)** definiert, der die Übereinstimmung der Mengen R und E ausdrückt:

$$r = \frac{|T|}{|R|}$$

Der Recall nimmt Werte zwischen 0 und 1 an. Ein Wert von 1 bedeutet, dass alle relevanten Dokumente in der Ergebnismenge enthalten sind, ein Wert von 0 entsprechend, dass keines der relevanten Dokumente gefunden wurde.
Ein weiterer Wert ist die ***Precision* (p)**, der die „Reinheit“ der Ergebnismenge ausdrückt:

$$p = \frac{|T|}{|E|}$$

Wie leicht nachvollzogen werden kann verhalten sich beide Werte divergent. Um dies verständlich zu machen, wird die Menge der Ergebnisse E betrachtet. Angenommen das System liefert alle existierenden Dokumente und damit auch alle relevanten Dokumente R. In dem Fall ergeben sich für den Recall der Wert $r = 1$ und für Precision $p \to 0$. Jetzt betrachten wir den Fall, dass nur ein relevantes Dokument gefunden wird. Damit erhalten wird die Werte $r = \frac{1}{|E|}$ und p=1.
Dieser Effekt stellt ein Problem dar, weil für Recall und/oder Precision auch gute Werte entstehen können, obwohl die Ergebnisse nicht qualitativ sind. Deswegen werden beiden Maße werden zu einem Maß verschmolzen, in dem ein harmonisches, gewichtetes Mittel über den beiden Werten berechnet wird:

$$\beta = \frac{(\beta^2 + 1) \cdot r \cdot p}{\beta^2 \cdot p + r}, \beta \in R^+$$

Hohe Werte von β gewichten den Recall höher, kleine Werte dagegen die Precision.
Da jedoch die Ergebnisse der Trefferliste eine bestimmte Reihenfolge haben, ist die Einführung eines Maßes notwendig, der auch diesen Aspekt berücksichtigt. Hierfür gibt es einige Ansätze, von denen der MAP (Mean Average Precision) am geeignetsten ist. Im Folgenden wird dieser vorgestellt.

Mean Average Precision

Die Mean Average Precision (MAP) ist ein weitverbreitetes Maß für die Bewertung von IR-Systemen. Formal betrachtet hat man zu jeder Anfrage q eine Menge der relevanten Dokumente und eine Menge der Ergebnisdokumente, von der einige relevant sind. Für die Berechnung des MAP muss als Erstes die *Precision at k* ermittelt werden, in dem die Precision der Teilmenge bis zum k-ten Dokument berechnet wird.

Beispiel:
Es seien eine Menge der relevanten Dokumente $\{d_2, d_5, d_7\}$ sowie die Ergebnismenge $\{d_1, d_2, d_3, d_4, d_5, d_6, d_7\}$ gegeben, dann lauten die Precision-Werte an den Stellen k=2, k=3, k=5 und k=7:

- $P_2 = \frac{1}{2}$
- $P_4 = \frac{1}{4}$
- $P_6 = \frac{2}{6}$
- $P_7 = \frac{3}{7}$

Das Prinzip des MAP besagt nun, dass zunächst zu jedem Treffer der Ergebnismenge die *Precision at k* berechnet und aus allen der Durchschnitt gebildet wird, nämlich die *Average Precision*. Auf diese Weise wird ein nach Rang gewichteter Durchschnittswert ermittelt:

$$AP = \frac{1}{N_r} \sum_{k=1}^{n} P_k$$

mit
N_r - Anzahl der Treffer
P_k - Precision at k

Beispiel:
In dem vorigen Beispiel wurden die Werte für *k* zufällig ausgewählt. Die *Average Precision* berücksichtigt jedoch nur die relevanten Dokumente, in dem Fall also k=2, k=5 und k=7. Es ergeben sich somit folgende Ergebnisse:

- $P_2 = \frac{1}{2}$
- $P_5 = \frac{2}{5}$

- $P_7 = \frac{3}{7}$

Die *Average Precision* beträgt also:

$$AP = \frac{\frac{1}{2} + \frac{2}{5} + \frac{3}{7}}{3} = \frac{31}{70} \approx 0.44$$

Für die Mean Average Precision wird der Mittelwert über alle Anfragen ermittelt:

$$MAP = \frac{1}{|Q|} \sum_{q} AP(q)$$

5.2 Feinkonzept

5.2.1 Analyse der Benutzerdaten

Im Vorfeld der Arbeit wurde eine Log-Funktion erstellt, die die vom Benutzer ausgewählten Vorschläge protokolliert. Hierfür wurde die mobile Version des GEOFOX verwendet und über 415.932 Anfragen samt den gewählten Vorschlägen gesammelt. Aus den Log-Daten wurde unter anderem für jedes Suchobjekt seine Suchhäufigkeit ermittelt. In der Tabelle 5.1 sind einige Logg-Daten aufgeführt.

Name (gewählt)	Stadt (gewählt)	Name (eingegeben)	Stadt (eingegeben)	Typ (eingegeben)
Barmbek	Hamburg	brbmek	Hamburg	0
Am Dalmannkai (Hamburg)		Am Da	Hamburg	1
Stephansplatz	Hamburg	steph	Hamburg	0
Elbe EKZ	Hamburg	elbe	Hamburg	2

Tabelle 5.1: Gesammelte Benutzerdaten

In den ersten beiden Spalten ist der ausgewählte Ort aufgeführt, in der dritten und vierten die Benutzereingabe. Die fünfte Spalte zeigt den eingegebenen Objekttyp (0 - Station, 1 - Adresse, 2 - POI).
Zunächst wurden die geloggten Daten aufbereitet, vor allem wurden mehrfach vorkommende und fehlerhafte Einträge gelöscht. Die Zahl der Einträge wurde damit auf 115.947 reduziert. Danach wurden diverse Statistiken erstellt, beispielsweise die über die Häufigkeit bestimmter Wörter und Anzahl relevanter Dokumente zu gegebenen Anfragen. Die Auswertung der Statistiken führte zu folgenden Erkenntnissen:

1. Die Verteilung der Suchhäufigkeiten der Suchobjekte ist nicht linear, sondern stark exponentiell.
2. Einige Suchobjekte haben eine sehr hohe Suchhäufigkeit
3. Viele Suchanfragen werden inkorrekt und abgekürzt eingegeben.
4. Einige Begriffe werde fälschlicherweise getrennt oder zusammen geschrieben, z.B. *Borg Weg* statt. *Borgweg* oder *Hamburgerstraße* statt *Hamburger Straße*.
5. Es kommen viele Wörter vor, die den Objekttyp *STATION* kennzeichnen (*implizite Schlüsselwörter*), beispielsweise *U, S, Bahn*.
6. Adressen werden meistens mit einer Nummer versehen, z.B. *Borgweg 28*.
7. Die Eingabe des Objekttyps wird vor allem bei der Suche nach POIs größtenteils nicht vorgenommen.

8. Die Anzahl der relevanten Dokumente zu einer bestimmten Anfrage variiert zwischen 1 und 87. Über die Hälfte der Anfragen enthält weniger als zehn relevante Dokumente.

Aus den Erkenntnissen heraus wurden folgende Lösungsansätze formuliert:

1. Die Häufigkeit der Suchobjekte kann nicht direkt übernommen werden. Die Verteilung wird deshalb linearisiert und skaliert.
2. Implizite Schlüsselwörter werden für die Erkennung des Typs verwendet.
3. Für die Trennung der Terme wird ein Compound Splitter eingesetzt.
4. Die Berechnung der Mean Average Precision (MAP) wird modifiziert, weil für alle Anfragen mit weniger als k relevanten Dokumenten trügerische Werte geliefert werden (siehe Abschnitt 5.2.8).

5.2.2 Repräsentation der Suchobjekte

Die Wahl von Lucene wurde durch die Struktur der vorliegenden Daten maßgeblich beeinflusst. Die Suchobjekte weisen aus der fachlichen Sicht eine semistrukturierte Form auf und lassen sich somit problemlos in das von Lucene vorgegebene Dokumentformat überführen. Die Suchobjekte waren in zwei Datenbanken persisiert. Für die Zwecke der Indexerstellung galt es zunächst die Suchobjekte in ein konformes Datenformat zu überführen. Alle irrelevanten Daten, wie z.B. Koordinaten und Kennnummern wurden rausgefiltert und eine semistrukturierte Textdatei (csv) erstellt, deren wichtigsten Spalten in der folgenden Abbildung tabellarisch dargestellt werden.

Type	Village	Place	Combined Name
STATION	Hamburg	U Borgweg	U Borgweg
STATION_ALIAS	Hamburg	U Bahnhof Borgweg	U Borgweg
ADDRESS	Hamburg	Mittelweg	Hamburg, Mittelweg
POI	Hamburg	Alster	Hamburg, Alster
ACRONYM	Hamburg	hbf	Hauptbahnhof

Tabelle 5.2: Repräsentation der Suchobjekte

Der Objekttyp *STATION_ALIAS* ist eine alternative Darstellung eines anderen, bereits vorhanden Objektes. Eine Zusammenführung der Aliase mit den Originalobjekten wurde bewusst nicht vorgenommen, weil es sich auf die Längennorm und damit auch auf das Scoring auswirkt. Das Selbe gilt für den Typ *ACRONYM*.

Die Spalten *Type, Village* und *Place* werden in den Index übernommen, die Spalte *Combined Name* dient der reinen Präsentation. Der aus Platzgründen nicht dargestellte *Key* ist für die eindeutige Zuordnung zu den im GEOFOX-System gespeicherten Suchobjekten

zuständig, wird aber auch für die Filterung der Ergebnisdaten verwendet. Mit Filterung ist die Aussortierung gleicher, mehrfach vorkommender Suchobjekte aus der Ergebnismenge gemeint.
Das im ersten Zeileneintrag gespeicherte Objekt wird zu dem in der Tabelle 5.3 dargestellten Dokument. Die Repräsentation aller Suchobjekte entspricht diesem Muster. Als

Feldname	Term 1	Term 2	Term 3
Type	STATION_ALIAS		
Village	Hamburg		
Place	U	Bahnhof	Borgweg

Tabelle 5.3: Beispielsdokument

Beispiel für ein POI wurde aus Platzgründen ein Suchobjekt mit einem kurzen Namen ausgewählt, ist jedoch für diese Art von Suchobjekten nicht stellvertretend. Meistens enthalten POIs längere Namen wie beispielsweise „Hamburger Biergarten (Nähe Landhaus Walter)“ oder „Hamburger Kontakt- und Informationsstelle für Selbsthilfegruppen“. Die höhere Anzahl der Terme macht die POIs für die Evaluierung des IR-Systems interessant, da dort die globalen Gewichte eine höhere Rolle spielen.

5.2.3 Berücksichtigung der Präfixe

Um das Maß der Umsetzung des Konzepts möglichst hoch zu halten, wurden einige Funktionen von Lucene bereits im Vorfeld getestet. Dabei wurde ein Problem mit der *PrefixQuery* entdeckt, welches im Folgenden besprochen wird.
Es wurde angedacht sowohl nach falsch geschrieben als auch abgekürzten Terme sowie einer Kombination von Beiden zu suchen. Es ist jedoch in Lucene nicht möglich die FuzzyQuery und die PrefixQuery miteinander zu kombinieren.
Beispielsweise wird das Suchobjekt mit dem Namen *Stadthausbrücke* über die Eingabe der falsch geschriebenen Präfix *stadhaus* nicht gefunden. Des Weiteren verursacht PrefixQuery einen enormen Perfomanceverlust und erzeugt eine große Anzahl von Termanfragen, so dass die maximale Anzahl von 1024 schnell überschritten wird.

Die Lösung des Problems wird im Folgenden beschrieben:
Die Präfixe aller Terme werden als eigenständige Terme behandelt und dem Index hinzugefügt. Hierfür werden neue Felder definiert, dessen Gewichtung von dem Abstand des Präfix zum Originalwort anhängt.
Die Speicherung der Präfixe eines jeden Terms hat jedoch Vor- und Nachteile:

- (+) Steigerung der Performance beim Suchen, weil alle Präfixe bereits im Index vorhanden sind

- (+) Angemessene Gewichtung der Abstände zum Orginalwort möglich
- (-) Vergrößerung des Index
- (-) Höhere Anzahl an Gewichtungen
- (-) Höhere Anzahl an Feldern

Um die Größe des Index nicht über die gesetzte Vorgabe hinaus wachsen zu lassen, sollten nicht zu viele neue Felder eingefügt werden. Nach mehreren Versuchen hat sich für die Repräsentation der Präfixe des Feldes *Place* die in der Tabelle 5.4 beispielhaft dargestellte Indexstruktur herausgebildet.

Place	Pref1	Pref2	Pref3	Pref4
Bergedorf	bergedor bergedo	berged berge	berg ber	be b
Berliner Tor	berline berlin to	berli berl t	ber be	b
Harburg	harbur harbu	harb har	ha	h

Tabelle 5.4: Präfixfelder für das Feld *Place*

Insgesamt sind für die Aufnahme der Präfixe sechs Felder notwendig, zwei für den Ort (*Village*) und vier für den Namen (*Place*). Somit enthält jedes Dokument drei Gewichtungen für den Ort und fünf Gewichtungen für den Namen, insgesamt also acht feldbezogene Gewichtungen.
Die Möglichkeit der Gewichtung der Abstände zum Orginalwort erweist sich als besonders sinnvoll, weil dadurch zum einen das Fuzzy Prinzip des Systems weiter ausgebaut wird und zum anderen das Maß der Ähnlichkeit für Präfixvergleiche direkt gesteuert werden kann.

5.2.4 Berücksichtigung der Suchhäufigkeiten

Die aus den Log-Daten ermittelten Suchhäufigkeiten werden für die Gewichtung der Dokumente verwendet und im folgenden Abschnitt näher betrachtet. Es wurde als erstes eine Statistik erstellt aus der die Anfragehäufigkeit jedes Suchobjektes hervorgeht. Die Verteilung der Häufigkeiten zeigte eine starke Krümmung, der höchste Wert lag bei 13.229 (ca. 8,76% der Anfragen) und hob sich von den anderen Werte stark ab. Des Weiteren trat der größte Teil der Suchobjekte in den Log-Daten nicht auf.
Am folgenden Beispiel wird die Problematik der Suchhäufigkeitsverteilung grob vorgestellt (Tab. 5.5).
Das erste Problem ist die Verteilung der Häufigkeitswerte - über 95% der Suchobjekte haben eine Häufigkeit mit einem Wert kleiner als 100. Des Weiteren weisen die Häufigkeiten im oberen Bereich große Sprünge auf, beispielsweise unterscheidet sich der Wert für „Hauptbahnhof“ mit 13229 Anfragen von dem Folgenden um einen Wert von 9988.

Name	Häufigkeit
Hauptbahnhof	13229
Altona	3241
S Landwehr	109
Bille Bad	6
Lomerstraße	0

Tabelle 5.5: Häufigkeiten - ausgewählte Beispiele

Das zweite Problem ist die Größe der Werte. Alle wichtigen Bahnstation weisen eine Häufigkeit auf, die den Wert 1000 überschreitet. Eine direkte Verwendung der Häufigkeiten als Dokumentengewichtung ist also nicht sinnvoll, weil es zu einer zu starken Dominanz der Bahnstationen führt.
Ein weiteres Problem sind die in den Benutzerdaten nicht vorkommenden Suchobjekte, welche 87,5% aller Objekte ausmachen. Eine Gewichtung von 0 kann nicht verwendet werden, weil die Gewichtungen bei der Berechnung des Scores miteinander multipliziert werden. Da es sich fast ausschließlich um Objekte handelt, die entweder Adressen oder POIs sind, entsteht ein weiteres Problem: unter den oben genannten Suchobjekten haben ca. 8% die gleichen Namen (Feld *place*). Beispielsweise kommt „Dorfstraße“ in 342 Städten bzw. Dörfern vor. Die Eingabe „Dorfstraße“ würde demnach eine Liste mit 342 Dokumenten produzieren, die allesamt denselben Score haben.
Die Probleme der starken Krümmung und der großen Werte wird dadurch gelöst, dass die Verteilungsfunktion abgeflacht wird und jede Häufigkeit auf einen Wert zwischen 0 und 20 abgebildet wird.
Im Folgenden werden die Lösungsschritte detaillierter erläutert:

1. Das Intervall [0,1392] wird in 19 Teilintervalle geteilt. Die Intervallgrenzen werden mit der Funktion $0.1792 \cdot i^3 + 5i$ bestimmt, wobei i die Nummer des Intervalls repräsentiert. Das letzte Intervall wird manuell hinzugefügt und beinhaltet alle Häufigkeiten über der Obergrenze des Intervalls. Die gewählte Funktion sorgt dafür, dass die Intervallgrößen exponentiell verteilt sind. Je größer die Nummer des Intervalls desto größer wird das Intervall selbst. Die Verteilung der Intervallgrößen ist auf der Abb.5.2 dargestellt, in der folgenden Tabelle 5.6 sind alle Intervalle aufgezählt: Für die Suchobjekte aus dem vorigen Beispiel ergibt sich eine neue Verteilung, die in der Tab.5.7 dargestellt ist.
   Der erreichte Effekt ist für alle Häufigkeiten auf den Diagrammen 5.3 und 5.4 erkennbar. Bei der normierten Verteilung entsprechen die Werte der x-Achse den bereits vorgestellten normierten Häufigkeiten. Bei den statistisch ermittelten Häufigkeiten stellt ein Wert der x-Achse ein Intervall dar, in dem sich die betroffene Häufigkeit befindet. Für die Erzeugen der gleich großen Intervalle wurde nicht die Zahlenspanne von 0 bis 13299 sondern von 0 bis 2500 gewählt, weil sich sonst in den meisten Intervallen keine Suchobjekte befinden würden. Außerdem wurde die-

Index:	1	2	3	4	5
Intervall:	[0,5]	[6,11]	[12,20]	[21,31]	[32,47]
Index:	6	7	8	9	10
Intervall:	[48,69]	[70,96]	[97,132]	[133,176]	[177,229]
Index:	11	12	13	14	15
Intervall:	[230,294]	[295,370]	[371,459]	[460,562]	[563,680]
Index:	16	17	18	19	20
Intervall:	[681,814]	[815,965]	[966,1135]	[1136,1328]	[1329,∞]

Tabelle 5.6: Häufigkeitsintervalle für die Normierung

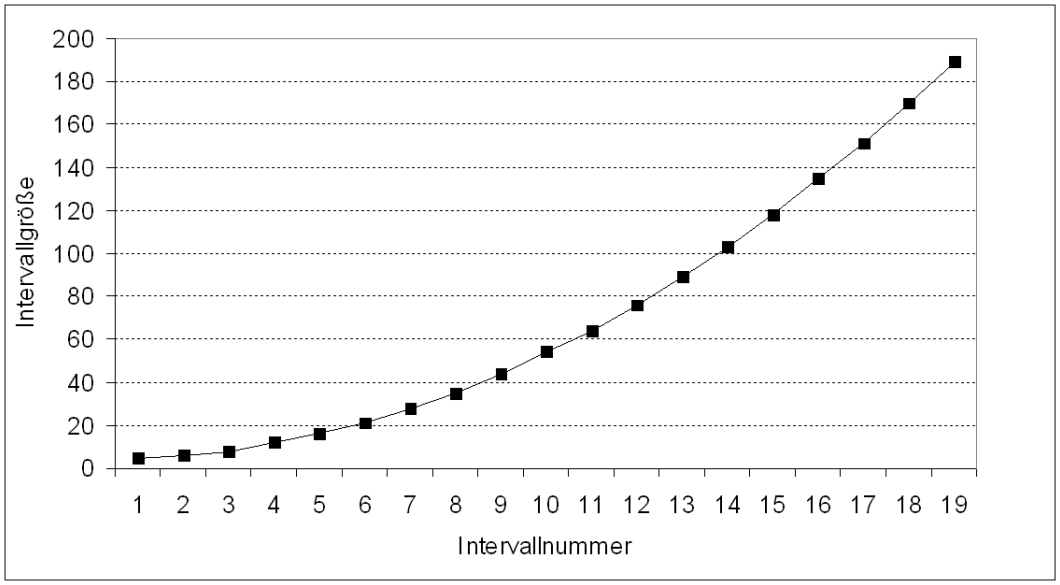

Abbildung 5.2: Verteilung der Intervallgrößen

ser Ansatz bei der Normierung ebenfalls angewendet und der Vergleich der beiden Kurven ist somit exakter. Zu Zwecke der Übersichtlichkeit sind die Verteilungen auf zwei Diagrammen dargestellt.

