Unscharfe Suche für Terme geringer Frequenz in einem großen Korpus

Dissertation zur Erlangung des Grades
Doktor der Kognitionswissenschaft
(PhD in Cognitive Science),
eingereicht am
Fachbereich Humanwissenschaften
der Universität Osnabrück
von

Karl Gerhards
aus
Vechta

Osnabrück, 2010

Danksagung

Herzlich danken möchte ich meinem Doktorvater Prof. Dr. Kai-Uwe Kühnberger für seine Bereitschaft, dieses Projekt zu betreuen. Ihm und Herrn PD Dr. Helmar Gust verdanke ich Ermutigung und zahlreiche hilfreiche Tipps und Hinweise.

Vielen herzlichen Dank den ungenannten Programmierern von Open Source- und Freeware-Programmen, ohne die diese Arbeit nicht entstanden wäre.

Ein besonderer Dank gilt schließlich meiner Frau Barbara für den Freiraum, den sie mir gegeben hat, um dieses Projekt zu verwirklichen.

Abstract

Until now infrequent terms have been neglected in searching in order to save time and memory. With the help of a cascaded index and the introduced algorithms, such considerations are no longer necessary.

A fast and efficient method was developed in order to find all terms in the largest freely available corpus of texts in the German language by exact search, part-word-search and fuzzy search.

The process can be extended to include transliterated passages.

In addition, documents that contain the term with a modified spelling, can also be found by a fuzzy search.

Time and memory requirements are determined and fall considerably below the requests of common search engines.

Zusammenfassung

Selten vorkommende Terme wurden bei der Suche bisher vernachlässigt, um Zeit und Speicherplatz zu sparen. Mit einem kaskadierten Index und den vorgestellten Algorithmen sind solche Rücksichten nicht mehr erforderlich.

Für das größte frei verfügbare Korpus mit Texten in deutscher Sprache wurde ein schnelles und effizientes Verfahren entwickelt, um alle Terme im Korpus mit exakter Suche, Teilwortsuche und unscharfer Suche aufzufinden.

Das Verfahren ist erweiterungsfähig um transliterierte Textstellen.

Darüber hinaus werden mit einer unscharfen Suche auch die Dokumente gefunden, die den Term in einer abgewandelten Schreibweise enthalten.

Zeit- und Speicherbedarf werden ermittelt und unterschreiten die Anforderungen verbreiteter Suchmaschinen erheblich.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Neben Fortpflanzung und Stoffwechsel ist die Mustererkennung das entscheidende biologische Prinzip. Für jede Sinneswahrnehmung müssen die Sinnesreize verarbeitet, mit bekannten Reizmustern verglichen und diesen zugeordnet werden. Unscharfe Suche bedeutet nichts anderes: eine Musterkombination mit bekannten Mustern vergleichen und unter den bekannten Mustern das am besten passende herausfinden. So betrachtet, hat die Natur unserer Erkenntnis und unserm Intellekt gegenüber einen Entwicklungsvorsprung von einer Milliarde Jahren, den wir so schnell wie möglich aufholen wollen.

Zur Suche von Termen in einem Text gibt es eine Vielzahl von Verfahren und eine schier unüberschaubare Literatur. Viele Einschränkungen, die früher gültig waren und besondere Berücksichtigung verlangten, sind heute weggefallen. Dazu zählen die Verfügbarkeit riesiger Hauptspeicher, günstige Preise für Festplatten und schnellen Hauptspeicher und schnelle Prozessoren. Andererseits haben sich die Anforderungen erhöht und die heute zu erfassenden Textmengen übersteigen die früher üblichen um Größenordnungen.

In dieser Arbeit soll der Frage nachgegangen werden, ob und wie neue Algorithmen und Verbesserungen bekannter Algorithmen Retrievalergebnisse verbessern können.

Suchverfahren bemühen sich üblicherweise, für einen Text relevante Terme zu finden und Texte nach der Häufigkeit des Vorkommens solcher relevanten Terme zu sortieren. Die am Häufigsten vorkommenden Wörter werden als Stopp-Wörter bezeichnet und als nicht relevant betrachtet. Die nur ein- oder zweimal vorkommenden Wörter, die in einem Vollformenwörterbuch zu einem gegebenen Korpus mehr als die Hälfte aller Wortformen liefern, werden ebenfalls ausgeblendet.

Hier stellt sich die Frage, ob dieses Vorgehen, das durch die früher stark beschränkte Rechnerkapazität bedingt war, heute noch zeitgemäß ist, oder ob die Berücksichtigung dieser Wörter das Retrievalergebnis verbessern können.

In dieser Arbeit wird gezeigt, dass

- der Einbezug von selten vorkommenden Wörtern in eine Dokumentensuche sinnvoll und vorteilhaft ist,

- selten vorkommende Wörter nur durch eine unscharfe Suche erfasst werden, und

- für die unscharfe Suche je nach Wortlänge verschiedene Algorithmen optimal sind.

Bekannte Algorithmen für die unscharfe Suche wurden reimplementiert und weiter verbessert.

Entsprechend dem Zipf'schen Gesetz [Zip49] bilden selten vorkommende Wörter einen erheblichen Anteil der Einträge in einem Vollformenwörterbuch zu einem gegebenen Korpus. Für das untersuchte Korpus Edition Gutenberg-DE[1] kommen ein Drittel aller Wortformen nur einmal im Korpus vor, ein weiteres Viertel zwischen zwei- und zehnmal (vgl. die Verteilung der Worthäufigkeiten in Abbildung 3.4).

Selten vorkommende Wörter wurden in Untersuchungen bisher ausgeblendet. Häufig wurden sie nicht einmal indexiert. In der Regel gehören sie nicht zum aktiven Wortschatz des Suchenden.

So bringt die Erweiterung durch eine unscharfe Suche zum Suchbegriff »geschossen« die Terme »geschosse« und »jeschossen«, die der Mundart entstammen. Die Erweiterung durch eine Suche nach anderen Wortformen bringt unter anderem die Terme »schießet« und »schießest«, die wieder durch eine unscharfe Suche nicht gefunden werden, ohne eine zu große Treffermenge einzubeziehen.

Verzichtet man auf eine unscharfe Suche, dann werden nur die eingegebenen Suchbegriffe gefunden. Solche Dokumente, die den Suchbegriff nur in alternativen Schreibweisen enthalten, können dann nicht mehr gefunden werden. Im untersuchten Korpus wurden jedoch in der Mehrzahl der Fälle mindestens ein weiterer relevanter Suchbegriff gefunden, der zum Nachweis weiterer Dokumente führte.

In Anbetracht der Größe des verwendeten Korpus ist für alle verwendeten Algorithmen ein Blick auf die benötigten Ausführungszeiten erforderlich. Die schlichte Volltextsuche nach dem Wort »Gnade« dauerte 10 Minuten und erbrachte fast 6 500 Dateien, in denen das Wort vorkommt. Solche Ergebnisse sind für den Anwender wenig hilfreich. Die Suchzeit strapaziert die Geduld des Anwenders und die gefundenen Dateien lassen sich in ihrer Menge nicht mehr überblicken.

Die inhaltliche Suche ist ein bislang ungelöstes Problem. Durch die Kombination einer Erweiterung des Suchbegriffs mit Hilfe vorhandener Wörterbücher und einem Abgleich mit dem im Korpus verwendeten Wörtern wird der Suchraum intelligent erweitert.

Eine unscharfe Suche mit einem Abgleich mit den im Korpus vorhandenen Wörtern erschließt auch Texte, in denen die Suchwörter in unüblichen Schreibweisen erscheinen.

[1] http://gutenberg.spiegel.de/

Wie sich jeder leicht selbst überzeugen kann, erschließt sich der Inhalt eines Satzes einem unerfahrenen Leser auch dann, wenn die Buchstaben im Inneren eines Wortes durcheinander gewürfelt sind.[2] Gleiches darf man auf einer höheren Ebene auch für ganze Texte postulieren. Es gibt Schnelllesekurse, in denen der Leser lernt, den Inhalt eines Textes durch „diagonales“ Lesen rascher zu erschließen als dies mit einem Wort-für-Wort-Lesen möglich ist. Diese Fähigkeit liegt in der originären Fähigkeit aller Organismen zur Mustererkennung begründet. Mustererkennung ist ein basales biologisches Prinzip mit großer Fehlertoleranz, das für eine Simulation durch die exakt arbeitenden Rechenmaschinen durchaus nicht-triviale Algorithmen erfordert.

1.2 Problembeschreibung

Im klassischen Retrievalmodell benutzt der Leser Schlagwortkataloge und Indexe, um die Texte zu finden, die er sucht. Auch wenn diese Verzeichnisse mit großem Aufwand erstellt und sorgfältig gepflegt werden, erscheinen sie nach der Analyse von Hawking und Zobel [HZ07] nur wenig hilfreich zu sein. Selten vorkommende Terme machen einen großen Teil der Einträge in einem Vollformenlexikon aus. Während für die einmalige Indexierung die zur Ausführung benötigte Zeit für die Akzeptanz eines Verfahrens wenig relevant ist, ist die Geduld der Nutzer bei der Ausführung von Suchanfragen limitiert. Kurze Antwortzeiten sind ebenso essentiell wie hohe Präzision in der Ergebnisliste der Anfrage. Während bei häufig vorkommenden Suchbegriffen aufgrund der Vielzahl der Fundstellen ein Ranking höhere Priorität hat, liegt die Priorität bei selten vorkommenden Begriffen eher darin, den Begriff und mögliche alternative Schreibweisen möglichst vollständig im Korpus zu finden.

1.3 Ziele

Will man schnell ein Wort in einem umfangreichen Text finden, verwendet man einen Index. Bis heute ist unklar, wie ein optimaler Index für eine unscharfe Suche aussehen sollte. In dieser Untersuchung werden verschiedene Verfahren analysiert und belastbare Kriterien für die Performanceunterschiede der Verfahren erarbeitet.

Die klassische Methode erschließt Texte durch Schlagworte. Das Wiederauffinden eines solchen Textes erfordert, dass der Suchende das Schlagwort trifft.

[2] Ncah eienr Stidue der Cmabirdge Uinertvisy ist es eagl, in wlehcer Reheni floge die Bcuhstbaen in Woeretrn vokrmomen. Huaptschae, der esrte und ltzete Bcuhstbae snid an der rhcitgien Setlle. http://www.zeit.de/2006/07/S_36_Kleintext?page=all

Maschinengestützte Verbesserungen im Sinne einer höheren Toleranz mit Hilfe neuronaler Netze beschreibt [Man01]. Einen anderen Ansatz verfolgen [WAD07] mit einer Beschreibungssprache für Textmuster, die Verallgemeinerungen mit Hilfe von Ontologien erlaubt. Diese Verfahren erlauben eine extreme Reduktion der zu verarbeitenden Datenmengen, weil sie nur Wörter extrahieren, die auf die eine oder andere Art als relevant definiert wurden.

Der Preisverfall bei den Speichermedien zusammen mit kurzen Zugriffszeiten erlaubt heute auf solche Reduktionen zu verzichten und den gesamten Textbestand ungekürzt zu indexieren. In manchen Anwendungsfällen wie der Genomanalyse ist dieses Verfahren unerlässlich.

Ausgehend von regulären Ausdrücken mit Jokerzeichen wie * oder ? wurden zur fehlertoleranten Suche die Algorithmen *agrep* [WM92a, WM92b] oder *tre* [Lau01, Lau06] entwickelt. Eine Übersicht über die Leistungsfähigkeit verschiedener aktueller Algorithmen geben [NR02].

Ein stark vereinfachtes Verfahren einer Trigramm-basierten Suche stellt [Rap97] vor. Eine neue Idee beschreibt [Ess04, Ess05]. Hier werden aus im Text vorhandenen Wörtern mit neun bis zwanzig Zeichen regelbasiert und halbautomatisch „ähnliche“, im Text vorkommende Wörter erzeugt, mit denen die Suche erweitert werden kann (*query extension*).

In dieser Arbeit soll untersucht werden, welche Algorithmen und Verfahren sich für eine Suche nach selten vorkommenden Termen eignen und wie diese Algorithmen zu verbessern sind. Verschiedene ältere und neu entwickelte Verfahren zur unscharfen Indexierung von Volltexten werden auf ihre Leistungsfähigkeit hin untersucht. Die gefundenen Verfahren werden mit einigen kommerziellen Suchmaschinen verglichen.

1.4 Aufbau

Grundlage dieser Untersuchung ist ein nach der Arbeit von [Hei94] neu aufgebautes und optimiertes *SpaCAM* (*Sparsely Coded Associative Memory*), das eine Plattform für die Untersuchung der verschiedenen Codierungen zur unscharfen Suche bietet. Die Komplexität der Codierung spielt für die Suche keine nennenswerte Rolle, weil nur das einmalig notwendige Erstellen des Index Rechenzeiten zwischen weniger als einer Minute und zehn Minuten beanspruchte. Die Antwortzeiten für die unscharfe Volltextsuche über den Gesamtbestand von über 600 MB Fließtext lagen aufgrund der optimierten *SpaCAM*-Technologie stets unter 500 ms.

Die verwandten Verfahren zur Suffixarrayerstellung [KS03] eignen sich ideal zur Teilwortsuche. Zur unscharfen Suche eignen sie sich aber nur nach Modifikation und auch dann nur eingeschränkt.

Im folgenden Kapitel wird ein Überblick über den aktuellen Stand der Forschung gegeben. Kapitel 3 stellt den verwendeten Korpus vor. Im vierten Kapitel werden die verwendeten Algorithmen und Datenstrukturen vorgestellt. Darauf folgt im fünften Kapitel die Darstellung der erzielten Ergebnisse und der Vergleich mit weiteren kommerziellen Suchmaschinen. Im sechsten Kapitel werden die erzielten Ergebnisse bewertet, anschließend folgt noch ein zusammenfassender Abschnitt über Folgerungen und Ausblick.

Das Literaturverzeichnis schließt die Arbeit ab.

Im Anhang sind schließlich weitere Tabellen zu finden, die die im Hauptteil getroffenen Feststellungen belegen.

2 Stand der Forschung

Information Retrieval erscheint heute im Wesentlichen als die Domäne der Internet-Suchmaschinen, allen voran Google™[1]. Zu den Verfahren existieren heute eine große Anzahl an Patenten, allein die Suche nach »Information« AND »Retrieval« erbrachte im Jahr 2010 70 387 Titel in der Liste der amerikanischen Patentdatenbank[2]. Die Verfahren, die die Suchmaschinenbetreiber verwenden, sind teilweise öffentlich bekannt, in wesentlichen Teilen aber auch von diesen entwickelt und als solche Betriebsgeheimnis [Lew05]. Der Übersichtsarbeit von Mandl [Man05] ist die Grafik 2.1 entnommen:

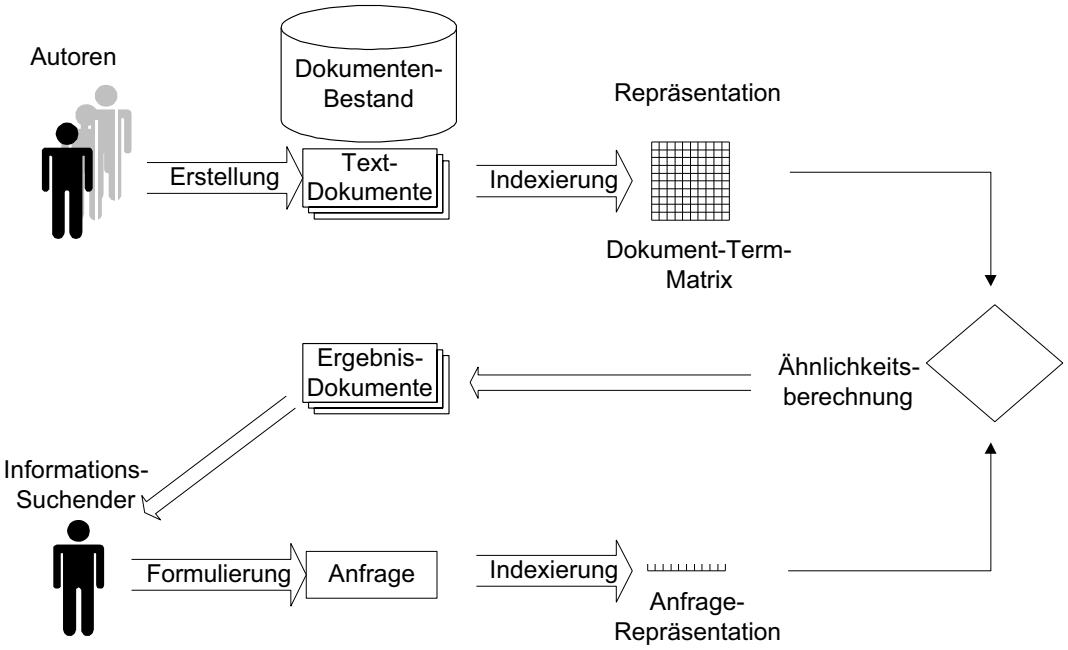


Abbildung 2.1: Der Information Retrieval Prozess

[1] Zur Verwendung des Zeichens ™ siehe http://www.markenbusiness.com/de/news.php?newsid=2182

[2] http://portal.uspto.gov

Eine wesentliche Entwicklung im Verständnis des Information Retrieval ist die Erkenntnis, dass es nicht ausreicht, wenn ein Suchbegriff in der Textmenge wiedergefunden wird, sondern dass die eingegebene Suchanfrage „unscharf“ ist und die gesuchten Texte mehr oder weniger genau zur Anfrage passen. Zadeh [Zad65, KF88, KY95] war der erste, der für diese Unschärfe mit der *fuzzy set theory* eine mathematische Formulierung fand. Diese Toleranz in Bezug auf Ungenauigkeit, Unsicherheit und teilweiser Richtigkeit bezeichnet Zadeh [Zad94, Zad01, Zad02] mit dem Begriff des Soft-Computings. Eine Einführung in die *fuzzy set theory* findet sich bei [Eid02].

Ein anderes mathematisches Modell wird mit dem Begriff des *rough set* beschrieben [PWZ88, Paw91], das im Abschnitt 2.4.1 näher erläutert wird.

2.1 Übersicht

Die ursprüngliche Aufgabe von Information Retrieval besteht in der Suche nach Textdokumenten. Eine klassische Einführung in das Gebiet Information Retrieval bieten Salton und McGill [SM83]. Eine aktuelle Einführung mit Schwerpunkt auf Algorithmen findet sich in [BYRN99]. Eine themenorientierte Einführung liegt mit [FRWH98] vor. Ein aktuelles Lehrbuch, das auch das Retrieval anderer Medien umfasst, wurde von Stock [Sto07] geschrieben. In Deutschland gibt es in der Gesellschaft für Informatik eine Fachgruppe Information Retrieval, auf deren Internetsite eine Übersicht über verschiede Forschungsgruppen zu finden ist[3]. Mandl [Man00] schreibt:

> „Für die Indexierung von großen Textmengen gibt es zwei Verfahren:
>
> - intellektuelle (oder manuelle) Indexierung:
>
> Ein menschlicher Indexierer evaluiert das Objekt, das in die Datenbasis eingefügt wird. Er vergibt dazu inhaltskennzeichnende Schlagwörter, die meist aus einem kontrollierten Vokabular stammen.
>
> - automatische (maschinelle) Indexierung:
>
> Ein Algorithmus analysiert das Objekt und extrahiert aus Textdokumenten alle vorkommenden Wörter (außer besonders häufig vorkommenden Stoppwörtern), reduziert eventuell morphologische Formen und speichert die Vorkommenshäufigkeit der Wörter. Die maschinelle Indexierung versucht nicht, den Text zu verstehen

[3] http://www.uni-hildesheim.de/fgir/index.php?option=com_content&task=view&id=17&Itemid=45

> und semantisch zu modellieren. Stattdessen werden Begriffe gewählt, die einen Text repräsentieren, ohne deren Zusammenhang zu analysieren."

Ob angesichts des Preisverfalls bei Datenträgern heute noch die oben genannten Einschränkungen erforderlich oder sinnvoll sind, oder ob ein Vollformenindex angezeigt ist, soll in dieser Arbeit überprüft werden. Auch Stoppwörter sind in ihrer Bedeutung nicht zu vernachlässigen. Wilbur und Sirotkin [WS92] behaupten:

> These terms, such as "the", "if", "but", "and", etc., have a grammatical function but reveal nothing about the content of documents. By removing such terms one can save space in indices and also, in some cases, may hope to improve retrieval.

Die Terme „be", „not", „or", und „to" lassen sich durchaus als Stoppwörter ansehen. Eliminiert man sie, dann kann das Hamletzitat „to be or not to be" nicht mehr gefunden werden [Sto07].

Die Literatur zum Thema ist mittlerweile unüberschaubar. Eine kurze Übersicht zum Thema gibt [Sin01]. In [TD04] wird beispielhaft gezeigt, wie verschiedene Algorithmen für ein optimales Ergebnis kombiniert werden.

2.2 Retrievalmodelle

Das manuelle und das maschinelle Verfahren zur Indexierung unterscheiden sich nicht nur in dem Arbeitsaufwand für denjenigen, der den Index erstellt. Das manuelle Verfahren benutzt in der Regel einen Schlagwortkatalog mit einer relativ geringen Anzahl von Schlagwörtern, die den Texten zugeordnet werden. Um eine Zahl anzugeben: Das Literaturverwaltungsprogramm Bibliographix[4] empfiehlt etwa 50 Schlagworte für ein wissenschaftliches Spezialgebiet. Diese Schlagworte müssen nicht in den Ursprungstexten enthalten sein, weil sie ja intellektuell vergeben wurden.

Maschinelle Verfahren herkömmlicher Art versuchen ebenfalls, sinnträchtige Schlagworte zu erfassen. Deshalb werden Stoppwörter entfernt und gegebenenfalls morphologische Formen auf Grundformen reduziert. Außerdem können verschiedene Maßzahlen berechnet werden. Beliebt sind

- die Angabe der *term frequency, tf*,
- die Angabe der inversen Dokument-Häufigkeit *IDF* und
- die Angabe der *tf-idf*

[4] www.bibliographix.de

Die *term frequency* ist definiert als

$$tf_{t,d} = \frac{n_{t,d}}{\sum_k n_{k,d}}, \tag{2.1}$$

wobei $n_{t,d}$ die Anzahl der Vorkommen des betrachteten Terms t im Dokument d ist und der Nenner die Anzahl aller Terme im Dokument d bezeichnet.

Die *IDF* definieren [BYRN99] als

$$IDF_t = \log \frac{n}{n_t} \tag{2.2}$$

Hierbei bezeichnet n die Anzahl aller Dokumente und n_t die Anzahl all derjenigen Dokumente, die den Term t enthalten. Die Wahl der Basis des Logarithmus ist dabei ohne Bedeutung [MRS08].

Wenn also der Term »Mensch« in 300 von 1000 Dokumenten vorkommt, ergibt sich

$$IDF_{Mensch} = \log_2 \frac{1000}{300} \approx 1{,}7 \tag{2.3}$$

und ein Term, der in jedem Dokument vorkommt, ergibt eine *IDF* von 0. Damit ist die *IDF* ein Hinweis auf den Filtereffekt eines Terms, ein Hinweis auf die Nutzbarkeit eines Terms, um Dokumente auszuwählen.

Beide Maße können miteinander multipliziert werden zur *tf-idf*:

$$tf\text{-}idf_{t,d} = tf_{t,d} \cdot IDF_t \tag{2.4}$$

Damit ordnet $tf\text{-}idf_{t,d}$ einem Term t in einem Dokument d ein Gewicht zu, welches

- am größten ist, wenn t häufig in einer kleinen Anzahl von Dokumenten vorkommt,
- kleiner ist, wenn entweder der Term seltener im Dokument vorkommt oder aber in vielen Dokumenten vorkommt, und schließlich
- am geringsten ist, wenn der Term in fast allen Dokumenten vorkommt.

Diese und davon abgeleitete Maße werden zur Berechnung von Ähnlichkeiten und zur Bestimmung relevanter Dokumente im Vektorraummodell (Abschnitt 2.2.3) und ähnlicher Modelle (Abschnitt 2.2.5) verwendet.

2.2.1 Definitionen

Um für begriffliche Klarheit zu sorgen, werden an dieser Stelle wichtige Begriffe definiert:

Begriff Bedeutungsinhalt eines Wortes, Vorstellungsinhalt [Wah85].
Auch „Denkeinheit, die aus einer Menge von Gegenständen unter Ermittlung der diesen Gegenständen gemeinsamen Eigenschaften mittels Abstraktion gebildet wird“ (DIN 2342).

Wort kleinste selbständige sprachliche Einheit mit eigener Bedeutung oder Funktion[5], auch: zusammenhängende Zeichenfolge.

Term sinnvoller Ausdruck, auch mehrere Wörter umfassend.

Konzept gleichbedeutend mit dem in WordNet verwendeten Begriff Synset, der eine Gruppe von Synonymen bezeichnet, die im Sinne des Information Retrievals untereinander austauschbar sind.[6]

Lemma Zitier- oder Grundform eines Wortes für den Eintrag in ein Wörterbuch

Morphem kleinste bedeutungstragende Einheit der Sprache auf der Inhalts- und Formebene im Sprachsystem. Es lässt sich auch als kleinste semantisch interpretierbare Konstituente eines Wortes. bezeichnen.[7]

Lexem Menge der Formen eines Lemmas.

Token Eine Zeichenkette, die im Originaltext steht und von solchen Zeichen begrenzt wird, die in der Liste der Worttrenner in Tabelle 7.2 aufgeführt sind.

2.2.2 Boolesches Modell

Das Boolesche Modell ist das älteste und einfachste Modell. Es basiert auf der elementaren Mengenlehre und ordnet die Dokumente für jeden Indexterm in zwei Mengen: Solche, die mit einem Term indexiert sind und solche, die das nicht sind. Aus diesen Mengen können dann mit den Operationen Schnittmenge, Vereinigungsmenge und Komplementärmenge die gewünschten Dokumente auch mit komplizierteren Booleschen Ausdrücken herausgefiltert werden.

Im Gegensatz zu den im folgenden beschriebenen Modellen ist im Booleschen Modell die Beschreibung der Suchatome wichtig. Unter Suchatom [Sto07] wird die kleinste Einheit verstanden, die im Index verzeichnet ist. Das Suchatom kann über einen regulären Ausdruck zu einer Menge von Suchatomen erweitert werden.

[5] http://www.canoo.net/services/Controller?input=wort&service=canooNet&lookup=caseSensitive

[6] http://en.wikipedia.org/wiki/Synset

[7] http://de.wikipedia.org/wiki/Morphem#Wort

Der reguläre Ausdruck ist eine Zeichenkette zur Beschreibung einer Menge von Zeichenketten. Zur Beschreibung werden als Metazeichen häufig Zeichen aus dem Zeichenvorrat {`[ ] ( ) { } | ? + * ^ $ \ .`} verwendet. Reguläre Ausdrücke werden näher erläutert in Abschnitt 3.3.2 auf Seite 60.

Mit regulären Ausdrücken kann man nach Wörtern suchen, die in unterschiedlicher Weise geschrieben werden: (Ph | F)otogra(ph | f)ie, Känguruh?, internationali[sz]ation, Schwein[se]braten, Schifff?ahrt, (E | Ae | Ä)ther, Mo[çcsz]ambi(k | que). Wie das letzte Wort zeigt, darf man dabei umso großzügiger auch falsche Schreibungen mit zulassen, je unwahrscheinlicher es ist, andere gültige Wörter zu treffen. Auch eine gleichzeitige Suche nach mehreren Formen eines Wortes unabhängig von Beugungsendungen lässt sich so erreichen: eine?[mnrs]? geschenkte?[mnrs]? G[aä]ule?s?[8]

2.2.2.1 Abstandsoperator

Eine Erweiterung des Booleschen Modells besteht in der Einführung der Abstandsoperatoren NEAR und ADJ, auch Proximity-Suche genannt. Damit soll ausgedrückt werden, dass die Suchatome im Dokument in räumlicher Nähe stehen sollen. Diese Nähe kann sich auf eine bestimmte Anzahl von Zeichen oder Wörtern beziehen. Je nach Abfragesprache lässt sich bestimmen, ob je zwei Suchatome eine bestimmte Reihenfolge aufweisen sollen oder ob die logischen Operatoren nur innerhalb einer begrenzten Umgebung des ersten Suchterms angewandt werden sollen. Weitere Operatoren können bestimmen, dass die Begriffe in einem Satz oder in einem Absatz stehen müssen (SENTENCE und PARAGRAPH).

2.2.2.2 Unscharfe Suche

Datenbanken speichern bisher in der Regel exakt definierte Daten und erwarten ebenso exakt definierte Abfragen. Mit regulären Ausdrücken konnte die Anzahl der abgefragten Suchatome auf verhältnismäßig einfache Weise erweitert werden. Die unscharfe oder Fuzzy-Suche erweitert die zweiwertige Boolesche Logik zu einem Kontinuum, in dem unscharfe Werte wie kurz oder warm statt der numerischen Angaben der Länge oder der Temperatur modelliert werden können. Dazu existiert bereits eine kommerzielle Anwendung.[9] Durch die Verwendung eines Synonymwörterbuches kann die Anfrage auf ähnliche Begriffe erweitert werden.