2. Dem Problem der vielen Suchobjekte mit gleichem Namen und gleicher Suchhäufigkeit wird damit begegnet, dass die Suchobjekte bereits in der Textdatei sortiert vorliegen. Als Sortierkriterium wird die Suchhäufigkeit der Städte bzw. Dörfer der Suchobjekte verwendet. Das führt dazu, dass bei gleichem Score die Suchobjekte in der gleichen Reihenfolge ausgegeben werden, wie sie eingelesen worden sind.

Modifikation der häufigkeitsbezogenen Dokumentengewichtung

Die Auswirkung der Gewichtung der Dokumente entsprechend der Suchhäufigkeiten wurde prototypisch getestet und modifiziert. Es hat sich nämlich gezeigt, dass die Gewichtung eines ganzen Dokumentes die Terme in den *village*-Felder zu stark gewichtet. Eine Gewichtung der *place*-Felder scheint mehr sinnvoll zu sein.

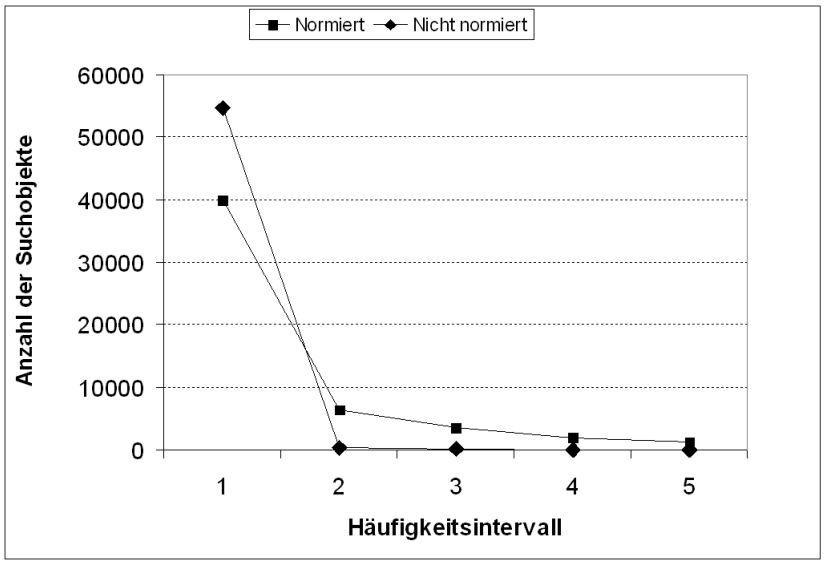

Abbildung 5.3: Vergleich der Häufigkeitsverteilung vor und nach Normierung (Intervalle 1 bis 5)

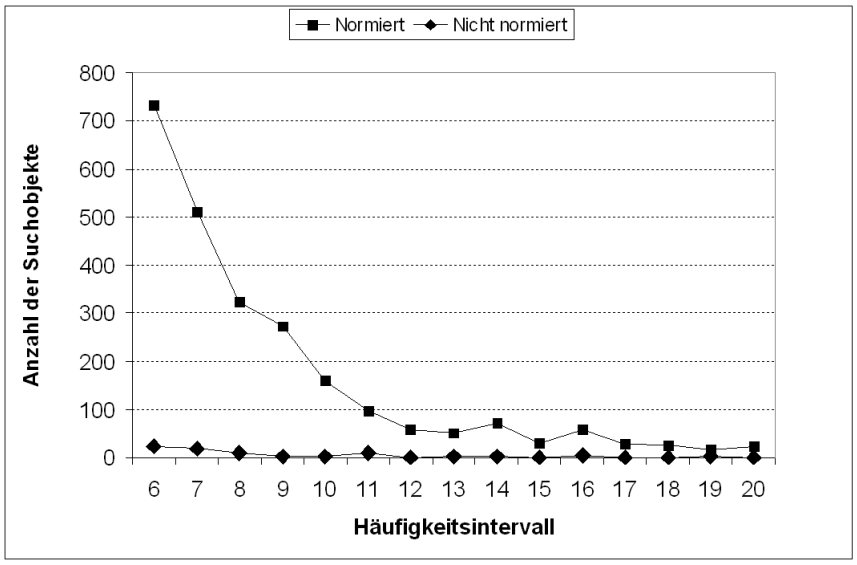

Abbildung 5.4: Vergleich der Häufigkeitsverteilung vor und nach Normierung (Intervalle 6 bis 20)

Name	Gewichtung
Hauptbahnhof	20
Altona	20
S Landwehr	14
Bille Bad	2
Lomerstraße	1

Tabelle 5.7: Normierte Häufigkeitsverteilung ausgewählter Beispiele

5.2.5 Behandlung inkorrekter Eingaben

Die in Lucene enthaltene Klasse *FuzzyQuery* ermöglicht das Finden von Termen mit Rechtschreibfehlern. Wie bereits besprochen bestand bezüglich der Performance der FuzzyQuery Unklarheit und es wurde eine Untersuchung im Vorfeld angedacht.
Wie erwartet bewirkt eine Verringerung der minimalen Ähnlichkeit keinen Performanceverlust und kann vernachlässigt werden. Eine Verringerung der minimalen Präfixlänge wirkt sich dagegen stark auf die Suchzeit aus. Bei einem Wert von 0 sind die Einbußen besonders hoch, die Suchzeit steigt dabei auf das zehnfache wie bei einem Wert von 1. Im Testfall wurde ein Index über zwei Spalten mit ca. 58.000 Dokumenten erstellt und eine Anfrage mit einem Suchterm gestartet. Die durchschnittliche Suchzeit von unter 2 ms bei einer min. Präfixlänge von 1 steigt auf über 15 ms, wenn die min. Präfixlänge auf 0 gesetzt wird. Es sei an dieser Stelle erwähnt, dass der Index nur aus zwei Spalten bestand und die Anfrage nicht erweitert wurde.
Es ist davon auszugehen, dass die Antwortzeit um einen beachtlichen Faktor steigen wird. Es ist jedoch in der nahen Zukunft ein neues Lucene-Release geplant, welches die FuzzyQuery auf eine andere Weise implementiert und einen 100-fachen Geschwindigkeitszuwachs verspricht (*http://blog.mikemccandless.com/2011/03/lucenes-fuzzyquery-is-100-times-faster.html*).

5.2.6 Compound Splitter

Compound Splitter werden in Information Retrieval Systemen zur Qualitätssteigerung eingesetzt [Dal06]. Ihre Aufgabe ist es lange, aus mehreren Teilwörtern zusammengesetzte Wörter in die Bestandteile zu zerlegen. So werden auf der einen Seite fälschlicherweise zusammengeschriebene Wörter korrigiert und auf der anderen Seite neue, ähnliche Suchbegriffe produziert. Compound Splitter enthalten Wörterbücher, in denen die Teilwörter nachgeschlagen werden und verursachen bei vielen Wörter einen Performanceverlust.
Für die Zwecke der vorliegenden Arbeit reicht es jedoch, wenn die in den Suchobjekten vorkommenden Teilwörter gespeichert werden.

Beispiel:

Der Benutzer sucht nach der U-Bahn Station „Lübeckerstraße“, deren Name korrekterweise getrennt geschrieben wird. Das System weiß nicht, dass die Schreibweise falsch ist, erkennt aber trotzdem, dass die Einzelteile der Eingabe die Wörter „Lübecker“ und „straße“ lauten. Aus den gewonnenen Teilwörtern wird eine neue Anfrage formuliert und als Alternative zur Originalanfrage hinzugefügt. Zur Steigerung der Robustheit des Systems wird das Wort „straße“ auch in falsch geschriebenen Varianten und in der Präfixschreibweise erkannt (z.B. „strase“, „str“ usw.).

5.2.7 Gewichtungen und Gewichtsparameter

Die **Gewichtungen** bilden ein zentrales Element zur Steuerung des Rankings. Sie werden nach globalen und lokalen Gewichtungen unterschieden.
Ein Gewichtung ist genau dann global, wenn dem betroffenen Element, Feld oder Term, in jedem Dokument das gleiche Gewicht zugewiesen wird. Die bereits erwähnten inverse Dokumentenfrequenz (*idf*) ist beispielsweise ein *globales* Gewicht, es wird für jeden Term einzeln bestimmt und gilt in der gesamten Dokumentenmenge. Auch die Gewichtungen der Felder *village* und *place* sowie ihrer Präfixfelder sind global.
Eine Gewichtung ist genau dann lokal, wenn das gewichtete Element in unterschiedlichen Dokumenten auch unterschiedlich gewichtet wird. Zu lokalen Gewichtungen gehören die Häufigkeitsgewichtung und die Längengewichtung.
Gewichtsparameter sind Einstellwerte, die die globalen und lokalen Gewichtungen skalieren. Die Skalierung ist für die Justierung des Systems notwendig. Beispielsweise ist der idf-Wert eine Gewichtung, die aus dem Vektorraum Modell stammt und für die vorliegende Problemstellung nicht unbedingt optimal sein muss. Die Häufigkeitsgewichtung kann ebenfalls justiert werden, es ist nämlich nicht klar, ob das gewählte Intervall von 1 bis 20 zu angemessenen Ergebnissen führt.
Gewichtungen von Anfragetermen sind in Lucene ebenfalls möglich und spielen bei der Formulierung einer booleschen Anfrage eine große Rolle, beispielsweise bei der Gewichtungen von bestimmten Anfrageteilen (OR-Query) oder Anfrageerweiterungen (Alternativanfrage).
Im Folgenden werden die Gewichtungen und Gewichtsparameter detaillierter erläutert, die Ermittlungen optimaler Werte bildet den Gegenstand des Kapitels 7.

Felder *village* und *place* - die beiden Felder werden unterschiedlich gewichtet, wobei *village* ein höheres Gewicht zugesprochen wird. Erstens haben die Terme dieses Feldes einen höheren Informationsgehalt, zweitens ist der Name eines Suchortes für den Benutzer einprägsamer und wird öfter verwendet.
Die Gewichtung der Präfixe wird im nächsten Unterabschnitt 5.2.7 diskutiert.

Häufigkeit - zentraler Parameter für die Beeinflussung der Reihenfolge der Ergebnisdokumente. Die aus den Benutzerdaten ermittelte Relevanz eines Suchobjektes fließt

direkt in die Gewichtung seines Dokumentes und wird beim Scoring berücksichtigt. Die Häufigkeit erhält zunächst einen Wert zwischen 1 und 20.

Häufigkeitsparameter - Steuert das globale Maß des Einflusses der Häufigkeit. Soll vor allem eine Über- bzw. Unterbewertung der Dokumentenhäufigkeit verhindern. Beträgt beispielsweise die Häufigkeit der Station *Hauptbahnhof Hamburg* den Wert 20 und der Gewichtungsparameter 0.5, so ergibt sich ein Gewichtungswert von $20 \cdot 0.5 = 10$.

Termfrequenz *tf* - dieser Wert wird auf 1 gesetzt, weil die Häufigkeit des Terms innerhalb des Dokumentes vernachlässigt wird.

IDF - Inverse Dokumentfrequenz - hängt von der Auftrittshäufigkeit eines Terms im Korpus ab und ist global gültig. Wird von Lucene berechnet.

IDF-Parameter - justiert die Gewichtung IDF.

Feldlänge - gewichtet den Term in Abhängigkeit von der Länge seines Feldes, wird von Lucene berechnet.

Längenparamter - justiert die Gewichtung bezüglich der Feldlänge.

Gewichtungen der Unteranfragen - die Unteranfragen werden unterschiedlich gewichtet. Eine aus dem erkannten Muster abgeleitete Anfrageerweiterung bekommt eine eigene Gewichtung, alternative Anfragen (z.B. durch Compound Splitting) ebenfalls.

Auf dem folgenden Bild (Abb.5.5) wird der Einfluss der Gewichte und Parameter auf die Berechnung des Scores gezeigt.

Gewichtung der Präfixfelder

Bei der Gewichtung der Präfixfelder wird die Entfernung des Präfixwortes zum Stammwort herangezogen. Das bedeutet, dass kürzere Präfixe ein kleineres Gewicht bekommen als Längere.

Beispiel 1:
Die Felder *Place*, *Pref1*, *Pref2*, *Pref3*, *Pref4* und *Pref5* werden zunächst exponentiell gewichtet (Tabelle 5.4). Das Feld *Place* wird mit dem Anfangswert von 6.0 und das Feld *Pref4* mit dem Endwert von 3.0 gewichtet, als Basis wird der Wert 1.3 gewählt. Die jeweiligen Gewichtungen werden in Tabelle 5.8 aufgeführt.

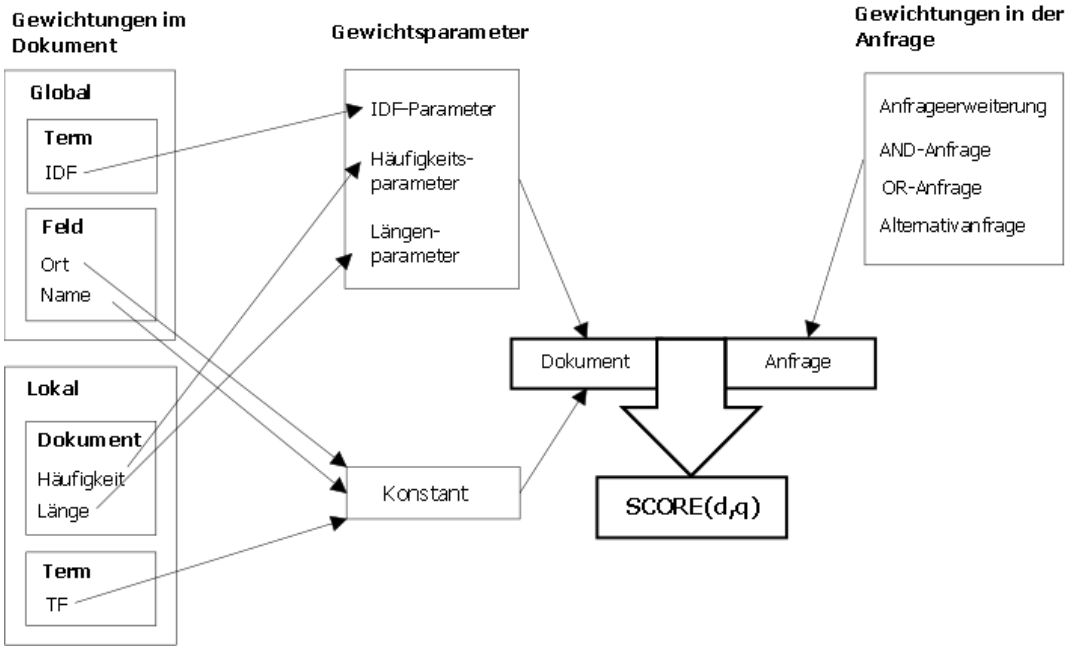

Abbildung 5.5: Gewichtungen und Gewichtsparameter

Place	Pref1	Pref2	Pref3	Pref4
6.0	4.94	4.12	3.48	3.0

Tabelle 5.8: Beispiel für exponentielle Gewichtsverteilung

Mit der Wahl der exponentiellen Verteilung wurde folgende Idee ausgedrückt:
Je höher der Abstand des Präfixes zum Originalterm, desto höher ist die Anzahl anderer Terme, die durch diesen Präfix ebenfalls repräsentiert werden. Damit verringert sich die Eindeutigkeit des Präfixes bezüglich des Originalterms. Mit einer abnehmenden Gewichtsbestrafung wird der Verlust der Eindeutigkeit kompensiert. Das Gleiche kann von der anderen Seite betrachtet werden. Wird ein Buchstabe weniger eingegeben, z.B. *Borgwe*, statt *Borgweg* so ist die Wahrscheinlichkeit dennoch gering, dass der Präfix mit dem Namen einer anderen Station identisch ist. Sollte es dennoch der Fall sein, so muss die andere Station eine deutlich höhere Gewichtung bekommen, weil sie direkt getroffen wird und somit *einzigartiger* ist.

5.2.8 Evaluierung

Nachdem im Abschnitt 5.1.1 das Konzept und die formalen Grundlagen der Qualität eines IR-Systems bereits vorgestellt wurden, werden in diesem Abschnitt die Maßnahmen zur Umsetzung der Qualitätsmessung beschrieben.
Wie bereits angesprochen wurden Benutzernanfragen und die ausgewählten Vorschläge geloggt. Auf diese Weise wurden Testdaten angesammelt, die für die Evaluierung des

Systems verwendet wurden. Zu jeder Anfrage eine Liste mit den von den Benutzern gewählten Dokumenten erstellt. Die Liste wird als die Menge der relevanten Dokumente aufgefasst und für die Berechnung der Average Precision verwendet.

Nach genauerer Betrachtung der Benutzerdaten ergab sich jedoch folgendes Problem: Circa 80% der untersuchten Anfragen sind weniger als 10 relevante Dokumente zugeordnet. In diesen Fällen liefert der im Abschnitt 5.1.1 vorgestellte Durchschnitt der *precistion at k* Werte (Average Precision) trügerische Ergebnisse, unter anderem wird der höchste Wert von 1 niemals erreicht.
Das Problem wird im Folgenden an einfachen Beispielen erläutert. Die Ergebnisliste soll dabei die Länge 10 haben.

Beispiel 1 Zu der Anfrage gibt es 22 relevante Dokumente und die Ergebnisliste enthält alle. Die average Precision beträgt somit genau den Wert 1.

Beispiel 2 Zu der Anfrage existieren genau 9 Dokumente und die Ergebnisliste enthält alle in den Positionen 1 bis 9. Es ergibt sich der Wert 0,9.

Beispiel 3 Die Anfrage soll genau ein relevantes Dokument liefern, in der Ergebnisliste steht dieses an erster Position. Die Average Precision beträgt 0,1.

In den vorgestellten Szenarien erhalten die Ergebnisse unterschiedliche Bewertungen, obwohl alle Ergebnismengen die höchst mögliche Qualität aufweisen. Es ist also eine Berechnungsmethode notwendig, der alle in den Szenarien vorgestellten Ergebnisse mit dem Wert 1 bewertet.

Auf den folgenden Diagrammen sind die Average Precision Werte für Anfragen mit einem und mit zwei relevanten Dokumenten dargestellt. Die Einträge auf der x-Achse bezeichnen die Position eines Dokumentes in einer Ergebnisliste der Länge 10. Der Verlauf für zwei Dokumente zeigt nur Positionen von 2 bis 10, weil die Position des ersten Dokuments auf 1 festgelegt ist. Der variable x-Wert ist also in beiden Fällen die Position genau eines Dokumentes.

Es ist am Diagramm zu erkennen, dass kein Wert von 1 erreicht werden kann, wenn die Anzahl der Dokumente kleiner als 10 ist. Des weiteren ist der Abfall der Precision bei 2 Dokumenten relativ gering. Ist das zweite Dokument auf Position 4, so beträgt die Precision 0,15, sinkt die Position auf den Wert 10, so ergibt sich ein Wert von 0,12.
Das Problem bei der Berechnung der Average Precision ist die feste Anzahl der Top Dokumente, durch die der kumulierte Wert der k-Precisions geteilt wird. Bei einer Menge von über 10 relevanten Dokumenten mag es durchaus sinnvoll sein, bei kleineren Mengen jedoch nicht.
Um für wenige Dokumente den Wert 1 zu erreichen, muss die Summe der k-Precisions

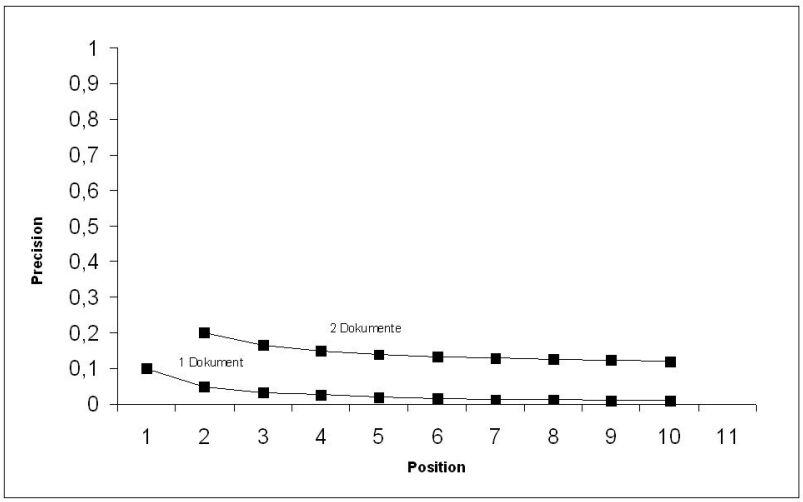

Abbildung 5.6: Average Precision für ein und zwei Dokumente

durch die Anzahl der relevanten Dokumente geteilt werden:

$$Avr.Prec. = \frac{p_k1 + p_k2 + ... + p_kn}{|R|}$$

Sind für eine Anfrage genau 2 Dokumente vorhanden und haben sie in der Ergebnisliste die Positionen 1 und 2, so ergibt sich folgendes Ergebnis:

$$Avr.Prec. = \frac{p_1 + p_2}{2} = \frac{\frac{1}{1} + \frac{2}{2}}{2} = 1$$

Im Folgenden wird das vorige Beispiel aufgegriffen und nach der neuen Methode berechneten Verläufe als Diagramme vorgestellt (Abb. 5.7. Wie bereits im Beispiel davor ist im Falle der zwei Dokumente das erste Dokument auf Position 1 gelegt.

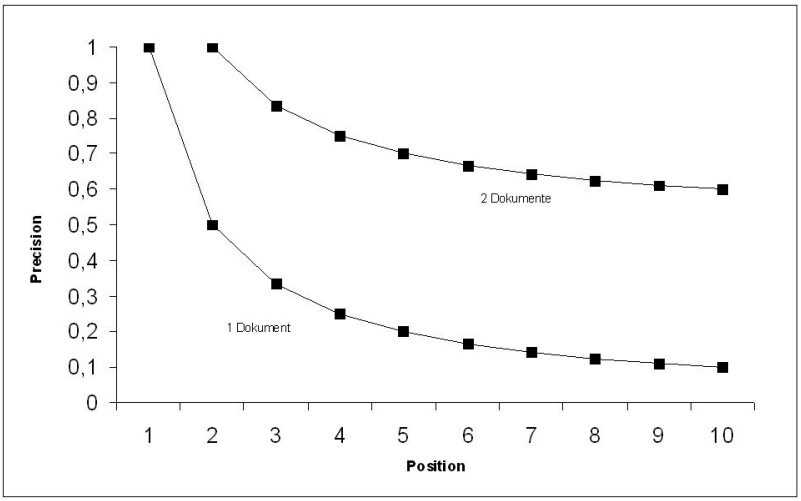

Abbildung 5.7: Modifizierte Average Precision für ein und zwei Dokumente

Beim genauen Hinschauen erkennt man, dass die modifizierte Formel der Formel für die Berechnung des Recalls ähnelt. Dort werden nämlich alle Treffer durch die Anzahl der relevanten Dokumente geteilt:

$$Recall = \frac{|T|}{|R|}$$

Dabei ist R die Menge der relevanten Dokumente und T die Treffermenge. Die neue Formel kombiniert die *Precision at k* und *Recall* und berücksichtigt somit alle drei relevanten Aspekte für die Bewertung der Ergebnisse: die Precision, den Recall und das Ranking. Das neue Maß heißt *Average Precision at R (APr)* und wird formell ausgedrückt mit:

$$APr = \frac{1}{|R|} \sum_{i=1}^{|R|} \frac{pos(d_i))}{i}$$

Dabei sollen die Dokumente d_i nach ihrem Ranking sortiert sein. Der Wert $pos(d_i)$ drückt die Position eines relevanten Dokuments in der Ergebnisliste aus.
Im Folgenden werden einige Beispiele aufgeführt, die die Vorteile der Average Precision at R erläutern. Es seien dabei D die Menge aller Dokumente, R die relevanten Dokumente, E die Menge der Ergebnisdokumente und T die Menge der Treffer mit $E \cap R$:

Beispiel 1: Die Ergebnismenge E enthält alle Dokumente und damit auch alle relevanten Dokumente. Die Standardformel für den Recall liefert den Wert 1. Für APr ergibt sich ein Wert von 1 nur dann, wenn die relevanten Dokumente an den ersten Positionen auftauchen: $APr = \frac{1}{|R|}(1 + \frac{2}{2} + ... + \frac{|R|}{|R|}) = 1$.

Beispiel 2: Die Ergebnismenge E enthält alle Dokumente und die relevanten Dokumente sind auf den letzten Rängen positioniert: $APr = \frac{1}{|R|}(\frac{1}{|E|-|R|} + \frac{2}{|E|-|R|+1} + ... + \frac{|R|}{|E|})$
Bei einem Korpus von 100 und 10 relevanten Dokumenten liefert die Formel dann den Wert: $APr = \frac{1}{10}(\frac{1}{91} + \frac{2}{92} + ... + \frac{10}{100}) = \frac{1}{10} \cdot 0,57 = 0,057$ Ein Korpus von 10000 Dokumenten dagegen ergibt einen Wert von: $APr = \frac{1}{10}(\frac{1}{91} + \frac{2}{92} + ... + \frac{10}{100}) = \frac{1}{10000} \cdot 0,57 = 0,00057$
Je größer die Dokumentenmenge im Vergleich zur Menge der relevanten Dokumente, desto kleiner ist der APr-Wert, wenn alle Ergebnisse auf den hinteren Rängen verteilt sind.

Beispiel 3: Die Ergebnismenge enthält keine relevanten Dokumente:
$APr = \frac{1}{|R|} \cdot 0 = 0$

Die Ermittlung der Mean Average Precision wird durch die Einführung des neuen Maßes nicht beeinträchtigt, die Modifikation tauscht lediglich einen Parameter aus. Das Ablaufschema ist auf der Abbildung 5.8 dargestellt.

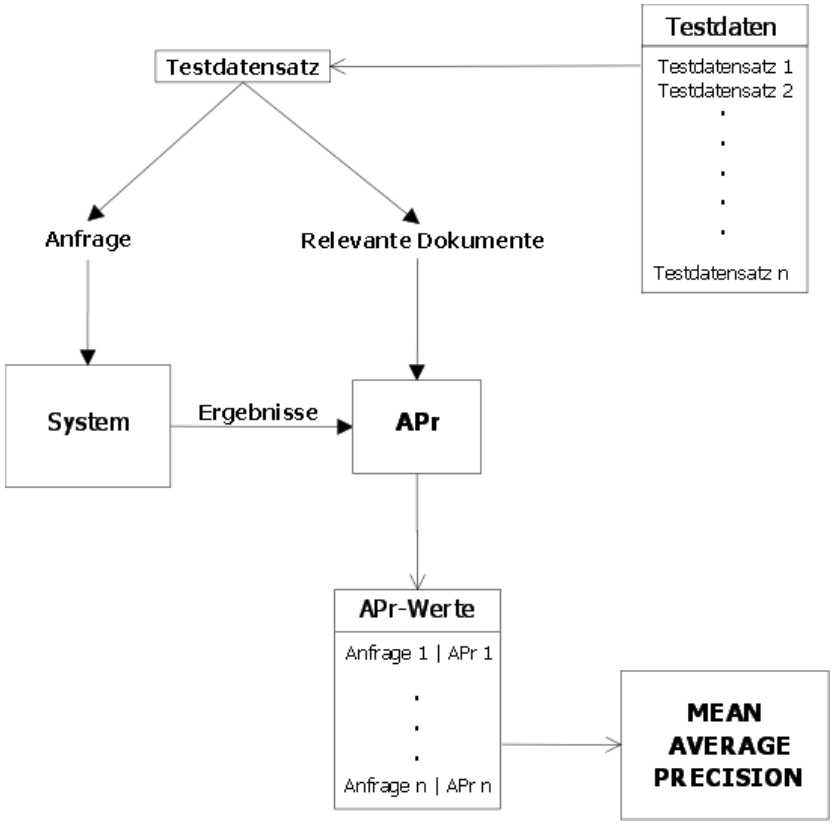

Abbildung 5.8: Ermittlung der MAP

6 Implementierung eines Prototyps

6.1 Überblick

Die Implementierung eines Prototyps soll zeigen, dass sich das Konzept umsetzten lässt.

6.2 Architektur

Die Architektur des Prototyps ist relativ einfach gestaltet und besteht aus drei Schichten, wobei es sich hier nicht um eine klassische Model-View-Controller-Architektur[1] handelt. Die Menge der Dokumente kann zwar als Modell aufgefasst werden und die GUI als View, doch in dem vorliegenden Fall wird das Modell kaum verändert, da der Index nur gelesen wird.
Die grafische Oberfläche bildet die erste Schicht, die mittlere Schicht besteht aus Hilfsklassen, die die in der GUI angebotenen Funktionen direkt umsetzen. Die dritte Schicht ist hauptsächlich aus den Lucene-Klassen zusammengesetzt, die direkt auf dem Index arbeiten.
Das Klassendiagramm ist in der Abbildung 6.1 dargestellt.

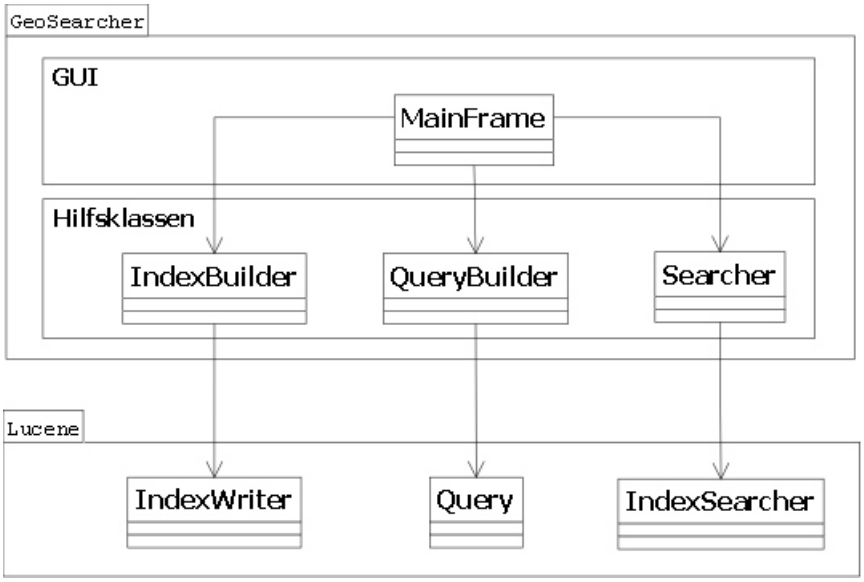

Abbildung 6.1: Klassendiagramm

[1] http://de.wikipedia.org/wiki/Model_View_Controller

6.3 Daten

Der Datenbestand besteht aus über 58.000 Dokumenten, wobei jedes Dokument genau ein Suchobjekt (Station, Adresse, POI) repräsentiert. Alle Suchobjekte sind in einer CSV-Datei untergebracht, die Zeilen stellen die Suchobjekte dar, die Zeilen die Felder.
In der Tabelle 6.1 werden die einzelnen Spalten der CSV näher erläutert:

Spaltename	Inhalt	Beispiel	Im Index
type	Typ des Suchobjektes	STATION	ja
town	Stadt- oder Dorfname	Winsen	ja
town1	Präfixe von town	Winse, Wins	ja
town2	Präfixe von town	Win, Wi	ja
place	Name	Ilmer Weg	ja
place1	Präfixe von place	Ilme We	ja
place2	Präfixe von place	Ilm W	ja
place3	Präfixe von place	Il	ja
place4	Präfixe von place	I	ja
occurrence	Normierte Häufigkeit	2	nein
key	Schlüssel	102425	nein
combiname	Ausgabetext	Winsen, Ilmer Weg	nein

Tabelle 6.1: Datenstruktur der Suchobjekte

Die Werte der **normierten Häufigkeit** werden nicht gesucht und dienen ausschließlich der Gewichtung der Dokumente. Die Gewichtung wird jedoch ebenfalls im Index gespeichert, somit werden sie indirekt übernommen.
Der Inhalt der Spalten *key* und *combiname* wird lediglich für die Verarbeitung der Ergebnisse verwendet und erscheint ebenfalls nicht im Index.
Die **Verteilung der Präfixe** auf die entsprechenden Spalten ist linear. Das bedeutet, dass der Abstand zum Originalterm bis auf einige wenige Ausnahmen gleichmäßig abnimmt. Eine Ausnahme tritt ein, wenn der Term weniger Buchstaben hat als die Anzahl der vorgesehene Spalten. Dieser Fall herrscht vor, wenn beispielsweise das Wort „Weg“ auf fünf Spalten verteilt wird. Die beiden letzten Präfixspalten bleiben somit leer.

6.4 Indexerstellung

Im folgenden Abschnitt wird die Erstellung des Index besprochen. Zunächst werden die betroffenen GUI-Elemente vorgestellt. (Abb. 6.2).
Zu Anfang liegen alle Daten im csv-Format vor, sind also tabellarisch strukturiert. In der Textbox *field to index* werden alle Spalten definiert, die in den Index aufgenommen werden sollen. Alle dort nicht definierten Spalten werden bei der Suche nicht berücksichtigt, können jedoch ausgelesen werden (siehe Abschnitt 6.3).

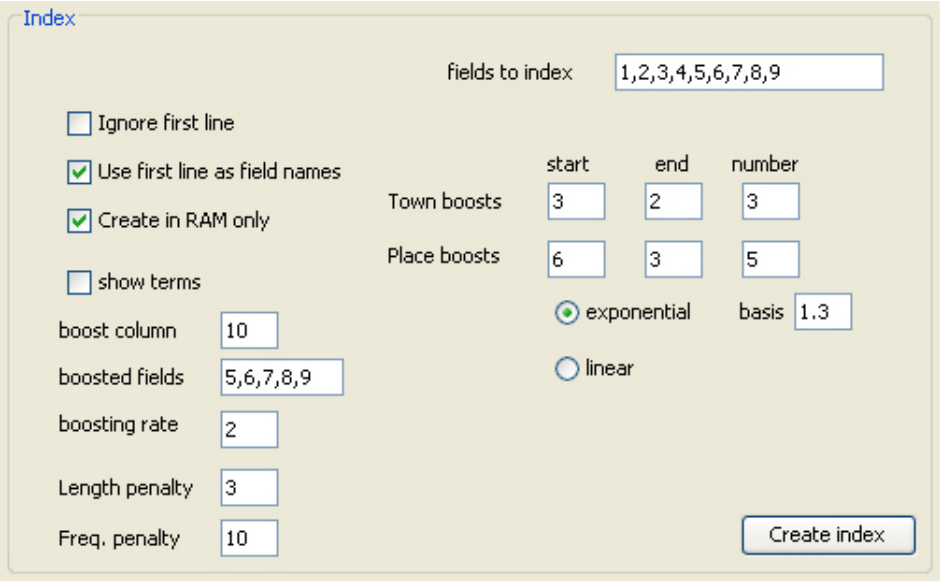

Abbildung 6.2: GUI-Indexerstellung

Das Setzen der Checkbox *Use first line as field names* bewirkt, dass die Einträge in der ersten Zeile des CSV-Datei als Feldnamen gedeutet werden. Die Option ist standardmäßig auf TRUE gesetzt.
Die Option *Create in RAM only* wird verwendet, wenn der Index im RAM erzeugt werden soll und nicht im Dateisystem. Der Zugriff auf den Index ist dann wesentlich schneller.
Die Angabe der *boost column* legt die Spalte fest, die die normierten Häufigkeiten enthält, die zu gewichtenden Felder werden direkt darunter in dem Feld *boosted fields* bestimmt.
Der Parameter *boosting rate* ist der Skalierfaktor für die häufigkeitsbezogenen Gewichtungen.
Der Parameter *length penalty* wirkt sich auf die Gewichtung aus, die anhand der Dokumentenlänge bestimmt wird. Der Wert des *idf (inverse document frequency)* wird dagegen von der *freq. penalty* skaliert.
Die Gewichtung der Namen und der Orte sowie ihrer Präfixe erfolgt über die Textboxen *Town boosts* und *Place boosts*. Die exponentielle Verteilungsfunktion der Felder wird aus dem Startwert, dem Endwert, der Anzahl der Stützstellen und der Basis ermittelt. Für die lineare Verteilung wird keine Basis benötigt.
Die Erstellung des Index ist in Abb.6.3 als Aktivitätsdiagramm dargestellt. Einige Aktionen werden im Folgenden näher erläutert:

- **Dokument erstellen, gewichten und Felder hinzufügen** - jedes Dokument wird in Lucene mit der Klasse *Document* repräsentiert. Einer Instanz der Klasse *Document* können beliebig viele Felder hinzugefügt werden, des Weiteren kann eine *Document*-Instanz gewichtet werden. Die Werte für die Gewichtung werden der Spalte *occurrence* entnommen.