[8] http://www.lrz-muenchen.de/services/schulung/unterlagen/regul
[9] http://www.hduw.de/Fuzzy/FA-VF-DB_Produkt.pdf

Schließlich kann die Suche algorithmisch auf ungenaue oder fehlerhafte Schreibweisen ausgedehnt werden. Näheres ist beschrieben im Abschnitt 2.4 ab Seite 42 über fehlertolerantes Retrieval.

2.2.3 Vektorraum-Modell

Die meisten anderen Modelle zum Information Retrieval lassen sich verschiedenen Varianten des Vektorraum-Modells zuordnen [Sal71, Sal86, Sal89]. Prinzipiell wird durch jeden Indexterm eine Dimension in einem hochdimensionalen Vektorraum aufgespannt. Mit der inversen Dokument-Häufigkeit *IDF* oder einer ähnlichen Maßzahl lässt sich für jeden Term der Abstand jedes Dokumentes vom Nullpunkt festlegen. Jedes Dokument bildet also einen Punkt in dem aufgespannten hochdimensionalen Raum, der durch einen Vektor bezeichnet werden kann.

Ebenso kann eine Anfrage als Punkt in diesem Vektorraum dargestellt werden. Dokumente, die zu der Anfrage passen, liegen dann in räumlicher Nähe zur Anfrage. Die verwendeten Vektoren sind dünn (engl. *sparse*) besetzt, weil jedes Dokument nur wenige Terme aus der Menge der indizierten Terme enthält. Weil die Vektoren aber umgekehrt sehr viele Dimensionen haben (nämlich die Anzahl aller Indexterme), entsteht in der naiven Implementierung von Algorithmen zur Verarbeitung das Problem eines extrem hohen Speicherbedarfs.

Für die Lösung dieses Problems gibt es zwei unterschiedliche Ansätze: Davis [Dav06] beschreibt effiziente Methoden zur Speicherung und Verarbeitung von dünn besetzten Matrizen. Unter http://www.cise.ufl.edu/research/sparse/ stehen viele unter der GNU GPL lizenzierte Algorithmen zur Verfügung.

Ein anderer Ansatz besteht darin, die Ausgangsdaten verlustbehaftet so weit zu komprimieren, bis sie handhabbar werden. Dazu gibt es wieder verschiedene Möglichkeiten, einerseits intellektuelle, wie das im folgenden beschriebene Verfahren mit Kontextvektoren, andererseits mathematische, die eher statistisch arbeiten.

2.2.3.1 Kontext-Vektor

Das zunächst von Gallant [Gal91] beschriebene Verfahren verwendet eine intellektuelle Zuordnung der in den Dokumenten identifizierten Terme auf wenige Basisterme, wie Abbildung 2.2 (entnommen aus [Man01]) zeigt:

Hier wird der Term Mosaik mit den Basiseigenschaften fest, künstlerisch und statisch konnotiert. Die intellektuelle Zuordnung ist sehr aufwändig, deshalb wurden Verfahren gesucht, diesen Schritt zu automatisieren. Eine ganz andere Definition lautet deshalb: Ein Kontextvektor ist eine Tabelle, in der pro Lexem (oder pro Lesart, Wortform, ...) alle möglichen Kontextwörter und ihre Frequenz aufgelistet sind. Im Patent von Liddy et al. [LPSLY99] wird beschrieben, wie durch

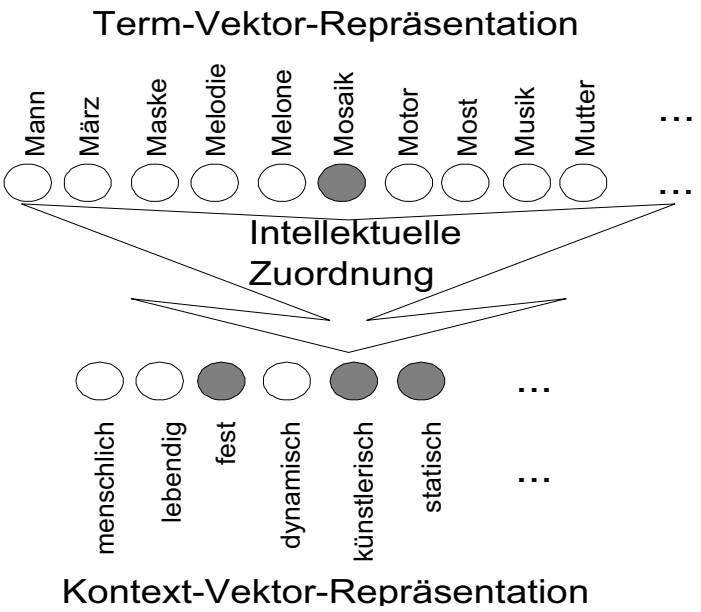


Abbildung 2.2: Kontext-Vektor

das automatische Nachschlagen in Wörterbüchern mit diesem Verfahren auch lexikalische Ambiguitäten aufgelöst werden können. Caid und Carleton [CC03] beschreiben eine Automatisierung dieses Verfahrens unter dem Namen *MatchPlus*. Die Abbildung 2.3 wurde dieser Arbeit zur Illustration entnommen.

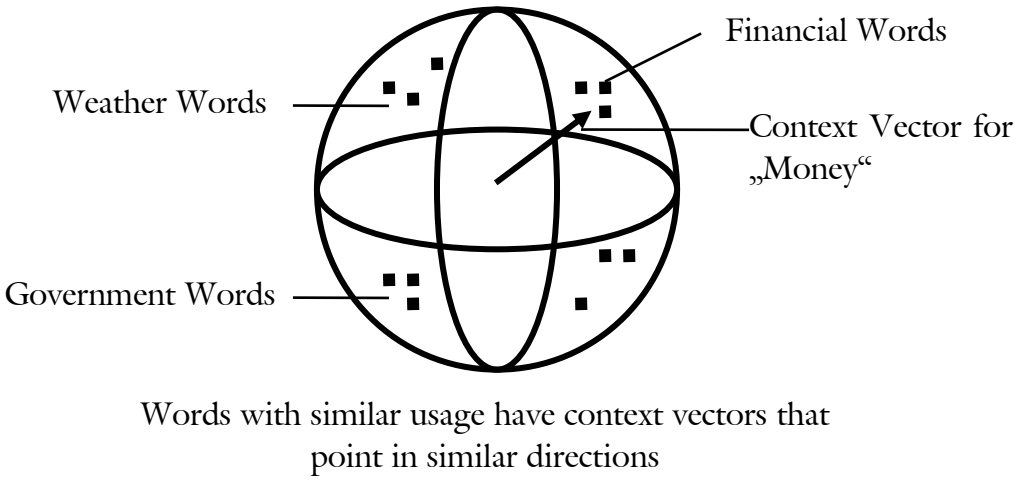


Abbildung 2.3: Beispiel für einen Kontext-Vektor

Der Algorithmus ist selbstlernend unter vorgegebenen Randbedingungen. So werden Stoppwörter entfernt. Zusammen gehörige Wörter wie »New York« werden als Einheit betrachtet und die Stammformen der Wörter gebildet. Anschließend kann der Nutzer ein Wort wählen und der Algorithmus liefert eine Liste all der Wörter, die im Zusammenhang mit diesem Wort häufig erscheinen.

Gleiches gilt für Dokumente: Aus den Termen eines Dokumentes wird ein Kontext-Vektor gebildet, der das Dokument beschreibt. Für das Text Retrieval kann der Nutzer einen Satz eingeben, aus dem mit dem gleichen Verfahren ein Kontextvektor berechnet wird. Die Ausgabe des Systems ist dann eine Liste der Dokumente mit einem ähnlichen Kontextvektor.

2.2.4 Cluster-Verfahren

Ausgehend von einem hochdimensionalen Vektorraum wünscht man sich Methoden, um sich in diesem Vektorraum zurechtzufinden. Cluster-Verfahren ermöglichen es, in diesem Vektorraum Gebiete zu finden, in denen Vektoren häufiger liegen, als es einer zufälligen Verteilung entsprechen würde. Ein Cluster ist eine Anhäufung von Punkten, in diesem Fall von Vektoren in einem n-dimensionalen Vektorraum.

Cluster-Verfahren können auf verschiedene Weisen eingeteilt werden. Abbildung 2.4 zeigt eine Einteilung.[10]

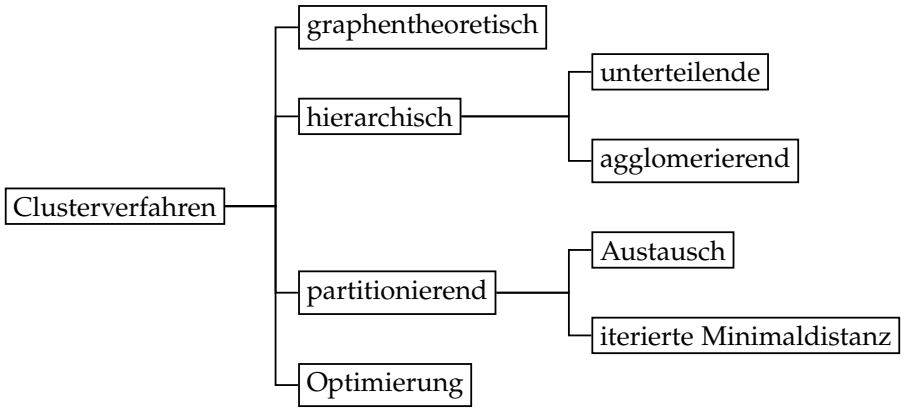


Abbildung 2.4: Einteilung der Clusterverfahren

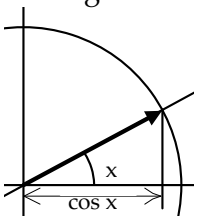
Um Anhäufungen zu beschreiben, ist ein Abstandsmaß erforderlich, mit dem die Abstände zwischen den Punkten im Vektorraum beschrieben werden. Viele solcher Abstandsmaße oder Distanzfunktionen wurden beschrieben.[11] Als Beispiele seien

[10] http://de.wikipedia.org/wiki/Clusteranalyse
[11] http://de.wikipedia.org/wiki/Distanzfunktion

genannt die L_P-Distanz oder Minkowski-Metrik [Min11] und die Cosinus-Distanz. Zunächst die Minkowski-Metrik:

$$L_P(\mathbf{v}_1,\mathbf{v}_2) = \sqrt[P]{\sum_{i=1}^{n} |\mathbf{v}_{1i} - \mathbf{v}_{2i}|^P}, \quad \mathbf{v}_1,\mathbf{v}_2 \in \mathbb{R}^n \tag{2.5}$$

Mit $P = 1$ erhält man die City-Block- oder Manhattan-Distanz, $P = 2$ ergibt den Euklidischen Abstand und für $\lim P \to \infty$ erhält man die Maximum- oder Chebyshev-Distanz.
Ein anderes Distanzmaß ist die Cosinus-Distanz. Hier werden nicht die Punkte selbst, sondern nur die Winkel zwischen den Vektoren verglichen. Als Abstandsmaß hat sich statt des Winkels zwischen den Vektoren $\mathbf{v}_1,\mathbf{v}_2$ der Cosinus x des Winkels zwischen den Vektoren durchgesetzt entsprechend Gleichung 2.6. [SB88]



Cosinus x

$$\cos x = \frac{\mathbf{v_1} \cdot \mathbf{v_2}}{\|\mathbf{v_1}\| \, \|\mathbf{v_2}\|} \tag{2.6}$$

2.2.4.1 *k-means*-Algorithmus

Der Untersucher gibt die Anzahl k der gewünschten Klassen vor, die ungefähren Mittelpunkte der Cluster im Merkmalsraum und die Anzahl der Iterationen. Der Algorithmus berechnet die Abstände der Vektoren $\mathbf{v}$ zu den Mittelpunkten und ordnet die Vektoren den nächstgelegenen Mittelpunkten zu. In einem iterativen Verfahren werden die Mittelpunkte der Cluster neu berechnet und so das Ergebnis optimiert. Werden in einem Schritt Cluster nicht mit Vektoren besetzt, dann entfallen sie im nächsten. Das Verfahren ist schnell, aber stark von der Separierbarkeit der Cluster und der ursprünglichen Verteilung der geschätzten Mittelpunkte abhängig. Werden keine Vorgaben gemacht, können diese auch vom Programm geschätzt werden.

2.2.4.2 ISODATA-Algorithmus

Der ISODATA-Algorithmus entspricht dem *k-means*-Algorithmus. Allerdings ist hier die Anzahl der Cluster nicht festgelegt, sondern der Algorithmus kann Cluster teilen, wenn die zugehörige Punktwolke besser durch zwei Zentren zu beschreiben ist. Cluster werden zusammengelegt, wenn der Abstand der Zentren und die Verteilung der Punktwolken dies nahelegen und Cluster werden gelöscht, wenn sie mit zu wenigen Punkten besetzt sind.

Beide Verfahren gehören zu den agglomerierenden Verfahren, sie gehen von den einzelnen Vektoren aus und versuchen diese zusammen zu fassen.

Ein unterteilendes Verfahren ist die *Support Vector Machine, SVM*:

2.2.4.3 Support Vector Machine

Die *Support Vector Machine, SVM* [Vap95, Vap98, Bur98, STC00, Kec01] ist ein anderes mathematisches Verfahren, um in einem Vektorraum verteilte Objekte zu Klassen zusammenzufassen. Das Verfahren ist allgemein einsetzbar zur Mustererkennung und es existieren etliche auch frei verfügbare Programme und Softwarebibliotheken, die das Verfahren implementieren. Das Verfahren gehört zu den lernenden Klassifikationsverfahren und benötigt deshalb einen Trainingsdatensatz, um die Grenzen zwischen den verschiedenen Klassen bestimmen zu können. Aufgrund der erforderlichen Matrizenrechnungen in hochdimensionalen Vektorräumen ist das Verfahren sehr ressourcenintensiv. In neuester Zeit konnten signifikante Beschleunigungen durch den Einsatz von in den Grafikkarten von Desktopsystemen enthaltenen Grafikprozessoren erzielt werden. [CSK08]

Der Einsatz einer *Support Vector Machine* zur Textklassifikation wird von Joachims [Joa98] beschrieben. Um Dokumente zu Klassen zusammenfassen zu können, werden sie analysiert. Neben dem klassischen Ansatz der Verwendung von Metadaten wie bibliographischen Daten und Schlagwörtern sollen in einem automatischen Verfahren relevante Informationen aus dem Text selbst extrahiert werden. Aus den Folgen von Zeichenketten werden deshalb Terme extrahiert, von denen man annimmt, dass sie für den Inhalt des Dokuments relevant sind. Yang [YP97] untersuchte verschiedene Methoden, um aus den in einem Dokument vorhandenen Termen die für das Dokument relevanten Terme herauszufinden.

Die als relevant befundenen Terme aus allen Dokumenten zusammen bilden gewissermaßen einen Schlagwortkatalog. Anders als im Kontextvektor-Modell spannen hier die Terme selbst (*binary independent model* [Fuh95, MRS08]) oder von ihnen abgeleitete Konzepte (*support vector machines* [CST00]) zusammen einen hochdimensionalen Vektorraum auf.

In [For03] wird herausgestellt, dass die *SVM* für die Textklassifikation herausragend geeignet ist und es werden weitere Methoden zur radikalen Reduktion des Vektorraums untersucht. [BLJ04] beschreiben eine Weiterentwicklung, die sie *support kernel machine* (*SKM*) nennen. Es besteht allerdings ein Wettstreit zwischen den verschiedenen Verfahren und häufig bewirkt eine Änderung der eingestellten Parameter einen Vorsprung des einen über das andere Verfahren. So zeigen [GM04], dass ohne eine radikalen Reduktion des Vektorraums (d. h. auf 4 bis 40

Vektoren) das Entscheidungsbaum-basierte Verfahren C4.5 in speziellen Fällen besser abschneidet als die *Support Vector Machine* oder auch als *k-means*-Verfahren.

Die extrahierten Terme werden gewichtet, z. B. mit der *tf-idf* gemäß ihrer Häufigkeit im gesamten Korpus und entsprechend ihrer Bedeutung für das spezielle Dokument (vgl. Gleichung (2.4) auf Seite 20).

In diesen Vektorraum werden die Dokumente entsprechend ihrer Koordinaten eingeordnet. So entstehen im Idealfall linear trennbarer Klassen klar unterscheidbare Punktwolken in einem hochdimensionalen Vektorraum. Der Idealfall tritt leider eher seltener ein, deshalb wurden Methoden entwickelt, um auch verschlungene Punktwolken noch unterscheiden zu können. Schließlich werden die Suchterme aus der Anfrage ebenfalls geeignet gewichtet, um mit den gefundenen Klassenvektoren verglichen zu werden.

Diese Wichtung erfolgt durch ein lernendes Verfahren. Dazu stellt [SB88] fest:

> "Without sophisticated term weighting systems, acceptable retrieval output is not obtainable either for initial searches or the feedback search iterations."

Der bisher schnellste Lernalgorithmus für die *SVM* ist der in [Pla99] beschriebene *minimal optimization algorithm*. Eine Implementierung der *SVM* unter Verwendung des *minimal optimization algorithm* ist bei [Mak00] zu finden.

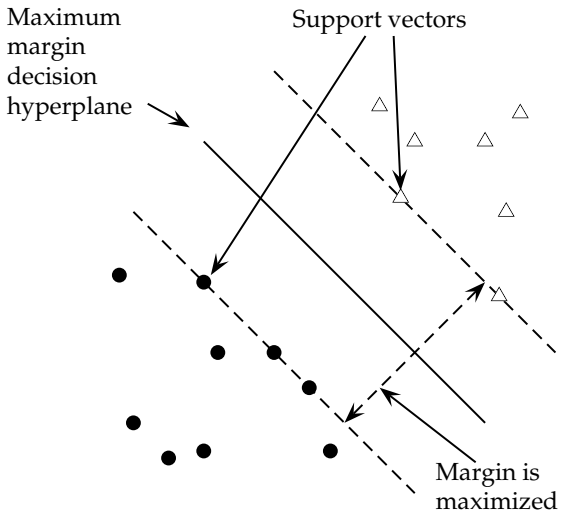


Abbildung 2.5: Beispiel für die Wahl der Stützvektoren einer SVM. Die Hyperebene wird so ausgewählt, dass die Summe der Quadrate der Abstände der nächstliegenden Trainingsvektoren zur Hyperebene maximal wird.[MRS08]

Die Support Vector Machine löst das oben angesprochene Problem der fehlenden linearen Unterscheidbarkeit. Dazu wird der Vektorraum der Eingabe durch eine nichtlineare Funktion, einen Kernel, in einen höherdimensionalen *feature-space* transformiert. Allerdings müssen Berechnungen nicht in diesem höherdimensionalen Raum durchgeführt werden, sondern können bei geeigneter Wahl des Kernels im Vektorraum der Eingabe durchgeführt werden.

Die Klassifikation mit Hilfe einer SVM läuft in folgenden Phasen ab: Zunächst bedarf es eines möglichst kleinen und möglichst guten Trainingsdatensatzes, der die Stützvektoren liefert, nach der die SVM benannt wurde.

Durch die Stützvektoren wird die Hyperebene bestimmt, die die Trainingsvektoren optimal in zwei Klassen einteilt.

Hyperebenen können nicht verbogen werden. Durch die Transformation in einen höherdimensionalen – im Zweifelsfall unendlichdimensionalen – Vektorraum lassen sich auf jeden Fall die Vektoren durch eine Hyperebene linear trennen.

Die optimale Trennung wird durch das Training bestimmt. Der Trainingsdatensatz enthält eine Menge von Paaren $\{(\mathbf{v}_1, y_1), ..., (\mathbf{v}_m, y_m) \mid \mathbf{v}_i \in \mathcal{V}, y_i \in \{-1,1\}\}$ von einem Vektor zusammen mit einer Entscheidung, ob dieser Vektor zur zu lernenden Klasse gehört oder nicht.

Optimal bedeutet in diesem Fall, dass unter den vielen möglichen trennenden Hyperebenen diejenige ausgesucht wird, für die die Summe der Quadrate der Abstände der nächstliegenden Trainingsvektoren zur Hyperebene maximal wird.

Für den Fall, dass sich auf diese Weise einzelne Trainingsvektoren nicht korrekt zuordnen lassen, können für diese Trainingsvektoren Schlupfvariablen eingeführt werden, die eine etwas unschärfere Anpassung ermöglichen.

Für den oben angesprochenen Fall, dass sich die Trainingsdaten nicht linear separieren lassen, können die Trainingsvektoren $\mathbf{v}$ durch eine Funktion

$$\phi : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, \mathbf{v} \mapsto \phi(\mathbf{v}) \tag{2.7}$$

mit $d_1 < d_2$ in einen höherdimensionalen Raum überführt werden, in dem eine solche Trennung möglich ist.

Weil die Klassifikationsregel statt mit Datenpunkten $\mathbf{v}$ mit Skalarprodukten $\langle \mathbf{v}_i, \mathbf{v}_j \rangle$ der Datenpunkte rechnet, werden auch im höherdimensionalen Raum statt der Datenpunkte Skalarprodukte $\langle \phi(\mathbf{v}_i), \phi(\mathbf{v}_j) \rangle$ verwendet, die wiederum durch eine positiv definite Funktion $k(\mathbf{v}_i, \mathbf{v}_j) = \langle \phi(\mathbf{v}_i), \phi(\mathbf{v}_j) \rangle$ ersetzt werden können.

Durch diesen sogenannten Kerneltrick kann eine SVM eine gute Klassifikationsrate und Rechenleistung trotz kompliziert strukturierter Daten erreichen.[12]

[12] Siehe auch: http://www.svms.org/tutorials/

2.2.4.4 Latent Semantic Indexing

Das Vektorraummodell wird in drei wesentlichen Punkten verbessert: Zunächst ist bekannt, dass verschiedene Personen den gleichen Sachverhalt mit unterschiedlichen Termen bezeichnen [FLGD83], ebenso sind sich menschliche Indexierer uneins [TB74]. Schließlich benutzen sowohl unerfahrene Benutzer [Lil54, Bat86] als auch Experten [Fid85] jeweils unterschiedliche Suchterme im Text Retrieval.

Zu diesen Phänomenen, die man allgemein unter dem Begriff der Synonymie subsumieren kann, kommt das Phänomen der Polysemie und der Homographie, dass nämlich Texte unterschiedliche Wörter mit gleicher Schreibweise enthalten. Schließlich sind im naiven Vektorraummodell je zwei Terme und die durch sie aufgespannten Dimensionen zueinander orthogonal. Sie werden als unabhängig betrachtet. Die Terme »Sonnabend« und »Samstag« haben in diesem Modell nichts miteinander zu tun. Um diesen Problemen abzuhelfen, wurde das als Latent Semantic Indexing (*LSI*) bezeichnete Verfahren entwickelt [DDF$^+$88, DDF$^+$90, LFL98]. Wesentliche Merkmale sind der Verzicht auf eine Reduktion der Wortformen, andererseits aber die Berücksichtigung der Dokumente im zu erstellenden Vektorraum. Dazu wird aus allen vorkommenden Termen und Dokumenten eine Term-Dokument-Matrix gebildet. Die [DDF$^+$90] entnommene Abbildung 2.6 illustriert das Verfahren.

Es gehen all diejenigen Terme in die weitere Bearbeitung ein, die mindestens zweimal in der Dokumentsammlung vorkommen (in Abbildung 2.6 *kursiv* gesetzt). Im ersten Schritt wird die absolute Dokumenthäufigkeit notiert, im nächsten Schritt werden die Dokumentgewichte entsprechend einer Formel $wdf \cdot IDF$ bestimmt. *wdf* bezeichnet hier die *word document frequency* und *IDF* die *inverse document frequency*. Beide können nach unterschiedlichen Formeln berechnet werden und beeinflussen die Güte des Retrievals.

Durch die Singulärwertzerlegung (Abk.: *SVD* für *Singular Value Decomposition*) werden nun die Beziehungen zwischen den Termen und Dokumenten analysiert: Jede (rechteckige) Matrix $\mathbf{X}$ lässt sich darstellen als das Produkt

$$\mathbf{X} = \mathbf{WSP}^T \tag{2.8}$$

einer Matrix $\mathbf{W}$ mit orthonormalen Spaltenvektoren, einer (quadratischen) Diagonalmatrix $\mathbf{S}$, die die Singulärwerte als nur positive, nach Größe geordnete Zahlen enthält und der Transponierten $\mathbf{P}^T$ einer Matrix $\mathbf{P}$ mit orthonormalen Spaltenvektoren.

Geometrisch-anschaulich lässt sich dieses Verfahren als eine Folge von Abbildungen beschreiben: die orthonormalen Abbildungen drehen oder spiegeln den Vektorraum ohne Abstände oder Winkel zu verändern und die Diagonalmatrix

Titles:
c1: *Human* machine *interface* for Lab ABC *computer* applications
c2: A *survey* of *user* opinion of *computer system response time*
c3: The *EPS user interface* management *system*
c4: *System* and *human system* engineering testing of *EPS*
c5: Relation of *user*-perceived *response time* to error measurement

m1: The generation of random, binary, unordered *trees*
m2: The intersection *graph* of paths in *trees*
m3: *Graph minors* IV: Widths of *trees* and well-quasi-ordering
m4: *Graph minors*: A *survey*

Terms	Documents								
	c1	c2	c3	c4	c5	m1	m2	m3	m4
human	1	0	0	1	0	0	0	0	0
interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
user	0	1	1	0	1	0	0	0	0
system	0	1	1	2	0	0	0	0	0
response	0	1	0	0	1	0	0	0	0
time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
survey	0	1	0	0	0	0	0	0	1
trees	0	0	0	0	0	1	1	1	0
graph	0	0	0	0	0	0	1	1	1
minors	0	0	0	0	0	0	0	1	1

A sample dataset consisting of the titles of 9 technical memoranda. Terms occurring in more than one title are italicized. There are two classes of documents - five about human-computer interaction (c1-c5) and four about graphs (m1-m4). This dataset can be described by means of a term by document matrix where each cell entry indicates the frequency with which a term occurs in a document.

Abbildung 2.6: Term-Dokument-Matrix

schrumpft oder expandiert die einzelnen Dimensionen und bewirkt so, dass Ähnlichkeiten durch Nachbarschaftsbeziehungen räumlich ausgedrückt werden.

Weil die Singulärwerte nach ihrer Größe geordnet sind und kleine Werte nur einen untergeordneten Einfluss haben, lässt sich diese Dekomposition auch zur Datenreduktion verwenden. Die Qualität des Retrievals steigt zunächst mit der

Anzahl der Dimensionen, um nach einem Optimum wieder abzusinken. Wie die Ermittlung der Dokumentengewichte ergibt auch die Wahl der Anzahl der verwendeten Dimensionen einen weiteren Freiheitsgrad zur Optimierung des Retrievalergebnisses bei Verwendung des *Latent Semantic Indexing*. Papadimitriou et al. [PTRV98] geben Hinweise auf eine Automatisierung der besten Wahl.

In der Arbeit von Deerwester et al. [DDF+90] wurden 2000 Dokumente (D) mit 7000 Indextermen (T) in einem 100-dimensionalen Raum (k) untersucht. Die Rechenkomplexität lag dabei bei $O((D+T)^2 \cdot k^2)$. Christy [CT06] bescheinigt dem von ihm untersuchten Verfahren bei einem allerdings sehr kleinen Korpus ein besseres Verhalten als bei der *SVM*. Am Rande sei bemerkt, dass das *Latent Semantic Indexing* auch ein Modell für die Struktur des menschlichen Gedächtnisses bilden kann, wie [Fol91] belegt.

2.2.5 Probabilistisches Modell

Im Probabilistischen Retrievalmodell [MK60] wird ein Wahrscheinlichkeitswert ermittelt, ob ein Dokument für eine Suchanfrage relevant ist [SJWR00a, MRS08]. Der Ähnlichkeitswert einer Suchanfrage zu einem Dokument ist dabei von der Häufigkeit der Suchbegriffe im Dokument abhängig. Dies ist eine Analogie zur *fuzzy*-Suche. Die angestrebte Unschärfe bei der Suche wird durch die Berechnung von Wahrscheinlichkeiten modelliert.

Ausgangsbasis ist wieder die Berechnung der inversen Dokument-Häufigkeit *IDF* (vgl. Gleichung (2.2) auf Seite 20).

Wir bestimmen eine Kontingenztabelle:

	relevant	nicht relevant	
Term enthalten	r	$n-r$	n
Term nicht enthalten	$R-r$	$N-n-R+r$	$N-n$
	R	$N-R$	N

Zunächst wird eine Term-Wichtungsfunktion *RW* eingeführt. Für jeden Term t sei n_t die Anzahl der Dokumente in einem Korpus mit n Dokumenten, in denen t vorkommt und r_t die Anzahl der relevanten Dokumente, in denen t vorkommt.

Solange keine Informationen über die Relevanz vorliegen müssen wir die Zahl der relevanten Dokumente R vorläufig schätzen.