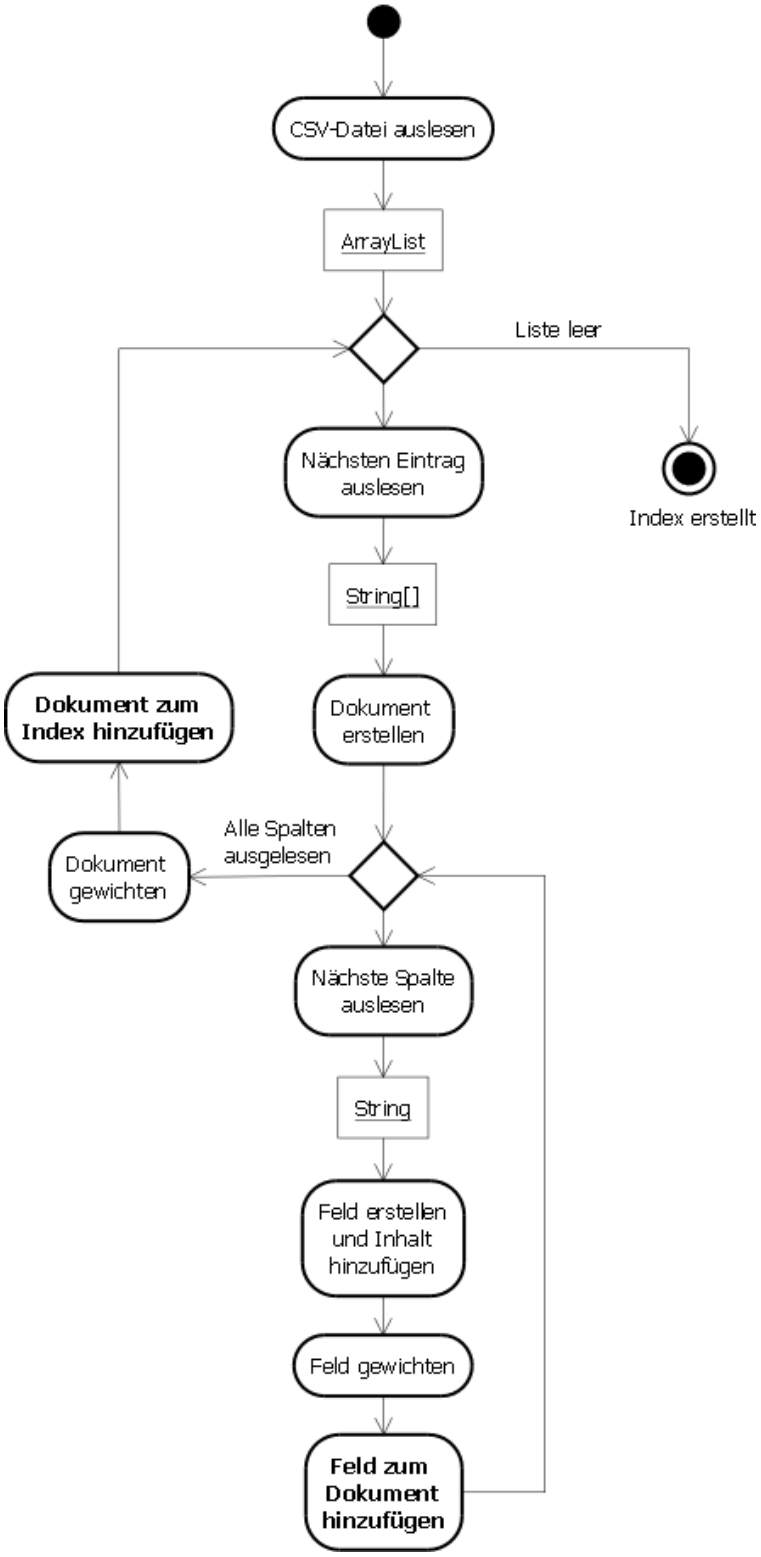

Abbildung 6.3: Aktivitätsdiagramm: Erstellung des Index

- **Feld erstellen und Inhalt hinzufügen** - die Felder der Dokumente werden durch die Klasse *Field* repräsentiert. Ähnlich der *Document*-Klasse können die *Field*-Instanzen gewichtet werden. Zusätzlich wird dem Feld der Inhalt zugewiesen, dieser kann aus mehreren Strings bestehen, die Zerlegung in Tokens wird automatisch vorgenommen. An dieser Stelle wird auch entschieden, ob der Inhalt indiziert werden soll. Nachdem ein *Feld*-Objekt erzeugt, konfiguriert und mit Inhalt gefüllt wurde, wird es dem Dokument hinzugefügt.
- **Dokument zum Index hinzufügen** - das konfigurierte und mit Feldern gefüllte *Document*-Objekt wird dem *IndexWriter*-Objekt übergeben und von ihm in das *Directory* (Index) eingefügt. Das *IndexWriter*-Objekt wird ebenfalls konfiguriert, beispielsweise um die Termnormalisierung zu parametrisieren.

Termnormalisierung
Die deutschen Umlaute „ä“,„ü“,„ö“ sowie „ß“ werden durch „ae“, „ue“, „oe“ und „ss“ ersetzt, die Großbuchstaben kleingeschrieben. Die Sonderzeichen werden durch das Leerzeichen ersetzt, die einzige Ausnahme bildet das Zeichen „’“, welches nur gelöscht wird. Das Compound-Splitting findet bei er Erstellung des Index nicht statt. Stattdessen wird es bei der Formulierung der Anfrage gebraucht und zur Erstellung von Alternativen verwendet.

6.5 Formulierung der Anfrage

Die Verarbeitung der vom Benutzer gestellten Anfrage und Erstellung eines wohldefinierten Ausdruckes war eine der zentralen Aufgaben bei der Implementierung des Prototyps.
Auf dem Aktivitätsdiagramm in Abb. 6.4 wird der Hauptalgorithmus vorgestellt.

In den folgenden Abschnitt werden die formulierten Unteranfragen und Anfrageerweiterungen erläutert. Hierfür wird eine an Lucene angelehnte Syntax verwendet:

- **AND** - logisches UND.
- **OR** - logisches ODER.
- **type: station** - der Term *station* wird nur im Feld *type* gesucht.
- **town: luebeck** - der Term *luebeck* wird in allen *town*-Feldern gesucht, d.h. in town, town1, town2.
- **place: holstenstrasse** - der Term *holstenstrasse* wird in allen *place*-Feldern gesucht, d.h. in place, place1, place2, place3 und place4.
- **luebeck** - der Term wird in allen *town*- und *place*-Feldern gesucht, d.h. in town, town1, town2, place, place1, place2, place3 und place4.

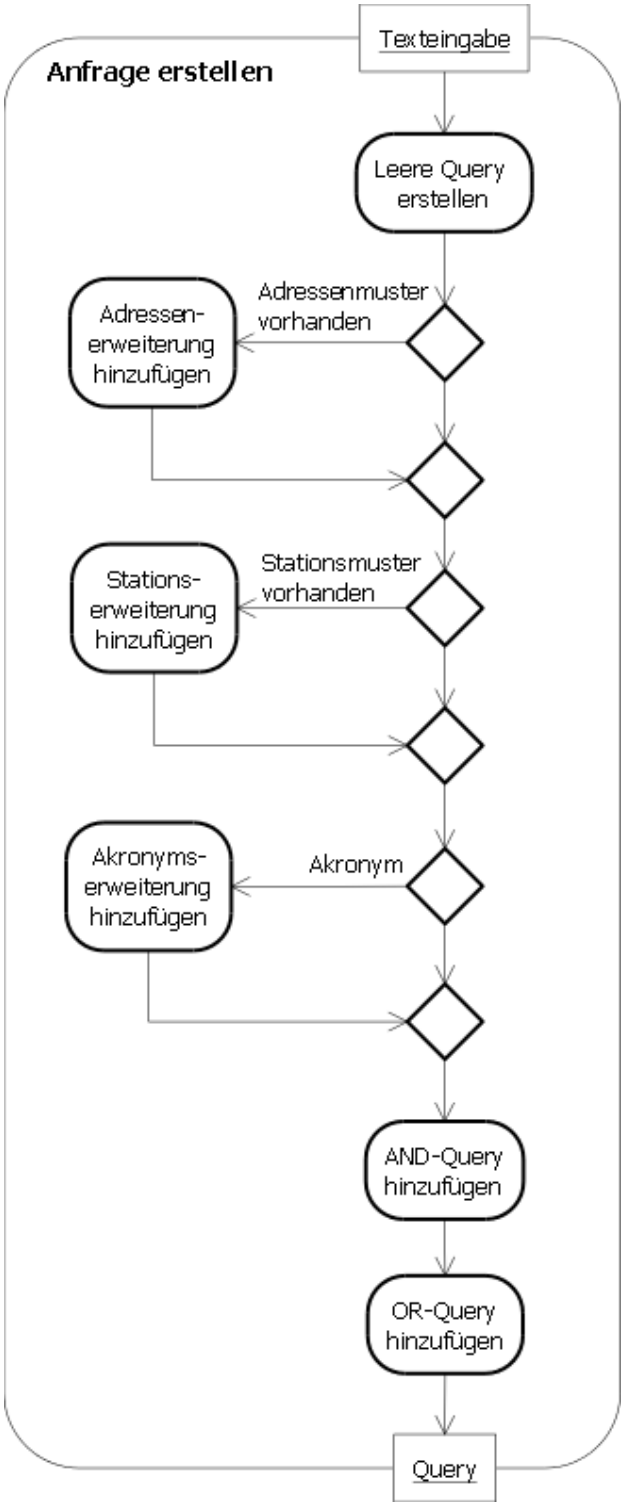

Abbildung 6.4: Aktivitätsdiagramm: Erstellung einer Anfrage

6.5.1 Termsuche

Die Anfrage nach einzelnen Termen bildet die unterste Stufe einer Anfrage. Zu Anfang dieses Kapitels wurde bereits angedeutet, dass zu jeder Termanfrage das Feld spezifiziert werden muss. Weil in den meisten Fällen die Bedeutung der Terme unbekannt ist, wird jeder Term in allen Feldern gesucht, wie das Beispiel in Abbildung 6.5 zeigt. An dieser

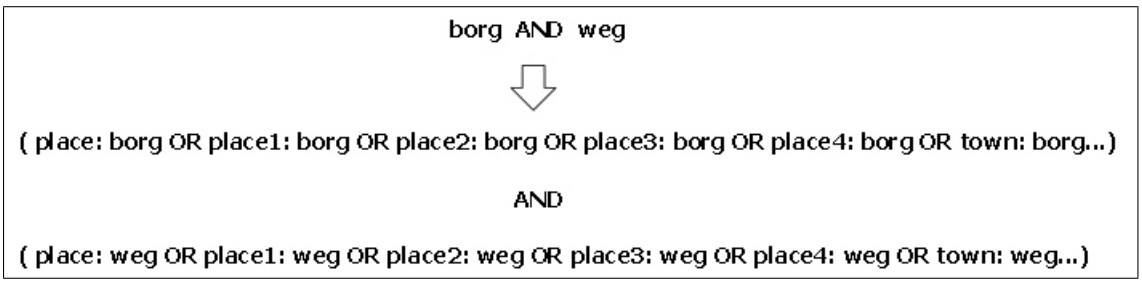

Abbildung 6.5: Terme werden in mehreren Feldern gesucht

Stelle ist erneut der Vorteil der Präfixfelder zu erkennen: bei Präfixen mit einem größeren Abstand zum Originalterm fällt die feldbasierte Gewichtung geringer aus.
Der Effekt der Suche in Präfixfeldern wird in der Abb.6.6 gezeigt.

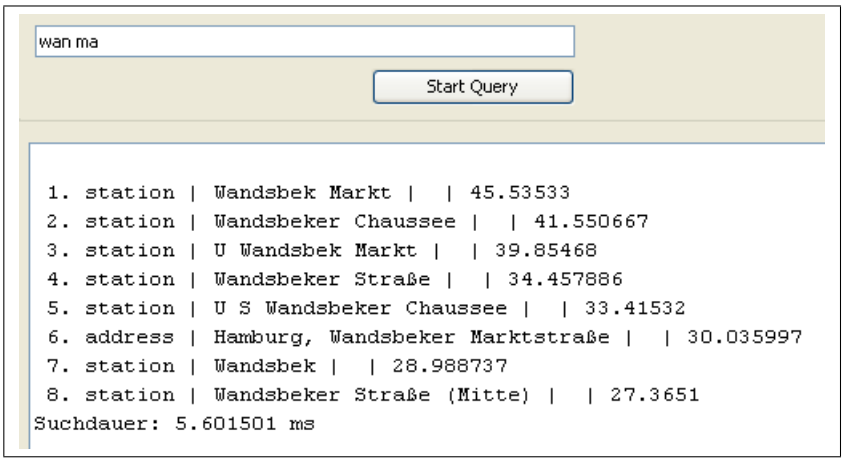

Abbildung 6.6: Über den Präfix gesuchte Objekte werden gefunden

6.5.2 AND- und OR-Query

Jede Eingabe wird in einem Query-Objekt gekapselt. Dieser enthält den logischen Ausdruck, der gemäß der Regeln des booleschen IR-Modells formuliert ist. Im einfachsten Fall enthält der Eingabestring keine Muster, so dass die formulierte Anfrage aus nur zwei Teilanfragen besteht, nämlich der AND- und OR-Query.
Der Aufbauprozess der AND-Query wird im Folgenden erläutert, die OR-Query wird analog dazu aufgebaut. Hierzu werde zunächst das Aktivitätsdiagramm für die Erstellung einer AND-Query betrachtet (Abb. 6.7).
Eine AND-Query ist dadurch gekennzeichnet, dass alle Tokens mit einem logischen

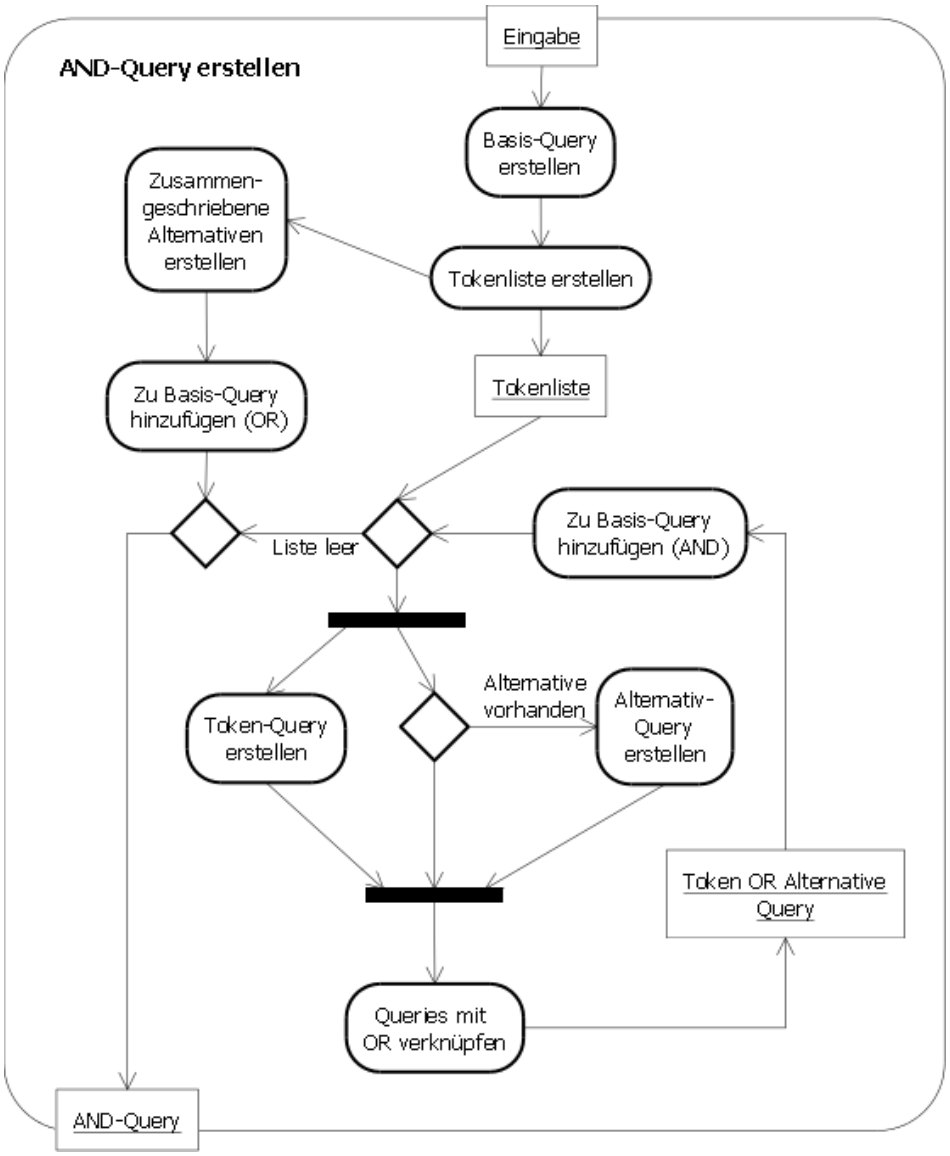

Abbildung 6.7: Aktivitätsdiagramm: Erstellung der AND-Query

UND miteinander verknüpft werden. Damit wird die Anforderung erfüllt, dass alle Suchwörter im Ergebnis vorkommen sollen (starke Bindung). Es kommt durchaus vor, dass eins der Terme unauffindbar ist, z.B. wenn das Wort falsch geschrieben ist und die Levenshtein-Distanz zu groß ist. Aus diesem Grunde wird zu der AND-Query zusätzlich eine OR-Query erstellt. Diese beiden Anfragen bilden somit das Grundgerüst einer jeden Anfrage (siehe Beispiel in Abb.6.8).

In der OR-Query braucht jeweils nur ein Term in irgendeinem Dokument vorkommen, die OR-Query ist damit „weicher“ als die AND-Query.

Am in der Abb.6.9 gezeigten Beispiel ist der Vorteil der OR-Query zu erkennen.

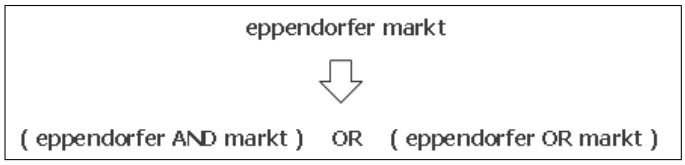

Abbildung 6.8: Grundgerüst einer jeden Anfrage

Abbildung 6.9: Die OR-Query sorgt dafür, dass ein Ergebnis erzeugt wird, ohne dass alle Terme im Dokument vorhanden sein müssen

6.5.3 Muster

Die Benutzereingaben enthalten Zeichen und Symbole, die auf einen Objekttypen hinweisen. In diesem Zusammenhang wird von *Mustern* gesprochen. Die Erkennung und Behandlung der Muster wird im folgenden Abschnitt besprochen.

Adressmuster

Sind in der Eingabe **Muster** vorhanden, wird eine Anfrageerweiterung erstellt und zu dem Grundgerüst hinzugefügt. Ein **Adressenmuster** ist dadurch gekennzeichnet, dass die Eingabe eine Zahl enthält, die jedoch nicht der erste Term ist. Alle Terme vor der Zahl werden dann als Name (place) und die danach als Stadt (town) interpretiert. Da es sich um eine Vermutung handelt wird die Originaleingabe nicht ersetzt, sondern erweitert.
Beispiel:
Der Eingabestring enthält eine Zahl und deutet auf eine Adresse hin. Es könnte jedoch auch eine Station sein oder ein POI wie z.B. *An der B 75* oder *Cafebar Nr. 1*. Die Abb. 6.10 zeigt die vorgenommene Erweiterung.

Die Funktionsweise der Gewichtung des Suchobjekttyps *ADDRESS* wird an dem Beispiel auf Abb. 6.11 erläutert. Der Benutzer sucht zwar nach einer Adresse, bekommt je-

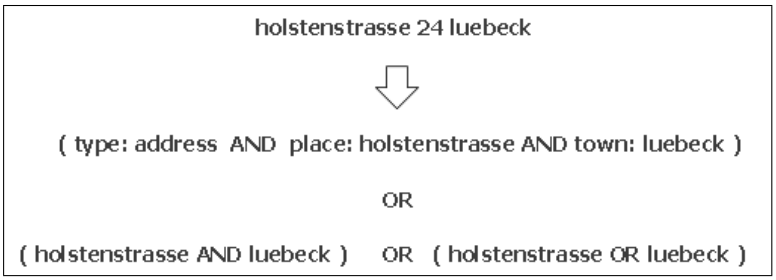

Abbildung 6.10: Anfrageerweiterung einer Eingabe mit vermuteter Adresse

doch auch andere Suchobjektypen mit den selben bzw. ähnlichen Termen repräsentiert.

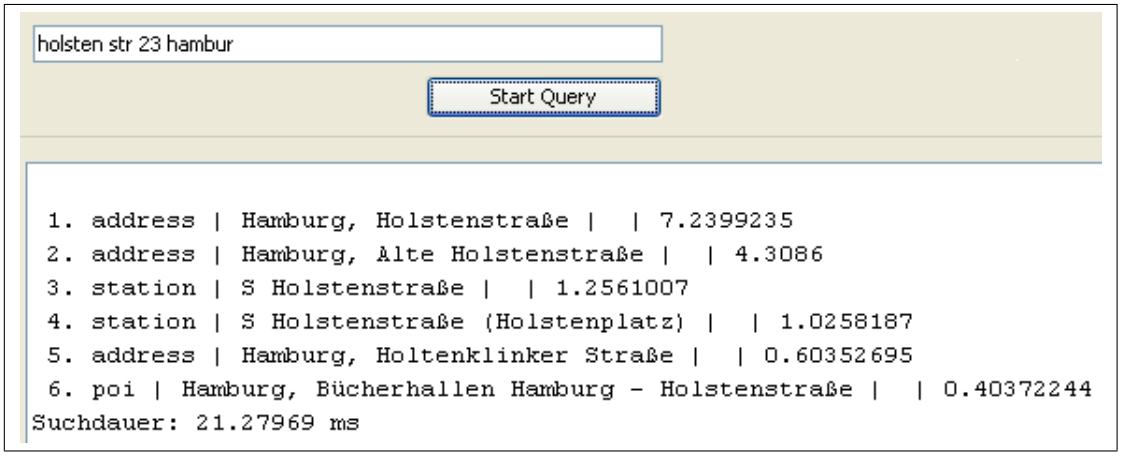

Abbildung 6.11: Beispiel für eine Anfrage mit einem Adressmuster

Stationsmuster

Ein weiteres Muster ist das **Stationsmuster**. Die Erkennung dieses Musters funktioniert analog zu dem Adressenmuster, die Terme werden jedoch in diesem Falle nach Schlüsselwörtern wie *U, S, Bahnhof* oder *Bf.* durchsucht. Die Erweiterung ist in der Abb. dargestellt.

Die Behandlung des Stationsmusters erfüllt eine weitere, wichtigere Aufgabe als die bloße Gewichtung des Objekttypen. Bei der Erstellung der Anfrageerweiterung werden nämlich die Schlüsselwörter korrigiert. Das hat den Vorteil, dass auch kurze, inkorrekt geschriebene Wörter, wie z.B. *bhn* stat *bahn* erkannt werden. Ein geringer Unterschied bei kürzeren Wörtern hat nämlich eine gravierendere Auswirkung auf das Ergebnis der Distanzfunktion von Lucene als bei Längeren. Kommt beispielsweise in der Eingabe der Term *sbhnf* vor, wird dieser durch *s bahnhof* ersetzt.

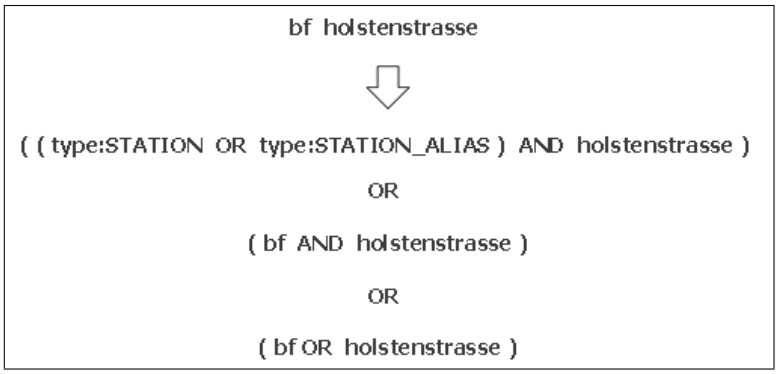

Abbildung 6.12: Erweiterung einer Eingabe mit vermuteter Station

6.5.4 Kürzel

Ein weiterer Aspekt bei der Anfrageformulierung ist die Behandlung von Kürzeln. Wie bereits erwähnt steht ein Kürzel immer für genau ein Objekt, beispielsweise *hbf* für Hauptbahnhof. Ein Kürzel besteht meistens aus weniger als drei Buchstaben und liefert immer, vorausgesetzt er ist richtig geschrieben, das richtige Dokument.
Im Grunde genommen ist das Kürzel einem Synonym in der Hinsicht ähnlich, dass es ein Suchobjekt beschreibt, das auch von anderen Wörtern repräsentiert wird. So können beispielsweise die Bezeichnungen *hbf*, *Hauptbahnhof* und *Main Station* als Synonyme betrachtet werden. Ein weiteres Beispiel wäre das Kürzel *flh*, welches die Objekte *Flughafen*, *Airport* und womöglich noch andere Orte am oder in der Nähe des Flughafens beschreibt. Es ist also sinnvoll auch die Dokumente zu suchen, die dem vom Kürzel repräsentierten Dokument ähnlich sind. Für diesen Zweck wird beim Erstellen des Index eine Liste mit allen Kürzeln und den dazugehörenden Beschreibungen (combiName) angelegt. Sollte in der Eingabe ein Kürzel vorhanden sein, werden zu der Anfrage die Terme der Beschreibung hinzugefügt.
Auf der Abb.6.13 sind Ergebnisse einer Beispielanfrage mit einem Kürzel dargestellt. Das eingegebene Kürzel lautete „flh“ und repräsentierte nur dan der ersten Position erschienene Dokument. Die restlichen Ergebnisdokumente lieferte die Anfrageerweiterung, welche die Termen „Hamburg“, „Airport“ und „Flughafen“ ODER-verknüpft enthielt. Das Schema der Anfrageerweiterung ist auf der Abb.6.14 zu sehen.

```
1. Hamburg Airport (Flughafen) |  | 10.634647
2. Hamburg Airport |  | 0.6783456
3. Hamburg, Hamburg Airport |  | 0.50611156
4. Flughafenstraße |  | 0.2263274
5. Hamburg, Flughafen Hamburg-Fuhlsbüttel |  | 0.20178363
```

Abbildung 6.13: Ergebnisliste der Anfrage „flh“

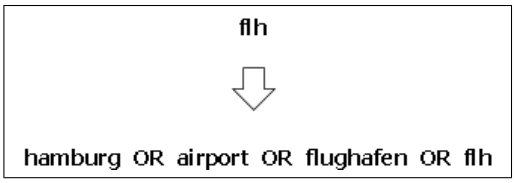

Abbildung 6.14: Erweiterung einer Eingabe mit einem Kürzel

6.5.5 Alternativen

Das Problem der fälschlicherweise zusammen geschriebenen Namen, wie z.B. *Berlinertor* statt *Berliner Tor* wird dadurch gelöst, dass eine *Alternative* erzeugt wird, die durch Aufsplittung des Termes entsteht (*compound splitting*).
Analog dazu werden wird in bestimmten Fällen zwei oder mehrere Terme zusammengefasst.
Die Beispiele auf den Abbildungen 6.15 , 6.16 und 6.17 geben mit Hilfe konkreter Beispiele einen Überblick über die möglichen Strukturen einer AND-Query, die Alternativterme enthält.

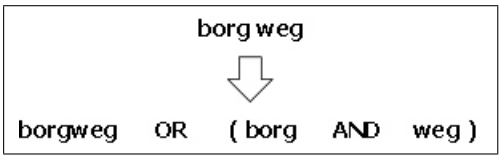

Abbildung 6.15: Beispiel für eine Anfrage - die AND-Query enthält auch den aus den einzelnen Termen zusammengesetzten Term

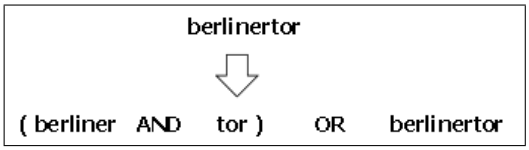

Abbildung 6.16: Beispiel für eine Anfrage - die AND-Query enthält als Alternative die getrennte Schreibweise des Eingabeterms

Das letzte Beispiel zeigt einen Nebeneffekt der Strategie. Obwohl die zusammengesetzte Schreibweise inkorrekt ist, wird trotzdem danach gesucht. Die Qualität der Ergebnisse wird dadurch zwar nicht beeinflusst, eine zusätzliche Anfrage verringert jedoch die Performance. Eine andere Lösung wäre es, das Compound-Splitting auf alle Terme bei der Indexerstellung anzuwenden. Dann bräuchte man nur noch alle Anfrageterme zu splitten und hätte das gleiche Ergebnis.
Dennoch bringt die verwendete Strategie einen erheblichen Vorteil: sie ist Tipp- und Rechtschreibfehlern toleranter. Ein Compound-Splitter enthält nämlich ein Wörterbuch und benutzt es um festzustellen ob ein Teilwort drin vorhanden es. Dieses Problem wird im Folgenden erklärt:

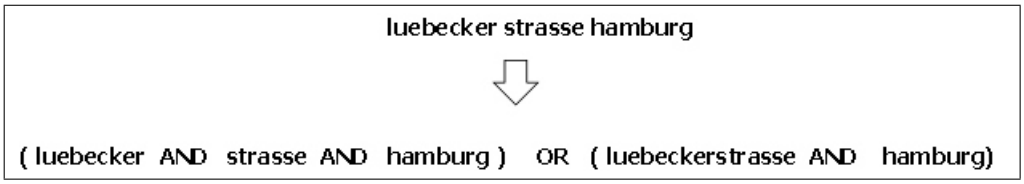

Abbildung 6.17: Beispiel für eine Anfrage - die AND-Query enthält auch inkorrekt geschriebene Alternativen.

Es werde jeder Term während des Indizierens gesplittet. Aus *holstenstrasse* werden dann zwei Terme, *holsten* und *strasse*, der Compound-Splitter hat nämlich den Term „strasse“ in seiner Liste. Wenn der Benutzer die Eingabe *Holstenstraße* tätigt, ist alles in Ordnung, sollte er jedoch stattdessen *Holstenstaße* eingeben, bekommt er kein Ergebnis. Das Wörterbuch des Compound-Splitters müsste demnach auch alle falsch bzw. abgekürzt geschriebenen Alternativen enthalten.
Die Toleranz der vorliegenden Lösung kann mit dem Vektorraum Modell geometrisch erklärt werden. Dort unterscheidet sich das Dokument mit dem Term *holstenstrasse* von dem Dokument mit den Termen *holsten* und *strasse* erheblich, für den Menschen jedoch nicht. Es ist also sinnvoll alternative Anfragevektoren mit syntaktisch ähnlichen Termen zu erzeugen und den abgefragten Raum zu vergrößern.
Die Implementierung der Compound-Splitting-Funktion basiert auf dem Lösungsansatz von De Rijke/Monz, bei dem alle Teilwörter gebildet und im Wörterbuch nachgeschlagen werden.
Auf der Abb.6.18 und Abb.6.19 wird anhand von zwei Beispielsanfragen der Effekt der Alternativanfragen aufgezeigt.

6.5.6 Tipp- und Rechtschreibfehler

Dem Problem der inkorrekten Terme wird begegnet, in dem die Terme mit Hilfe der FuzzyQuery-Klasse der Lucene-Bibliothek gesucht werden. Damit werden auch Terme akzeptiert, sich bis zu einem Grad vom Originalterm unterscheiden. Das Grad der Ähnlichkeit (*distance*) wird mit der Formel $dist = 1 - \frac{Levenshteindistanz}{min(Term1,Term2)}$ bestimmt. Beim Erzeugen einer Instanz der Klasse FuzzyQuery werden die minimale Distanz und die Nicht-Fuzzy-Präfixlänge angegeben. Alle Terme, die einen höheren Wert als die minimale Distanz aufweisen, werden als Treffer gewertet. Die minimale Nicht-Fuzzy-Präfixlänge definiert den ersten Teil des Terms, der bei der Berechnung der Levenshteindistanz nicht berücksichtigt wird.
Die bereits gezeigten Beispiele mit Anfragen verdeutlichen den positiven Effekt der *FuzzyQuery*.

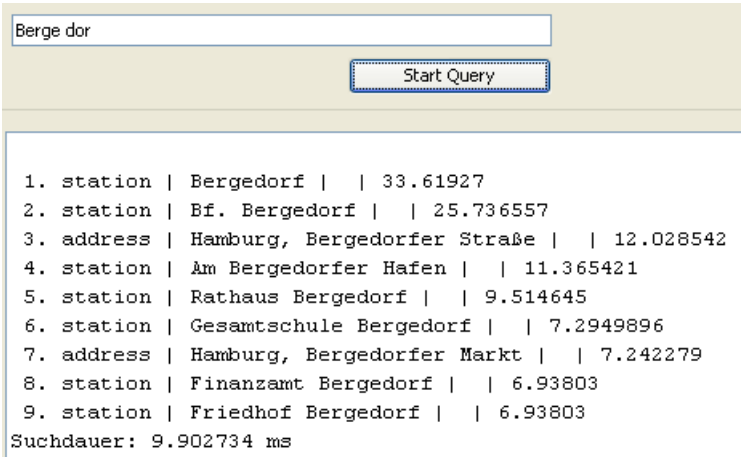

Abbildung 6.18: Fälschlicherweise getrennt geschriebene Terme werden akzeptiert

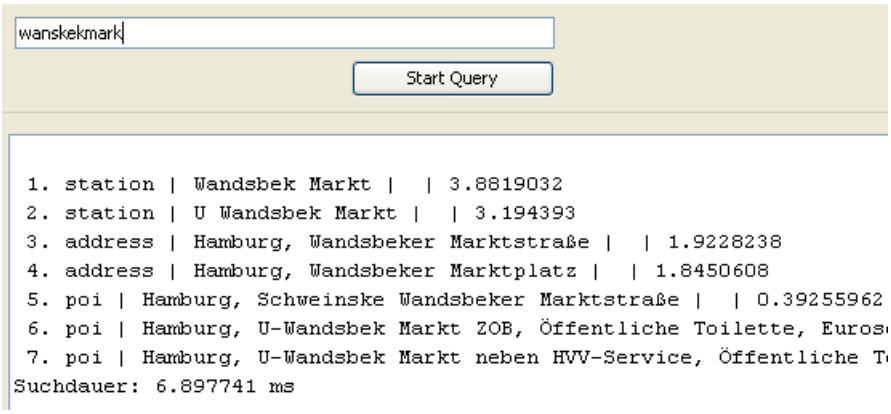

Abbildung 6.19: Fälschlicherweise zusammengeschriebene Terme werden akzeptiert

6.5.7 Anfragegewichtungen

Alle vorgestellten Anfrageerweiterungen sowie die AND- und OR-Query werden mit unterschiedlichen Werten gewichtet. Diese Entscheidung ist vor allem bei der Betrachtung der AND- und OR-Query verständlich. Die UND-Verknüpfung ist „strenger“ als ODER und jeder so erreichte Treffer „wertvoller“ als jeder einzelne Treffer einer ODER-Anfrage. Somit muss die AND-Query höher gewichtet werden.
Die aus einem Muster abgeleiteten Anfragen müssen ebenfalls gesondert gewichtet werden. Es soll im Folgenden ein Beispiel für eine Eingabe einer Adresse betrachtet werden, zu der es auch eine gleichnamige Station gibt, beispielsweise *Borgweg 24*. Da die Stationen eine viel höhere Anfragehäufigkeit aufweisen, kann die Addressenerweiterung von der Originalanfrage überstimmt werden. In dem angeführten Beispiel ist der unterschied sehr hoch, die S-Bahn Haltestelle *U-Bahn Borgweg* hat eine Gewichtung von 18, die Adresse *Borgweg* nur 1.

6.6 Similarity

In der vorliegenden Implementierung werden die für die Berechnung der Ähnlichkeit verwendeten Funktionen an die konzeptuellen Anforderungen angepasst. Diese geben vor, dass sowohl die Längennorm (*LengthNorm*) als auch die inverse Dokumentenfrequenz (*inverse document frequency - idf*) skalierbar sein sollen. Des Weiteren soll die Termfrequenz auf einen konstanten Wert von eins gesetzt werden. Aus diesen Gründen kann die *DefaultSimilarity*-Klasse der Lucene-Bibliothek nicht verwendet werden.
Stattdessen wird eine neue Klasse implementiert, die von der DefaultSimilarity-Klasse abgeleitet ist und die entsprechenden Methoden überschreibt:

idf(int docFreq, int numDocs) - die standardmäßig implementierte Formel lautet:

$$idf = log\left(\frac{numDocs}{docFreq+1}\right)+1$$

wobei:
numDocs - Anzahl aller Dokumente
docFreq - Anzahl der Dokumente, die den Term enthalten
Nach der Einführung des Parameters *Frequency Penalty* ergibt sich folgende Formel:

$$idf = \left(log\left(\frac{numDocs}{docFreq+1}\right)+1\right) * (1 - 0.05 \cdot freqPenalty)$$

Der Parameter *freqPenalty* verringert somit den *idf*-Wert multiplikativ mit der Schrittlänge von 0.05. Ab einem Wert von 20 wird der Ausdruck in der letzten Klammer auf 0.05 gesetzt, das bedeutet der *idf*-Wert kann niemals kleiner werden als ein zwanzigstel des ursprünglichen Wertes.

computeNorm(String field, FieldInvertState state) - die von Lucene verwendete Formel lautet:

$$lengthNorm = boost * \frac{1}{\sqrt[2]{termNum}}$$

wobei:
boost - Gesamtgewichtung aus den Feld- und Dokumentengewichtungen
termNum - Anzahl der Terme im Dokument
Die *lengthPenalty* wird analog zu *freqPenalty* zu der Originalformel hinzugefügt:

$$lengthNorm = boost * \frac{1}{\sqrt[2]{termNum}} * (1 - 0.05 \cdot lengthPenalty)$$

tf(float freq) - der Rückgabewert dieser Methode wird einfach auf den Wert Eins gesetzt. Somit spielt die Termfrequenz für die Gewichtung keine Rolle mehr.

6.7 Evaluierung

Wie bereits erwähnt wird bei der Evaluierung die MAP-Methode angewendet. Hierfür wird die Klasse *Evaluator* erstellt, deren Instanz die Benutzerdaten ausliest und Anfragen an das Suchsystem stellt. Die Ergebnisdokumente werden dann mit den vom Benutzer gewählten Dokumenten (relevante Dokumente) verglichen und die Average Precision berechnet. Auf der Abbildung 6.20 ist der Evaluierungsprozess in der Form eines Aktivitätsdiagramms dargestellt.

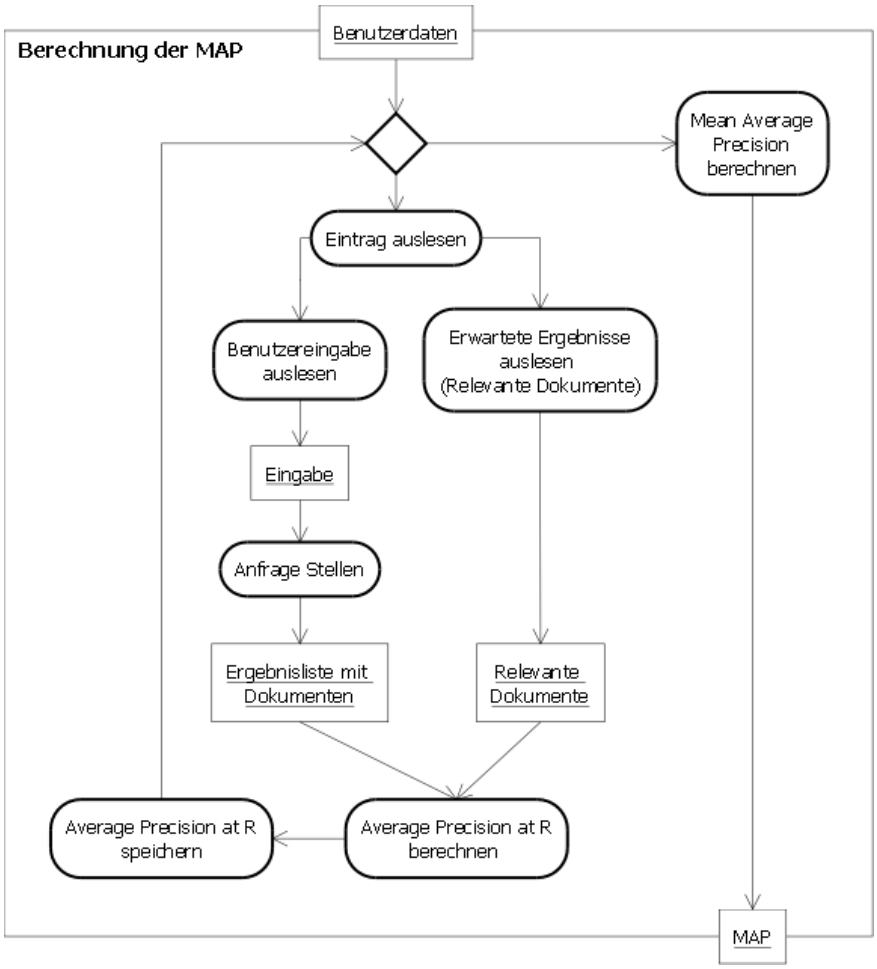

Abbildung 6.20: Berechnung der Mean Average Precision

Das neue System orientiert sich an Nutzern mobiler Geräte, deswegen werden Evaluierung des Systems Testdaten verwendet, die aus mobilen Endgeräten stammen. Die Qualität der Ergebnisse lässt sich mit dem bestehenden System nicht objektiv vergleichen, weil die Nutzerentscheidungen von den Gegebenheiten des alten Systems beeinflusst wurden. Das bedeutet im Klaren, dass bei der Evaluierung einige Vorteile dem neuen System gegenüber entstehen.
Die Gründe werden im Folgenden aufgeführt:

- Im bestehenden System werden die Suchobjekte über zwei Textfelder und eine Auswahlbox eingeben und werden somit genauer spezifiziert.
- Das bestehende System liefert bei inkorrekten bzw. ungenauen Eingaben lange Ergebnislisten mit intuitiv nicht nachvollziehbaren Ergebnissen (siehe 3.1). Diese Vorschläge werden ebenfalls ausgewählt und treten in den Log-Daten auf.
- Die Eingabe von Präfixen wird vom Benutzer nicht besonders präferiert.
- Die Angabe des Typs und der Stadt wird zum großen Teil nicht getätigt.
- Eingaben ohne Ergebnisse wurden nicht geloggt.