R ist zunächst unbekannt und wird iterativ bestimmt, indem nach einem Retrieval die Relevanz der gefundenen Dokumente bewertet wird. Bei einem weiteren Suchlauf können dann die Suchterme t mit einer Gewichtungsfunktion RW_t gewichtet werden.

$$RW_t = \log \frac{(r_t + 0{,}5)(N - n_t - R + r_t + 0{,}5)}{(R - r_t + 0{,}5)(n_t - r_t + 0{,}5)} \tag{2.9}$$

Die Addition von 0,5 zu den jeweiligen Anzahlen von Dokumenten ist keine theoretische Erfordernis, sondern ein von den Bedürfnissen der Praxis initiiertes Vorgehen, um die so berechneten Werte in der Praxis robuster zu machen. Verschiedene Modifikationen betreffen die Art und Weise, in der weitere Verfahren eingeführt werden, um die relevanten Terme in einem Dokument zu finden oder um Stoppwörter auszuschließen. Häufig beinhalten diese Modifikationen weitere freie Parameter, die für ein Feintuning verwendet werden, um die Qualität des Retrievals zu verbessern.

2.2.5.1 Modell der *divergence from randomness*

Eine Variante respektive Erweiterung des probabilistischen Modells ist das in Glasgow erforschte Modell der *divergence from randomness* [AVR02]. Dieses Modell beschreibt einen nicht-parametrischen Ansatz, d. h. es kommt ohne eine Lernphase und ohne zusätzliche, wie in Gleichung 2.9 beispielhaft verwendete Parameter aus. Allerdings verlangt es die Angabe einer Wahrscheinlichkeitsverteilung und unterschiedliche Wahrscheinlichkeitsverteilungen führen zu unterschiedlichen Retrievalmodellen [ACR01]. Eine aktuelle Implementierung ist als Open Source Software unter dem Namen Terrier (TERabyte RetrIEveR) verfügbar[13]. Die grundlegende Idee besteht darin, den Unterschied in der Termhäufigkeit innerhalb eines Dokumentes mit der Termhäufigkeit im Korpus heranzuziehen, um die Relevanz eines Dokuments für eine gegebene Suchanfrage zu beurteilen.

> "The more the divergence of the within-document term-frequency from its frequency within the collection, the more the information carried by the word t in the document d. In other words the term-weight is inversely related to the probability of term-frequency within the document d obtained by a model M of randomness:"[Har74][14]

$$\text{weight}(t|d) \propto -\log \text{Prob}_M(t \in d \,|\, \text{Collection}) \tag{2.10}$$

[13] http://ir.dcs.gla.ac.uk/terrier/index.html
[14] http://ir.dcs.gla.ac.uk/terrier/doc/dfr_description.html

Amati et al. schreiben dazu [ACR01]:

> The fundamental weighting Formula is the product of two *information content* functions:
>
> $$w = Inf_1 \cdot Inf_2 \tag{2.11}$$
>
> Both Inf_1 and Inf_2 are decreasing functions of two probabilities P_1 and P_2, respectively. The function Inf_1 is related to the whole document collection D, whilst Inf_2 to the elite set E_t (this notion roots back to Harter's work [Har75]) of the term t, namely the set of all documents in which the term t occurs. P_1 is obtained as follows. We assume that words which bring little information are randomly distributed on the whole set of documents. By contrast, informative words diverge from the randomic behaviour and therefore they receive little probability according to a suitable model of randomness for Information Retrieval.

2.2.6 Ranking

Werden viele Dokumente gefunden, dann ist das Ranking von entscheidender Bedeutung. Durch das Ranking werden die Dokumente nach Relevanz sortiert, damit nur die wichtigsten Dokumente analysiert werden müssen. Das berühmteste Rankingverfahren für das Internet wurde von Google unter dem Namen PageRank™ entwickelt, hat das Unternehmen groß gemacht und gilt im wesentlichen als Betriebsgeheimnis (vgl. Abschnitt 6.6.2).

Ein öffentliches und wissenschaftlich erprobtes Rankingverfahren nennt sich Okapi BM25 [RWJ+94, RWJ+95, RWHB98, SJWR00a, SJWR00b, CZR05] nach seinem ersten Einsatz für die City University London und wird ständig weiterentwickelt [WFXF04].

In einer vereinfachten Version wird der BM-25-Score eines Dokumentes nach Gleichung 2.12 bestimmt.

Definition. Sei Q eine Abfrage mit n Suchtermen $q_1, ..., q_n$, dann wird der BM25-Score eines Dokumentes D bestimmt durch

$$\text{score}(D,Q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{avgdl})} \tag{2.12}$$

Dabei ist $f(q_i, D)$ die Häufigkeit des i-ten Suchterms im Dokument D. Das Dokument D enthält $|D|$ Wörter, und $avgdl$ ist die durchschnittliche Anzahl Wörter, die die Dokumente im Korpus enthalten. k_1 und b sind freie Parameter, die üblicherweise zu $k_1 = 2{,}0$ und $b = 0{,}75$ gewählt werden.

Definition. Die $\underline{IDF(q_i)}$ ist das Gewicht des Suchterms q_i als inverse Dokumenthäufigkeit und wird häufig nach Gleichung 2.13 berechnet.

$$IDF(q_i) = \log \frac{N - n(q_i) + 0{,}5}{n(q_i) + 0{,}5} \tag{2.13}$$

Hier ist N die Anzahl der Dokumente im Korpus und $n(q_i)$ ist die Anzahl der Dokumente im Korpus, die q_i enthalten.

2.3 Speicherverfahren

Verschiedene Verfahren zur Indexierung eines Korpus sind in [MRS08][15] beschrieben. Sie lassen sich nach unterschiedlichen Aspekten einteilen. Für ein statisches Korpus wie das hier verwendete spielt beispielsweise eine bequeme Update-Funktion keine Rolle. Andererseits sind insbesondere bei einem großen Korpus Speicherplatz und Zugriffszeit zu optimierende Kenngrößen.

Schließlich kann, wie bei B-Bäumen[16] [BM72] oder Tries[17], insbesondere einem Patricia-Trie[18] [Mor68], der Text selbst im Index enthalten sein, oder aber im Index ist nur ein Zeiger auf den Text enthalten.

Ein weiterer Aspekt ist die Granularität des Index: Soll jedes Teilwort im Korpus direkt auffindbar sein, oder wird auf größere Teilstücke, Sätze, Absätze oder ganze Dokumente verwiesen. Wie so häufig, führt eine sachgerechte Mischung der verschiedenen Algorithmen zu einem optimalen Ergebnis. Der implementierte Aufbau ist in Abbildung 4.1 auf Seite 67 beschrieben. In diesem Kapitel werden zunächst die Grundlagen der verwendeten Verfahren beschrieben.

2.3.1 Suffix-Array

Will man in einem Text nach beliebigen Zeichenfolgen auch im Inneren eines Wortes suchen, so genügt auch ein Vollformenindex bei umfangreichen Texten nicht den Anforderungen an eine kurze Laufzeit, weil jedes Wort durchsucht werden muss. Bereits 1973 wurde von Weiner [Wei73] ein Algorithmus vorgestellt, mit dem jede Zeichenfolge in einem Text in linearer Zeit indexiert werden kann. Das

[15] http://nlp.stanford.edu/IR-book/html/htmledition/index-construction-1.html

[16] http://de.wikipedia.org/wiki/B-Baum

[17] http://www.nist.gov/dads/HTML/trie.html: Äs defined by me, nearly 50 years ago, it is properly pronounced "tree" as in the word "retrieval". At least that was my intent when I gave it the name "Trie". The idea behind the name was to combine reference to both the structure (a tree structure) and a major purpose (data storage and retrieval)." Edward Fredkin

[18] http://www.itl.nist.gov/div897/sqg/dads/HTML/patriciatree.html

amerikanische *National Institute of Standards and Technology (NIST)* [Ede07, Bla09] definiert den *Suffix Tree* als: *"A compact representation of a trie corresponding to the suffixes[19] of a given string[20] where all nodes[21] with one child[22] are merged with their parents[23]."*

Eine mehr mathematische Definition lautet (zit. nach [Sti05]):

Im folgenden sei w ein Wort der Länge n über dem Alphabet Σ der Mächtigkeit σ. Außerdem sei $@ \notin \Sigma$ ein Sondersymbol (Textende). Mit $w_i, 1 \leq i \leq n+1$, bezeichnen wir das Suffix $w[i \ldots n]$. Wir gehen davon aus, dass auf $w \cup \{@\}$ eine totale Ordnung definiert ist, wobei $@$ das kleinste Element bezüglich dieser Ordnung ist.

Die Grundidee ist, den Suchwort-Baum, den *suffix trie* (siehe Abbildung 2.7 auf Seite 37) für die Menge der Suffixe des Textes $w@$ zu konstruieren. Durch das Sondersymbol ist gewährleistet, dass kein Suffix von $w@$ das Präfix eines anderen Suffixes von $w@$ ist, d. h. dass es für jedes Suffix α von w ein Blatt mit der Pfad-Beschriftung $\alpha@$ gibt. Das Blatt für das i-te Suffix ist mit i beschriftet.

Definition. Es sei $w \in \Sigma^*$ ein Wort mit $|w| = n$ und $@ \notin \Sigma$ ein Sonderzeichen.
Der Suffix-Suchwortbaum (*suffix trie*) von w ist Trie$(w_1@, w_2@, \ldots, w_n@, @)$.

Der Suffix-Suchwortbaum wird jedoch nicht selbst benutzt, da er eine quadratische Anzahl von Knoten besitzt. Verantwortlich dafür sind innere Knoten mit dem Ausgangsgrad 1. Deshalb wird der Suchwort-Baum modifiziert – es entsteht der Suffixbaum (siehe Abbildung 2.8 auf Seite 37) mit einer linearen Anzahl von Knoten.

Definition. Es sei $w \in \Sigma^*$ ein Wort und $@ \notin \Sigma$ ein Sonderzeichen. Der Suffixbaum (*suffix tree*) von Trie(w) von w entsteht, indem man im Suffix-Trie von w jeden maximalen Weg, dessen innere Knoten einen Ausgangsgrad von 1 besitzen, durch eine Kante mit der Beschriftung dieses Weges ersetzt.

Es lässt sich zeigen, dass die Zahl der Knoten von Trie(w) höchstens $2n$ ist mit $n = |w|$.

Ein solcher Suffixbaum ist vielseitig verwendbar, benötigt jedoch auch sehr viel Speicherplatz, in der Regel etwa das neunfache des Textes. Für die meisten Fälle ist das Suffix-Array (vgl. Abbildung 2.9 auf Seite 38) eine speichersparende leistungsfähige Alternative. Das Suffix-Array benötigt für jedes Zeichen des Textes

[19] http://www.nist.gov/dads/HTML/suffix.html
[20] http://www.nist.gov/dads/HTML/string.html
[21] http://www.nist.gov/dads/HTML/node.html
[22] http://www.nist.gov/dads/HTML/child.html
[23] http://www.nist.gov/dads/HTML/parent.html

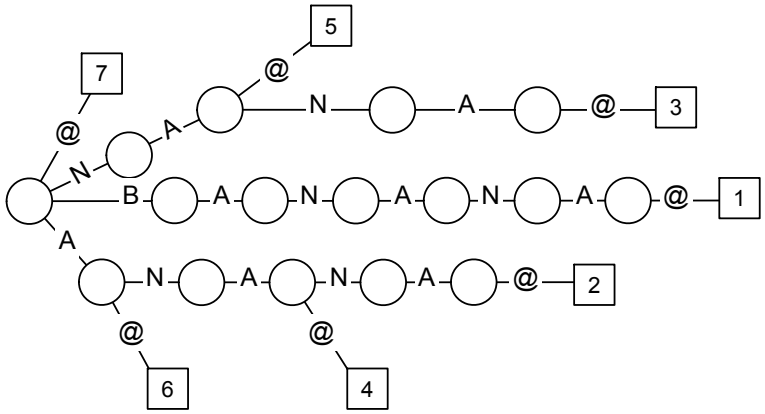


Abbildung 2.7: Der Suffixtrie für den String BANANA. Jeder Substring wird mit dem Sonderzeichen @ beendet. Die Blätter des Baums sind durch Quadrate markiert und mit i beschriftet. Jede Kante ist mit einem Zeichen von w beschriftet. Für jedes Blatt i ergeben die Beschriftungen der Kanten auf dem Pfad von der Wurzel zu i aneinander gehängt das Suffix von w, das an Index i beginnt. Somit enthält der Suffixtrie alle Suffixe von w.

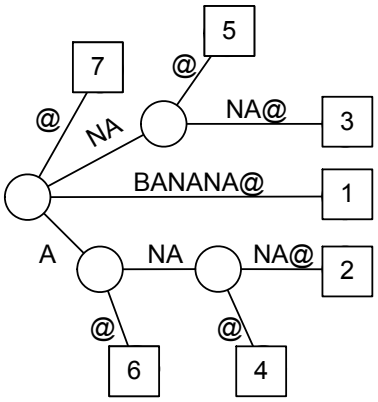


Abbildung 2.8: Der Suffixbaum für den String BANANA. Jede Kante ist mit einem Teilstring von w beschriftet. Jeder innere Knoten hat mindestens zwei Kinder, deren Kantenbeschriftungen nie mit dem gleichen Symbol beginnen. Für jedes Blatt i ergeben die Beschriftungen der Kanten auf dem Pfad von der Wurzel zu i aneinander gehängt das Suffix von w, das an Index i beginnt. Somit enthält der Suffixbaum alle Suffixe von w, wobei mehrfach auftretende Teilstrings nur einmal enthalten sind.

1	BANANA@		7	@	
2	ANANA@		6	A@	
3	NANA@		4	ANA@	
4	ANA@	$\Rightarrow$	2	ANANA@	$\Rightarrow$ [7 6 4 2 1 5 3]
5	NA@		1	BANANA@	
6	A@		5	NA@	
7	@		3	NANA@	

Abbildung 2.9: Das Suffixarray für den String BANANA. Alle Suffixe werden lexikographisch sortiert. Der im Array gespeicherte Index zeigt auf den Beginn des Suffixes.

Speicherplatz für einen Zeiger, d. h. wenn der Zeichenvorrat des Textes *single-byte*-codiert werden kann und der Text nicht größer als 2^{32} Zeichen (4 GB) ist, dann benötigt das Suffix-Array als Speicherplatz das vierfache des Textes. Die Definition folgt wieder [Sti05]:

Definition. Es sei $w \in \Sigma^*$ mit $|w| = n$. Das Suffix-Array A von w ist das $(n+1)$-dimensionale Feld A_w mit $A_w[i] = j$ genau dann, wenn w_j das lexikographisch i-te Suffix von w ist.

Die exakte Suche nach einem Wort P der Länge m erfolgt mit Hilfe des Suffix-Arrays von w durch binäre Suche. P kommt an der Position k genau dann vor, wenn die ersten m Zeichen von w_k mit P übereinstimmen. Die Vorkommen von P in w bilden damit ein zusammenhängendes Intervall des Suffix-Arrays A_w. Bei der binären Suche vergleicht man im ersten Schritt das Suchwort P mit dem Suffix w_k, das durch die mittlere Position in A_w gegeben ist. Ist P lexikographisch größer als die ersten m Zeichen von w_k oder ist w_k ein Präfix von P, so setzt man die Suche im rechten Teil des Arrays fort. Ist P lexikographisch kleiner als die ersten m Zeichen von w_k, so sucht man im linken Teil des Arrays weiter. Stimmt P mit den ersten m Zeichen von w_k überein, so gibt es ein Vorkommen von P, und man hat einen Punkt im Intervall zu P gefunden.

Für die Erzeugung eines Suffix-Arrays wurden verschiedene Algorithmen vorgestellt [KS03, S.5, 9, 28, 29, 30, 31].

2.3.2 Arrays der Listen der q-gramme

Schließlich stellt das Array der Listen der q-gramme eine weitere Möglichkeit der vollständigen Indexierung eines Textes dar.[KS03]

Definition. Es seien $w \in \Sigma^*$ mit $|\Sigma| = \sigma$, $|w| = n$ und $q \in \mathbb{N}$. Das Array der Listen der q-gramme von w ist das σ^q-dimensionale Feld L_σ wobei $L_\sigma[\alpha]$ für jedes $\alpha \in \Sigma^q$ die Liste der Vorkommen von α in w in ihrer natürlichen Reihenfolge ist.

In einer naiven Implementation benötigt jede q-gramm-Liste für jedes Zeichen des Textes Speicherplatz für einen Zeiger.

Für eine exakte Suche ist es nicht erforderlich, dass für jedes $q \leq n$ eine Liste erzeugt wurde. Es reicht vielmehr aus, wenn das Wort in Teilworte zerlegt werden kann, für die es eine Liste gibt. Dann muss man nur noch die Listen der Teilworte miteinander vergleichen.

Der Speicherbedarf des Arrays kann bei geschickter Implementierung dem Speicherbedarf des Suffix-Arrays entsprechen. Die Suche ist etwas aufwendiger, dafür ist kein Rückgriff auf den Text w erforderlich. Das ist vorteilhaft, wenn der Zugriff auf den Text zeitaufwendig ist, weil der Text für eine Speicherung im Hauptspeicher zu groß ist.

2.3.3 Neuronale Netze

Neuronale Netze erhalten ihren Namen aus dem biologischen Vorbild. Neuronen sind die Nervenzellen, die mikroskopisch netzartig untereinander verbunden sind. Neuronale Netze dienen der Informationsweiterleitung, der Informationsverarbeitung, der Mustererkennung und dienen als Gedächtnis. Sie sind das Vorbild für die hier besprochenen künstlichen neuronalen Netze, wobei das Wort „künstlich“ weggelassen wird, wenn keine Verwechselungsgefahr besteht. Künstliche neuronale Netze sind völlig anders aufgebaut als biologische neuronale Netze. Sie bilden aber Teilfunktionen biologischer Netze mehr oder weniger genau nach.

Künstliche neuronale Netze lassen sich nach verschiedenen Kriterien einteilen [Zel97]. Zunächst interessieren die topologischen Merkmale: Ein Netz kann ein-, zwei- oder mehrschichtig aufgebaut sein. Die Neuronen können untereinander verschieden stark vernetzt sein, und sie können auch mit sich selbst vernetzt sein.

Eine andere Einteilung betrifft die Art der Signalweiterleitung: Die Eingangssignale können von Schicht zu Schicht weitergeleitet werden, dies ist der einfache Fall, sogenannte *Feedforward*-Netze. Sie können sich auch in jeder Schicht ausbreiten, oder es kann Rückverbindungen von tiefer liegenden Schichte zu vorgelagerten Schichten geben. Die beiden letztgenannten Arten werden als rekurrente oder rückgekoppelte Netze bezeichnet. Die Art der Rückkopplung wird weiter unterschieden in

- *direct feedback*: ein Ausgang eines Neurons wirkt auf einen Eingang desselben Neurons,

- *indirect feedback*: ein Ausgang eines Neurons wirkt auf einen Eingang eines vorgelagerten Neurons,
- *lateral feedback*: ein Ausgang eines Neurons wirkt auf einen Eingang eines Neurons der selben Schicht.

Schließlich lassen sich die verwendeten Signale unterscheiden in binäre, digitale und analoge Signale. Je nach Art des Netzes sind diese Netze unterschiedlich leistungsfähig. Allen Netzen gemeinsam ist, dass sie zunächst nur eine Struktur vorgeben. Die genaue Art der Signalverarbeitung muss jeweils noch gelernt werden.

Kleinste Einheiten dieser Netze sind stilisierte Neuronen, wie sie von [MP43] zuerst beschrieben worden sind. Diese Neuronen haben eine Ausgangsleitung und können eine beliebige Anzahl aktivierender und hemmender Eingangsleitungen haben, an denen Signale anliegen. Diese Signale werden durch Gewichte w bewertet. Die Neuronen berechnen aus den Zuständen an den Eingangsleitungen einen Wert, den sie mit einem gespeicherten Schwellwert Θ vergleichen, und geben entsprechend diesem Vergleich selbst einen Ausgangswert aus. Die Funktion, die den Ausgangswert berechnet, heißt Aktivierungsfunktion. Diese Neuronen werden zu Netzen verknüpft.

Die Gewichte werden häufig entsprechend der Hebbschen Regel [Heb49] eingestellt

$$\Delta w_{ij} = \eta \cdot a_i \cdot o_j \tag{2.14}$$

mit Δw_{ij} : Veränderung des Gewichtes von Neuron j zu Neuron i (also die Änderung der Verbindungsstärke dieser beiden Neuronen),

η : „Lernrate“ (ein geeignet zu wählender konstanter Faktor),

a_i : Aktivierung von Neuron i und

o_j : Ausgabe von Neuron j, das mit Neuron i verbunden ist.

Die Aktivierungsfunktion muss eine Lyapunov(Ляпунов)-Funktion sein, damit das Netz dynamisch stabil ist. Eine Lyapunov-Funktion

$$E : \mathbb{R}^n \to \mathbb{R} \tag{2.15}$$

hat in einem dynamischen System die Eigenschaft eine positiv-definite Funktion zu sein, d. h. es gilt

$$\begin{aligned} E(0) &= 0 \\ E(x) &> 0 \quad \forall x \in U \setminus \{0\} \end{aligned}$$

für eine Umgebung U von $x = 0$.

Im Abschnitt 4.4 wird ein zweischichtiges *feedforward*-Netz vorgestellt werden. Die Neuronen in diesem Netz arbeiten nur mit binären Signalen und ein einzelnes

Neuron wird durch ein einzelnes Bit im Speicher repräsentiert. Als Besonderheit hat dieses Netz eine vorgeschaltete Codierungsfunktion, mit der das Eingabemuster auf die Neuronen der Eingabeschicht abgebildet wird, dazu kommt eine Decodierungsfunktion, mit der die Ausgabe wieder in ein Muster umgewandelt wird.

2.3.4 Assoziativspeicher

Ein Assoziativspeicher ist eine Speicherform, bei der mit der Assoziation von Inhalten gearbeitet wird, um auf einzelne Speicherinhalte zuzugreifen. Das menschliche Gedächtnis arbeitet u. a. mit Assoziationen. So verbinden wir beispielsweise mit bestimmten Gegenständen Erinnerungen an bestimmte Erlebnisse.

Um eine unscharfe Suche zu ermöglichen, müssen Suchbegriff und Wörterbucheintrag „unscharf“ auf einander abgebildet werden. Eine Möglichkeit, dies zu erreichen, ist die Speicherung mit einem Assoziativspeicher. Hier werden nach einem zuvor festgelegten Algorithmus Buchstaben, Buchstabengruppen oder Phoneme auf einzelne Bits abgebildet. Durch die Abbildung auf Bits wird einerseits eine sehr starke Kompression erreicht, die weitere Suchfunktionen beschleunigt. Andererseits werden mehrere Zeichen zusammengefasst, wodurch in der Umkehrung, d. h. beim Retrieval mehrere Token zurückgegeben werden. Werden beispielsweise ›ay‹, ›ai‹, ›ei‹ und ›ey‹ auf ein Bit abgebildet, so liefert eine Anfrage nach »Meier« sowohl »Meier« wie auch alle anderen phonetisch gleichen Wörter wie »Maier«, »Mayer« und »Meyer« zurück.

Die Qualität des Retrievals ist stark abhängig von der Anpassung des Abbildungsalgorithmus an die Datengrundlage. Für die Indexierung amerikanischer Namen westeuropäischer Herkunft wurde von Russell 1918 [Rus18, Rus22] ein Algorithmus („SoundEx[24]“) entwickelt und patentiert[25], mit dem Karteikarten nach akustischer Erfassung der Namen wieder aufgefunden werden konnten.

Wenn man in einem auf einer Tastatur geschriebenen Dokument Wörter trotz vorhandener Tippfehler wieder finden will, werden sich andere Schreibvarianten finden, als wenn ein zuvor tippfehlerfreies Dokument ausgedruckt und im OCR-Verfahren wieder digitalisiert wurde.

Schließlich ergeben sich wieder andere Schreibweisen durch Unsicherheiten des Autors in der Orthographie und durch Änderungen und temporale und regionale Eigentümlichkeiten in der Orthographie. Zur inhaltsbasierten Suche schlug deshalb bereits Zuse 1943 [Zus45, Gil90] eine Speicherform vor, die durch ihren Inhalt

[24] http://www.creativyst.com/Doc/Articles/SoundEx1/SoundEx1.htm
[25] U.S. Patente 1 261 167, 1 435 663

adressiert werden sollte. Ein solcher Speicher wird Assoziativspeicher (*Content Addressable Memory, CAM*) genannt.

In herkömmlichen Rechnern ist der Hauptspeicher als *Random Access Memory, RAM* organisiert. Jede Speicherzelle hat eine Adresse und über diese Adresse kann auf den Inhalt der Speicherzelle zugegriffen werden. Im Gegensatz dazu wird beim Datenzugriff im Assoziativspeicher der gesamte Speicher in einem Takt gleichzeitig mit dem eingegebenen Muster verglichen. Als Ausgabe erhält der Nutzer eine Liste der Adressen, in denen das Muster gefunden wurde oder auch gleich die mit dem Muster verknüpften Daten. Wenn der Assoziativspeicher nicht nur 0 und 1 speichern kann, sondern auch X als Symbol für ein Bit mit beliebigem Wert, spricht man von einem ternären CAM. In diesem Fall ist es sinnvoll, das gespeicherte Muster selbst zurückzugeben. Ein solcher Speicher ist natürlich ideal für eine unscharfe Suche.

Leider ist solcher Speicher in der Herstellung sehr teuer, weil jede Speicherzelle nicht nur ihren Zustand speichern muss wie im RAM, sondern darüber hinaus die Auswertelogik zur Verfügung stellen muss.

Weil bei jedem Datenzugriff sämtliche Auswertelogiken im gesamten Speicher aktiviert werden, ist auch der Energieverbrauch eines solchen CAMs wesentlich höher als der eines RAMs. Einen Überblick über den aktuellen Stand bei der Entwicklung des Assoziativspeichers geben [PS06]. Während ein aktueller RAM-Chip 8 Gigabit speichert, speichert ein CAM-Chip 0,5 Megabits (Music Semiconductors, 2008[26]).

Assoziativspeicher wird heute für kleine Caches verwendet, ansonsten ist der Aufwand zu hoch und die Trefferbestimmung aufgrund der aufwendigen Logik zu langsam [SK03].

Um derartige Funktionen zu realisieren, wird deshalb gerne auf neuronale Netze zurückgegriffen.[27] In Kapitel 4.4 auf Seite 73 wird jedoch eine Softwarelösung vorgestellt, die die geforderten Eigenschaften besitzt und den Anforderungen entsprechend ausreichend schnell ist.

2.4 Unscharfe Suche oder fehlertolerantes Retrieval

2.4.1 *Rough Set*

Zur Beschreibung der Treffermenge einer unscharfen Suchanfrage eignet sich am Besten ein *rough set* („grobe“ Menge) im Unterschied zum *crisp set* („scharfe“ Menge) der herkömmlichen mathematischen Definition einer Menge. Ein *rough set*

[26] http://www.music-ic.com/index.php?section1=products&section2=function
[27] http://www.at-mix.de/assoziativspeicher.htm

kann anschaulich beschrieben werden als eine Approximation einer Menge mit unscharfen Rändern durch zwei Mengen im herkömmlichen mathematischen Sinn, die die untere oder innere und die obere oder äußere Schranke der Menge in einem gegebenen Universum bezeichnen. Im Tutorial von [KPPS98] werden dazu über 500 Literaturhinweise gegeben.

Wir gehen aus von einem Informationssystem $\mathcal{A} = (U, A)$, wobei U eine Menge von Objekten – das Universum – ist und A eine nicht-leere, endliche Menge von Attributen, die den Elementen aus U zugeordnet sind.

In einem solchen Informationssystem $\mathcal{A} = (U, A)$ kann man für eine Teilmenge $B \subseteq A$ eine Äquivalenzrelation $IND_{\mathcal{A}}(B) = \{(x, x') \in U \times U \mid \forall a \in B : a(x) = a(x')\}$ definieren.

Eine Äquivalenzrelation $R \subseteq U \times U$ ist eine Relation, die reflexiv[28], symmetrisch[29] und transitiv[30] ist. Sie partitioniert eine Menge restlos in nichtleere und disjunkte Untermengen, die als Äquivalenzklassen bezeichnet werden. Diese Partition von U wird mit $U/IND(B)$ oder U/B bezeichnet.

Der Name $IND_{\mathcal{A}}(B)$ leitet sich ab von *indiscernibility relation* (Ununterscheidbarkeitsrelation). Wenn $(x, x') \in IND_{\mathcal{A}}(B)$, dann sind die Objekte x und x' bezüglich der Attribute von B nicht mehr unterscheidbar (*indiscernible*). Die Äquivalenzklassen der bezüglich B nicht unterscheidbaren Objekte aus U werden als $[x]_B$ bezeichnet. Liegen nur die Informationen aus B vor, dann lässt sich eine Teilmenge $X \subseteq U$ approximieren durch eine untere oder innere Schranke $\underline{B}X$ und eine obere oder äußere Schranke $\overline{B}X$ mit $\underline{B}X = \{x \mid [x]_B \subseteq X\}$ und $\overline{B}X = \{x \mid [x]_B \cap X \neq \varnothing\}$.

In neueren Definitionen[SV97, SV00, SGM05] wird diese Definition, insbesondere die Forderung nach einer Äquivalenzrelation, noch verändert:

So kann es sinnvoll erscheinen, wenn auf die Symmetrieeigenschaft verzichtet wird und bei der Suche nach „Götze“ auch der „Gott des Geldes“ gefunden wird, bei der Suche nach „Gott“ jedoch Verweise auf den Term „Götze“ unterdrückt werden.

Ähnlich sinnvoll ist es, wenn auf die Forderung nach Transitivität verzichtet wird. Ein Beispiel mag eine Reihe Tassen voller Kaffee sein, in denen der Zuckergehalt successive erhöht wird. Den Inhalt zweier benachbarter Tassen wird man als ähnlich empfinden. Ob zwischen dem Inhalt der ersten und dem Inhalt der letzten Tasse noch eine Ähnlichkeit besteht, hängt wohl eher vom Zuckergehalt ab.

Wenn man auf Symmetrie und Reflexivität verzichtet, bildet man damit einen Teil der Vielgestaltigkeit der Sprache ab: Synonyme haben in der Sprache nur in einem bestimmten Zusammenhang die gleiche Bedeutung, meistens unterscheiden

[28] reflexiv: $\forall x \in U : xRx$

[29] symmetrisch: $\forall x, y \in U : xRy \Rightarrow yRx$

[30] transitiv: $\forall x, y, z \in U : xRy \wedge yRz \Rightarrow xRz$

sie sich aber auch dann noch in Nuancen. Je nach Häufigkeit eines Suchterms kann der Einbezug von Synonymen die Ergebnisqualität einer unscharfen Suche verbessern.

Die innere Schranke $\underline{B}X$ eines *rough sets* sei die Menge aller gefundenen Verweise, die auf eine exakte Kopie des Suchterms zeigen.

Außerhalb der äußeren Schranke $U \setminus \overline{B}X$ liegen alle Verweise, die für die Suchanfrage sicher irrelevant sind. Im Bereich zwischen diesen beiden Schranken, der *boundary region* oder dem *border set* sind alle Verweise zu finden, die für die gegebene Anfrage möglicherweise interessant sind. Wie viele Verweise diesem Bereich zugeordnet werden, hängt insbesondere von der Granularität des Bereichs zwischen diesen Schranken ab. Den gefundenen Termen werden Attribute zugeordnet, wie z. B. die Relevanz.

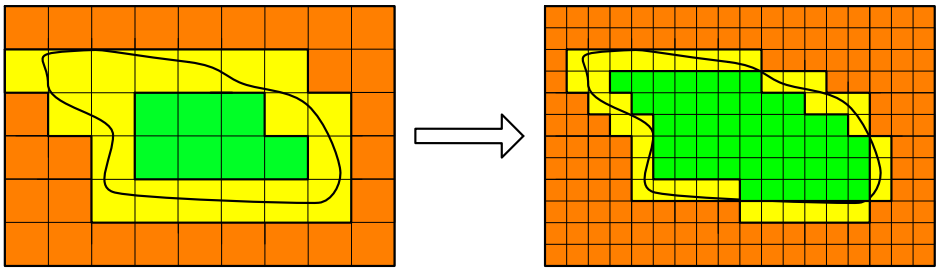


Abbildung 2.10: *rough set* und Granularität: Der rote Bereich bezeichnet das *outer set*, der gelbe Bereich bezeichnet das *border set* und der grüne Bereich bezeichnet das *inner set* des eingetragenen *rough sets*.

Die Granularität G ist eine Abbildung der Potenzmenge $\mathcal{P}$ der Attribute A: $\mathcal{P}(A) := \{U \mid U \subseteq A\}$ auf die Menge der reellen Zahlen $\mathbb{R}$:

$$G : \mathcal{P}(A) \to \mathbb{R} \tag{2.16}$$

Granularität definiert [ZYG08] als Verbesserung der Definition von [LWQ09, XZZ09] mit den bisher verwendeten Symbolen folgendermaßen:

Definition. Sei $\mathcal{A} = (U,A)$ ein Informationssystem, und $\forall P \subseteq A$, eine Partition $U/IND(P) = \{P_1,P_2,...,P_m\}$, dann kann Granularität wie folgt definiert werden:

$$G(P) = \frac{1}{|U|^2 - 1} \left(\frac{|U|}{\sum_{i=1}^{m} 1/|P_i|} - 1 \right) \tag{2.17}$$

In Zahlen ausgedrückt bewegt sich G in einem Bereich von 0, wenn die Partition $|U|$ Elemente hat und 1, wenn die Partition nur ein Element hat:

$$U/IND(P) = \{\{u_i\} \mid u_i \in U\} \qquad \Rightarrow G(P) = 0$$
$$U/IND(P) = \{\{U\}\} \qquad \Rightarrow G(P) = 1$$

Für die in Abbildung 2.10 dargestellten beiden Universen mit $m = 3$ und $|U| = 54$ bzw. $|U| = 216$ ergeben sich Werte für die Granularität G von $G = 0{,}0122$ bzw. $G = 0{,}0855$.

Im *border set*, dem Bereich zwischen innerer und äußerer Schranke, besteht ein besonderes Bedürfnis, dass die Einträge nach Relevanz sortiert werden. Um diese Relevanz darzustellen, eignet sich eine Erweiterung des *rough sets*, der Dominance-based Rough Set Approach [GMS01]. Dazu müssen die Kriterien, die zum Einbezug eines Terms in das *border set* führen, partiell geordnet sein.

2.4.2 Algorithmen

Zur Stringsuche in Texten wurden bereits viele Algorithmen entwickelt. Mit den einfachsten lässt sich nur ein gegebenes Muster in einem Text auffinden. Die weiteren Entwicklungen bezogen sich einerseits darauf, mehr als einen String effektiv gleichzeitig zu suchen, andererseits auch ungenaue Suche und reguläre Ausdrücke als Suchmuster zuzulassen. Ein weiteres entscheidendes Kriterium ist die Mächtigkeit des zulässigen Alphabets. Für die Suche in DNA-Ketten mit ihren vier Basen sind andere Algorithmen effektiver als für die Suche in Texten mit einem Alphabet von mehr als zwanzig Zeichen.

Algorithmen, die (kurze) Zeichenketten mit (langen) Zeichenketten vergleichen, sie darin suchen, lassen sich nach verschiedenen Kriterien einteilen: Nach Zeit- oder Speicherkomplexität, nach dem zugrunde liegenden algorithmischen Prinzip: Suchbäume, Automaten, Bit-parallele Verfahren und einige weitere Prinzipien sowie Kombinationen der vorgenannten. Andere Einteilungskriterien sind die Fähigkeit des Algorithmus, mehrere Zeichenketten gleichzeitig zu suchen oder eine unscharfe, fehlertolerante Suche zuzulassen. Schließlich kann noch unterschieden werden, ob ein Algorithmus auf eine Vorverarbeitung wie z. B. eine Indexierung angewiesen ist (er *offline* arbeitet), oder ob eine solche nicht erforderlich ist (*online*). Außerdem gibt es auch Algorithmen, die auf komprimierten Texten arbeiten können, was bei hohen I/O-Kosten vorteilhaft sein kann.

Navarro und Raffinot [NR02] haben verschiedene aktuelle Algorithmen auf ihre Eignung für die Stringsuche in Texten untersucht. Ein wichtiges Entscheidungskriterium für die Wahl eines optimalen Algorithmus ist dabei die Mächtigkeit des zugrunde liegenden Alphabets. Prinzipiell unterscheidet sich die Suche nach einer

bestimmten Basensequenz in einem DNA-Strang nicht von der Suche nach einem Wort in einem Text. Es hat sich jedoch gezeigt, dass bei einer Beschränkung auf ein Alphabet von nur vier Zeichen andere Algorithmen vorteilhafter sind als wenn ein umfangreicheres Alphabet mit einem Zeichenvorrat von 23 oder 150 Zeichen verwendet wird.

2.4.3 AGREP

Reguläre Ausdrücke beschreiben ein Suchmuster genau. Für die Onlinesuche existiert ein Algorithmus *grep* für *get regular expressions*. Wu und Manber [WM91] entwickelten eine Variante für die unscharfe Suche, den sie *agrep (Approximate GREP)* nannten. Er wurde seither weiterentwickelt und ist frei für den privaten und nicht kommerziellen Gebrauch, steht aber nicht unter der GPL[31]

Eine Neuimplementierung, *TRE* von Laurikari [Lau06] in der Version 0.7.5 ist unter der LGPL[32] lizensiert. Das Paket *TRE* ist eine Bibliothek von Prozeduren, die reguläre Ausdrücke um eine unscharfe (*fuzzy*) Suche erweitern.

Für die Form regulärer Ausdrücke gibt es eine Norm, den *POSIX*-Standard [IEE04] den die Autoren verschiedener Implementierungen unterschiedlich genau einhalten. Der Autor von *TRE* war bemüht, die Spezifikation möglichst genau zu erfüllen.

Die in *TRE* implementierte unscharfe Suche verwendet die *Levenshtein*-Distanz (vgl. 4.4.1 auf Seite 74, um das Suchergebnis um ungefähr passende Begriffe zu erweitern. Dabei können für die Operationen *Einfügen*, *Löschen* oder *Ersetzen* von Zeichen unterschiedliche Kosten angegeben werden.

Optional kann angegeben werden, dass die *Levenshtein*-Distanz ein bestimmtes Maß nicht übersteigen soll oder es kann auch nach den Treffern mit der geringsten *Levensthein*-Distanz gesucht werden.

Die *worst-case*-Zeitkomplexität liegt bei $\mathcal{O}(M^2N)$, wenn M die Länge des regulären Ausdrucks ist und N die Länge der zu durchsuchenden Zeichenkette. Der Algorithmus arbeitet also online und greift nicht auf einen vorher berechneten Index zurück.

Leider waren die verwendeten Bibliotheken nicht kompatibel.

2.4.4 Anapherresolution

Wesentliche Terme werden in Dokumenten nicht immer gleich ausgedrückt. Abgesehen von vielfältigen Umschreibungen, wie sie gerade in belletristischen Texten

[31] http://www.gnu.org/licenses/gpl-3.0.html
[32] http://www.gnu.org/licenses/lgpl.html

üblich sind, werden häufig Ausdrücke verwendet, die auf andere Begriffe verweisen [Sto07]. Diese Ausdrücke werden Referenzausdrücke genannt. Die Begriffe, auf die sie sich beziehen, heißen Referenten. Verweist der Referenzausdruck auf einen vorher oder später eingeführten Begriff, so liegt eine Anapher vor. Kommt der Referenzausdruck gar nicht im Text vor, weil er ausgelassen wurde, dann spricht man von einer Ellipse. Das Ziel der Anapherresolution ist die korrekte Auflösung von Anaphern und Ellipsen durch das Zusammenführen von Referent und zugehörigem Referenzausdruck. Besonders wichtig ist die Anapherresolution für die korrekte Termwichtung bei Vektorraum-Modellen und probabilistischen Modellen. Für die englische Sprache existiert ein einfacher Algorithmus zur Auflösung von Pronomen [Hob78]. Insbesondere bei der Auflösung von Personennamen konnten Pirkola und Järvelin [PJ96]) erhebliche Recallsteigerungen feststellen. Für andere Begriffe erschien dies nicht so wichtig. Andererseits hielten diese Autoren es für unwahrscheinlich, dass in naher Zukunft ein funktionierendes System entwickelt würde. Sogar negative Effekte konnten bei manchen Anfragen und Klassen von Anaphern festgestellt werden [DL90].

Andererseits stehen in der Regel Referent und Referenzausdruck innerhalb eines Absatzes, zumindest innerhalb eines Abschnitts. Da im vorgestellten System die Suche nach einem Begriff eine Stelle in einer Datei liefert, liegt die Nachbarschaft des Begriffs im Blickfeld des Nutzers.

2.5 Relevance Feedback

Allein die Phänomene der Polysemie und Homonymie bedingen, dass eine Suchanfrage relevante und irrelevante Dokumente zurück liefern kann. Weil Suchanfragen häufig nur sehr kurz sind, nur aus wenigen Termen bestehen, sind die Anfragen mehrdeutig. In nutzerfreundlichen Systemen besteht deshalb die Option, dass der Nutzer in einer zurück gelieferten ersten Liste relevante und irrelevante Dokumente ankreuzen kann. Das System kann deshalb zunehmend bessere Hypothesen über die Suchanfrage aufstellen und ausgehend von dieser so präzisierten Suchanfrage eine neue Liste von Dokumenten erstellen.

Die ursprüngliche Idee besteht darin, aus den t Termen $i \in \{1, \ldots, t\}$ mit den Wichtungen q_i der Suchanfrage $Q := (q_1, \ldots, q_t)$ durch Neugewichtung der Suchterme eine neue Suchanfrage $Q' := (q_1', \ldots, q_t')$ zu erzeugen.[Roc66, Roc71, Ide71, Sal90]

Einen aktuellen Überblick über verschiedene automatische und halbautomatische Relevance Feedback Systeme geben Ruthven und Lalmas [RL03] in ihrer Übersichtsarbeit. Ein interessanter Ansatz ist auch der von [OMY06], bei der Bestimmung von Relevanz nur wenige, vom Nutzer als nicht relevant beurteilte Dokumente heranzuziehen.

Bei der Indexierung für ein solches System sind nur solche Terme wichtig, die Dokumente unterscheidbar machen. Deshalb werden Stoppwörter und solche Terme entfernt, die in vielen Dokumenten vorkommen. Die verbleibenden Terme sollten auf ihre Stammformen reduziert werden und erhalten ein Gewicht entsprechend ihrer *tf-idf* (siehe Gleichung (2.4) auf Seite 20).

Durch das Feedback des Nutzers kann dann die Kontingenztabelle neu berechnet werden, wie sie in Abschnitt 2.2.5 auf Seite 32 für das probabilistische Modell beschrieben wurde.

Relevance Feedback ist aber nicht nur für das probabilistische Modell vorteilhaft. Logikbasierte Modelle, insbesondere solche, die ein Inferenz-System verwenden, können die Suchanfrage durch ontologisches Wissen erweitern und so beispielsweise für eine Anfrage nach Tieren Dokumente über Hunde oder Katzen liefern, auch wenn der Term »Tier« im Dokument selbst nicht erscheint.

In gleicher Weise kann eine Suchanfrage mit Hilfe eines Synonymwörterbuchs oder Thesaurus erweitert werden.

Harman [Har88, Har92] und Lundquist et al. [LGF97] stellten fest, dass eine Erweiterung der Suchanfrage um bis zu zwanzig Terme die Ergebnisqualität verbessern könnte. Die Zahl zwanzig kam unter anderem dadurch zustande, dass zwanzig Terme in einer Feedback-Bildschirmmaske gerade noch übersichtlich dargestellt werden können, in der der Nutzer relevante Terme ankreuzen kann. Ein ähnliches Feature wird in der vorliegenden Arbeit bei der Termerweiterung durch eine unscharfe Suche genutzt.

Darüber hinaus wurde Relevance Feedback ohne Benutzerfeedback, also ohne Relevanz-Informationen untersucht. Auch solches Pseudo Relevance Feedback konnte die Ergebnisqualität steigern, insbesondere, wenn die erste Suchanfrage Dokumente mit hoher Relevanz generierte. Lieferte die erste Suchanfrage nur Dokumente mit geringer Relevanz, dann entstand häufig eine *query drift*. Beim Pseudo Relevance Feedback wird aus den am besten bewerteten Dokumente einer ersten Suchanfrage eine weitere Anfrage generiert, deren Ergebnisse dem Nutzer präsentiert werden.

Im folgenden Kapitel wird das untersuchte Korpus näher in Augenschein genommen.

3 Material

3.1 Verwendete Hard- und Software

Alle Programme wurden ausgeführt auf einem Rechner mit einem Prozessor AMD Athlon® 2200+ mit 1,8 Ghz und 2 GB RAM. Als Betriebssystem wurde Microsoft® Windows® XP Home SP3 verwendet. Diese Arbeit wurde geschrieben mit LaTeX 2_ε. Die erhaltenen Daten wurden mit Microsoft® Excel 2002 ausgewertet. Für den Import von Excel-Diagrammen in LaTeX 2_εwurden die Diagramme mit FreePDF[1] und Ghostscript[2] in eine .pdf und eine .eps-Datei gedruckt und der .eps-Datei die Daten der Boundingbox entnommen. Zur Bildbetrachtung, -bearbeitung und Formatkonvertierung wurde Adobe® Photoshop® 6.0 benutzt. Als Programmieroberfläche für Visual Basic-Programme wurde die Express Edition aus Microsoft® Visual Basic 2005 verwendet.

3.2 Das verwendete Korpus

3.2.1 Beschreibung

Bei den Dateien des Projekts Gutenberg (Edition 8) handelt es sich um Romane, Erzählungen, Novellen, Dramen und Gedichte in deutscher Sprache von über 500 Autoren – Literatur von der Antike bis ins 20. Jahrhundert – in mehr als 80 000 Text- und Bilddateien. Das Gesamtvolumen benötigt 1,39 GB in über 3 600 Ordnern. Die Texte liegen im HTML[3]-Format vor, die Ausgabe wird mit Hilfe von 29 Cascaded Style Sheets (CSS) formatiert. Im Ordner pic und seinen 15 Unterordnern liegen 283 Dateien mit weniger als 3 MB. Außerdem enthält die DVD 2 536 Bilder im jpg-Format. Das Korpus enthält 111 760 879 laufende Wortformen[4], 1 410 643 verschiedene Wortformen in 66 190 Dokumenten.

[1] http://freepdfxp.de/

[2] http://www.ghostscript.com/

[3] Hyper Text Markup Language

[4] Die genaue Zahl der laufenden Wortformen und der verschiedenen Wortformen hängt von der Auswahl der Zeichen ab, die als Worttrenner definiert werden und kann um etwa 2 % schwanken. Die Liste der als Worttrenner definierten Zeichen ist im Anhang in Tabelle 7.2 auf Seite 169 zu finden.

3.2.2 Statistische Auswertung

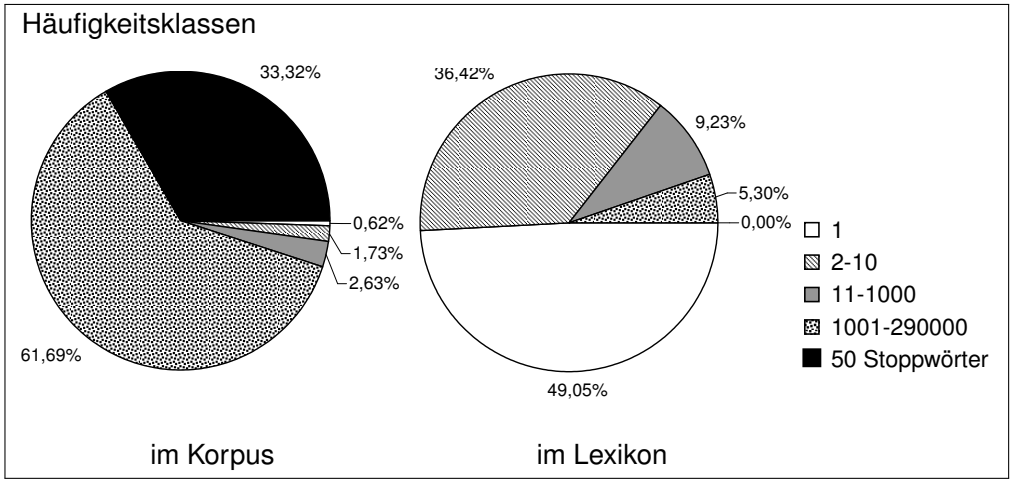


Abbildung 3.1: Wortformen in Häufigkeitsklassen. Die fünfzig am häufigsten vorkommenden Wörter machen 33 % aller laufenden Wortformen im Korpus aus, aber nur 0,035 ‰ der Wörterbucheinträge. Nur einmal im Korpus vorkommende Wortformen bilden 0,6 % der Wörter im Korpus und 49 % der Einträge im Vollformenlexikon. Die hier interessierenden Wortformen, die zwei bis zehn mal im Korpus vorkommen, bilden 35 % der Wörterbucheinträge.

Abbildung 3.1 zeigt die Verteilung der Wortformen im Korpus und im Vollformenlexikon nach Häufigkeitsklassen. 62 % aller Wörter im untersuchten Korpus haben eine mittlere Häufigkeit im Bereich von 1 000 bis knapp 300 000 Vorkommen. Ein weiteres Drittel sind die sogenannten Stoppwörter (und, die, der, in, ...), die hier als die fünfzig häufigsten Wörter definiert wurden (vgl. Abbildung 3.4 auf Seite 53). Nur 2,4 % der Wörter im Korpus kommen zwischen einem und zehn mal vor. Im Wörterbuch bilden diese Wörter jedoch 85 % aller Einträge.

Im Abschnitt 6.2 ab Seite 133 wird ein anderes Indexierungsverfahren diskutiert, das sich speziell für häufig zu aktualisierende Korpora eignet. Abbildung 3.2 bestätigt die dort genannten Zahlen: Die 5 % häufigsten Wörter im Wörterbuch ergeben 95 % der laufenden Wortformen im Korpus.

Abbildung 3.3 zeigt in einem Histogramm die Verteilung der Anzahl der Zeichen in den Dateien im Korpus. Die Abszisse ist entsprechend einer geometrischen Reihe aufgeteilt. Sie enthält zur Klasseneinteilung die gerundeten Werte der Normreihe DIN R10. Die Ordinate gibt die Anzahl der Dateien im Bereich zwischen den angegebenen Anzahlen von Zeichen an. Mehr als die Hälfte der Dateien enthalten zwischen 2 000 und 20 000 Zeichen, der Median liegt bei 8 370 Zeichen.

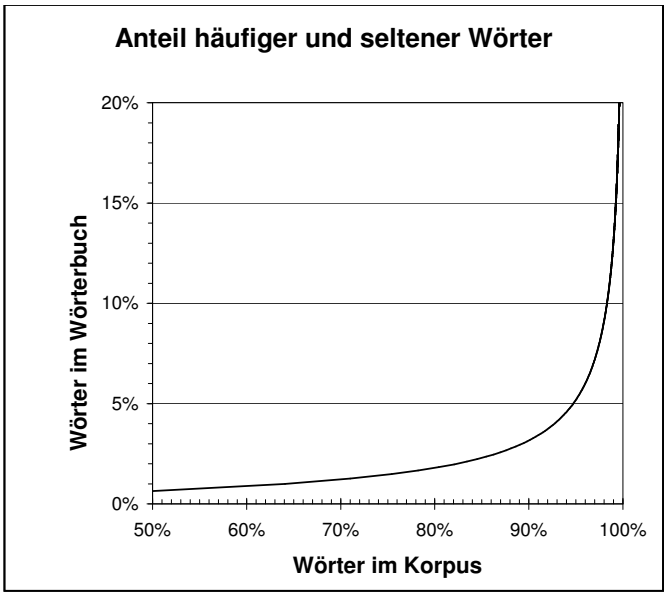


Abbildung 3.2: Anteil häufiger und seltener Wörter. Die 5 % häufigsten Wörter im Wörterbuch ergeben 95 % der laufenden Wortformen im Korpus.

Diese Verteilung wird auch zugrunde gelegt, wenn es um die Verteilung der selten vorkommenden Terme in den Dateien geht (vgl. Abbildung 5.1 auf Seite 101).

Die folgenden vier Diagramme wurden aus kryptographischem Interesse eingefügt:

Abbildung 3.4 zeigt in logarithmischer Skalierung die fünfzig häufigsten Wörter und ihre Anzahl im Korpus. Groß- und Kleinschreibung wurde nicht unterschieden. Die fünfzig häufigsten Wörter bilden mit 40 114 553 laufenden Wortformen bereits ein Drittel der laufenden Wortformen des gesamten Korpus. Bei Speichermangel oder Performanceproblemen bietet es sich an, diese Wörter als Stoppwörter nicht zu indexieren. Mit einer aktuellen Rechnerausstattung ist das für das hier betrachtete große Korpus nicht erforderlich. Alle Wortformen wurden indexiert und für den Primärindex wurden auch Groß- und Kleinschreibung unterschieden.

Abbildung 3.5 gibt eine Übersicht über die Wortlänge, d. h. die Anzahl der Zeichen in einem Wort. Wegen der logarithmischen Skala sind nur Häufigkeiten größer als eins aufgeführt. Wörter mit 10 Zeichen Länge kommen mit 152 556 Vorkommen im Vollformenlexikon am häufigsten vor. Die häufigen „Wörter“ mit nur einem oder zwei Zeichen sind Zahlen, Fußnotenzeichen und andere Kombinationen von solchen Zeichen, die nicht als Worttrenner deklariert wurden.

Abbildung 3.6 zeigt die relative Häufigkeit der Buchstaben im Vollformenwörterbuch. Zum Vergleich sind mit einer roten Raute ◆ die von [Beu05] angegebenen

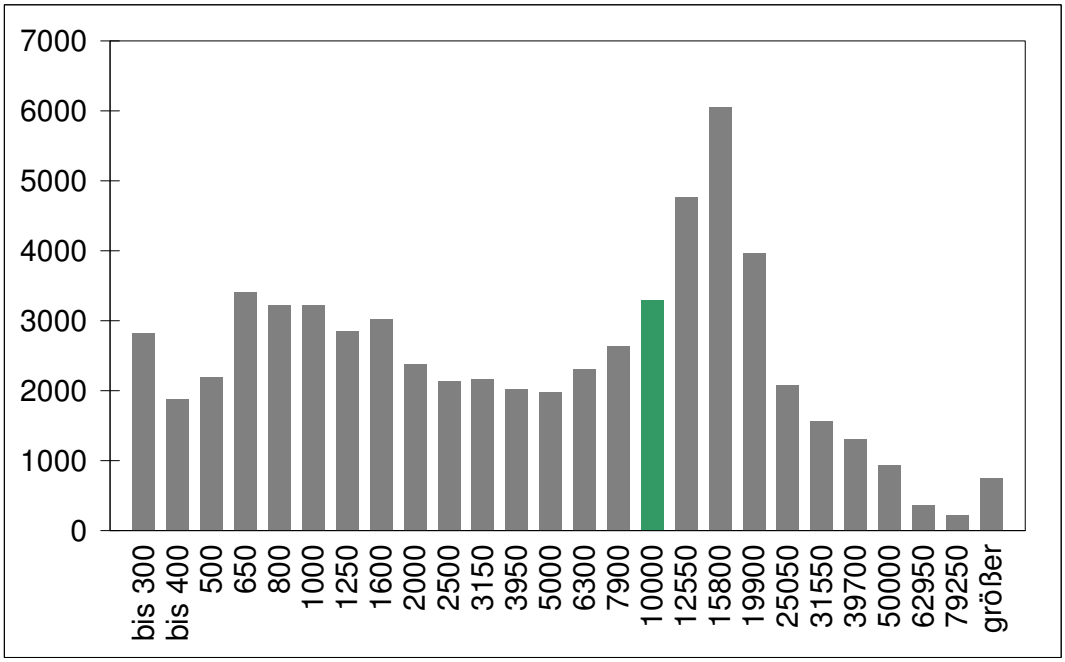


Abbildung 3.3: Häufigkeit der Dateilängen. Auf der Abszisse sind in geometrischer Reihe (gerundete Werte der Normreihe DIN R10) die Anzahl der Zeichen im Text eingetragen. Die Ordinate gibt die Anzahl der Dateien in der entsprechenden Häufigkeitsklasse an. Der Median liegt bei 8 370 Zeichen und wurde grün markiert.

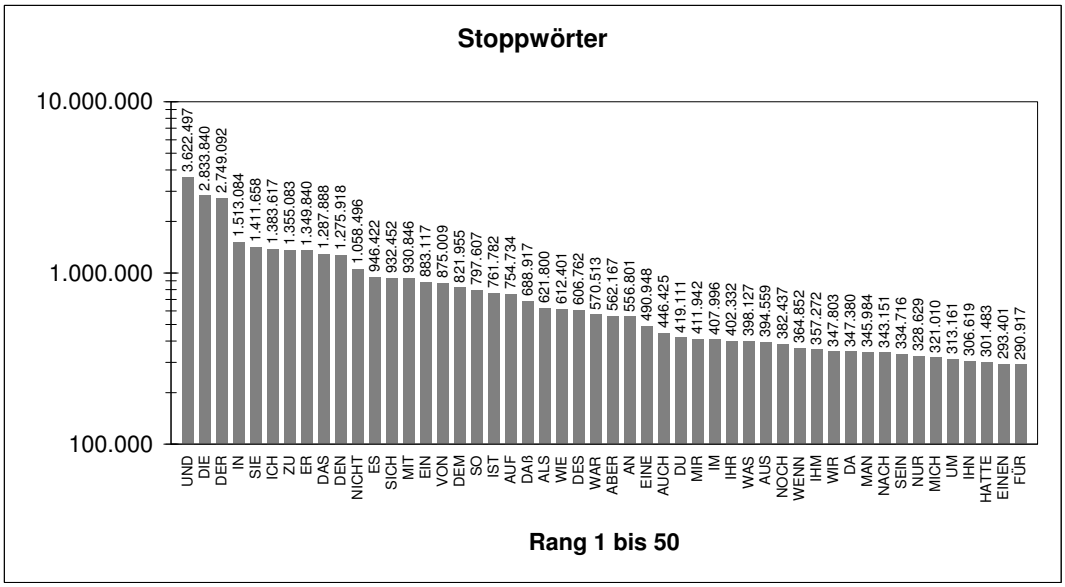


Abbildung 3.4: Die 50 häufigsten Wortformen nach Häufigkeit. Angegeben sind die fünfzig häufigsten Wortformen, sortiert nach ihrem Rang. Groß- und Kleinschreibung wurde nicht unterschieden. Die Ordinate ist logarithmisch skaliert. Die exakten Zahlen sind über den Balken notiert.