Das letzte Problem ist auf die getypte Zweifeldsuche zurückzuführen, weil der Benutzer eine Einfeldsuche bevorzugt und für seine Eingaben oft nur das erste Feld nutzt. Nichtsdestotrotz erscheint ein Vergleich beider System sinnvoll und interessant.
Die Log-Einträge sind aufgrund ihrer Datenstruktur für die Evaluierung ungeeignet. Jeder Eintrag besteht aus einer Anfrage und dem zu der Anfrage relevanten Dokument, es wird jedoch eine Liste aller relevanten Dokumente benötigt. Hierfür wird die Collection *TreeMap* verwendet, in der jede Anfrage auf eine Liste der relevanten Dokumente abgebildet wird (Abb. 6.21). Durch die Zusammenfassung der relevanten Dokumente verringert sich die Anzahl der Einträge von 115.947 auf 100.570.

Für die Evaluierung des neuen Systems wird die Angabe der Stadt nur übernommen, wenn es vom Benutzer auch beabsichtigt war. Dies wird überprüft, indem der eingegebene und gewählte Stadtname verglichen werden. Sollten sie unterschiedlich sein oder kein Stadtname vorliegen, erfolgt die Eingabe ohne den Stadtnamen. Die Eingabe und der Vergleich des Typen wird analog ausgeführt (siehe Abb. 6.22).

6.7.1 Ergebnisse

Die Ergebnisse der Evaluierung beider Systeme waren zufriedenstellend. Es ist davon auszugehen, dass die manuelle Wahl der Gewichtungen und Gewichtsparameter nicht optimal ist und weiter optimiert werden kann.
Die Verbesserung wurde mit der Evaluierung der Benutzerdaten der PC-Version bestätigt. Die Ergebnisse sind in den Tabelle 6.2 aufgelistet.

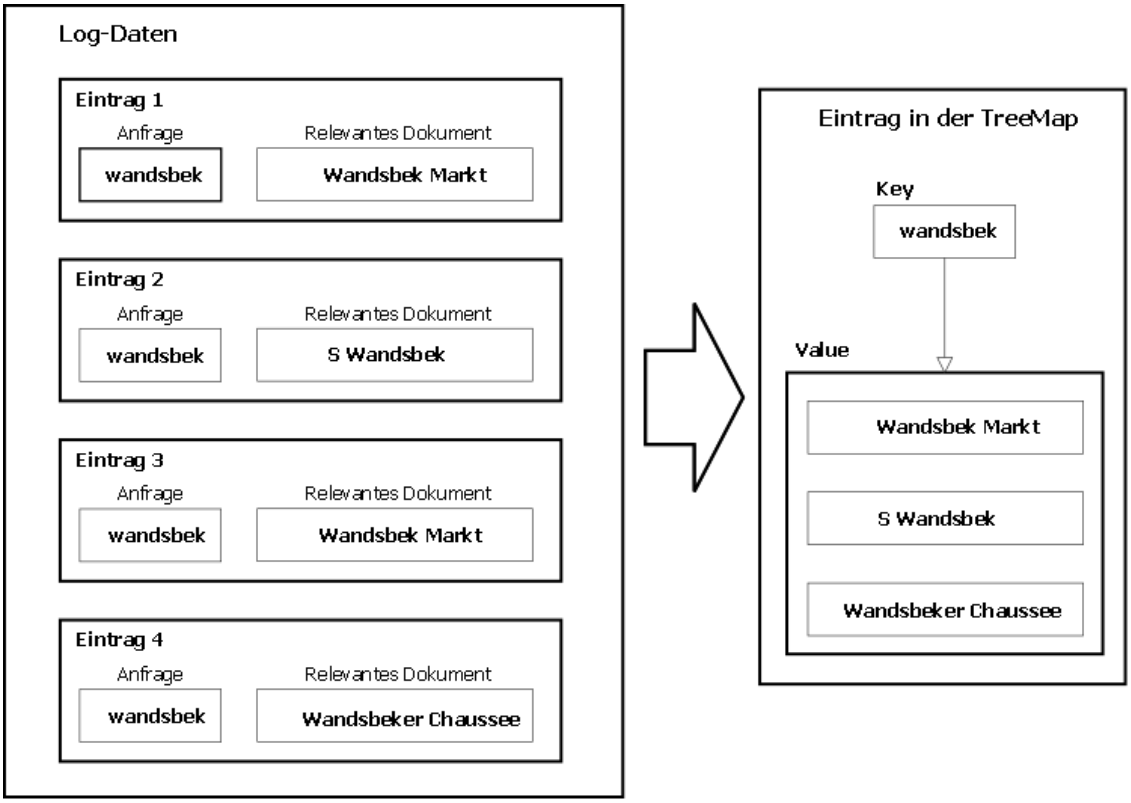

Abbildung 6.21: Transformation der Log-Daten

Mobile Version	Anzahl der Anfragen: 100.570
System	MAP
Alt	0,75965
Neu	0,77023
PC Version	Anzahl der Anfragen: 395.458
System	MAP
Alt	0,74342
Neu	0,76231

Tabelle 6.2: Ergebnisse der Evaluierung

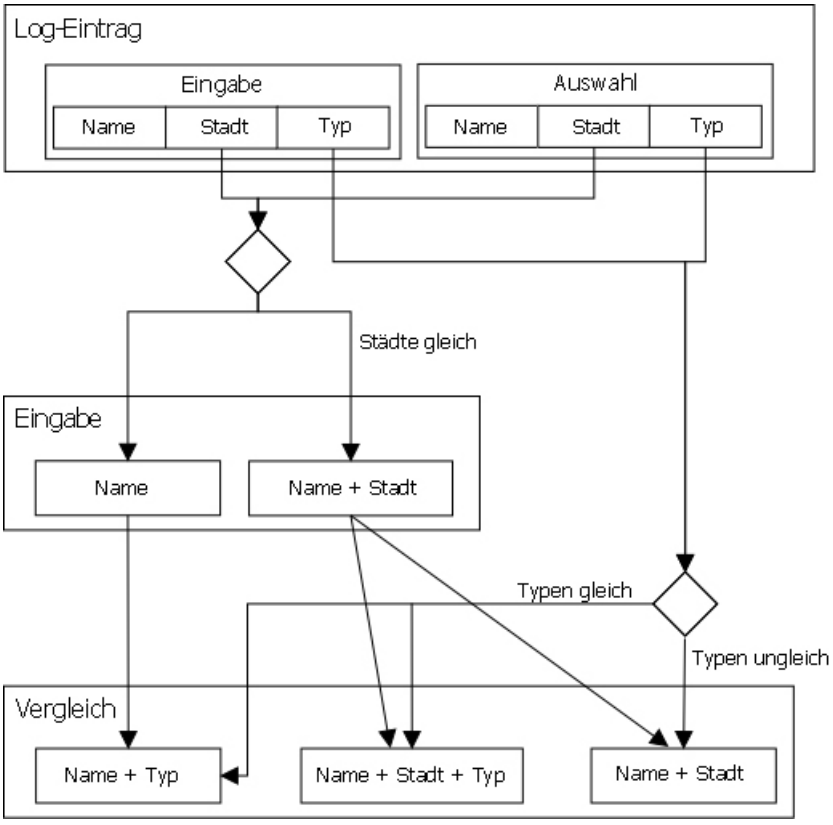

Abbildung 6.22: Eingabe und Vergleich der Log-Daten

7 Optimierung

Das Gebiet der Optimierung beschäftigt sich damit optimale Parameter eines meist komplexen Systems zu finden. Es existieren diverse Optimierungsverfahren, die auf bestimmte Problematiken spezialisiert sind, sie alle unterliegen folgender, grundlegender Klassifikation [Alt02].

- Linear bzw. nichtlinear
- Mit Nebenbedingungen bzw. ohne
- Global bzw. lokal
- Skalar- bzw. Vektoroptimierung

Zu den bekanntesten linearen Optimierungsverfahren zählen das Simplex-Verfahren und die verschiedenen Innere-Punkte-Verfahren. Diese beinhalten meistens lineare Nebenbedingungen, sie werden hauptsächlich in Scheduling, Produktions- und Tourenplanung eingesetzt.
Die nichtlinearen Verfahren sind weitaus komplexer. Ein nichtlineares Optimierungsproblem liegt genau dann vor, wenn die Zielfunktion oder die Nebenbedingungen nichtlineare Funktionen sind. Im Gegenteil zur linearen Optimierung ist die Bestimmung eines globalen Extremwertes in der nichtlinearen Optimierung deutlich unklarer und komplexer. Ein großes Problem besteht darin, dass nicht mehr zwischen lokalem und globalem Optimum unterschieden werden kann.
Für die Ermittlung eines globalen Optimums in einem unbekannten Suchraum sind Algorithmen notwendig, die an mehreren Stellen gleichzeitig suchen und lokale Optima verlassen. Diese Kriterien erfüllten zum großen Teil sogenannte „evolutionäre Algorithmen“, sie werden im Abschnitt 7.1 vorgestellt. Es handelt sich dabei um nichtlineare, globale Verfahren mit Nebenbedingungen, die häufig in Multiparameter-Systemen eingesetzt werden. Für weitere Informationen über Optimierungsverfahren sei an dieser Stellen auf weiterführende Literatur verwiesen.

7.1 Evolutionäre Algorithmen

Die evolutionären Optimierungsverfahren sind Verfahren, die vor allem für Systeme mit mehreren reelwertigen Parametern und einem unbekannten Suchraum sind [Rod95]. Sie

orientieren sich vor allem am Vorbild der biologischen Evolution [EE03]. Den Suchraum bilden Vektoren ganzer oder reeller Zahlen, welche als Genom von Individuen aufgefasst werden können. Ähnlich dem biologischen Vorbild werden diese durch Rekombination erzeugt und durch Mutation verändert. Bei der Rekombination werden ausgewählte Individuen miteinander gekreuzt und Neue erzeugt, bei der Mutation werden die Vektoreinträge (Genome) verändert. Für jedes Individuum wird mit Hilfe einer Fitnessfunktion die Fitness ermittelt. Das Ziel der Optimierung ist die Ermittlung des Individuums mit der höchsten Fitness. Die Erreichung dieses Zieles kann oft nicht nachgewiesen werden, deswegen werden oft auch Pseudo-Optima akzeptiert. Es handelt sich dabei um Werte, die sich in der Nähe des Optimums befinden.
Evolutionäre Algorithmen können als „Suchverfahren“ bezeichnet werden, weil sie den Suchraum erforschen. Aus dem Grunde sind sie für Prolemklassen geeignet, über die entweder kein Hintergrundwissen existiert oder die über eine sehr hohe Komplexität verfügen.
Es existieren zahlreiche Varianten der evolutionären Algorithmen, von denen einige im folgenden kurz vorgestellt werden:

- $(\mu + \lambda)$ **- Evolutionsstrategie** - aus jeder Generation werden μ Eltern ausgesucht und λ Nachkommen durch Klonen erzeugt und mutiert. Aus der Gesamtheit der Eltern und Nachkommen werden für die nächste Generation die besten ausgewählt.

- (μ, λ) **- Evolutionsstrategie** - bei dieser Strategie werden die Eltern nur für die Erzeugung der Nachkommen (durch Klonen) verwendet und haben eine Lebensdauer von genau einer Generation. Damit wird, zumindest ansatzweise, die biologische Evolution besser nachgeahmt.

- $(\mu/\rho, \lambda)$ **- Evolutionsstrategie** - bei dieser Strategie werden die Nachkommen nicht durch Klonen sondern durch Rekombination von ρ-Elternindividuen erzeugt.

7.2 Strategievarianten

Im Rahmen der Untersuchung werden diverse Varianten eines evolutionären Algorithmus untersucht [Tie07]. Ziel war es die Variante zu ermitteln, die mit möglichst wenigen Individuen und einer geringen Anzahl an Generationen zu einem hohen Fitnesswert führt. Dabei spielte die Zeit eine große Rolle, da die Ermittlung des MAP-Wertes eine beachtliche Zeit in Anspruch nimmt. Des Weiteren wurde das Ziel verfolgt den gesamten Suchraum möglichst vollständig zu durchsuchen und das Steckenbleiben in lokalen Optima zu vermeiden.
Wie bereits erwähnt werden neue Genome durch Rekombination und Mutation erzeugt. Die Auswahl entsprechender Strategien beeinflusst die Vererbung und Entstehung von

Merkmalen und somit die Effektivität. Angestrebt wird immer eine sowohl breite Streuung der Merkmale, um möglichst alle lokalen Optima zu erfassen, als auch die konzentrierte Vererbung von Merkmalkonstellationen, die zu einem globalen Optimum führen könnten. Beide Ansätze stehen im Widerspruch zueinander, so dass je nach Strategiewahl unterschiedliche Verläufe der Fitness zu erwarten waren.
In den folgenden Abschnitten werden die relevanten Aspekte und mögliche Probleme besprochen sowie Lösungsansätze vorgestellt.

7.2.1 Population

Die Population ist die Gesamtheit aller Individuen. Zu Anfang der Untersuchung muss entschieden werden, wie die Startpopulation gestaltet werden soll und wie viele Individuen die Population im Laufe der Optimierung erhalten soll.
Aus der Unkenntnis des Terrains und der Komplexität der Zielfunktion ergibt sich zunächst das Problem der Parameterstreuung innerhalb der Startpopulation. Die Parameterstreuung sollte möglichst breit verteilt sein und trotzdem nicht allzu viele Individuen erfordern (max. 100). Ein intuitiver Ansatz ist zunächst die Startpopulation durch Mutation zu erzeugen, wobei die Schrittweite groß gewählt werden muss, um eine breite Streuung zu gewährleisten. Eine weitere Idee ist es von jedem Parameter einige Werte zu wählen und alle mit allen zu kombinieren. Bei genau 17 Parametern ergeben sich jedoch zu viele Individuen, zur Veranschaulichung nehme man für jeden Parameter genau zwei Werte: die Kombination der Parameter ergibt genau $2^{17} = 131072$ Möglichkeiten. Im Laufe des manuellen Testprozess haben sich jedoch einige Parameter herausgebildet, die einen größeren Einfluss auf die Ergebnisse haben als andere. Hierzu zählt vor allem die Gewichtung der Relevanz und der minimalen Ähnlichkeit. Ausgehend von den beiden Parametern kann die Anzahl der Individuen deutlich eingeschränkt werden. Bei einer Wahl von jeweils fünf Werten ergeben sich genau 25 Kombinationsmöglichkeiten. Genauere Informationen diesbezüglich werden im Rahmen einer Voruntersuchung ermittelt.

7.2.2 Rekombination

Der Prozess der Rekombination besteht aus zwei Schritten: der Selektion und der Kreuzung. Bei de Selektion werden einige Individuen einer Population anhand bestimmter Kriterien ausgewählt, der Zweck der Kreuzung ist die Erzeugung neuer Individuen aus zwei oder mehreren vorher ausgewählten Individuen.
Die Selektion verfolgt das Ziel diejenigen Individuen auszuwählen, die sich mit großer Wahrscheinlichkeit in der Nähe eines (möglicherweise lokalen) Optimums befinden. Durch den Erhalt der Erbinformationen wird sichergestellt, dass das Optimum nicht verlassen wird. Für die Selektion wurden folgenden Strategien untersucht:

- **TOP N** - es werden nur die besten Individuen ausgewählt, gefundene Optima werden schnell erklommen.
- **ROULETTE** - den Individuen wird zunächst entsprechend ihrer Fitness eine Wahrscheinlichkeit zugeordnet. Im zweiten Schritt werden iterativ Zufallszahlen erzeugt und mit den Wahrscheinlichkeiten verglichen. Ist die Wahrscheinlichkeit höher als die Zufallszahl, so wird das Individuum ausgewählt. Schwächer Individuen haben eine höhere Chance zu überleben als bei TOP N.
- **UNIQUE ROULETTE** - analog zu der ROULETTE-Strategie mit der Einschränkung, dass die ausgewählten Individuen keine identischen Ausprägungen beinhalten.
- **TOURNAMENT** - es werden zufällig ausgewählte Individuen paarweise ausgewählt und das Individuum mit der höheren Fitness selektiert. Diese Strategie stellt eine gute Mischung zwischen TOP N und ROULETTE dar.

Der Zweck der Kreuzung ist das Erzeugen neuer Individuen, die möglicherweise bessere Fitness aufweisen. Auf diese Weise wird das Optimum angenähert. Die Kreuzung (Crossover) wurde mit Hilfe zweier Strategien durchgeführt:

- **MEAN** - aus den Parametern der Eltern wird ein arithmetischer Durchschnittswert gebildet
- **UNIFORM** - es werden nach dem Zufallsprinzip einige Gene des einen oder des anderen Elternteils übernommen

Um die Streuung der Genome möglichst breit zu gestalten ist es sinnvoll die Kreuzung von ähnlichen Individuen zu unterbinden (Family Check) [Tie07]. Dafür gibt es mehrere Ansätze:

- **NO COMMON PARENTS** - zwei Individuen mit den selben Eltern sind für die Kreuzung unzulässig. Bei nur einem Elternteil ist die Kreuzung erlaubt.
- **NO PARENT** - keines der Elternteile darf die selbe sein.
- **MIN DISTANCE** - die Elternindividuen müssen einen Mindestabstand zueinander haben, damit sie ein neues Individuum zeugen können.

Der zuletzt genannte Ansatz orientiert sich stark am biologischen Vorbild und umfasst bei einer richtigen Wahl der minimalen Distanz implizit auch die anderen Ansätze. Er steigert ebenfalls die Effizienz, weil die Erzeugung fast identischer Individuen vermieden wird. Im Falle einer Kreuzung von Individuen, die sehr nah an einem Optimum sind, sollte die minimale Distanz jedoch verkleinert werden. Die Funktion des Parameters ist der Mutationsschrittweite sehr ähnlich. Beide beeinflussen nämlich die Geschwindigkeit der Suche, die Mutationsschrittweite die Suche nach den Optimumsbereichen, die minimale Genomdistanz die Suche nach dem Optimum selbst.

7.2.3 Mutation

Der Mutation wird eine wichtige Bedeutung beigemessen, weil sie in erster Linie dazu dient, neue Punkte des Parameterraums zu erforschen. Mutationen erzeugen neue Genome, also Parameterkombinationen, die nur durch Selektion und Kreuzung nicht entstehen können.
Die Mutation ist eine zufällige Veränderung des Genoms. Diese bestimmt sowohl die Geschwindigkeit der Suchraumdurchsuchung als auch die Abdeckung. Es hat sich gezeigt, dass eine stochastisch gewählte Schrittweite am Besten geeignet ist. Hierfür wird eine Normalverteilung mit dem Erwartungswert Null empfohlen, wobei die Standardabweichung die Schrittweite repräsentiert. Es ist weiterhin sinnvoll die Schrittweite im Laufe des Optimierungsprozesses an den Fortschritt anzupassen. Dies wäre beispielsweise angebracht, wenn alle Individuen keine nennenswerten Verbesserungen mehr erzielen und die Gefahr besteht, dass sie in lokalen Optima steckengeblieben sind und andere Optima nicht gefunden wurden. Die Adaption der Schrittweite kann auch von Individuum zu Individuum variieren. Die durch vorherige Kreuzung ausgewählter Eltern erzeugten Individuen könnten eine kleinere Schrittweite bekommen, um das Optimum besser zu approximieren, die nur durch Mutation entstandenen Individuen eine größere, um den Suchraum schneller abzusuchen.
Jeder Parameter wird nur innerhalb seines zulässigen „Fensters“ mutiert, d.h. er besitzt einen minimal und maximal zulässigen Wert.