Häufigkeiten eingetragen. Hier wurden jedoch ›ä‹, ›ö‹ und ›ü‹ als jeweils zwei Buchstaben ›ae‹, ›oe‹ und ›ue‹ gezählt, was die Abweichungen bei den Vokalen ungefähr erklärt. Groß- und Kleinbuchstaben wurden zusammengefasst, die Ligatur ›ß‹ separat gezählt. (Die Zahlen schwanken je nach verwendetem Korpus recht stark, eine Auflistung älterer Auszählungen findet sich bei [Bau94], Seite206.)

Abbildung 3.7 gibt eine Übersicht über die Häufigkeit, mit der ein bestimmter Buchstabe an einer bestimmten Position auftritt. Die Zahlen an der Ordinate beziehen sich auf die Anzahl der Zeichen im Wörterbuch, nicht im Korpus. Der besseren Übersicht halber wurde die Ordinate logarithmisch skaliert. Auch für diese Darstellung wurden Groß- und Kleinbuchstaben zusammengefasst. Zu erkennen sind Unregelmäßigkeiten, die sich aus den Gesetzlichkeiten der Wortbildung ergeben: das ›e‹ ist zwar generell der häufigste Buchstabe, wird aber in erster Position vom ›s‹ deutlich übertroffen und in dritter Position vom ›r‹. Die Häufung der Vokale in den Spitzenplätzen an zweiter Position überrascht nicht, eher der Peak des ›e‹ an fünfter Position.

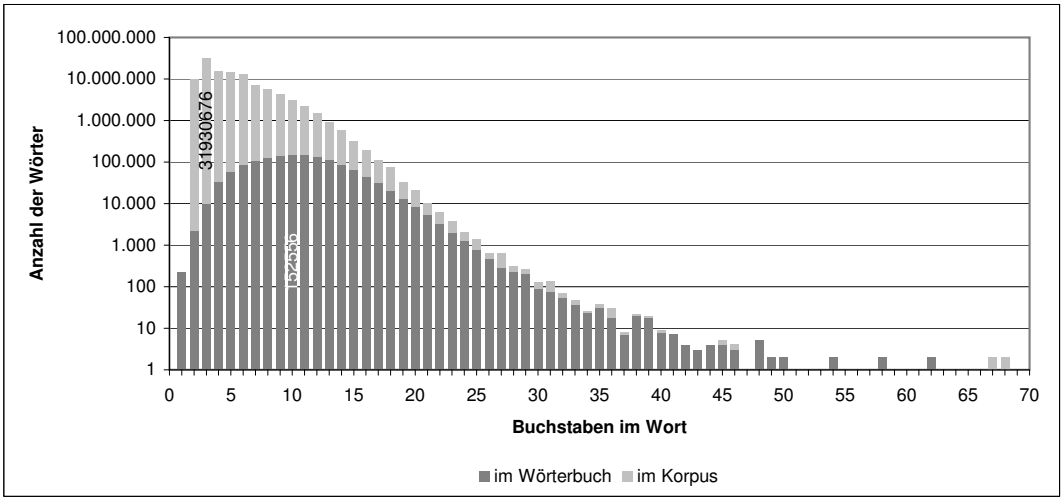


Abbildung 3.5: Wortlänge im Korpus und im Vollformenlexikon. Angegeben sind die Anzahl der Wörter im Korpus (hellgrau) bzw. im Vollformenlexikon (dunkelgrau) mit der angegebenen Anzahl Zeichen. Aufgrund der logarithmischen Skalierung der Ordinate sind Häufigkeiten von eins nicht angegeben. Weitere Erläuterungen im Text.

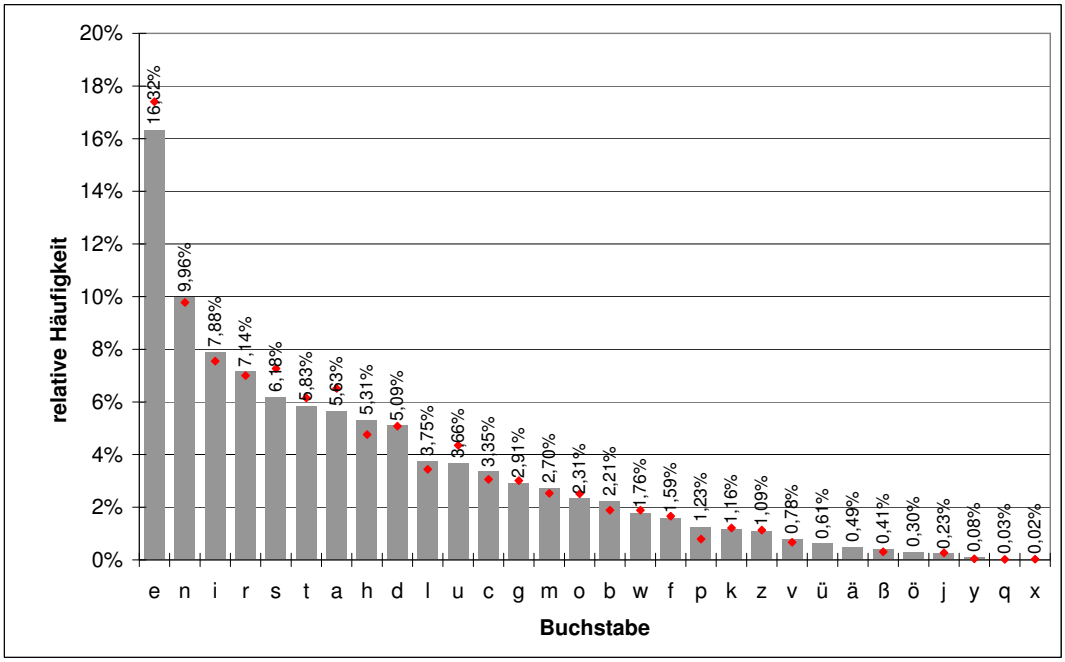


Abbildung 3.6: Häufigkeit der Buchstaben im Korpus

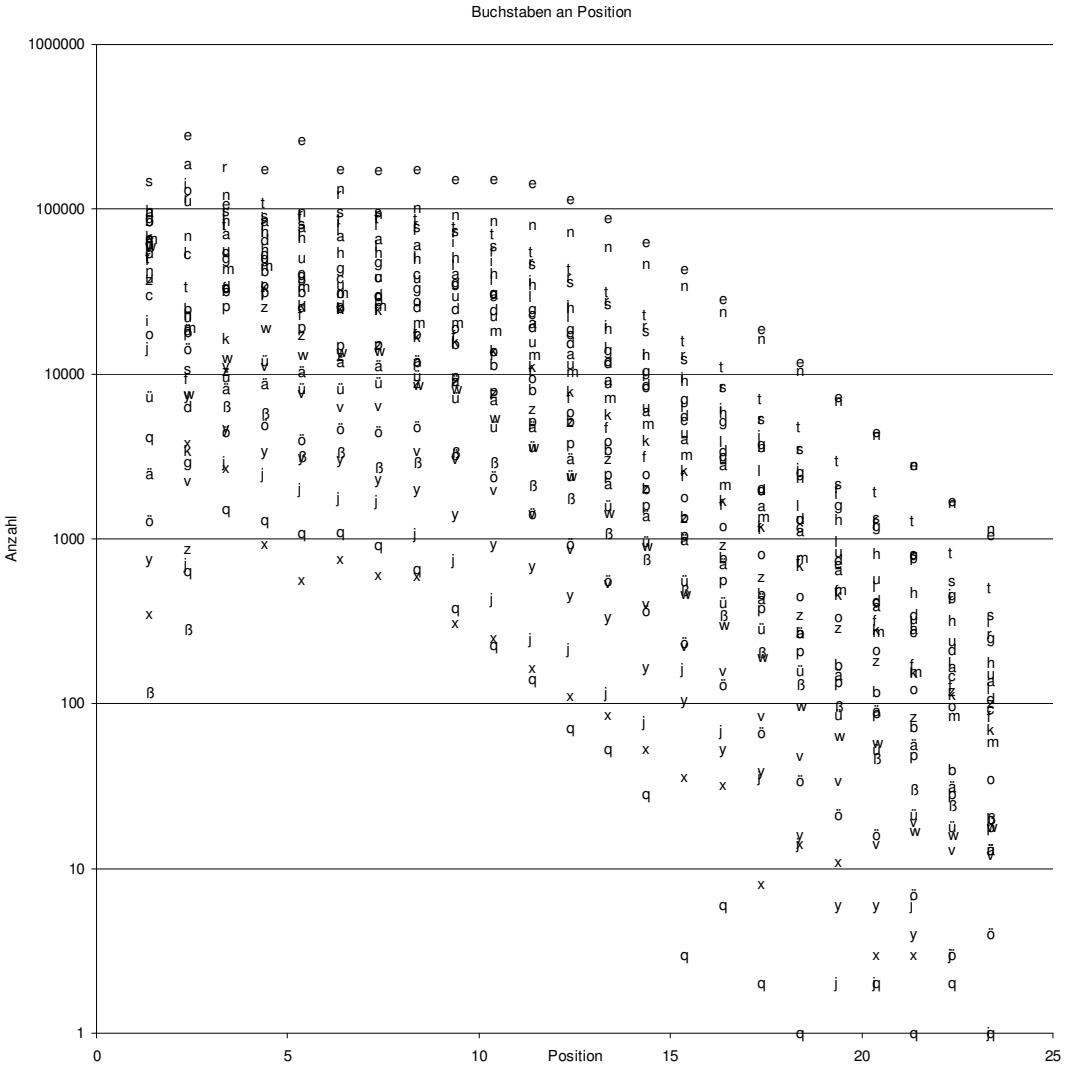


Abbildung 3.7: Buchstabe an Position

3.2.3 Verwendeter Zeichensatz

Alle Daten, die Zeichenketten enthielten, wurden im UCS-2-Format[5] (Little Endian) auf der Festplatte gespeichert. Die verwendete Programmiersprache VisualBasic 2005 speichert Zeichenketten intern im Hauptspeicher immer in diesem Format ab. Abgesehen von dem vergrößerten Platzbedarf auf der Festplatte konnten durch diese Festlegung keine Performanceeinbußen beobachtet werden. Theoretisch ist bei der binären Suche je Suchvorgang ein weiterer Plattenzugriff erforderlich, d. h. bei einer festgelegten Puffergröße von 1 024 Bytes, einer Datensatzlänge von weniger als 128 Zeichen und einer Größe der zu durchsuchenden Datei von 64 MB müssen nicht mehr als $\log_2(2^{26}/7) \approx 25$ Puffer eingelesen werden, um den gesuchten Datensatz in einer sequentiell geordneten Datei zu finden, wenn ein Zeichen mit einem Byte codiert wurde. Wurde ein Zeichen mit zwei Bytes codiert, dann passen nicht sieben sondern nur drei Datensätze in den Puffer und es müssen $\log_2(2^{26}/3) \approx 26$ Puffer eingelesen werden.

Mit einer festen Länge von 16 Bit können maximal 2^{16} = 65 536 Zeichen codiert werden. Unicode erlaubt die Codierung weiterer Zeichen, indem statt zwei Bytes vier Bytes verwendet werden. Diese Zeichen heißen *Supplementary Characters*, die zugehörigen Codepoints heißen *Supplementary Code Points* und liegen im Bereich zwischen U+10 0000 und U+10 FFFF. In der Codierung Unicode-16 werden diese Zeichen dargestellt durch ein *Surrogate Pair*, bestehend aus einer *high-surrogate code unit* mit Werten aus dem Bereich $D800_{16}$ bis $DBFF_{16}$ und einer *low-surrogate code unit* mit Werten aus dem Bereich $DC00_{16}$ bis $DFFF_{16}$. Solche *Supplementary Characters* kamen im Korpus nicht vor, tatsächlich wurden im Korpus Zeichen aus dem Bereich U+0009 bis U+2323 verwendet.

Im Abschnitt 6.1 auf Seite 128 wird die Problematik der verschiedenen Codierungen näher betrachtet.

3.3 Aufbereitung des Korpus (HTML-Normalisierung)

Das Zeichen ›é‹ kann wahlweise definiert sein als

```
&eacute;
&#233;
```

[5] Das *Universal Character Set* (UCS) ist eine Zeichenkodierung, die in der internationalen Norm ISO/IEC 10646 definiert ist. Aufgrund der Beschränkung auf 16 Bit können nur 2^{16} Zeichen codiert werden. Aufgrund der gewünschten Kompatibilität mit UNICODE wird darauf verzichtet, die 2^{11} möglichen Werte zu nutzen, die in UTF-16 für die *surrogate code points* im Bereich U+D800 bis U+DFFF genutzt werden sowie die Werte U+FFFE und U+FFEF. Dies entspricht der *Basic Multilingual Plane*. Deshalb können mit dieser Codierung nur 63486 Zeichen dargestellt werden. Für alle praktischen Belange im europäischen Schriftraum ist dies dasselbe wie Unicode.

```
&#xE9;      (hexadezimal)
&#x00E9;    (hexadezimal)
&#769;e     (Akzent acute, e)
&#x0301;e   (Akzent acute, e)
e&#714;     (Kleines e mit Strich drüber nach rechts oben)
e&#x02CA;   (Kleines e mit Strich drüber nach rechts oben)
```

In der ersten Definition wird das benannte Entity verwendet, in den beiden folgenden wird ein numerisches Entity verwendet, in der fünften bis siebten Definition wird ein einzelnes Unicode- oder ANSI-Zeichen durch eine Kombination zweier Zeichen dargestellt: ein Akut mit dem Grundbuchstaben ›e‹.

HTML bedeutet, dass der Text mit verschiedensten Tags markiert wird. Da es in dieser Arbeit um die Verbesserung der Qualität von Precision und Recall geht, müssen diese Tags differenziert betrachtet werden.

Tags in HTML werden durch spitze Klammern (`<...>`) eingefasst. Will man spitze Klammern im Text erscheinen lassen, dann sollten diese durch die HTML-Entities `<` und `>` codiert werden. Dies ist leider nicht immer der Fall.

Im Suchergebnis soll Text, der vom Browser nicht angezeigt wird, auch nicht angeführt werden. Umgekehrt soll Text, den der Browser anzeigt, auch gefunden werden können. Deshalb werden im Rahmen der Vorverarbeitung Tags entfernt. Dazu sind die Tags in verschiedene Gruppen einzuteilen. Zunächst können alle Metadaten entfernt werden. Die Suche nach Dokumenten, die anhand von Metadaten indexiert wurden, ist nicht Gegenstand dieser Arbeit. Zur Indexierung sollte deshalb ausschließlich die zwischen den Tags `<BODY>` und `</BODY>` stehende Zeichenkette verwendet werden.

Eine genauere Überprüfung der verarbeiteten Dateien zeigt jedoch, dass dieser Ansatz zu kurz greift: Der Spezifikation entsprechend kann das Tag mit Attributen erweitert sein. Das einleitende Tag kann beispielsweise auch `<body color=ff00cc>` lauten. Dies ist bei 258 von 63516 betrachteten Dateien der Fall. Aber auch wenn auf die schließende Klammer verzichtet wird und als Start-Tag das Token `<BODY` verwendet wird, werden nicht alle relevanten Dokumente eingeschlossen, bei 37 Dateien fehlte ein einleitendes `<BODY>`-Tag. Von diesen 37 Dateien enthalten 9 tatsächlich keinen Text, weil sie *Frames*[6] beschreiben. Drei Dateien werden als *temp.htm* bezeichnet. Deren Inhalt erscheint unter einem anderen Dateinamen im gleichen Verzeichnis. Bei 25 Dateien fehlt das `<BODY>`-Tag tatsächlich und sie enthalten relevanten Inhalt. In diesen Fällen wird das Dokument von

[6] Ein *Frame* ist ein Teilbereich einer HTML-Seite, in dem eine andere HTML-Seite dargestellt werden kann. Das einzelne Segment wird dabei als *Frame* (dt. Rahmen) bezeichnet, die Definition aller *Frames* als *Frameset*.

den verwendeten Browsern dennoch richtig dargestellt. Die Dokumente beginnen mit der Spezifikation <!DOCTYPE HTML oder auch dem Tag <HTML>, dann folgt entweder ein Head-Block, das Tag <TITLE> oder auch direkt eine mit <H1> oder auch mit <H2> ausgezeichnete Überschrift.

Ebenso, wie der Dokumentenbeginn gelegentlich fehlerbehaftet ist, schließen nicht alle Dateien regelkonform. In 46 Dateien fehlt das schließende </BODY>-Tag, obwohl der Dokumenteninhalt mit einem <BODY>-Tag eingeleitet wurde.

Auch in einem regelkonformen Dokument gibt es Text außerhalb von Tags, der nicht zum eigentlichen Inhalt des Dokuments gehört. Die Tags <SCRIPT> und </SCRIPT> fassen Programmcode ein, der auch im Body eines Dokumentes stehen darf, aber nicht Gegenstand des Retrievals sein soll.

Innerhalb der zu indexierenden Zeichenkette gibt es Tags, die Absätze markieren und durch ein entsprechendes Zeichen ersetzt werden können. Andere Tags müssen durch *Whitespace*, in diesem Fall ein Leerzeichen, ersetzt werden. Dann gibt es Tags, die ersatzlos gestrichen werden sollten und schließlich gibt es fehlerhafte Auszeichnungen und Zeichenketten, die wie HTML-Tags aussehen, aber als normaler Text erscheinen sollen.

<embed> gehört nicht zum HTML-Standard, sondern ist ein Relikt aus Netscape-Zeiten. Damit werden aber auch aktuell noch beliebige Objekte, insbesondere Multimediainhalte eingebettet. Um einen lern- und erweiterungsfähigen Algorithmus zum Suchen und Ersetzen dieser Tags zu bilden, wird folgendes System verwendet.

Als eigentlicher Algorithmus für die Such- und Ersetzfunktion werden reguläre Ausdrücke verwendet. Reguläre Ausdrücke sind eine sehr leistungsfähige Sprache für dieses Problem. Friedl [Fri08] führt umfassend in diese Sprache ein und beschreibt auch die Unterschiede in verschiedenen Dialekten.

Der bedingte Trennstrich (*SOFT HYPHEN*) erfordert eine besondere Behandlung, die auf Seite 130 erläutert wird.

Das Gedächtnis dieses lernfähigen Algorithmus wird als XML-Datei modelliert. Die Abarbeitung der Knoten eines XML-Baumes ist nicht vorhersehbar. Weil die Reihenfolge der Abarbeitung aber wichtig ist (die <SCRIPT>...</SCRIPT>-Abschnitte müssen zuerst entfernt werden), erhalten die Knoten ein Attribut *Priority*, das die richtige Reihenfolge bestimmt.

3.3.1 Vorverarbeitung

Die Originaltexte liegen im HTML-Format vor. Der Umgang mit großen Dateien erfordert zusätzliche Überlegungen. Um so wenig wie möglich auf langsamen Auslagerungsspeicher zugreifen zu müssen, werden die Texte aus den über 63 000

```
<HTML>

<HEAD>
<TITLE>Anzengruber: Der Gwissenswurm, I. Akt, 5. Szene</TITLE>
<meta name="type"      CONTENT="COMEDY">
<meta name="booktitle" CONTENT="Der Gwissenswurm">
<meta name="author"    CONTENT="Ludwig Anzengruber">
<meta name="year"      CONTENT="1986">
<meta name="publisher" CONTENT="Philipp Reclam jun.">
<meta name="address"   CONTENT="Stuttgart">
<meta name="isbn"      CONTENT="3-15-000215-X">
<meta name="title"     CONTENT="Der Gwissenswurm, I. Akt, 5. Szene">
<meta name="pages"     CONTENT="10-11">
<meta name="sender"    CONTENT="gerd.bouillon@t-online.de">
<link rel="stylesheet" HREF="../../css/drama.css" TYPE="text/css">
<meta name="firstpub" content="1874">
</head>
```

Abbildung 3.8: Beispiel eines Dateiheaders

Originaldateien extrahiert und in einer einzigen Sammeldatei zusammengefasst, die den gesamten Text fortlaufend enthält.

Die Metadaten werden nicht übernommen. Die DVD enthält Indexe und Verzeichnisse, die die in den Metadaten enthaltenen Informationen auswerten und auf die passenden Dateien verlinken, so dass die Nutzer auf sie zugreifen können. Der Zugriff über diese Daten ist der typische Zugriff von Bibliothekssystemen und nicht Gegenstand dieser Untersuchung. So wurden alle Ordner mit Metadaten und Bildern übersprungen. Dazu gehören Ordner, die die Zeichenketten `autoren`, `info`, `js`, `suchverl`, `svg` oder `xml` im Ordnernamen tragen. Die meisten Ordner enthalten Dateien, die `0htmldir` im Dateinamen tragen. Diese Dateien enthalten Verweise auf die Dateien des Ordners und sind zur Navigation im Verzeichnisbaum gedacht. Diese Dateien wurden ebenfalls nicht berücksichtigt. Schließlich kann jede HTML-Datei Metadaten enthalten, wie sie in dem in Abbildung 3.8 beispielhaft gezeigten Header zu sehen sind. Diese Daten wurden übersprungen und nur der von den Tags `<BODY` und `</BODY>` eingefassten Zeichenkette gebildete *body* der HTML-Dateien in die Sammeldatei aufgenommen.

Clematide [Cle07] und Heyer et al. [HQW06] empfehlen, in diesem Schritt auch alle Tags zu entfernen. Dabei würde man weitreichende Strukturinformationen verlieren: Bekannt ist das Tag `<P>`: Es markiert Absätze. In ASCII-Text wird es je

nach Betriebssystem durch CR-LF (Windows), LF (Unix) oder CR (Apple) ersetzt. Dies ist aber nicht das einzige absatzbildende Tag.

Tabelle 7.3 im Anhang auf Seite 172 gibt eine Übersicht über absatzbildende Tags der Spezifikation für HTML 4.01.[7]

Im Unterschied dazu sind in Tabelle 7.4 (Anhang, Seite 173) Tags für die Schriftformatierung aufgelistet. Solche Tags sollten nicht durch Wortgrenzen ersetzt werden, sondern ersatzlos gestrichen werden.

Alle verbleibenden Tags können durch Leerzeichen ersetzt werden.

3.3.2 Reguläre Ausdrücke

Reguläre Ausdrücke sind eine mächtige Sprache, um in Texten Zeichenketten zu suchen und zu ersetzen. Ein profundes Lehrbuch wurde von Friedl [Fri08] erstellt. Ein Testwerkzeug für reguläre Ausdrücke in den .NET-Sprachen VB und C# liefert der RegexDesigner.NET[8]. Für die oben genannten Tags wurden die im folgenden beschriebenen regulären Ausdrücke benutzt.

Als erstes muss der vom `SCRIPT`-Tag eingeschlossenen Stellen entfernt werden. Diese Stellen können beliebigen Programmcode enthalten. Der zugehörige reguläre Ausdruck lautet:

```
(?im)<SCRIPT(|.|\n)*?/SCRIPT>
```
(3.1)

Mit der Zeichenkette `(?im)` werden Optionen eingestellt. Und zwar wird Groß- und Kleinschreibung nicht unterschieden (`i`), und über Zeilenenden wird hinweggegangen (`m`). Es folgt der Text, nach dem wörtlich gesucht wird (`<SCRIPT`) und eine durch die Klammern (`(..)`) markierte Gruppe, durch die nach `\s` oder . oder `\n` gesucht wird. `\s` bezeichnet ein beliebiges *white-space*-Zeichen, der Punkt steht für ein beliebiges Zeichen und `\n` bezeichnet einen Zeilenvorschub. Diese Gruppe kann zwischen null mal und beliebig oft wiederholt (`*`) werden, jedoch nicht öfter, als bis der Parser auf die Zeichenfolge (`/SCRIPT>`) trifft, dafür steht das `›?‹`, das den *lazy*-Modus aktiviert.

Die Gruppe der in Tabelle 7.3 beschriebenen absatzbildenden Tags wird durch diesen regulären Ausdruck erfasst:

```
"(?i)</{0,1}" & _
"(BLOCKQUOTE|BR|CAPTION|CENTER|CITE|CODE|" & _
"COLGROUP|COL|DIR|DIV|DD|DL|DT|" & _
"FIELDSET|FRAMESET|FRAME|" & _
```

[7] Auszug aus http://www.w3.org/TR/html401/index/elements.html

[8] http://www.sellsbrothers.com/tools/

```
"H[1-6]|HR|IFRAME|ISINDEX|LI|MAP|MENU|OL|P|SAMP|" & _
"TABLE|TBODY|TD|TFOOT|THEAD|TH|TITLE|TR|UL)" & _
"(>|(\s[^>]*/{0,1}>))"
```

Hier wird wieder mit `(?i)` als Option festgelegt, dass Groß- und Kleinschreibung nicht berücksichtigt werden soll. Anschließend wird eine öffnende spitze Klammer ›<‹ gesucht, der kein oder ein Schrägstrich folgt (`</{0,1}`). Anschließend muss ein Codewort aus der folgenden Aufzählung folgen. Die Reihenfolge der Wörter ist zunächst beliebig. Wenn in der Aufzählung jedoch zwei unterschiedlich lange, aber gleich beginnende Wörter vorkommen, dann muss das längere zuerst aufgezählt werden. Deshalb muss `COLGROUP` vor `COL` stehen und `THEAD` vor `TH`.

Die oben aufgeführten sechs Zeichenketten werden durch den Unterstrich ›`& _`‹ programmintern zu einer einzigen zusammengefügt.

In der letzten Zeile wird festgelegt, dass der Ausdruck entweder durch eine schließende spitze Klammer ›>‹ abgeschlossen werden soll oder aber durch ein *white-space*-Zeichen (`\s`), dem eine beliebige Anzahl von Zeichen folgen, solange sie keine schließende spitze Klammer sind (`[^>]*`), denen wiederum kein oder ein Schrägstrich, codiert durch (`/{0,1}`), folgt und schließlich eine schließende spitze Klammer.

Dieser erste Schritt des Erstellens der Sammeldatei und der Dateiliste einschließlich der Zeichenersetzung mit den beschriebenen regulären Ausdrücken dauerte auf dem in Abschnitt 3.1 beschriebenen System für die mehr als 63 000 Dateien sieben Minuten.

Um aus dieser Datei auf die Quelldateien zugreifen zu können, wird ein Index erstellt, der als Schlüssel die Position des ersten Zeichens aus der Quelldatei in der Sammeldatei enthält. Dieser Schlüssel verweist auf einen Eintrag, der Pfad und Dateinamen der Quelldatei enthält. Allein der Verzicht auf die Strukturdaten bewirkt eine Datenkompression um 50 %. Trotzdem ist die Sammeldatei mit über 1,3 GB noch zu groß, um sie im Hauptspeicher zu halten.

Je nach Größe des zur Verfügung stehenden Hauptspeichers wird deshalb die Sammeldatei in handliche Stücke aufgeteilt. Die Aufteilung darf jedoch nur an Zeilenendemarkierungen (CRLF[9]) erfolgen, damit durch eine abschnittsweise Bearbeitung nicht zufällige neue Wörter gebildet werden.

Aus der Sammeldatei werden die Wortlisten erstellt. Die Wortlisten enthalten jedes vorkommende Wort und die zugehörige Häufigkeit. Dabei werden zunächst alle Zeichen getrennt betrachtet. So werden für »Das«, »DAS« und »das« drei verschiedene Einträge angelegt.

[9] Carriage Return - Line Feed im Windows-Bereich, Line Feed im Unix-Bereich

Um in der Textsammlung all die Textdateien aufzufinden, in denen mehrere verschiedener Suchwörter gemeinsam auftreten, wurde folgendes Verfahren gewählt:

Die Texte wurden aus dem `BODY` der HTML-Dateien extrahiert. Sämtliche Tags und alle Satzzeichen wurden entfernt und die HTML-codierten Sonderzeichen (»—« [–], »ä« [ä]) mit einer Umsetzungstabelle (vgl. Tabelle 7.1 im Anhang auf Seite 7.1) decodiert. Der so erzeugte Fließtext wird fortlaufend in einer Datei gespeichert und gleichzeitig in einer Indexdatei die Pfade zu den Ursprungsdateien aufgezeichnet. Außerdem wird in der Indexdatei die Position des ersten Zeichens der Ursprungsdatei in der Sammeldatei gespeichert.