7.2.4 Algorithmus

Für die Erzeugung neuer Generationen wurde die $(\mu/\rho, \lambda)$ - Evolutionsstrategie ausgewählt. Diese Strategie gibt lediglich einen groben Ablauf vor, so dass die genauen Schritte im Vorfeld überlegt werden mussten. Das Hauptproblem war die Unkenntnis des Suchraums. Sollten alle Optima in einem bestimmten Bereich auftreten, ist die Reihenfolge der Schritte anders zu wählen, als wenn sie verteilt sind. Ein weiterer relevanter Aspekt in diesem Zusammenhang war die Festlegung der richtigen Balance zwischen den “suchenden„ und den “erklimmenden“ Individuen. Die Suchenden Individuen werden durch Mutation erzeugt, die Erklimmenden durch Selektion und Cross Over. Wird die Selektion strenger gewählt, so steigt auch der Selektionsdruck. Dann konvergiert jedoch die Population gegen ein Optimum, dass möglicherweise ein lokales Optimum ist.

7.3 Untersuchung

Bei der Untersuchung werden mehrere Ziele verfolgt. Zum einen soll ein Überblick über den Suchraum gewonnen und die Frage nach der Existenz lokaler Optima geklärt werden. Zum anderen sollen geprüft werden, wie weit die bereits erwähnten Optimierungsstrategien für die Problemstellung geeignet sind. Es wird davon ausgegangen, dass die

Klärung dieser Fragen zum letzten Ziel führt, nämlich zur Verbesserung der Mean Average Precision.

Untersuchung 1
Als Erstes wurden alle Strategien miteinander kombiniert, wobei aus zeitlichen Gründen Testdaten mit 17.988 Anfragen verwendet wurden. Die Ergebnisse sollten einen Überblick über die Entwicklung der Fitnesswerte sowie über die möglichen Cluster der Parameterwerte verschaffen. Des Weiteren sollte untersucht werden, ob einige Strategien eher die Tendenz haben lokale Optima zu finden als andere. Eine weiteres Ziel war die Beantwortung der Frage, ob die gewählten Parameterfenster groß genug waren, d.h. ob das Optimum nicht außerhalb davon lag.
Als Erstes wurde die Annahme getroffen, dass alle Optima benachbart sind und sich innerhalb eines überschaubaren Teilraumes befinden. Die Startpopulation wurde durch eine breit gestreute Mutation einen einzelnen Individuums erzeugt, wobei die gewählten Parameter des Individuums ihres zulässigen Fensters lagen. Die Lage und Größe der Parameterfenster waren Erfahrungswerte. Die Erzeugung neuer Generation erfolgte durch Selektion einiger Individuen, die dann gekreuzt und mutiert wurden. Gemäß der Annahme sollte auf diese Weise der die Optima beinhaltende Bereich relativ schnell ermittelt und das Optimum gefunden werden.

Einstellungen
Die einzelnen Strategien wurden miteinander kombiniert, so dass insgesamt 24 Konstellationen untersucht wurden. Sonstige Optimierungsparameter wurden auf folgende Werte gesetzt:

- Populationsgröße: 50
- Abbruchbedingung: keine Verbesserung um mind. 0,001 innerhalb von 5 Generationen
- Mutationswahrscheinlichkeit: 50%
- Mutationsschrittweite: 2
- Crossover: 40%

Ergebnisse
Die Ergebnisse der Voruntersuchung entsprachen den Erwartungen, brachten jedoch auch neue Erkenntnisse.
Wie im Vorfeld erwartet, lieferten verschiedene Selektionsstrategien unterschiedliche Verläufe, dabei führte die Strategie TOP N mit Family Check zu den Individuen mit den besten Fitnesswerten. Der große Vorteil der TOP N Selektionsstrategie ist der, dass die besten Individuen erst dann absterben, wenn sie von den Übrigen überholt werden. Der

Werte der maximalen Fitness nehmen im Verlaufe der Optimierung niemals ab (Abb. 7.1). Bei den übrigen Selektionsstrategien ist die Selektion nicht so streng, so dass auch schwächere Individuen für die Rekombination ausgewählt werden. Der Verlauf der Fitnesswerte unterliegt in dem Falle größeren Schwankungen (Abb. 7.2). Die Analyse der

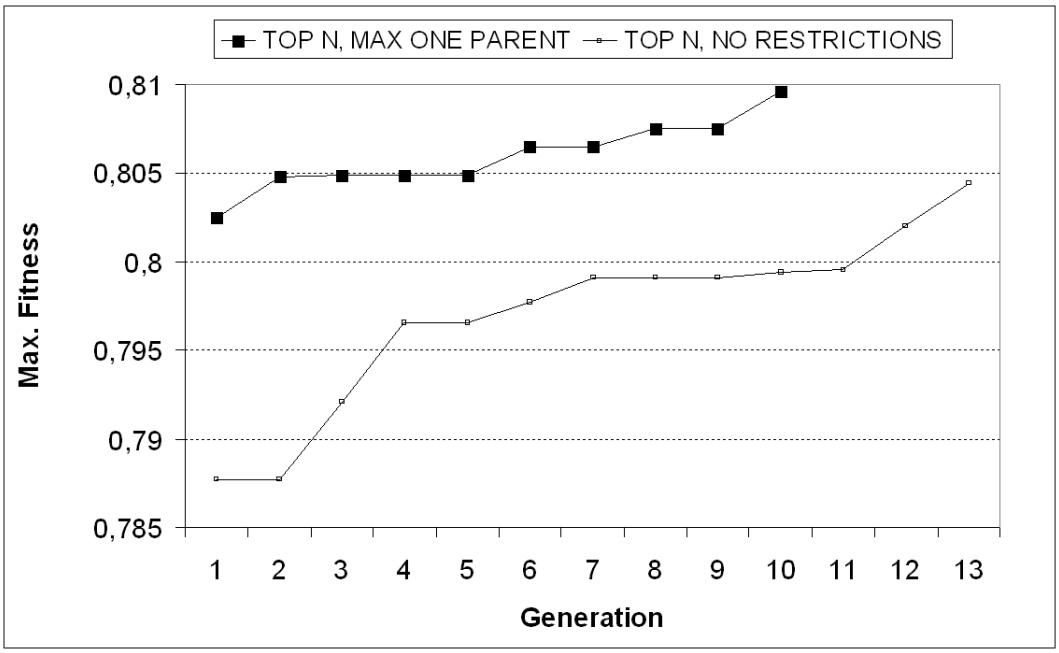

Abbildung 7.1: Optimierungsverlauf der Selektionsstrategie TOP N

Ergebnisdaten liefert folgende Resultate:

Fitnesswerte der ersten Generation: Die fünfzig Individuen der durch zufallsbasierte Mutation erzeugten Startgeneration weisen sehr unterschiedliche Fitnesswerte auf. Die Besten davon erreichen fast die Spitzenwerte. Die Streuung der Parameterwerte der Startindividuen erweist sich daher als sinnvoll und sollte eventuell optimiert werden.

Steigerung der Fitnesswerte: Es ist in den Optimierungsläufen mit hoher Gesamtsteigerung zu beobachten, dass die Verbesserung der maximalen Fitness nicht kontinuierlich und sprunghaft auftritt. Die die Anzahl der Generationen ohne Steigerung der Fitness muss höher gesetzt werden.

Parameter: Die Parameterwerte der besten Individuen (siehe 7.1) sind unterschiedlich gestreut.Diese Beobachtung im Folgenden detaillierter analysiert.
Zunächst werden einige Parameterwerte der fünf besten Individuen tabellarisch dargestellt (7.1). Es sei erwähnt, dass jedes Individuum mit einer anderen Strategievariante erzeugt wurde. Somit sind jegliche durch Verwandtschaft entstandene Ähnlichkeiten ausgeschlossen.

Der Wert der Skalierung der Häufigkeitsgewichtung befindet sich, mit Ausnahme

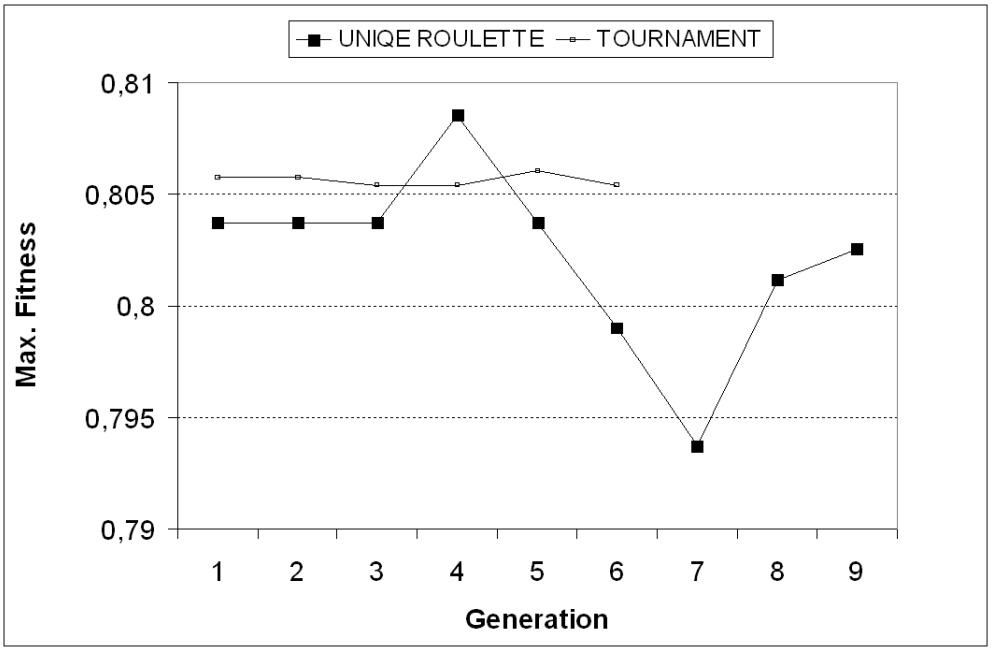

Abbildung 7.2: Optimierungsverlauf der Selektionsstrategien Roulette und Tournament

Fitness	Häuf.Par.	Min. Ähnl.	Placemax	Placemin	Len.pen.	Addr.	OR
0.8099	2,4	0,74	6,4	1,2	-5	2,0	0,80
0.8097	3,0	0,75	1,2	1,0	-2	2,0	0,74
0.8097	3,0	0,73	1,8	1,0	-1	2,0	1,0
0.8096	2,98	0,75	3,02	3,02	0	1,0	0,83
0.8095	3,0	0,75	3,9	1,0	-4	2,0	0,6

Tabelle 7.1: Ausgewählte Paramter der TOP 5 Individuen (Fitness, Gewichtsparameter der Häufigkeit, minimale Ähnlichkeit der *FuzzyQuery*, maximale Gewichtung der *place*-Felder, minimale Gewichtung der *place*-Felder, Bestrafung der Feldlänge-eine negative Bestrafung ist gleich einer positiven Gewichtung), Gewichtung des Adressenmusters, Gewichtung der OR-Query

des ersten Individuums, am Rande des ihm zugewiesenen Intervalls [0,4 3,0]. Im Verlauf der weiteren Untersuchung wird die obere Schranke auf 5,0 erhöht.
Die minimale Ähnlichkeit der *FuzzyQuery* unterliegt einer geringen Streuung und wird auf den Intervall 0,73 bis 0,75 gesetzt.
Die willkürlich gestreuten Werte der Gewichtungen der *place*-Felder deuten auf die nicht vorhandene Relevanz der Präfixgewichtung. Diese Erkenntnis wird zunächst auf die geringe Anzahl der Präfixanfragen seitens der Benutzer zurückgeführt.

Eine bessere Übersicht über die Parameterwerte liefern die in den Abbildungen dargestellte Diagramme. Es sind dort die Parameterwerte der sieben besten Individuen dargestellt. Zur besseren Übersichtlichkeit werden für die Darstellung zwei Diagramme ver-

wendet.

Es ist auf den Diagrammen zu beobachten, dass einige Paramterwerte die Grenzen ihres Intervalls erreichen und andere bei vielen Individuen gleich sind. In der nächsten Untersuchung werden die Beobachtungen berücksichtigt.

Untersuchung 2
Die Analyse der ermittelten Parameter der besten Individuen führt zu neuen Ideen bezüglich der Mutation:

- Bei Parametwerten an der Intervallsschranke werden die Intervalle verschoben
- Bei leicht streuenden Werten wird das Intervall verkleinert
- Bei großer Streuung wird die Annahme getroffen, dass die Paramterwerte irrelevant sind. Diese werden auf einen konstanten Wert gesetzt und bei der Mutation nicht berücksichtigt.

Die großen Unterschiede der Fitnesswerte in der Startpopulation fordern eine bessere Abdeckung des Suchraumes durch die zu Anfang erzeugten Individuen. Da die Anzahl der variierender Parameter eingeschränkt wird, muss die bisher gewählte Anzahl von 50 jedoch nicht erhöht werden. Das Startindividuum wird aus den „Gewinnern“ der Voruntersuchung gewählt. Die Bildung von Clustern wird in der Startpopulation dadurch vermieden, dass jedes erzeugte Individuum einen Mindestabstand zu allen bisher erzeugten Individuen aufweisen muss.
Das Erklimmen der Optima und die Durchsuchung der Raumes werden parallel ausgeführt. Die selektierten Individuen werden miteinander gekreuzt und mutiert. Für die Mutation wird eine Schrittweite von 1 gewählt und die Cross Over Strategie UNIFORM kombiniert mit den Familienstrategie MAX ONE PARENT und NO RESTRICTIONS verwendet. Die restlichen Individuen werden lediglich mit der Schrittweite von 2 mutiert. Die Strategien werden unterschiedlich gewählt, weil die Mutation in den beiden Fällen einen andern Zweck erfüllt:

- Die selektierten Individuen sollen sich dem Optimum nähern und werden deswegen gekreuzt. Das Cross Over sorgt dafür, dass sich die Nachfolgerindividuen von den Elternindividen nicht entfernen. Dies ist zwar erwünscht, sorgt aber dafür, dass der Spielraum der Parameter eingeengt wird. Aus diesem Grunde werden die Nachfolgeindividuen ebenfalls mutiert, jedoch mit einer geringeren Schrittweite.
- Die Mutation der nicht selektierten Individuen verfolgt das Ziel den Suchraum abzusuchen. Aus dem Grunde wird hierfür eine größere Schrittweite gewählt.

Die sonstigen Optimierungsparameter lauten:

- Selektionsstrategie: TOP N

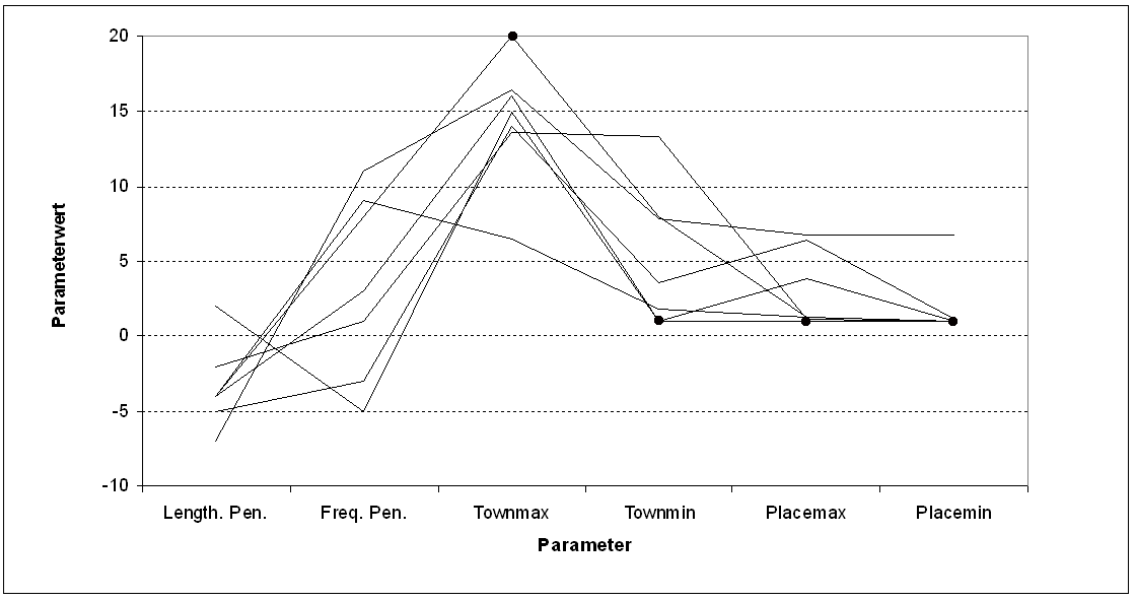

Abbildung 7.3: Paramterwerte der sieben besten Individuen (1a)

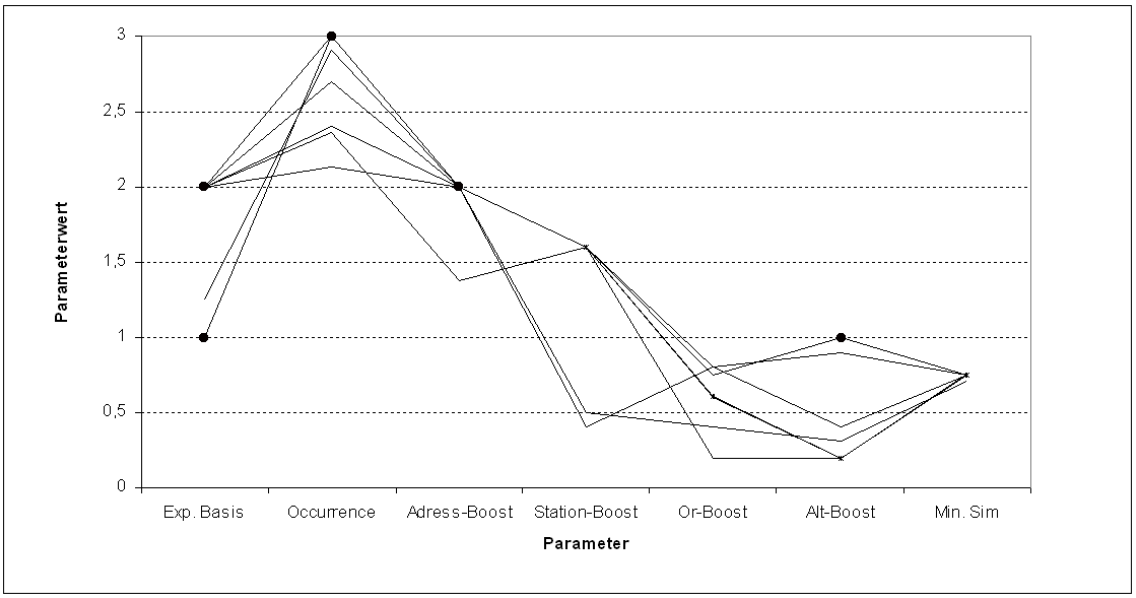

Abbildung 7.4: Paramterwerte der sieben besten Individuen (1b)

- Mutationsstrategie: UNIFORM
- Familienstrategie: MAX ONE PARENT und NO RESTRICTIONS
- Populationsgröße: 50 Individuen
- Mutationswahrscheinlichkeit: 50 %
- Mutationsschrittweite: 1
- Crossover: 40 %
- Abbruchbedingung: keine Verbesserung um mind. 0,0001 innerhalb von 10 Generationen

Aus zeitlichen Gründen werden für den Optimierungslauf nur 1863 Anfragen verwendet. Aus den Ergebnissen sollen weitere Erkenntnisse gewonnen und weitere Läufe gestartet werden.Die Größe der Intervalle wurde stark verringert und einige Parameter auf einen konstanten Wert festgelegt.

Ergebnisse

Zunächst werden die Parameter der besten fünf Individuen betrachtet (Tab.7.2).

Fitness	Häuf.Par.	Min. Ähnl.	Placemax	Placemin	Len.pen.	Addr.	OR
0,7920	4,0	0,75	6,0	1,5	2	2,0	0,5
0,7910	3,6	0,75	1,0	1,0	2	1,0	1,0
0,7907	4,0	0,75	5,5	1,0	2	2,0	0,9
0,7903	2,0	0,75	1,0	1,0	2	2,0	0,9
0,7901	4,0	0,75	1,0	1,0	2	1,0	0,5

Tabelle 7.2: Ausgewählte Paramter der TOP 5 Individuen (Fitness, Gewichtsparameter der Häufigkeit, minimale Ähnlichkeit der *FuzzyQuery*, maximale Gewichtung der *place*-Felder, minimale Gewichtung der *place*-Felder, Bestrafung der Feldlänge-eine negative Bestrafung ist gleich einer positiven Gewichtung), Gewichtung des Adressenmusters, Gewichtung der OR-Query

Die ermittelten Parameter unterscheiden sich von denen der Voruntersuchung, was auf die Testdaten zurückzuführen ist. Die Erhöhung der Invervallgrenze für den Parameter der Häufigkeitsgewichtung erwies sich als sinnvoll, die Werte der meisten der in der Tabelle dargestellten Individuen überschreiten die im letzten Lauf gesetzte Grenze deutlich. Auffällig ist der Wert für die minimale Ähnlichkeit, der genau der festgesetzten Obergrenze entspricht.

Für einen besseren Vergleich werden die Parameter der sieben besten Individuen zunächst grafisch dargestellt (Abb. 7.5 und 7.6).

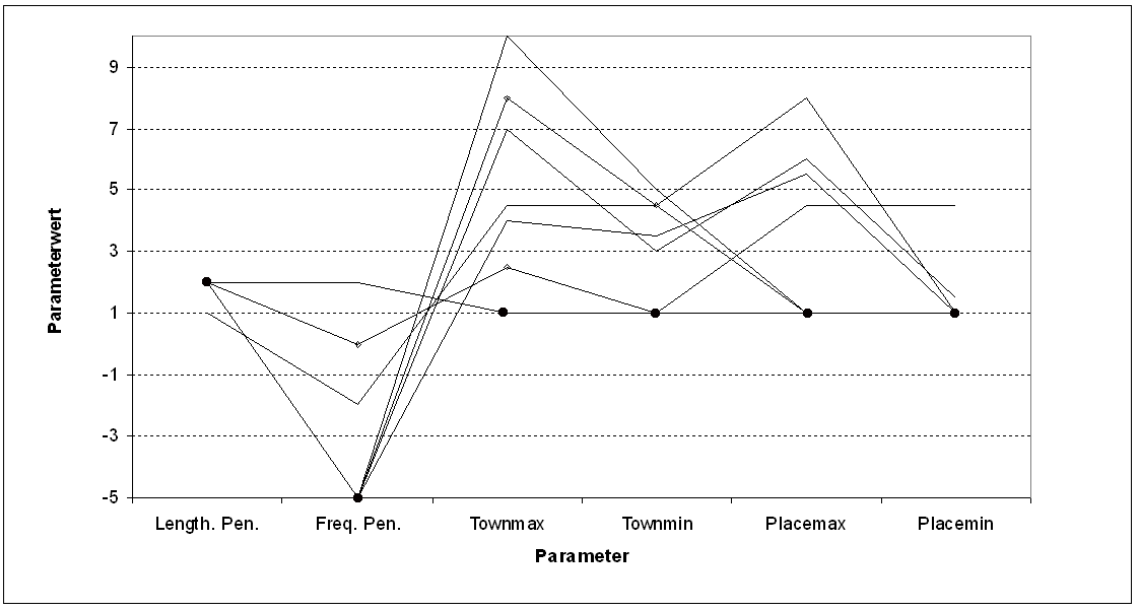

Abbildung 7.5: Paramterwerte der sieben besten Individuen (2a)

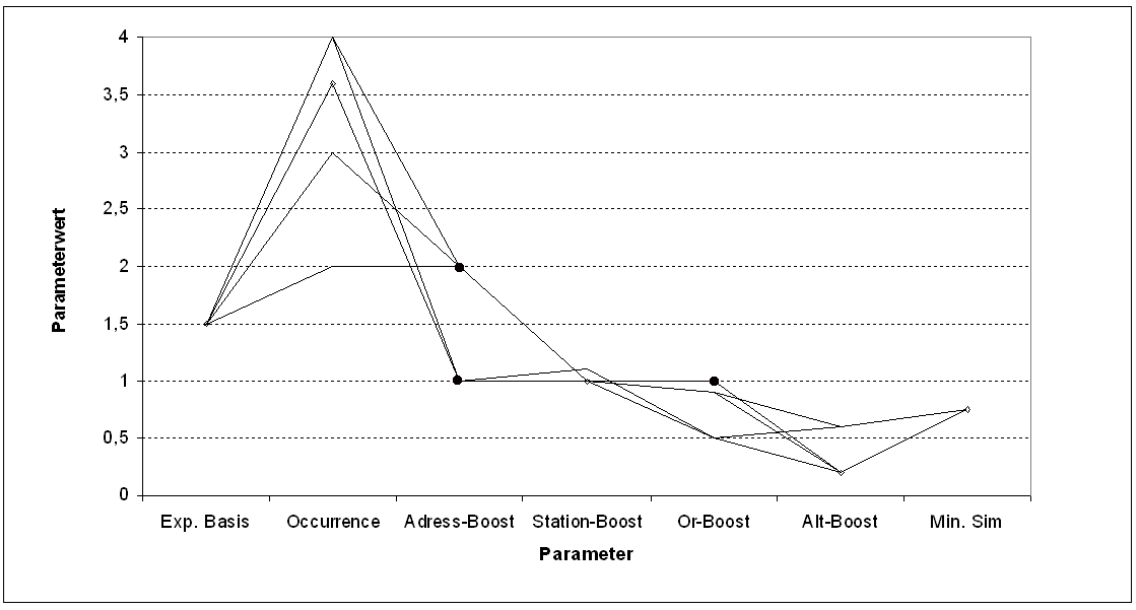

Abbildung 7.6: Paramterwerte der sieben besten Individuen (2b)

Auffällig ist erneut eine Ähnlichkeit der Verläufe, was auf ein globales Optimum hindeutet. Diese Vermutung wird im nächsten Lauf überprüft, in dem der Selektionsdruck verringert wird. Bei den mit schwarzen Punkten markierten Parametern die Intervallgrenzen neu angepasst.

Untersuchung 3
In folgender Untersuchung wird der Suchraum flächendeckender erforscht und der Cross Over auf 20% reduziert.
Die Optimierungsparameter lauten:

- Selektionsstrategie: TOP N
- Mutationsstrategie: MEAN
- Familienstrategie: NO RESTRICTIONS
- Populationsgröße: 50 Individuen
- Mutationswahrscheinlichkeit: 50 %
- Mutationsschrittweite: 1 bei den selektierten, 2 bei den mutierten Individuen
- Crossover: 20 %
- Abbruchbedingung: keine Verbesserung um mind. 0,0001 innerhalb von 10 Generationen

Ergebnisse
Die maximale Fitness konnte nicht verbessert werden. Dennoch werden einige Parameterwerte der drei besten Individen vorgestellt und diskutiert. Die Werte für *PlaceMax* und *PlaceMin* werden nicht mehr betrachtet, weil sie in den meisten Fällen gleich sind, stattdessen werden die Werte für *TownMax* und *TownMin* gezeigt.

Fitness	Häuf.Par.	Min. Ähnl.	Townmax	Townmin	Len.pen.	Addr.	OR
0,7890	4,6	0,76	10,0	4,0	1,0	3,0	0,6
0,7889	5,0	0,76	9,5	1,0	-7,0	1,2	1,0
0,7880	2,0	0,78	8,0	5,0	-3,0	2,0	1,0

Es ist auffällig, dass die Werte für TownMax und TownMin erneut weit auseinander liegen. Weiterhin ist der Wer für den Häufigkeitsparameter auf den Grenzwert gestiegen. Da die Fitnesswerte den aus dem vorigen Lauf unterliegen, werden die Parameterwerte nicht genauer analysiert. Die schlechteren Fitnesswerte werden in erster Linie auf den geringeren Selektionsdruck zurückgeführt. Die Anzahl der selektierten Individuen wird daher in der nächsten Untersuchung wieder erhöht.

Untersuchung 4
In der folgenden Untersuchung werden folgende Optimierungsparamter verwendet:

- Selektionsstrategie: TOP N
- Mutationsstrategie: UNIFORM
- Familienstrategie: MAX ONE PARENT
- Populationsgröße: 60 Individuen
- Mutationswahrscheinlichkeit: 50 %
- Mutationsschrittweite: 1 bei den selektierten, 2 bei den mutierten Individuen
- Crossover: 50 %
- Abbruchbedingung: keine Verbesserung um mind. 0,0001 innerhalb von 10 Generationen

Ergebnisse
Die beste Fitness konnte erneut nicht übertroffen werden, wurde jedoch angenähert:

Fitness	Häuf.Par.	Min. Ähnl.	Townmax	Townmin	Len.pen.	Addr.	OR
0,7909	2,4	0,77	7,5	1,0	0,0	3,0	1,0
0,7908	4,4	0,77	6,0	3,0	-1,0	3,0	1,0

Es wurde beobachtet, dass die maximale Fitness in zwei Sprüngen erreicht wurde, von denen der erste direkt bei der Erzeugung der Startpopulation und der zweite nach neun Generationen auftrat. Eventuell ist die Abbruchbedingungen anzupassen und die maximale Anzahl der Generationen ohne Verbesserung erhöht werden.

7.4 Fazit

Die Ergebnisse der Optimierung sind zufriedenstellend. Das primäre Ziel die Parameter zu verbessern wurde erreicht. Es konnte auch gezeigt werden, dass die Parameterwerte an die Log-Daten angepasst werden.
Es besteht dennoch Unklarheit über die genaue Position der Optima im Suchraum, die Frage nach den lokalen Optima kann ebenfalls nicht beantwortet werden. Der Suchraum konnte nur bedingt eingeschränkt werden. Einer der Gründe ist der für eine Untersuchung notwendige Zeitaufwand, der wiederum auf die niedrige Preformance der in LUCENE implementierten *FuzzyQuery* zurückzuführen ist.
Für die Beantwortung de verbliebenen Fragen werden folgende Lösungsansätze vorgeschlagen:

- Die Optima müssen von mehreren, getrennten Populationsgruppen gleichzeitig erklommen werden, die nicht miteinander gekreuzt werden.
- Die Anzahl der suchenden Individuen muss erhöht werden, muss jedoch variabel sein. In der ersten Phase wird sie hochgehalten und nimmt zu Ende immer mehr ab.
- Die Optimierung muss auf mehrere Rechner verteilt werden
- Für die Suche muss LUCENE 4.0 eingesetzt werden, damit wird die Antwortzeit enorm verkürzt.

8 Zusammenfassung

8.1 Fazit

Die vorliegende Arbeit verfolgte mehrere Ziele. Das primäre Ziel war die Erstellung eines Suchsystems, welches die Suche kurzer, semistrukturierter Dokumente mit Hilfe der Infomation Retrieval Techniken ermöglicht. Die implizite Forderung nach einer Verbesserung der Qualität gegenüber dem eingesetzten System wurde zum einen vollständig erfüllt, zum anderen wurde ein adaptives System gebaut, welches an die Forderungen seitens des Benutzers angepasst werden kann. Mit Hilfe der eingesetzten Konzepte des Information Retrieval (Boolesches Modell, Vektorraum Modell) und des verwendeten Frameworks LUCENE wurde ein mächtiges, adaptives und flexibles Tool implementiert, welches problemlos für die Einfeldsuche produktiv eingesetzt werden kann.
Die Erfüllung der gestellten Forderungen wird im Folgenden diskutiert:

- **Keine Schlüsselwörter** - das System erkennt Schlüsselwörter und erkennt den Typen des gesuchten Objektes (100%).
- **Berücksichtigung von Rechtschreibfehlern** - Terme mit Rechtschreibfehlern werden erkannt, die exakte Toleranz kann an die Benutzerdaten angepasst werden (100%).
- **Erkennung von Präfixen** - das System erkennt alle Präfixe, die Toleranz kann ebenfalls angepasst werden (100%).
- **Akzeptanz von Akronymen** - das System erkennt nur Kürzel, sondern erweitert die Anfrage um die hinter dem Kürzel stehenden Terme (100%).
- **Bevorzugung kurzer Dokumenteninhalte** - im Laufe der Optimierung hat sich gezeigt, dass diese Forderung zu einer geringeren Qualität führt und wird verworfen (0%).
- **Vermeidung leerer Ergebnislisten** - es braucht lediglich ein Term der Anfrage auffindbar sein, um eine Ergebnisliste zu präsentieren. Hierbei kann es sich auch um einen inkorrekt geschriebenen Präfix handeln (100%).
- **Berücksichtigung der Relevanz** - Die Anfragehäufigkeit der Suchobjekte wurde berücksichtigt, aufgrund der ungünstigen Verteilung ist die Sammlung weitere Benutzerdaten jedoch sinnvoll. Es ist ebenfalls empfohlen die neuen Daten über die Einfeldsuche zu sammeln (80%).

- **Erfüllung der Benutzererwartung** - die Erfüllung der Benutzererwartung wurde mit Hilfe der evolutionären Optimierung approximiert. Für eine exaktere Untersuchung sind jedoch vollständig Jurorbewertungen unerlässlich, die sowohl die relevanten als auch die irrelevanten Dokumente spezifizieren (80%).
- **Flexibilität** - das System ist bezüglich der Änderung der Dokumentenstruktur, der Erweiterung der Inhalte und des Hinzufügens neuer Dokumente flexibel gestaltet (100%).
- **Kurze Antwortzeit** - die Forderung nach einer Antwortzeit unter eine halben Sekunde wurde nur teils erfüllt. Für Anfragen mit mehr als vier Termen und einem vorhanden Muster benötigt das System ca. 600 ms, was auf die niedrige Performance der in LUCENE implementierten FuzzyQuery zurückzuführen ist. Im neuen Release ist dieses Problem jedoch behoben (80%).
- **Einfache Integration** - das Suchsystem ist vollständig in Java geschrieben und lässt sich in das GEOFOX-System problemlos integrieren (100%).

Das neue System kann mit wenigen Anpassungen für die Bewältigung anderer Aufgaben eingesetzt werden:

1. Die Suche nach besonderen Orten (POI) ist im bestehenden System zwar implementiert, wird jedoch selten benutzt (ca 2-5% aller Anfragen). Die Attraktivität einer solchen Suche könnte durch eine Erweiterung des Dokumentenbestands um neue POIs und das Hinzufügen ausführlicherer Informationen gesteigert werden, was mit dem neuen Systems problemlos umgesetzt werden könnte.
2. Suche nach Seiten im GEOFOX und im HVV-Portal. Die beiden Internetpräsenzen verfügen über eine Suchfunktion, der Vorteil des neuen Systems ist jedoch die Anpassungsfähigkeit an die Präferenzen des Benutzers.
3. Einfeldsuche für den Start- und Zielort in einem Textfeld.
4. Eine Implementierung auf einem Android-Smartphone ist ebenfalls möglich. Aufgrund der begrenzten Kapazität des Arbeitsspeichers müsste das LUCENE-Framework auf die notwendigsten Klassen und der Index reduziert werden. Der mögliche Grad der Reduzierung des Index wird auf 50% (8 MB) geschätzt.
5. Suche nach Produkten und ihre Klassifizierung im E-Commerce.

8.2 Ausblick

Die Ergebnisse der Evaluierung bestätigten die höhere Qualität des neuen Systems gegenüber dem Alten. Im Laufe der Optimierung wurden die Systemparamter untersucht und optimiert. Diese beiden Aspekte können jedoch genauer erforscht werden.

Zum einen muss eine effiziente Methode für die Beschaffung qualitativer Benutzerdaten überlegt werden. Ob sie automatisch oder manuell durchgeführt wird, sei zunächst dahingestellt. Die Qualität der Daten spielt hierbei eine besondere Rolle, denn eine große Anzahl von geloggten Anfragen führt zu einem hohen zeitlichen Aufwand bei der Optimierung. Gewünscht wären an dieser Stelle ausführliche Jurorbewertungen, die zu den Anfragen die relevanten, die irrelevanten und die Reihenfolge der Relevanten beinhalten.
Die Optimierung erwies sich als sehr zeitaufwendig, was auf die evolutionären Algorithmen zurückzuführen ist. Bei einer durchschnittlichen Antwortzeit von 100 ms und 20.000 Anfragen dauert die Berechnung der Mean Average Precision ca. 18 Minuten. Eine Population von 100 Individuen und ein Lauf von 10 Generationen benötigt somit 18.000 Minuten, das heißt ca. 30 Stunden. Sollen Anfragen nach möglichst vielen Suchobjekten herangezogen werden und es werden 500.000 Anfragen ausgewertet, ergibt sich ein Zeitaufwand von über 750 Stunden, das heißt ca. 31 Tagen. Aus dem Grunde müssten andere Optimierungsmethoden untersucht werden, die weniger zeitaufwendig sind.

Literaturverzeichnis

[Alt02] ALT, Walter: *Nichtlineare Optimierung: Eine Einführung in Theorie, Verfahren und Anwendungen*. 1.Aufl. Vieweg Verlag, 2002

[Cai02] CAI, Guoray: *GeoVSM: An Integrated Retrieval Model For Geographical Information*. The Pennsylvania State University 002K Thomas Building, University Park, PA 16802, 2002

[Dal06] DALIANIS, Hercules: *Improving search engine retrieval using a compound splitter for Swedish*. Stockholm University : Department of Computer and System Sciences, Stockholm University, 2006

[EE03] EIBEN, Agoston E. ; E.SMITH, James: *Introduction to evolutionary computing*. Springer Verlag, 2003

[Fou] FOUNDATION, Apache S.: *Lucene*. `http://lucene.apache.org/`

[Got09] GOTTRON, Dr. T.: *Information Retrieval, Vorlesungsskript*. Univesität Mainz, 2009

[McC10] MCCANDLESS: *Lucene in Action*. 2. Aufl. Stamford : Manning Publication Co., 2010

[MRS08] MANNING, Christopher D. ; RAGHAVAN, Prabhakar ; SCHÜTZE, Hinrich: *Introduction in Information Retrieval*. Cambridge University Press, 2008

[Rod95] RODEMEYER, Christian: *Entwicklung und Erprobung eines Evolutionären Algorithmus zur rechnergestützten Optimierung der Maschinenbelegung in einem Druckgußwerk*. Universität Gesamthochschule Essen : Fachbereich Wirtschaftswissenschaften, Universität Gesamthochschule Essen, 1995

[Tie07] TIEDEMANN, Jörg ; BASILI, Roberto (Hrsg.) ; PAZIENZA, Maria T. (Hrsg.): *Lecture Notes in Computer Science*. Bd. 4733: *A Comparison of Genetic Algorithms for Optimizing Linguistically Informed IR in Question Answering*. Springer, 2007. – 398–409 S. `http://dx.doi.org/http://dx.doi.org/10.1007/978-3-540-74782-6_35`. `http://dx.doi.org/http://dx.doi.org/10.1007/978-3-540-74782-6_35`. – ISBN 978–3–540–74781–9

[VA10] VIKAS, Om ; ARORA, Pooja: *Ranking Strategy Using Hybrid Model*. AKGEC Ghaziabad, India, 2010