In einer weiteren Datei „Wortliste“ wird für jedes unterschiedliche Wort aus der Sammeldatei ein Eintrag erzeugt und die Häufigkeit des Vorkommens gespeichert. Dabei wird nach Groß- und Kleinschreibung unterschieden. Schließlich wird ein Wortindex erzeugt. Dazu wird in der Sortierung der Wortliste fortlaufend jedes Wort der Sammeldatei indexiert. Um die Verweise zu finden, wird deshalb in der Wortliste zu jedem Eintrag die Position gespeichert, ab der das Wort im Index indiziert ist. Der Eintrag über die Vorkommenshäufigkeit in der Wortliste, multipliziert mit der Zahl der Bytes eines Indexeintrags (hier vier für *Integer* in VisualBasic 2005) entspricht deshalb der Differenz zwischen den Positionseinträgen zweier aufeinander folgender Wörter.

Um mehrere Wörter zu finden, müssen nun nur noch die Listen der Indexeinträge der Suchwörter mit Hilfe des Dateiindex verglichen werden. Auch die Forderung, nur Texte mit räumlich benachbarten Suchtermen zurück zu geben, lässt sich so rasch erfüllen.

Dieser Wortindex vermag eine Sammeldatei bis zu 4 Gigabyte Größe zu indexieren. Der Wortindex selbst benötigt bei diesem Vorgehen halb so viel Platz wie die Sammeldatei.

3.3.3 Tokenisierung

Das Ziel des Indizierens ist das Wiederfinden eines Ausdrucks im Korpus und seine hervorgehobene Darstellung im Browser. Dazu muss die Vorverarbeitung prinzipiell reversibel erfolgen: die ermittelten Token sollen im Original wieder aufgefunden werden. Dies ist ein Problem, wenn, wie häufig verlangt, Wortergänzungen vom Tokenizer vorgenommen werden. `An- und Umbauten` kann zu `Anbauten und Umbauten` ergänzt werden, wie aber soll die Zeichenkette `Anbauten` im Originaltext gefunden und ausgezeichnet werden?

Unter Token wird deshalb wie in den Definitionen 2.2.1 auf Seite 20 im folgenden eine Zeichenkette verstanden, die im Originaltext steht und von solchen Zeichen

begrenzt wird, die in der Liste der Worttrenner in Tabelle 7.2 Seite 169 aufgeführt sind.

Eine richtige Reihenfolge der Operationen beim Tokenisieren lautet:

1. HTML-Tags entfernen
2. Komplexe Token isolieren
3. Abkürzungen isolieren
4. Interpunktionen isolieren
5. Getrenntes zusammenfügen
6. An Leerzeichen trennen
7. Markierungen entfernen.

Dies ist nur eine Möglichkeit. Denn ob zuerst die komplexen Token oder die Abkürzungen isoliert werden, spielt keine Rolle. Wichtig ist, dass beide Schritte vor dem Isolieren der Interpunktionen kommen, damit beispielsweise Punkte bei Abkürzungen wie in »usw.« nicht verloren gehen. Auch die Reihenfolge der beiden Operationen „Interpunktionen isolieren“ und „Getrenntes zusammenfügen“ kann vertauscht werden, solange beide Prozesse stattfinden, bevor an Leerzeichen getrennt wird. Zu beachten ist auch, dass zuallererst die HTML-Tags entfernt werden und ganz zum Schluss die Markierungen, die durch das wiederholte Isolieren entstanden sind.[10]

3.4 Transliterationssysteme

Zu unterscheiden sind Transkription und Transliteration: Ein Transkriptionssystem soll eine halbwegs richtige Aussprache eines fremdsprachlichen Wortes ermöglichen für Sprecher, die die Quellsprache und ihr Schriftsystem nicht kennen. Ein Transliterationssystem wurde früher benötigt, wenn Schriftreproduktionssystemen Zeichensätze fehlten. Heute, wo mit Unicode jedes Zeichen dargestellt werden kann, dient ein Transliterationssystem im wesentlichen der Sortierung fremdschriftlicher Wörter in ein System mit eingeschränktem Zeichensatz. Der Duden [Dos96, S. 85] nennt für eine Übertragung griechischer Zeichen in den eingeschränkten lateinischen Zeichensatz ein Transkriptions- und zwei Transliterationssysteme, allerdings unter Zuhilfenahme des Makrons (Längestrichs) (¯) (U+0304) als diakritischem Zeichen. Im englischsprachigen Raum wurde früher das System BGN/PCGN 1962[11] verwendet. Dieses System wurde abgelöst durch das System ELOT 743 1987[12], das

[10] http://arvo.ifi.uzh.ch:9090/clab/tokenizing/set_tokenisierung/index.jsp

[11] United States Board on Geographic Names and the Permanent Committee on Geographical Names for British Official Use

[12] http://www.eki.ee/wgrs/

heute unter anderem von den Vereinten Nationen und der amerikanischen Library of Congress verwendet wird [Dev88, Bar97]. Eine Übersicht über die genannten verschiedenen Systeme gibt [Ped05][13].

Während die bisher genannten Transliterationssysteme auch diakritische Zeichen verwenden und ein griechisches Zeichen je nach Kontext in verschiedene lateinische Zeichen umsetzten, nutzt das System Beta-Code [Gra04] konsequent nur Zeichen aus dem 7-Bit ASCII-Satz und verwendet einen eineindeutige Abbildungsalgorithmus. Beta-Code wird unter anderem genutzt für den Thesaurus Linguae Graecae[14] und das Projekt Perseus[15], beide digitalisieren alte griechische Quelltexte.

Für die unscharfe Suche wird aus Gründen der besseren Performance auf einen eingeschränkten Zeichensatz zurückgegriffen. Deshalb eignet sich der Beta-Code für diesen Anwendungsfall in besonderem Maße. Für die Implementierung wurde jedoch die Verwendung der diakritischen Zeichen ausgeschlossen. Die Probleme der Transformation zwischen Unicode-Zeichensatz und HTML-Codierung bleiben wie auch bei der Codierung der anderen Texte bestehen. Die Tabelle 7.5 auf Seite 174 erläutert die Codierung.

Α Β Γ Δ Ε Ζ Η Θ Ι Κ Λ Μ Ν Ξ Ο Π Ρ Σ Τ Υ Φ Χ Ψ Ω	gamma exceptions
α β γ δ ε ζ η θϑ ι κ λ μ ν ξ ο π ρ σς τ υ φϕ χ ψ ω	γγ γκ γξ γχ
a b g d e z ae th i k l m n x o p r s t u,y ph ch ps o	ng nk nx nch

Abbildung 3.9: Transliterationstabelle Gutenberg

Die (von Gutenberg-DE unabhängige) Site Gutenberg.org, auf die sich aber auch Gutenberg-DE stützt, verwendet eine eigene Transliterationstabelle[16], die in Abbildung 3.9 dargestellt ist.

Für Benutzer des Betriebssystems Mac OS X gibt es eine frei erhältliche Software, die Beta-Code in Unicode umwandelt[17]. Das Ergebnis sieht dann so aus[18] wie in Abbildung 3.10 gezeigt.

[13] http://transliteration.eki.ee/pdf/Greek.pdf
[14] http://www.tlg.uci.edu, ein Projekt der University of California
[15] http://www.perseus.tufts.edu
[16] http://www.gutenberg.org/wiki/Gutenberg:Greek_How-To
[17] http://www.sourcecod.com/sophokeys/
[18] http://www.sourcecod.com/sophokeys/screenshot.png

You type:

mh=nin a)/eide qea\ *phlhi+a/dew *)axilh=os
ou)lome/nhn, h(\ muri/' *)axaioi=s a)/lge' e)/qhke,
polla\s d' i)fqi/mous yuxa\s *)/ai+di proi/+ayen
h(rw/wn, au)tou\s de\ e(lw/ria teu=xe ku/nessin
oi)wnoi=si/ te pa=si, *dio\s d' e)telei/eto boulh/,
e)c ou(= dh\ ta\ prw=ta diasth/thn e)ri/sante
*)atrei/+dhs te a)/nac a)ndrw=n kai\ di=os *)axilleu/s.

You see:

μῆνιν ἄειδε θεὰ Πηληϊάδεω Ἀχιλῆος
οὐλομένην, ἣ μυρί' Ἀχαιοῖς ἄλγε' ἔθηκε,
πολλὰς δ' ἰφθίμους ψυχὰς Ἄϊδι προΐαψεν
ἡρώων, αὐτοὺς δὲ ἑλώρια τεῦχε κύνεσσιν
οἰωνοῖσί τε πᾶσι, Διὸς δ' ἐτελείτο βουλή,
ἐξ οὗ δὴ τὰ πρῶτα διαστήτην ἐρίσαντε
Ἀτρεΐδης τε ἄναξ ἀνδρῶν καὶ δῖος Ἀχιλλεύς.

Abbildung 3.10: Beginn der Ilias

4 Algorithmen und Datenstrukturen

Dieses Kapitel beschreibt die verwendeten Algorithmen und Datenstrukturen, ihre Entwicklung und ihre Optimierungen.

4.1 Architektur der Indexstruktur

Abbildung 4.1 zeigt eine vereinfachte Übersicht über den Ablauf der Indexierung und des Retrievals:

1. Aus den HTML-Dateien des Korpus werden die Texte extrahiert und die HTML-Tags entfernt, gleichzeitig werden in der Dateiliste die Dateipfade der HTML-Dateien und deren Anfänge in der Sammeldatei notiert.

2. Aus der Sammeldatei wird das Vollformenlexikon erstellt.

3. Der invertierte Index wird erstellt. Dazu werden im Vollformenlexikon die Bereiche notiert, in denen die Verweise auf die Fundstellen in der Sammeldatei notiert sind.

4. Auf dem Vollformenlexikon können beliebig viele weitere Indexe erstellt werden. Hier abgebildet ist das SpaCAM (Abschnitt 4.4), aber auch die Suffix-Suche (Abschnitte 2.3.1 und 4.2.2.1) und die Webservices (Abschnitte 4.8 und 4.7) werden an dieser Stelle implementiert.

5. Die Nutzereingabe kann Vorschläge aus dem Vollformenlexikon übernehmen oder als unscharfe Suche oder Teilwortsuche erweitert werden.

6. Mit Hilfe der Dateiliste wird dem Nutzer die Originaldatei aus dem Korpus präsentiert. An dieser Stelle müssen die Mehrdeutigkeiten bei der HTML-Decodierung berücksichtigt werden.

Um eine gute Performance für die Volltextsuche zu erzielen, wurde ein kaskadierter Index aufgebaut. Aus den originalen HTML-Dateien wurde der von den <BODY> </BODY>-Tags eingefasste Inhalt extrahiert und in der Codierung UCS-2[1]

[1] *Universal Character Set* (UCS), Teilmenge von Unicode, vgl. die Fußnote auf Seite 56.

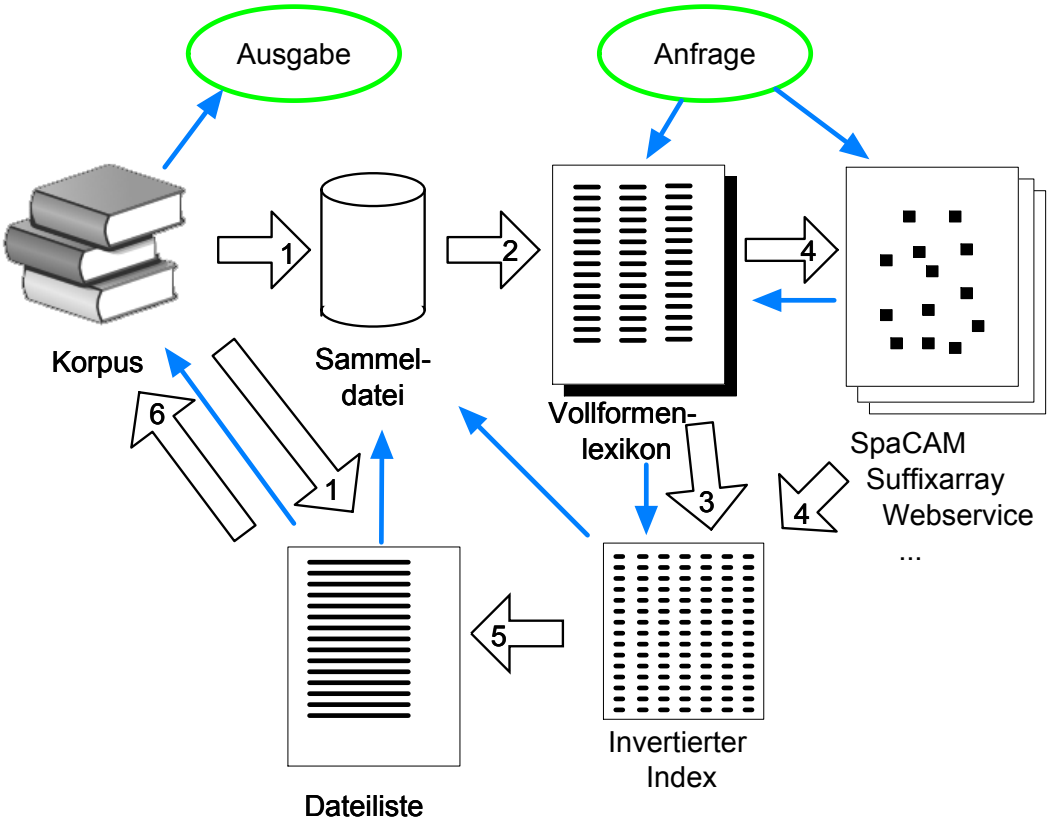


Abbildung 4.1: Vereinfachte Übersicht über den Ablauf der Indexierung und des Retrievals. Die breiten Pfeile bezeichnen die Arbeitsschritte für die Indexierung (1 – 4) und das Retrieval (3 – 6). Die schmalen Pfeile zeigen die Verweise innerhalb der Datenstrukturen.

als fortlaufender Text in einer Datei gespeichert[2]. Diese Datei wird als Sammeldatei bezeichnet.

Gleichzeitig wurde in einer Datei ein Verzeichnis der aufgenommenen Dateien erstellt, in dem für jede aufgenommene Datei der Beginn des Inhalts in der Sammeldatei notiert ist.

Bei der Extraktion des Inhalts wurden entsprechend der Empfehlungen von [HQW06, Cle07] Formatierungstags entfernt. Dabei muss unterschieden werden, ob die Tags absatzbildend sind. Solche Tags sind in Tabelle 7.3 auf Seite 172 aufgeführt und werden gemäß der HTML-Spezifikation 4.01[3] durch Zeilenendemarkierungen ersetzt. Tabelle 7.4 listet Tags für die Schriftformatierung auf. Solche Tags können auch innerhalb von Wörtern stehen und werden deshalb ersatzlos entfernt. Andere Tags werden durch Leerzeichen ersetzt. Das verwendete Korpus enthielt keine Zeichen, deren Codepunkt mit mehr als zwei Byte dargestellt werden musste. Für weitere Indexierungsschritte konnte also von einer konstanten Länge von 16 Bit je codiertem Zeichen ausgegangen werden.

Im nächsten Schritt wurde aus der Sammeldatei eine Vollformenwortliste erstellt. Die Vollformenwortliste ist zu diesem Zeitpunkt als `directory (of string, integer)` organisiert und enthält außer der Wortform als `key` die Häufigkeit der Wortform in der Sammeldatei als `value`. Sie kann vollständig im Hauptspeicher gehalten werden. Jedes Wort der Sammeldatei wird im `directory` nachgeschlagen und wenn nicht vorhanden, eingefügt mit einem `value` (Häufigkeit) von 1 oder, wenn vorhanden, wird `value` um eins inkrementiert.

4.2 Indexierung

Mit diesen Informationen kann in einem weiteren Schritt der invertierte Index mit den Verweisen auf die Vorkommen der Wortformen in der Sammeldatei erstellt werden und die Vollformenwortliste um den Verweis auf den ersten Eintrag im invertierten Index ergänzt werden. Der zur Verfügung stehende Hauptspeicher reichte nicht aus, um den gesamten Index im Hauptspeicher zu halten. In einer trivialen Implementierung, in der jeder Zeiger einzeln auf die Festplatte geschrieben wurde, benötigte die Indexerstellung 150 Minuten CPU-Zeit oder 17 Stunden Rechnerzeit.

Deshalb wurde entsprechend einem Vorschlag von Bansal und Modha [BM04] die CAR[4]-Strategie implementiert, um bei nicht ausreichendem Hauptspeicher

[2] In dieser Richtung ist die Codierung noch prinzipiell eindeutig. Die wenigen Codierungsfehler in der Quelle werden ignoriert.

[3] vergleiche http://www.w3.org/TR/html401/index

[4] Clock with Adaptive Replacement

den Festplattenspeicher als Auslagerungsspeicher effizient nutzen zu können. Es handelt sich hierbei um eine Strategie, schnellen Cache-Speicher – in unserem Fall RAM – durch langsamen Speicher – in unserem Fall Festplattenspeicher – zu ersetzen, ohne allzu große Performanceeinbußen erleiden zu müssen.

4.2.1 CAR-Strategie

Zunächst gehen wir davon aus, dass der Speicher eingeteilt ist in Segmente fester Größe, genannt Seiten (*pages*). Optimal ist eine Seitengröße in der Größe eines I/O-Puffers, z.B. 4 kB. Der Speicher, der die Seiten enthält, heißt *Cache*. Der Cache wird verwaltet durch das *Cachedirectory*. Das Cachedirectory enthält vier doppelt verlinkte Listen: zwei zyklische Ringspeicher ("*CLOCK*"), T_1 und T_2 mit Verweisen auf aktuell im Cache gehaltene Seiten und zwei Listen, B_1 und B_2. Die Listen B_1 und B_2 sind LRU-Listen[5]. Diese Listen verweisen auf nicht mehr im Cache befindliche Seiten. Sie speichern die Erinnerung, dass diese Seite kürzlich im Cache gewesen ist. Die Gesamtgröße des Cache ist durch den verfügbaren Hauptspeicher begrenzt. Die Größen der beiden Speicher T_1, T_2 sind jedoch variabel. Der Algorithmus achtet darauf, dass T_1 und B_2 etwa die gleiche Größe haben und T_2 und B_1. Außerdem darf $|T_1| + |B_1|$ die Cachegröße nicht übersteigen. Die Größen von T_1 und T_2 werden je nach Bedarf ständig angepasst.

Der Algorithmus wurde entsprechend der Beschreibung in Abbildung 4.2 aus [BM04] implementiert.

Wenn eine Seite in B_1 gefunden wird, dann wird die Seite erneut geladen und ihr Verweis von B_1 nach T_1 verschoben und T_1 vergrößert. Falls umgekehrt eine Seite in B_2 gefunden wird, dann wird T_1 verkleinert. Eine Seite, die in B_1 gefunden wird, wird in T_1 unmittelbar hinter dem Ringzeiger am *TAIL* eingefügt. Gleiches gilt für B_2 und T_2. Der Ringspeicher enthält für jede Seite ein *page reference bit*, das für eine neu eingefügte Seite auf 0 gesetzt wird. Wird eine Seite in T_1 oder T_2 gefunden, dann wird deren *page reference bit* auf 1 gesetzt.

Sobald der Ringzeiger von T_1 auf eine Seite mit einem *page reference bit* von 1 trifft, dann wird diese Seite hinter den Ringzeiger von T_2 verschoben und das *page reference bit* auf 0 gesetzt. Trifft der Ringzeiger von T_1 auf eine Seite mit einem *page reference bit* von 0, dann wird diese Seite aus T_1 entfernt und an das MRU-Ende von B_1 angefügt.

[5] Diese Listen haben zwei Enden: neue Objekte werden beim *most recently used* (MRU) Ende eingefügt und wandern durch das weitere Einfügen neuer Objekte zum *least recently used* (LRU) Ende, wo ihr Speicherplatz wieder freigegeben wird. Wird jedoch ein Objekt während seines Verbleibs in der LRU-Liste angefordert, dann wird es der Liste entnommen und wieder am MRU-Ende eingefügt. Die Objekte werden natürlich nicht körperlich im Speicher verschoben, sondern es werden bei diesen Operationen nur Zeiger umgehängt.

INITIALIZATION: Set $p = 0$ and set the lists T_1, B_1, T_2, and B_2 to empty.

CAR(x)
INPUT: The requested page x.
1: if (x is in $T_1 \cup T_2$) then /* cache hit */
2: Set the page reference bit for x to one.
3: else /* cache miss */
4: if ($|T_1| + |T_2| = c$) then
/* cache full, replace a page from cache */
5: replace()
/* cache directory replacement */
6: if ((x is not in $B_1 \cup B_2$) and ($|T_1| + |B_1| = c$)) then
7: Discard the LRU page in B_1.
8: elseif (($|T_1| + |T_2| + |B_1| + |B_2| = 2c$) and ($x$ is not in $B_1 \cup B_2$)) then
9: Discard the LRU page in B_2.
10: endif
11: endif
/* cache directory miss */
12: if (x is not in $B_1 \cup B_2$) then
13: Insert x at the tail of T_1. Set the page reference bit of x to 0.
/* cache directory hit */
14: elseif (x is in B_1) then
15: Adapt: Increase the target size for the list T_1 as: $p = \min\{p + \max\{1, |B_2|/|B_1|\}, c\}$
16: Move x at the tail of T_2. Set the page reference bit of x to 0.
/* cache directory hit */
17: else /* x must be in B_2 */
18: Adapt: Decrease the target size for the list T_1 as: $p = \max\{p - \max\{1, |B_1|/|B_2|\}, 0\}$
19: Move x at the tail of T_2. Set the page reference bit of x to 0.
20: endif
21: endif

replace()
22: found = 0
23: repeat
24: if ($|T_1| >= \max(1, p)$) then
25: if (the page reference bit of head page in T_1 is 0) then
26: found = 1;
27: Demote the head page in T_1 and make it the MRU page in B_1.
28: else
29: Set the page reference bit of head page in T_1 to 0, and make it the tail page in T_2.
30: endif
31: else
32: if (the page reference bit of head page in T_2 is 0), then
33: found = 1;
34: Demote the head page in T_2 and make it the MRU page in B_2.
35: else
36: Set the page reference bit of head page in T_2 to 0, and make it the tail page in T_2.
37: endif
38: endif
39: until (found)

Fig. 2. Algorithm for Clock with Adaptive Replacement. This algorithm is self-contained. No tunable parameters are needed as input to the algorithm. We start from an empty cache and an empty cache directory. The **first key point** of the above algorithm is the simplicity of line 2, where cache hits are not serialized behind a lock and virtually no overhead is involved. The **second key point** is the continual adaptation of the target size of the list T_1 in lines 16 and 19. The **final key point** is that the algorithm requires no magic, tunable parameters as input.

Abbildung 4.2: CAR – der Algorithmus [BM04, S. 7]

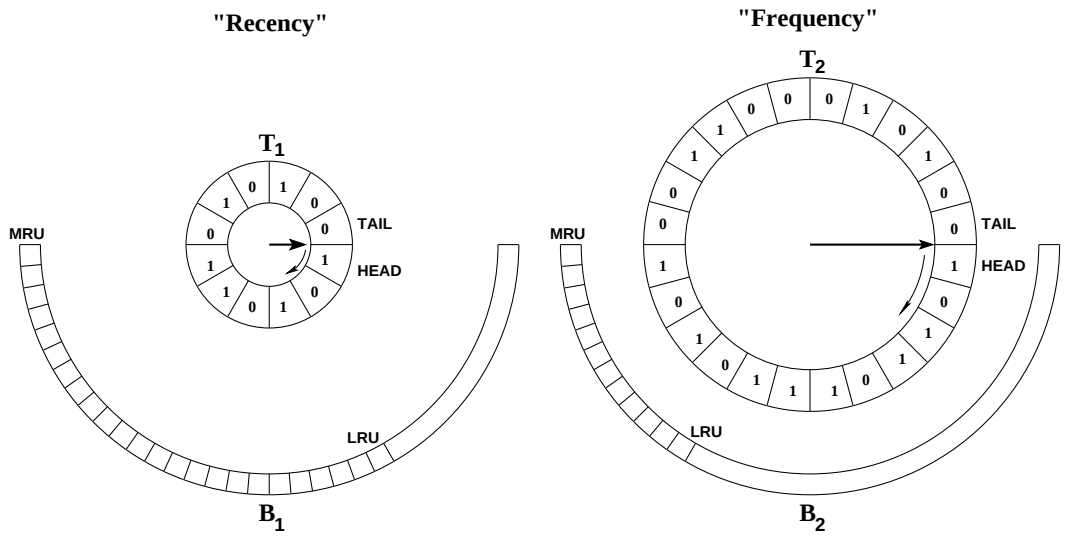


Abbildung 4.3: CAR – CLOCK with adaptive replacement [BM04, S. 5]

Sobald der Ringzeiger von T_2 auf eine Seite mit einem *page reference bit* von 1 trifft, dann wird das *page reference bit*dieser Seite auf 0 gesetzt. Trifft der Ringzeiger von T_2 auf eine Seite mit einem *page reference bit* von 0, dann wird diese Seite aus T_2 entfernt und an das MRU-Ende von B_2 angefügt.

Mit der Reservierung der Maschine für diese Aufgabe, dem temporären Verzicht auf eine Auslagerungsdatei und der oben beschriebenen Speicherstrategie konnte der Zeitbedarf für die erstmalige Indexierung von vielen Stunden auf wenige Minuten reduziert werden.

4.2.2 Weitere Indexe

Für die exakte Suche kann direkt auf den invertierten Index zurückgegriffen werden.

In Abschnitt 6.2 auf Seite 133 werden weitere Möglichkeiten zur Indexerstellung diskutiert.

Von diesen Dateien ausgehend können nun weitere Indexdateien erstellt werden.

Da im invertierten Index auch Groß- und Kleinschreibung differenziert wurde, wurde für die Suche ohne Berücksichtigung der Groß- und Kleinschreibung ein weiterer Index parallel zum in Abbildung 4.1 in Schritt 4 abgebildeten SpaCAM gebildet.

4.2.2.1 Suffixarray

Für die Teilwortsuche wurde ebenfalls parallel dazu ein Suffix-Array (vgl. Abschnitt 2.3.1) nach dem *skew*-Algorithmus von [KS03] erzeugt, der in linearer Zeit $\mathcal{O}(n)$ arbeitet. Der Algorithmus ist im Original in C geschrieben und wurde in VisualBasic reimplementiert. Zur Laufzeit- und Speicherplatzoptimierung wurde aus dem Vollformenlexikon eine Textdatei erzeugt, die alle vorkommenden Wortformen in Großbuchstaben – getrennt durch Leerzeichen – enthielt. In der gewählten Codierung UCS-2 (vgl. Abschnitt 6.1 auf Seite 128ff) benötigte diese Datei 29 MB Speicherplatz. Das darauf aufbauende Suffix-Array benötigte 59 MB Speicherplatz. Die Erzeugung des Suffix-Arrays dauerte auf dem in Abschnitt 3.1 auf Seite 49 beschriebenen System 58 Sekunden.

Für die unscharfe Suche wurde aus Effizienzgründen nicht das gesamte Korpus indexiert, sondern nur das aus dem Korpus erstellte Vollformenlexikon. Nachdem der Benutzer den oder die Suchterme und die zugehörigen Optionen eingegeben hat, werden diese Terme ausgewertet und zu einer Liste der im Index vorhandenen Begriffe erweitert. Diese Liste kann im Sinne des Relevance Feedbacks (vgl. Abschnitt 2.5 auf Seite 47f) vom Benutzer überprüft und gekürzt werden. Anschließend gibt das System eine Liste der Dateien, die die Indexterme enthalten.

4.3 Lernen

Will man die Software nicht ständig neu kompilieren, dann sind lernende Algorithmen vorteilhaft. Eine einfache Art, um dies zu bewerkstelligen, sind solche Algorithmen, die mehrere Freiheitsgrade bieten, die durch Parameter gesteuert werden. Diese Parameter können dann in einer separaten Datei gespeichert werden und die Wirkung des Algorithmus beeinflussen.

Beispielhaft für einen solchen Algorithmus wurde das Modul zur Datenerfassung gestaltet. Die Ausgangstexte liegen im HTML-Format vor und sollen in einer Datei gesammelt werden. Diese Datei soll im *Universal Character Set* mit zwei Byte je Zeichen UCS-2 kodiert sein. Weil diese Datei den reinen Text enthalten soll, müssen aus dem <BODY> der HTML-Dokumente alle Tags entfernt werden.

Der HTML-Standard verändert sich jedoch. Es gibt Tags, die waren gebräuchlich, sind aber nicht im Standard vorhanden, andere Tags umschließen Programmtexte (<SCRIPT>...</SCRIPT>), die nicht indexiert werden sollen. In Zukunft mögen weitere Tags hinzukommen. Deshalb wurde ein Verfahren gewählt, das sowohl programmgesteuert wie auch manuell lernt, Tags zu erkennen und durch welche Zeichenketten sie ersetzt werden sollen (siehe Seiten 58ff).

4.4 SpaCAM

Für die unscharfe Suche wurde das in Hildesheim entwickelte SpaCAM reimplementiert und weiter entwickelt, insbesondere in seiner Leistungsfähigkeit unabhängig von der Hardware um einen Faktor 2 400 beschleunigt. Der Name SpaCAM leitet sich ab von *Sparsely Coded Associative Memory*. In diesem Abschnitt wird nach einigen Begriffsbestimmungen eine Übersicht über Aufbau und Funktion des SpaCAM gegeben.

Definition. Eine Zeichenkette bezeichnet eine endliche Folge von Symbolen aus dem Alphabet Σ, die ohne Zwischenraum hintereinander geschrieben werden.

Das Wörterbuch enthält eine Liste aller Zeichenketten, die im Korpus vorkommen. Zeichenketten im Korpus werden durch ein Symbol aus der Liste der Zeichenkettenbegrenzer D separiert.

Definition. Der Merkmalsraum ist die Menge aller möglichen Zeichenketten der Länge n über dem Alphabet Σ_2.

Definition. Eine Codierung c über den Alphabeten Σ_1 und Σ_2 ist eine Relation der Form

$$c : \Sigma_1^* \to \Sigma_2^n \tag{4.1}$$

Sie ordnet Zeichenketten aus Symbolen des Alphabets Σ_1 Zeichenketten aus dem Alphabet Σ_2 der Länge n zu. Der Stern * bedeutet wie üblich die Kleenesche Hülle. Während die Länge der Zeichenketten im Definitionsbereich von c nicht begrenzt ist, wird die Länge n des Merkmalsvektors durch die Codierung festgelegt.

Diese Relation muss keine eineindeutige Abbildung sein, in unserem Fall der unscharfen Suche reicht es aus, wenn sowohl mehrere verschiedene Zeichenketten auf eine codierte Zeichenketten abgebildet werden, als auch verschiedene codierte Zeichenketten auf eine Zeichenkette aus dem Wörterbuch verweisen. Die Umkehrrelation d: $\Sigma_2^n \to \Sigma_1^*$ wird als Decodierung bezeichnet.

Definition. Ein Merkmalsvektor ist eine Zeichenkette der Länge n aus dem Bildraum von c. In diesem Zusammenhang werden die Zeichen der Zeichenkette auch Merkmale genannt.

4.4.1 Abstandsmaße

Um die Ähnlichkeit verschiedener Zeichenketten zu beschreiben, definiert man ein Abstandsmaß. Einfachstes Beispiel eines solchen Abstandsmaßes ist die Hamming-Distanz [Ham50] Δ_H. Hier werden für zwei Zeichenketten w_1, w_2 gleicher Länge k

die Anzahl der Positionen bestimmt, in denen sich die Symbole $\sigma_{1i} \in w_1$, $\sigma_{2i} \in w_2$ unterscheiden ($i \in \{1, \ldots, k\}$).

$$\Delta_H := \sum_{i=1}^{k} \sigma_{1i} \neq \sigma_{2i} \tag{4.2}$$

Für Zeichenketten verschiedener Länge ist die Hamming-Distanz nicht definiert.

Eine Erweiterung für Zeichenketten unterschiedlicher Länge und das gebräuchlichstes Maß zur Bestimmung des Unterschiedes zwischen zwei Zeichenketten ist die Levenshtein-Distanz, auch als Edit-Distanz bezeichnet [Lev65]. Der Abstand zweier Zeichenketten wird bestimmt durch die Anzahl der Operationen Einfügen, Löschen und Ersetzen von Symbolen, wobei diesen Operationen auch unterschiedliche Kosten zugeordnet werden können.

Eine Verfeinerung der Levenshtein-Distanz ist die Damerau-Levenshtein-Distanz. Im Unterschied zur Levenshtein-Distanz werden zur Berechnung der Damerau-Levenshtein-Distanz für vertauschte Symbole in einer Zeichenkette nicht zwei Ersetzungen berechnet, sondern eine Vertauschung. Schließlich können solche Vertauschungen buchstabenabhängig mit unterschiedlichen Kosten belegt werden [Fri96], um tippfehlerabhängige Vertauschungen geringer zu gewichten.

Weitere sinnvolle Abstandsmaße werden später im Zusammenhang mit den speziellen Codierungen besprochen.

4.4.2 Übersicht

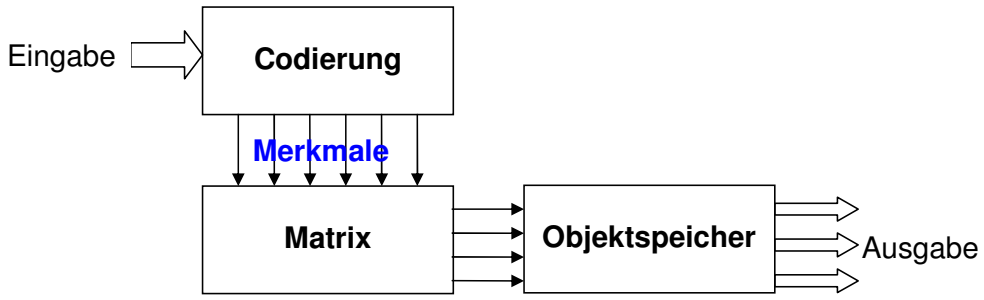


Abbildung 4.4: Ablaufprinzip der unscharfen Suche mittels eines SpaCAMs. Aus einem Objekt der Eingabe werden durch die Codierungsfunktion die binären Merkmale extrahiert. Ein Bitvektor, der die Merkmale repräsentiert, wird mit den in der Matrix gespeicherten Merkmalen aller bekannten Objekte verglichen. In der Matrix implizit gespeicherte Zeiger verweisen auf die bekannten Objekte, die im Objektspeicher abgelegt sind. Die ähnlichsten Objekte werden ausgegeben.

Die Eignung eines SpaCAM für fehlertolerantes Retrieval in einem großen Textcorpus hat bereits Hagström [Hag94, Hag96] festgestellt. In den letzten zwölf Jahren hat sich allerdings das Verständnis von „groß“ in Bezug auf den Korpusumfang verändert: Die beiden von Hagström [Hag96] verwendete Korpora umfassten 9,5 Mio Zeichen für ein Wörterbuch und 34 Mio Zeichen für eine Textsammlung. Das in dieser Arbeit verwendete Korpus umfasst über 600 Mio Zeichen. Ebenso war damals der Hauptspeicher mit 16 MB groß bemessen und ist heute mit 1 GB eher knapp bemessen.

Heitland [Hei94] implementierte eine Reihe von Suchalgorithmen und stellte fest, dass ein SpaCAM in Bezug auf die Suchgeschwindigkeit andere von ihm untersuchte Verfahren deutlich schlug. Hervorzuheben ist dabei die Erkenntnis, dass jedes Suchverfahren optimiert werden kann, wenn die Struktur der zugrunde liegenden Daten bekannt ist. Außerdem sind für unterschiedliches Datenmaterial unterschiedliche Algorithmen optimal. Dabei muss nicht nur nach der Art der Daten unterschieden werden, wie Übereinstimmungen in DNA-Strängen gegen Suche in Texten, sondern Algorithmen können auch für Texte in verschiedenen Sprachen verschieden optimiert werden, weil zum Beispiel die durchschnittliche Wortlänge eines englischen Textes kürzer ist als die eines deutschen Textes.

Entscheidend für die Schnelligkeit der Suche und die Effizienz der Speicherung ist beim SpaCAM eine optimale Codierung. Diese optimale Codierung zu finden ist nicht trivial [Hag96]. Sie hängt ab von den Eingabedaten und deren Struktur und auch von der Suchabfrage [Hei94]. Luba [Lub01] liefert die mathematische Analyse und damit das Rüstzeug für eine effektive Wahl einer geeigneten Codierung.

Viele neuronale Netzwerke erreichen ihre höchste Speicherleistung bei möglichst vollständig besetzen Gewichtsmatrizen. Im Gegensatz dazu verwendet ein SpaCAM (*Sparsely Coded Associative Memory*) eine dünn besetzte Matrix zur Speicherung des Wissens. Das SpaCAM wird ursprünglich als zweischichtiges Netzwerk modelliert, d. h. für Eingabe und Ausgabe existiert je ein Neuron, dass hier jedoch nur ein Speicherbit beansprucht.

Weiterhin gilt:

- Die Propagierungsfunktion der Netzaktivität erfolgt nur vorwärtsgerichtet (*feedforward*).
- Die Lernregel ist Hebb-ähnlich. Es gilt $\Delta w_{ij} = \eta \cdot a_j \cdot o_i$, wobei Δw_{ij} die Veränderung der Verbindungsstärke von Neuron i zu Neuron j ist, η eine geeignet zu wählende Lernrate (ein konstanter Faktor), a_j die Aktivierung von Neuron j und o_i die Ausgabe von Neuron i, das mit Neuron j verbunden ist. [Heb49]
- Es werden ausschließlich binäre Synapsen verwendet.
- Zur Abbildung des Musters auf die Eingangsneuronen werden ähnlichkeitserhaltende Codierungsfunktionen benutzt.

- Ein- und Ausgangscodierung sind spärlich, das heißt, dass der Anteil der aktivierten Neuronen gering ist.
- Die Neuronen der Eingangsschicht sind vollständig mit den Neuronen der Ausgangsschicht verbunden.

Wegen der erforderlichen Codierungsfunktionen können beliebige Muster unscharf aufeinander abgebildet werden. So sind übersetzungs- und thesaurus-ähnliche Funktionen möglich. Das exakte Verhalten des SpaCAM hängt von der Anzahl der Muster und von der Dichte der Einsen im Eingangs- und Ausgangsvektor ab. In der Praxis hat sich gezeigt, dass sich in Bezug auf Speicherausnutzung, Zeitaufwand und Musterdiskriminierung spärliche Codierungen als günstig erweisen. Bei der spärlichen Codierung ist nur ein geringer Teil der Vektoreinträge (möglichst gleichverteilt) auf 1 gesetzt. Weiterhin sollten die Codierungen ähnlichkeitserhaltend sein, um eine fehlertolerante Erkennung zu ermöglichen [SK03]. Eine effiziente Implementierung erfordert deshalb eine intelligente Wahl (oder Modellation) dieser Codierungen.

Da kein Quellcode verfügbar war, wurde das SpaCAM nach den Angaben von [Hei94] neu implementiert. Aufgrund der Größenordnung der zu bearbeitenden Datenmenge mussten die vorgeschlagenen Verfahren zum Teil erheblich modifiziert werden, um akzeptable Antwortzeiten zu erhalten. So konnte die Antwortzeit des Systems durch Änderungen der Datenstrukturen und trotz Verlagerung von Indexen aus dem Hauptspeicher auf die Festplatte von acht Minuten auf weniger als 200 ms reduziert werden.

4.4.3 Aufbau der Matrix

In diesem Abschnitt geht es um den Aufbau der Matrix als zentralem Speicher. Die ebenso wichtigen Codierungen entsprechend Abbildung 4.4 (S. 74) werden im folgenden Abschnitt ab Seite 79 behandelt.

In einer naiven Implementierung enthält die Matrix eine Zeile für jeden Eintrag im Objektspeicher. Damit kann die Nummer der Zeile in der Matrix als Verweis auf den entsprechenden Eintrag im Objektspeicher dienen. Die durch die Codierung festgelegten Merkmale eines Objektes werden als Bitvektor in der Matrix abgelegt. So entspricht jede Spalte einem möglichen Merkmal im Merkmalsraum. Nach Vorstellung von [Rom05] sollten Eingabe- und Ausgabe-Vektor in zwei Zeilen abgelegt werden. Dies ist für die hier zu lösende Aufgabe nicht erforderlich. Alle Objekte des Objektspeichers – hier: des Vollformenlexikons – wurden mit der jeweils verwendeten Codierung in Merkmalsvektoren (vgl. Definition auf Seite 73 überführt und diese in der Matrix zeilenweise abgelegt.

Eine dünn besetzte Matrix wird effizient gespeichert, indem nicht jedes Element explizit gespeichert wird, sondern nur die wenigen nicht leeren Elemente. Diese Elemente werden in Form verketteter Listen gespeichert. Je nach Bedarf kann dies zeilen- und/oder spaltenweise erfolgen in Form einfach oder doppelt verketteter Listen. Im Fall des SpaCAMs reichen schon einfache Zeigerarrays für jede Spalte. Die Zeigerarrays werden auf der Festplatte gehalten und nur bei Bedarf in den Hauptspeicher geladen. Um auf die Spalten zugreifen zu können, wird eine Verwaltungsstruktur aufgebaut, die als *Header* bezeichnet wird.

4.4.4 Der Header

Der Header des SpaCAM enthält die Verweise auf die verwendeten Spalten. Jedes Element der Headers entspricht deshalb einem Merkmal aus dem Merkmalsraum.

Zur Illustration dient das folgende vereinfachte Beispiel:

Variable	Zeichenvorrat	Beschreibung
Σ_1	{a, b, c}	der zulässige Symbolvorrat
D	{.}	Zeichenkettenbegrenzer
Σ_2	$\{0_b, 1_b\}$	Bits
c	$\{a \to 001, b \to 010, c \to 100\}$	die Relation: Transliteration der ersten vier Zeichen in ein verkettetes Bitarray

Die Zeichenkette .abba. wird so zu 001 010 010 001 codiert. Jedes Bit dieses verketteten Bitarrays, des Merkmalsvektors, ist verknüpft mit einem Verweis auf eine Spalte des SpaCAMs. Eine gute Codierung erzeugt eine dünn besetzte Matrix, d. h. die Anzahl der Nullen in der codierten Zeichenkette soll die Anzahl der Einsen deutlich übertreffen. Im nachfolgenden Suchschritt brauchen dann nur noch die Spalten betrachtet werden, für die das entsprechende Bit des Merkmalsvektors eine Eins enthält.

4.4.5 Die Spalten

Jede Spalte des SpaCAMs ist ein einfaches Zeigerarray. Eine Null im Merkmalsvektor wird nicht explizit gespeichert. Wenn das entsprechende Bit des Merkmalsvektors auf Eins gesetzt ist, dann steht in der Liste die Zeilennummer. Jedes Element dieses Zeigerarrays, das nicht leer ist (im Merkmalsvektor für den entsprechenden Zeileneintrag steht eine Eins statt einer Null) enthält als Eintrag die Zeilennummer des Merkmalsvektors.

In diesem Zeigerarray werden aufsteigend sortiert Verweise auf die Einträge im Wörterbuch gespeichert. Eine codierte Zeichenkette stellt sich deshalb im SpaCAM

durch einen Verweis auf einen Wörterbucheintrag in jeder betroffenen Spalte dar. Andere Verweise in den Spalten verweisen auf andere Wörterbucheinträge, die partiell genauso codiert wurden. Werden nun alle Spalten überprüft, die im Header als relevant markiert wurden, dann wird die Zeichenkette im Wörterbuch, die im Sinne der Codierung die größte Ähnlichkeit mit der gesuchten Zeichenkette aufweist, die höchste Anzahl von Verweisen auf sich ziehen.

4.4.6 Arbeitsweise

Das SpaCAM wird im Rahmen der Indexerstellung gefüllt mit den Merkmalsvektoren, die durch die Codierung aus den Objekten im Objektspeicher – den Zeichenketten aus dem Vollformenlexikon – gebildet werden.

Eine Suchanfrage wird auf die gleiche Weise codiert. Dadurch erhält man einen Merkmalsvektor. Für jedes mit einer Eins codierte Bit im Merkmalsvektor wird über den Header die entsprechende Spalte des SpaCAMs gefunden und in den Hauptspeicher geladen. Die Spalte ist ein Zeigerarray. Die Zeiger werden repräsentiert durch Integer, die die entsprechenden Zeilen im Objektspeicher beziffern.

Die Arrays sind aufgrund des Indexierungsverfahrens bereits aufsteigend sortiert. Durch einen einfachen *merge*-Schritt erhält man ein aufsteigend sortiertes Array mit einer Multimenge. Wurden alle Spalten betrachtet, dann erhält man eine aufsteigend sortierte Multimenge, deren Elemente in unterschiedlichen Häufigkeiten vorkommen. Für diese Multimenge wird ein Histogramm erstellt. Das Element oder die Elemente mit den häufigsten Vorkommen zeigen auf die Zeile oder Zeilen im Objektspeicher, die der Suchanfrage im Sinne der Codierung am ähnlichsten sind.

Indem nicht nur die häufigsten, sondern auch die zweithäufigsten oder nachfolgende Elemente als Rückgabewerte akzeptiert werden, kann die Genauigkeit der unscharfen Suche gesteuert werden. Das Verfahren wird in Abbildung 4.7 auf Seite 90 illustriert und in Abschnitt 4.6.2.1 auf Seite 93 näher beschrieben.

Durch diese Unschärfe kann das SpaCAM Fehler beim Lernen als auch bei der Anfrage ausgleichen. Es ist in hohem Maße fehlertolerant. Auch nachträgliche Veränderungen an der Matrix (z. B. durch „Umkippen“ von Bits, technische Defekte, teilweise Zerstörung o. ä.) kann die Matrix in einem gewissen Bereich ausgleichen.

Weitere Details zur Implementierung werden in Abschnitt 4.6 auf Seite 87 besprochen, nachdem im folgenden Kapitel die Codierungen beschrieben wurden.

4.5 SpaCAM-Codierung

Wie bereits erwähnt, ist die Codierung eines SpaCAM von entscheidender Bedeutung für die Qualität des Retrievalergebnisses. Die Codierung ist derjenige Mechanismus, der die Daten in einen binären Vektor transformiert [Hag96].

Vorgelegt seien die Mengen $X := \{0,1,\ldots,255\}$ und $Y := \{0,1\}$.

Definition. Die SpaCAM-Codierung c ist eine Funktion

$$c : X^m \to Y^n \tag{4.3}$$

wobei m die Länge der zu codierenden Daten und n die Länge des Inputvektors beschreibt und somit beide endlich sind.

Die wichtigste Anforderung an die Codierung ist die so genannte ähnlichkeitserhaltende Eigenschaft. Dieses besondere Kennzeichen ist für die Fehlertoleranz unerlässlich und bedeutet, dass mehrere, zueinander ähnliche Textmuster auch auf ähnliche Inputvektoren in Bezug auf die gleiche Position der Einsen abgebildet werden.

Ähnlichkeit muss definiert werden. In Anlehnung an von van Rijsbergen [Rij79] vorgestellte verschiedene Ähnlichkeitsmaße wird aus praktischen Erwägungen für das SpaCAM folgendes Ähnlichkeitsmaß verwendet:

Definition. Seien $\mathbf{y}^a$, $\mathbf{y}^b \in Y^n$ binäre Inputvektoren, $\mathbf{y}^a$ der Eingabevektor mit einer Länge von a Bit, $\mathbf{y}^b$ ein gespeicherter Vergleichsvektor mit einer Länge von b Bit.

Seien $|\mathbf{y}^a| = \sum_{i=1}^{n} \mathbf{y}_i^a$ und $\left|\mathbf{y}^a \cap \mathbf{y}^b\right| = \sum_{i=1}^{n} \min\left(\mathbf{y}_i^a, \mathbf{y}_i^b\right)$.

Für zwei binäre Inputvektoren $\mathbf{y}^a$, $\mathbf{y}^b \in Y^n$ ist der SpaCAM-Koeffizient $SCC(\mathbf{y}^a, \mathbf{y}^b)$ gegeben durch

$$SCC\left(\mathbf{y}^a, \mathbf{y}^b\right) = \frac{\left|\mathbf{y}^a \cap \mathbf{y}^b\right|}{|\mathbf{y}^a|} \tag{4.4}$$

Der SpaCAM-Koeffizient wird also definiert als das Verhältnis der Zahl der Einsen in der UND-Verknüpfung von Eingabe- und Vergleichsvektor zur Zahl der Einsen im Eingabevektor. Hagström [Hag96] benutzte als Eingabealphabet die 40 Zeichen aus dem Zeichenvorrat

$$A = \{\texttt{0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ\#}\}.$$

Kleinbuchstaben wurden zu Großbuchstaben transformiert. Das Zeichen ›#‹ zeigte die Wortgrenzen an und wurde vor und hinter jedes Wort gestellt. Er verwendete

eine Triplettcodierung, die je drei aufeinander folgende Zeichen auf eine Eins im Eingabevektor abbildet. Die Codierungsfunktion c_T lautet

$$c_T : \mathrm{X}^k \to \mathrm{Y}^{|A|^3}, \text{ wobei } \forall j \in \{0, \ldots, |A|^3 - 1\} \text{ gilt:} \tag{4.5}$$

$$\mathbf{y}_j := \begin{cases} 1, \text{falls } j = |A|^2 \cdot \mathbf{x}_i + |A| \cdot \mathbf{x}_{i+1} + \mathbf{x}_{i+2}, \quad i = 0, \ldots, L(\mathbf{x}) - 3 \\ 0 \text{ sonst} \end{cases} \tag{4.6}$$

Bei einer angenommenen maximalen Wortlänge von 10 erhält man so 10 Tripletts, die als Merkmale aufgefasst werden. Ein Triplett hat eine von $40^3 = 64\,000$ Ausprägungen, was für 10 Tripletts 640 000 Spalten[6] des SpaCAM erfordert. Die Triplettcodierung kann jedoch zu Problemen führen, ähnliche Wörter hinreichend ähnlich abzubilden. Dieser Punkt wird in Abschnitt 4.5.1.2 näher betrachtet.

Die Codierung soll ähnlichkeitserhaltend sein. Im Folgenden wird diese Forderung durch einige Kriterien näher spezifiziert.

1. Die Gleichheit von Eingabe- und Vergleichsvektor erhält den höchsten Score. Unterschiede bewirken Abzüge.
2. Änderungen in der Flexion eines Wortes sollen nur einen geringen Abzug bewirken.
3. Buchstabenvertauschungen sollen nur einen geringen Abzug bewirken.
4. Typische Fehler der Texteingabe über die Tastatur (m | n, ö | ä) sollen nur einen geringen Abzug bewirken.
5. Für Texte aus OCR-Verfahren sind manche Buchstaben oder Buchstabenkombinationen sehr schwer zu unterscheiden: In Groteskschriften unterscheiden sich das kleine l und das große I je nach Schriftart fast oder gar nicht (l | I), rn und m sind ebenfalls kaum zu unterscheiden.
6. Wünschenswert ist, dass Vorsilben, die den Bedeutungsgehalt nicht wesentlich ändern (ge-), die Ähnlichkeit nicht beeinflussen.
7. Wünschenswert ist, dass Stammvokaländerungen durch Flexion starker Verben die Ähnlichkeit nicht beeinflussen (lauf – lief).

Im SpaCAM werden lediglich Zeiger gespeichert. Zur Suche im SpaCAM wird das Suchwort nach dem gleichen, vom SpaCAM unabhängigen Algorithmus codiert,

[6] In der Originalarbeit sind 64 000 Spalten angegeben, ein Rechenfehler.

mit dem die Inhalte des SpaCAMs gelernt wurden. Die Qualität dieser Codierung entscheidet über die Qualität des Retrievalergebnisses. Verschiedene Codierungen wurden implementiert und überprüft.

4.5.1 Spezielle Codierungen

Das Ausgabealphabet Σ_2 sei definiert durch $\{0_b, 1_b\}$.

Aus Effizienzgründen wird bei den meisten Codierungen die Länge der codierten Zeichenkette im Bildraum der Codierung c konstant gewählt. Wenn die zu codierenden Zeichenketten kürzer sind als der zur Verfügung stehende Platz, dann werden die nicht vorhandenen Zeichen durch Nullen dargestellt und belegen bei der gewählten Repräsentation der Matrix keinen Speicherplatz. Längere Eingabezeichenketten werden abgeschnitten und vergrößern die Treffermenge bei der ungenauen Suche.

Die im folgenden untersuchten Codierungen codieren eine Zeichenkette in einen Bitarray fester Länge. Die Länge dieses Bitarrays unterscheidet sich von Codierung zu Codierung. Als Abstandsmaß wird die Hamming-Distanz der Bitarrays verwendet.

4.5.1.1 Standardcodierung

Eine einfache Form der Codierung weist jedem Buchstaben $\sigma \in \{A, ..., Z\}$ einen Binärwert $c(\sigma) \in \{00001_b, ..., 11010_b\}$ zu. Groß- und Kleinbuchstaben werden nicht berücksichtigt. Diakritische Zeichen werden ignoriert, aus ›ä‹ wird zunächst ›A‹, das zu 00001_b codiert wird. ›ß‹ kann zu ›SS‹ umgewandelt werden. Die Zeichenkette wird zu einer Zeichenkette konstanter Länge von 10 oder 20 Symbolen gekürzt oder mit Leerzeichen verlängert. Das Leerzeichen wird mit 00000_b codiert. Die Binärwerte werden als Bitarrays konstanter Länge in der Reihenfolge der Symbole der Eingabezeichenkette verkettet. So erhält man einen Bitvektor mit $10 \cdot 26 = 260$ oder $20 \cdot 26 = 520$ Elementen.

Tabelle 4.1: Vorverarbeitung für die Codierungen Standard10 und Standard20.

	Standard10	Standard20
Abendständchen	ABENDSTAND	ABENDSTANDCHEN

Ein Tippfehler in Form eines fehlerhaften Zeichens führt bei dieser Codierung zu einem geringen Abstandsmaß von 1. Ein fehlendes Zeichen zu Beginn der Zeichenkette bewirkt jedoch wegen der Verschiebung der anschließenden Teilzeichenfolge eine maximale Unähnlichkeit.

In einer versuchsweise optimierten Form können vorab weitere Ersetzungen im Sinne einer kaskadierten Codierung vorgenommen werden. Entsprechend einer rudimentären Stammbildung wurden die Endungen {CHEN | EN | ER | HEIT | KEIT | S | UNG} bei allen Wörtern mit mehr als 8 Zeichen Länge entfernt. Der Buchstabe ›E‹ wurde entfernt. ›TH‹ wurde durch ›T‹ ersetzt, ›SS‹ und ›ß‹ durch ›S‹, ›ä‹, ›ö‹, ›ü‹ durch ›A‹, ›O‹ und ›U‹. Diese Codierung ist in den Ergebnissen mit Optimiert10 (zehn berücksichtigte Symbole) bzw. Optimiert20 (zwanzig berücksichtigte Symbole) gekennzeichnet.

Tabelle 4.2: Vorverarbeitung für die Codierungen Optimiert10 und Optimiert20.

	Optimiert10	Optimiert20
Abendständchen	ABNDSTAND	ABNDSTAND

4.5.1.2 *n*-Tupel-Codierung

Hier wird die Eingabezeichenkette in Tupel von n aufeinander folgenden Buchstaben zerlegt. Aus praktischen Gründen werden für n gewöhnlich nur die Zahlen 2 oder 3 gewählt. Vorteilhaft ist es, wenn Wortanfang und -ende durch ein zusätzliches Symbol markiert werden.

Das Alphabet Σ_2 enthält also ein Symbol für alle n-Tupel, die sich aus den verwendeten Symbolen der Eingabezeichenkette plus einem Symbol für die Zeichenkettenbegrenzung bilden lassen. Tabelle 4.3 auf Seite 83 listet die untersuchten Varianten der n-Tupel-Codierungen auf.

Das Alphabet A_1 := {`@ABCDEFGHIJKLMNOPQRSTUVWXYZ`} besteht aus den Buchstaben A bis Z und einem Zeichen `@` für Wortanfang und Wortende.

Beim Alphabet $A_2 := A_1 \cup$ {`ÄÖÜ0123456789`} werden zusätzlich die Ziffern und {`ÄÖÜ`} codiert. Groß- und Kleinschreibung wird im Fall der unscharfen Suche nicht unterschieden. Kürzere Wörter werden bei rechtstrunkierenden Codierungsvarianten rechts mit Leerzeichen aufgefüllt, bei linkstrunkierenden Varianten links.

Wir untersuchten zwei kurze Varianten, hier als Bigramm10 und Trigramm10 bezeichnet, bei denen das Eingabealphabet A_1 verwendet wurde und es wurden fünf lange Varianten untersucht, bei denen das Eingabealphabet A_2 verwendet wurde. Im ersten Fall wurden nur die ersten 10 Zeichen codiert, im zweiten Fall wurden 20, 30 oder 40 Zeichen codiert, falls das Wort die entsprechende Länge aufwies. Kürzere Wörter wurden mit Leerzeichen aufgefüllt.

Tabelle 4.3: Varianten der n-Tupel-Codierungen

Codierung	Tupelgröße n	Alphabet	Anzahl codierter Zeichen	Trunkierung	Länge des Bitvektors	Beispiel für die Vorverarbeitung `Hühneraugen-essenzbereitungs-versuche`
Bigramm10	2	A_1	10	rechts	$10 \cdot 27^2 = 7\,290$	`HUHNERAUGE`
Trigramm10	3	A_1	10	rechts	$10 \cdot 27^3 = 19\,683$	`HUHNERAUGE`
Bigramm20	2	A_2	20	rechts	$20 \cdot 40^2 = 32\,000$	`HÜHNERAUGENESSENZBER`
Trigramm20	3	A_2	20	rechts	$20 \cdot 40^3 = 1\,280\,000$	`HÜHNERAUGENESSENZBER`
Bigrammreverse20	2	A_2	20	links	$20 \cdot 40^2 = 32\,000$	`NZBEREITUNGSVERSUCHE`
Bigrammreverse30	2	A_2	30	links	$30 \cdot 40^2 = 48\,000$	`RAUGENESSENZBEREIT UNGSVERSUCHE`
Bigrammreverse40	2	A_2	30	links	$40 \cdot 40^2 = 64\,000$	`HÜHNERAUGENESSENZ BEREITUNGSVERSUCHE`

Alphabet A_1 :={`@ABCDEFGHIJKLMNOPQRSTUVWXYZ`}
Alphabet A_2 :={`@ABCDEFGHIJKLMNOPQRSTUVWXYZÄÖÜ0123456789`}

Bei den Varianten Bigramreverse30 und Bigrammreverse40 wurden 30 bzw. 40 Zeichen codiert, falls das Wort die entsprechende Länge aufwies, anderenfalls wurde das Wort linksseitig mit Leerzeichen aufgefüllt.

In allen Fällen wurde ›ß‹ durch ›`SS`‹ ersetzt und bei akzentuierten Vokalen der Akzent entfernt. Andere Zeichen wurden aus der Eingabezeichenkette entfernt. Für die im Korpus recht häufigen griechischen Wörter in griechischer Schrift kann das griechische Alphabet durch ein Transliterationssystem wie dem Beta-Code[7] (vgl. Tabelle 7.5 auf Seite 174) in Zeichen des Eingabealphabets transliteriert werden.

Üblicherweise werden die Eingabezeichenketten rechts abgeschnitten, wenn sie länger sind als die Codierung das vorgibt. In den Varianten Bigrammreverse(20, 30,

[7] http://www.tlg.uci.edu/BetaCode.html
http://www.cs.utk.edu/~{}mclennan/OM/Beta-codes.html

40) wurde der Effekt einer Linkstrunkierung untersucht. Immerhin werden die meisten Komposita im Deutschen gebildet, indem eine differenzierende Attributierung dem Objektbegriff vorangestellt wird.[8]

Die Tabellen 7.7 und 7.6 im Anhang listen lange Wörter aus dem Korpus auf.

4.5.1.3 Konsonantische Codierung

Vokale transportieren aufgrund ihrer höheren Häufigkeit tendenziell weniger Informationen im Sinne der theoretischen Informatik (gemessen in der Einheit Shannon nach ISO 2382-16, Kurzzeichen Sh) und sind eher von einer Lautverschiebung betroffen als Konsonanten[9]. (Abbildung 3.6 enthält die Verteilung der Buchstaben im Korpus). Der Bitvektor des Bildraums enthält deshalb ein Bit für jede mögliche Kombination aus zwei Konsonanten und anschließend je ein Bit für jeden Vokal. Diakritische Zeichen werden wieder ignoriert. Die Position eines Zeichens in der Zeichenkette wird bei dieser Codierung nicht direkt abgebildet. Partiell wird sie durch die Folge der Konsonanten-Bigramme dargestellt. Die Länge des Bitvektors beträgt $(26 - 5)^2 + 5 = 446$.

Um Konsonanten noch stärker zu gewichten, können bei der Abstandsberechnung die Hammingdistanzen für den Teilbereich der Konsonantenkombinationen stärker gewichtet werden als die Distanzen in dem Teilbereich mit den Vokalen. In der Variante KonsonatischPlus wurde außer der oben beschriebenen Kombination aus zwei aufeinander folgenden Konsonanten eine Kombination aus jedem Konsonanten mit seinem Nach-Nachfolger codiert. Das Wort »Arabeske« wurde also in der einfachen Version mit {.. bs .. rb .. sk .. a .. e ..} codiert und in der erweiterten Version mit {.. bs .. rb .. sk .. bk .. rs .. a .. e ..} codiert. Die konsonantische Codierung kann als eine Variante der Bigramm-Codierung aufgefasst werden.

[8] Die Fälle, in denen dies anders ist, werden eher als Zusammenrückung oder Amalgamierung bezeichnet, wie beispielsweise »Taugenichts« oder »Nimmersatt« [Buß02]. (Man unterscheidet hier auch noch zwischen dem Kopf, dem Wortteil, das die grammatischen Eigenschaften festlegt und dem Kern, dem Wortteil, das das semantische Zentrum des Wortes bestimmt. In der Regel fallen jedoch Kopf und Kern zusammen.)

[9] Lsst mn d Vkl n nm Txt wg, wrd dsr schwrr lsbr, br ncht nlsrlch.
Lässt man die Vokale in einem Text weg, wird dieser schwerer lesbar, aber nicht unleserlich.

Tabelle 4.4: Vorverarbeitung für die Codierungen Konsonantisch und KonsonantischPlus

	Konsonantisch	KonsonantischPlus
Abendständchen	BNDSTNDCHNAE	BNDSTNDCHNAE
Codierte Symbole	BN ND DS ST TN DC CH HN A E	BN BD ND NS DS DT ST SN TN TD DC DH CH CN HN A E

Im Bitarray ist für jedes codierte Symbol ein Bit gesetzt. Die ursprüngliche Reihenfolge der Bigramme und der Vokale ist im Bitarray nicht erkennbar. Die Codierung KonsonantischPlus enthält außer den Konsonantenbigrammen der Codierung Konsonantisch zusätzlich Bigramme aus einem Konsonanten und einem konsonantischen Nach-Nachfolger.

4.5.1.4 Kölner Phonetik

Die Kölner Phonetik[10] ist ein vom Soundex-Verfahren[11] abgeleitetes und an die deutsche Sprache angepasstes Verfahren, um Personennamen, die mündlich übertragen werden, fehlertolerant zu codieren. Das Verfahren wurde hier in der von Gollmer[12] implementierten Version untersucht. Obgleich die Qualität des Retrievals nicht sonderlich gut ist, wird es hier, weil weit verbreitet, zu Vergleichszwecken mit überprüft.

Tabelle 4.5: Vorverarbeitung für die Codierungen Soundex und Kölner Phonetik

	Soundex	Kölner Phonetik
Abendständchen	B532	016826846

Die Soundex-Codierung ist nur der Vollständigkeit halber aufgeführt und wurde nicht untersucht.

4.5.2 Weitere Codierungen

Außer den beschriebenen Codierungen lassen sich beliebig viele andere vorstellen. Wörter können anhand ihrer Länge, ihrer Ober- und Unterlängen, weiterer visueller und vieler anderer Merkmale beschrieben werden. Ergebnisse neurolinguistischer Forschung können hier weitere Möglichkeiten aufzeigen. Coltheart und andere

[10] http://de.wikipedia.org/wiki/K%C3%B6lner_Phonetik

[11] http://de.wikipedia.org/wiki/Soundex
Das Soundex Verfahren, beschrieben in [Knu73], wurde von Margaret K. Odell und Robert C. Russell entwickelt und patentiert (U.S. Patents 1261167 (1918), 1435663 (1922)).

[12] http://www.vbarchiv.net/tipps/details.php?id=1754

[CRP+01, ZPC00] modellieren ein *Dual Route Cascaded Model of Visual Word Recognition and Reading Aloud* (*DRC*). Ausgehend von speziellen Eigenschaften der verwendeten Schriftart werden Einheiten modelliert, die zunächst diese Eigenschaften abbilden, dann zu Einheiten für die verwendeten Buchstaben führen, um später über Merkmale wie Buchstabenhäufigkeiten zu Einheiten zu führen, die Silben repräsentieren, aus denen dann die Wörter gebildet werden. Jeder dieser Schritte kann in dem von den Autoren verwendeten neuronalen Modell Rückwirkungen auf den vorangegangenen Schritt haben [CDJB77].[13]

4.5.3 Qualitätskriterien für Codierungen

Auf der Suche nach einer Codierung, die sowohl ein erwünschtes hohes Recall als auch eine hohe Precision liefert, sind Prädiktoren wünschenswert. Die Analyse der untersuchten Codierungen liefert dafür Hinweise.

4.5.3.1 Anzahl der verschiedenen Codes

Das einfachste Maß ist die Angabe, wie viel mögliche Ausprägungen ein Merkmal annehmen kann. Dieses Maß hat jedoch nur geringe Aussagekraft, es erlaubt bestenfalls eine Aussage über eine maximale obere Schranke der Auflösung. So erlaubt eine Trigrammcodierung mit einem Alphabet von 26 Zeichen $26^3 = 17576$ verschiedene Kombinationen, von denen jedoch nur wenige tatsächlich vorkommen. Wenn je Eintrag im Wörterbuch 10 Merkmale gespeichert werden, würde diese Codierung maximal

$$\binom{17576}{10} \approx 3{,}85569 \cdot 10^{15}$$

verschiedene Einträge darstellen können.

Sinnvoller als diese theoretische Angabe ist die Zahl der tatsächlich verwendeten Codes. Aber auch diese Zahl gibt noch keine Auskunft über die Verteilung der Codes: Haben die meisten Wörter einen eindeutigen Code, oder werden eher Gruppen von Wörtern auf einen Code abgebildet? Schließlich entscheidet der Nutzer über die Qualität der Codierung: Wenn die exakte Suche nicht ausreicht, können die Parameter der unscharfen Suche mit geringem Aufwand so eingestellt werden, dass das Suchergebnis die intensional gesuchten Begriffe enthält, andererseits aber nicht wesentlich darüber hinaus geht? Die Auswahl ist natürlich relativ zum verwendeten Korpus. Dieser Frage wird in Abschnitt 5.1.1 nachgegangen.

[13] http://www.maccs.mq.edu.au/~ssaunder/DRC/

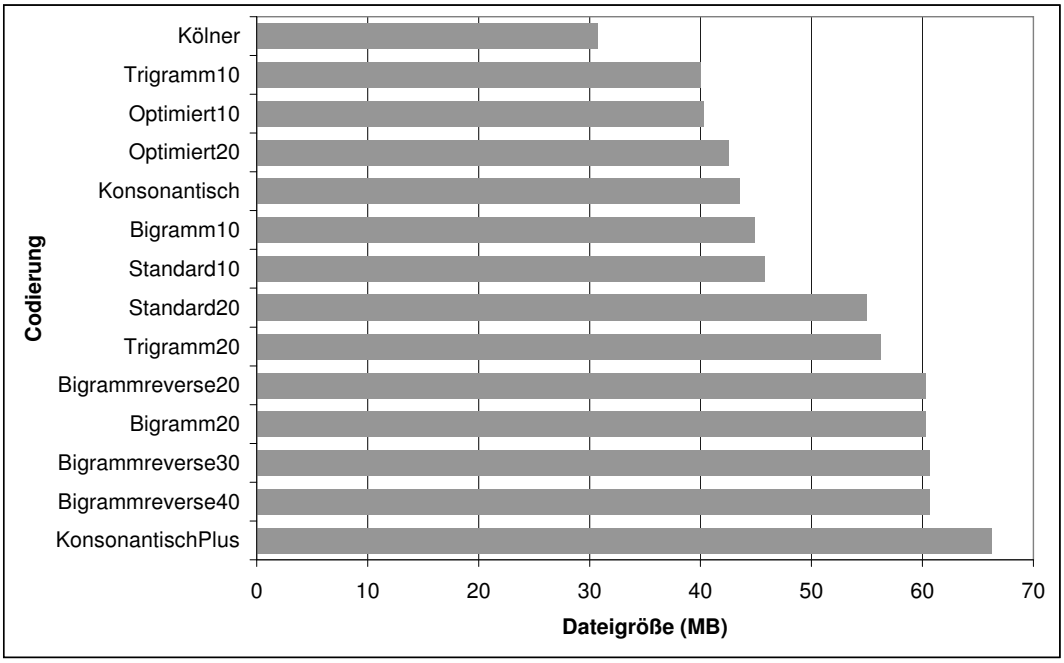


Abbildung 4.5: Größe der SpaCAM-Codierung

Die Größe der Indexdatei für die unscharfe Suche variiert je nach gewähltem Verfahren. Aufgrund der Repräsentation der Daten in einer dünn besetzten Matrix (*sparse coded matrix*) sind die Unterschiede in den Dateigrößen jedoch eher gering. Abbildung 4.5 zeigt die Größe der gebildeten Indexdateien. Die Dateigrößen belegen die effiziente Speichernutzung durch die gewählte Repräsentation der Daten in Form einer dünn besetzten Matrix.

4.6 Details zur Implementierung des SpaCAM

Prinzipiell ist die Verwendung von Hauptspeicher schneller als die Auslagerung von Daten auf die Festplatte. Ein Zugriff auf eine beliebige Stelle einer Festplatte dauert heute etwa 10 ms.[14] Ein Zugriff auf eine Speicherstelle dauert dagegen nur 100 ns, ist also um einen Faktor 100 schneller. Windows kann jedoch bei Speicherknappheit Hauptspeicher auf eine Festplatte auslagern und den Hauptspeicher bereinigen. Dieser Vorgang dauerte auf dem im Abschnitt 3.1 beschriebenen System bis zu drei Sekunden, währenddessen das System keine Reaktion zeigt. Ein solcher Zustand ist nicht nutzerfreundlich und sollte tunlichst vermieden werden.

[14] http://www.itwissen.info/definition/lexikon/Zugriffszeit-access-time.html

In einer naiven Implementierung benötigt eine Matrix der Größe m Spalten × n Zeilen mit Datenelementen, die d Bytes belegen M Bytes Speicherplatz: $M = m \cdot n \cdot d$.

Der Platzbedarf für den vollständigen Index der dünn besetzten Matrix in der Form der Zeiger-Zeiger-Implementierung beträgt mit $m = 32\,000$, $n = 1\,400\,000$ und $d = 4$ Bytes 167 GB Speicherplatz.

Weil eine dünn besetzte Matrix zum größten Teil leere Datenelemente enthält, verzichtet man auf die Darstellung dieser leeren Elemente. Statt dessen werden Listen gespeichert, die Verweise auf die wenigen zu speichernden Datenelemente enthalten. Wie diese Listen aussehen, hängt von den gewünschten Zugriffsmöglichkeiten ab: Wenn man sich in der Matrix beliebig bewegen will, müssen diese Listen in jeder Richtung verkettet sein. Dies ist für den Fall des SpaCAM jedoch nicht erforderlich.

Der tatsächlich erforderliche Speicherbedarf für die Matrix wurde in Abbildung 4.5 auf Seite 87 für die verschiedenen Codierungen bereits dargestellt.

4.6.1 Optimierungen

Auch in Zeiten hoch getakteter Prozessoren und automatisch optimierender Compiler kann eine unglückliche Wahl von Datenstrukturen die Leistungsfähigkeit eines Systems unter die Akzeptanzschwelle eines durchschnittlichen Nutzers sinken lassen. Durch passend gewählte Datenstrukturen und Algorithmen können enorme Gewinne in der Leistung erzielt werden. Eine solche Änderung bewirkte im vorliegenden Fall eine Verbesserung der Antwortzeit von acht Sekunden auf weniger als 200 ms und soll im folgenden beschrieben werden.

Für das Retrieval müssen die relevanten Spalten des SpaCAMs Feld für Feld betrachtet werden. Alle Felder mit gleichen Werten müssen zusammengeführt werden und nach Anzahl der Übereinstimmungen sortiert werden.

Ein naheliegender Weg zur Lösung dieser Aufgabe ist es, eine sortierte Liste zu erstellen, in der der Feldinhalt als Schlüssel für einen Wert verwendet wird, der die Häufigkeit enthält. Beim Bearbeiten der Felder einer Spalte wird dann entweder ein neuer Schlüssel an passender Stelle in die sortierte Liste eingefügt und diesem Schlüssel der Wert eins zugeordnet, oder, wenn der Schlüssel bereits vorhanden ist, wird der zugeordnete Wert um eins erhöht.

Anschließend kann ein Histogramm der enthaltenden Werte erstellt werden und alle Schlüssel, deren Werte unter der vom Nutzer vorgegebenen Relevanzschwelle liegen, können wieder entfernt werden.

Außer den in Visual Basic 2005 enthaltenen Klassen gibt es für diese Aufgabe besser geeignete Strukturen, wie den red-black-tree[15], der in einer Implementierung von NGenerics[16] zur Verfügung gestellt wurde.

Obwohl alle Daten im Hauptspeicher gehalten wurden und diesen erheblich beanspruchten, war die Verarbeitungsgeschwindigkeit äußerst unbefriedigend. Die Ausnutzung von Vorwissen über die Struktur der vorliegenden Daten kann die Performance erheblich verbessern. [Hei94] schlägt für große SpaCAMs die Verwendung einer segmentierten Matrix vor, damit nicht die volle Matrix im Hauptspeicher gehalten werden muss. Hier werden die Spalten in Segmente unterteilt, die blockweise einzeln von der Festplatte geladen werden können. Der Header muss die dazu erforderlichen Informationen enthalten.

Bei der Suche muss jedoch nur ein sehr geringer Anteil der Spalten überhaupt durchsucht werden, nämlich genau die Spalten, die durch eine Eins im Bitarray der codierten Suchzeichenkette als relevant markiert sind. Das sind bei den hier vorgestellten Codierungen nicht mehr als zehn oder zwanzig Spalten. Eine Spalte enthält in der 2-Tupelcodierung bei den im Wörterbuch vorhandenen 1,4 Millionen Einträgen die in Abbildung 4.6 dargestellte Anzahl von Verweisen, das sind für die zehn Spalten mit den meisten Einträgen (318 103 bis 122 000 Einträge, im Diagramm grün markiert) zusammen 1,8 Millionen Einträge.

Als Suchergebnis sollten für die intellektuelle Bearbeitung nicht mehr als 500 Wörterbucheinträge zurückgeliefert werden. Schon diese Zahl verlangt eine weitere Filterung, damit die Übersicht nicht verloren geht.

Wenn jeder Verweis durch eine vier-Byte-Integer-Zahl, der Länge eines Zeigers, ausgedrückt wird, sind bei der SpaCAM-Suche maximal 80 MB zu verarbeiten.

Statt wie in der Originalarbeit vorgesehen [Hei94] die Spalten zu segmentieren, ist es sinnvoller, die Spalten einzeln und komplett auf der Festplatte zu halten, um bei Bedarf eine komplette Spalte zu laden. Konsekutive Byte-Zugriffe erfolgen immer erheblich schneller als zufällige Zugriffe. Deshalb werden zunächst zwei Spalten geladen. Weil die Einträge in den Spalten aufgrund des Lernverfahrens bereits aufsteigend sortiert sind, reicht für das Zusammenführen von zwei Spalten bereits ein einfacher *merge*-Schritt. Wenn die Spalten n_1 und n_2 Elemente enthalten, sind dazu höchstens $(n_1 + n_2)$ Vergleiche erforderlich. So können alle erforderlichen Spalten als Integerarrays nacheinander geladen werden und zu einem sortierten Array zusammengefasst werden. Für dieses Array wird ein Histogramm erstellt, das die Häufigkeiten gleicher Verweise erfasst.

Beispielhaft ist in Abbildung 4.7 die Ausgabe der Konsolenversion für den Suchbegriff »Kaufrausch« wiedergegeben:

[15] http://en.wikipedia.org/wiki/Red_black_tree

[16] http://www.codeplex.com/NGenerics/Release/ProjectReleases.aspx?ReleaseId=2027

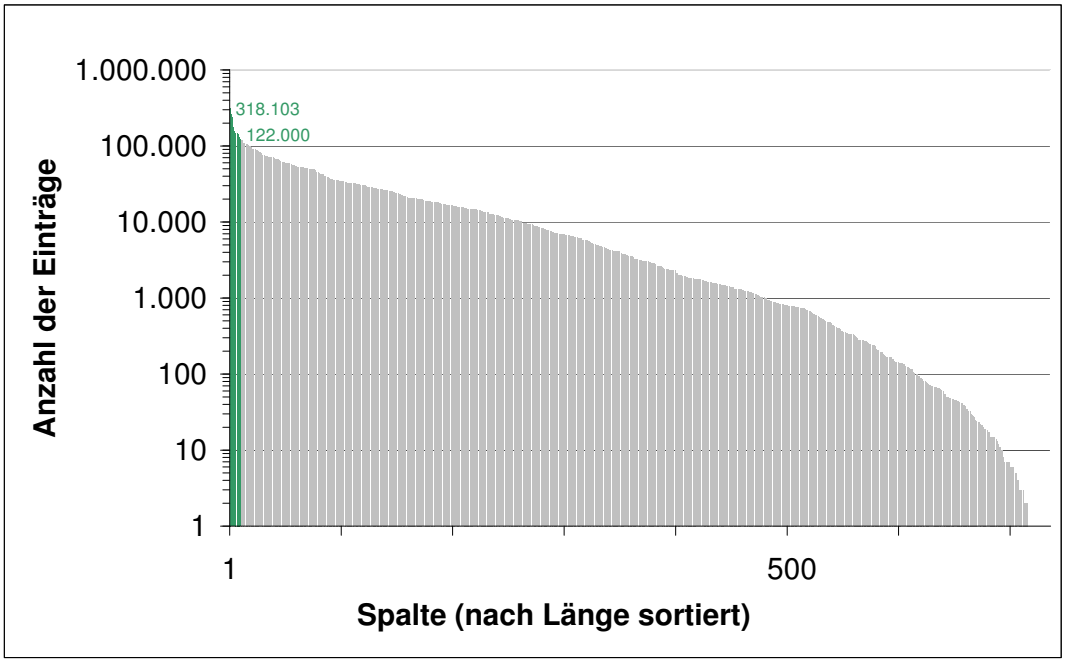


Abbildung 4.6: Spaltenlängen des SpaCAM

```
Kaufrausch
10 : @K AU CH FR H@ KA RA SC UF US
1 : 975071
2 : 222985
3 : 55206
4 : 14932
5 : 3063
6 : 409
7 : 49
8 : 3
Suchzeit: 0,59375 Sekunden
Treffer Worthäufigkeit Wort
   8         1          Kaffeerausch
   8         2          Kanonenrausch
   8         1          Aufrausch
```

Abbildung 4.7: Bigrammsuche nach »Kaufrausch«. Für die Bigrammsuche werden die zehn Merkmale (die Bigramme) dargestellt. In den folgenden Zeilen ist aufgelistet, wie viele Wörter des Lexikons in 1 .. 8 Merkmalen übereinstimmen. Das Wort selbst ist im Lexikon nicht enthalten.

Unter der Eingabe des Suchworts (bei den untersuchten Codierungen werden Groß- und Kleinbuchstaben nicht unterschieden) werden zur besseren Anschaulichkeit die extrahierten Merkmale dargestellt. In den *n*-gramm-Codierungen wird das

Suchwort von ›@‹ eingerahmt, um Wortanfang und -ende zu markieren. Insgesamt erhält man zehn Merkmale, nämlich die zehn verwendeten Bigramme »@K AU CH FR H@ KA RA SC UF US«. Die Bigramme sind alphabetisch sortiert. Darunter folgt ein Histogramm. In diesem Histogramm wird angegeben, wie viele Wörter des Wörterbuchs in wieviel Merkmalen mit dem Suchbegriff übereinstimmen. Das Wort selbst wurde im Wörterbuch nicht gefunden. Im Wörterbuch wurden drei Wörter gefunden, die in acht Merkmalen mit dem Suchbegriff übereinstimmen. Diese Wörter werden ausgegeben.

Das Löschen eines Eintrags in einem Suchbaum ist recht aufwendig. Wenn nur wenige Elemente aus einem Suchbaum übernommen werden sollen, ist es vorteilhafter, aus diesen Elementen einen neuen Suchbaum zu erzeugen. Anschließend wird der ursprüngliche Suchbaum in einem Schritt komplett gelöscht. Das gleiche Prinzip gilt auch für ein Array, weil das Löschen eines Elements aus einem Array ein Umkopieren des Array bewirkt, so dass das Löschen eines Elements für ein Array mit n Elementen ein $O(n)$-Vorgang ist.

4.6.1.1 Laufzeit gegen Speicherplatz

Je nachdem, wie viel Ressourcen vorhanden sind, kann ein Programm mit Hilfe eines höheren Speicherbedarfs die Laufzeit reduzieren. So kostet ein Index Speicherplatz, reduziert dafür aber die Suchzeit. Dieser Zusammenhang ist nicht linear, sondern regelmäßig logarithmisch mit Diskontinuitäten: Ein Index ermöglicht eine binäre Suche (Laufzeitverhalten $\mathcal{O}(\log n)$), muss aber erst erstellt werden (Laufzeitverhalten der Indexerstellung $\mathcal{O}(n)$). Für häufige Suchvorgänge ist deshalb das Erstellen eines Index zweckmäßig, für nur einmalige Suchvorgänge nicht. Das tatsächliche Verhalten eines Programms kann nicht immer vorhergesehen werden. Wenn viel Hauptspeicher gebraucht wird, können sich die Ausführungszeiten eines Programms ad libitum verlängern, weil der Speicher vom Betriebssystem ständig reorganisiert wird.

An manchen Stellen sind deshalb Abwägungen erforderlich, wie viel langsamer Plattenspeicher genutzt wird und wie viel schneller Hauptspeicher zur Verfügung steht. Die Größe des zur Verfügung stehenden Hauptspeichers ist dabei nicht konstant, sondern ändert sich bei dem gewählten Betriebssystem Windows XP® SP3 unvorhersehbar durch im Hintergrund ablaufende Prozesse. Weil auch das Auslagern von Hauptspeicher in Datendateien Speicherressourcen durch die Anlage von Dateistreamen und Puffern benötigt, hat sich folgendes Vorgehen bewährt: Wenn vom ursprünglich vorhandene Hauptspeicher nur noch zehn Prozent frei sind, wird bei nächster Gelegenheit der Speicher programmgesteuert bereinigt.

Dieses Vorgehen soll illustriert werden am Beispiel der Extraktion der Wörterliste aus der Textsammlung *Sammeldatei*.

Die Textsammlung ist größer als der zur Verfügung stehende Hauptspeicher. Außer der Wörterliste soll auch die Häufigkeit des Vorkommens der einzelnen Wörter erfasst werden. Das schnelle Aufsuchen eines bereits erfassten Wortes erfordert eine verlinkte Baumstruktur, die ein Einfügen in $\mathcal{O}(|k|)$ Zeit (mit $|k|$ als Anzahl der Buchstaben des einzufügenden Wortes) ermöglicht und die weitere Ressourcen kostet. Beim Einlesen aus der Textsammlung wird der Zustand des Hauptspeichers periodisch abgefragt. Wird der verbleibende Hauptspeicher knapp, dann wird die erhaltene Wörterliste in eine Datei geschrieben und aus dem Speicher gelöscht. Ist die gesamte Textsammlung eingelesen, können die erhaltenen Wörterlisten in einem rekursiven Verfahren zusammengefügt werden.

4.6.1.2 Kostenbetrachtung

Für den Nutzer sind lange Suchzeiten unerfreulich. Von einem Bürorechner wird erwartet, dass er sofort reagiert. Für Programmoptimierungen wurde deshalb von folgendem Szenario ausgegangen:

Festplattenplatz ist bei modernen PCs in der Regel ausreichend vorhanden. Eine externe Platte mit 500 GB kostet nach einem aktuellen Angebot (2008) 100 €. Eine Arbeitsstunde kann mit 5 € für einen fiktiven Amateur bis 200 € für eine mittlere Führungskraft kalkuliert werden.

Diese Werte wurden zugrunde gelegt, wenn es um Optimierungen ging. Früher waren wegen der vergleichsweise teuren und deshalb knapp vorhandenen Hardwareausstattung zahlreiche Tricks und Kniffe der Programmierer erforderlich, um akzeptable Antwortzeiten eines Systems zu erreichen. Heute erscheinen Hardwareressourcen meist mehr als ausreichend vorhanden. Auf Programmiertricks zur Optimierung wurde deshalb prinzipiell verzichtet und die programminterne Optimierung dem Compiler überlassen. Das bedeutet nicht den Verzicht auf intelligente Algorithmen. Diese sind nach wie vor erforderlich. An den entsprechenden Stellen wird deshalb auf den verwendeten Algorithmus hingewiesen. Unverzichtbar für die verwendeten Datenmengen war allerdings ein ausreichend bemessener Hauptspeicher (2 GB), wobei sich die allgemeine Verkehrsauffassung für die zu Verfügung stehende Größe des Hauptspeichers während der Erstellung dieser Arbeit von „groß“ zu „klein“ verändert hat.

4.6.2 Freie Parameter

Die graphische Benutzeroberfläche erlaubt zur unscharfen Suche mit Hilfe des SpaCAMs die Angabe verschiedener Optionen. Abbildung 4.8 zeigt das zugehörige

Dialogfeld. In der linken Hälfte werden die verfügbaren Codierungen angezeigt. Aus diesen verfügbaren Codierungen kann der Benutzer eine Codierung auswählen.

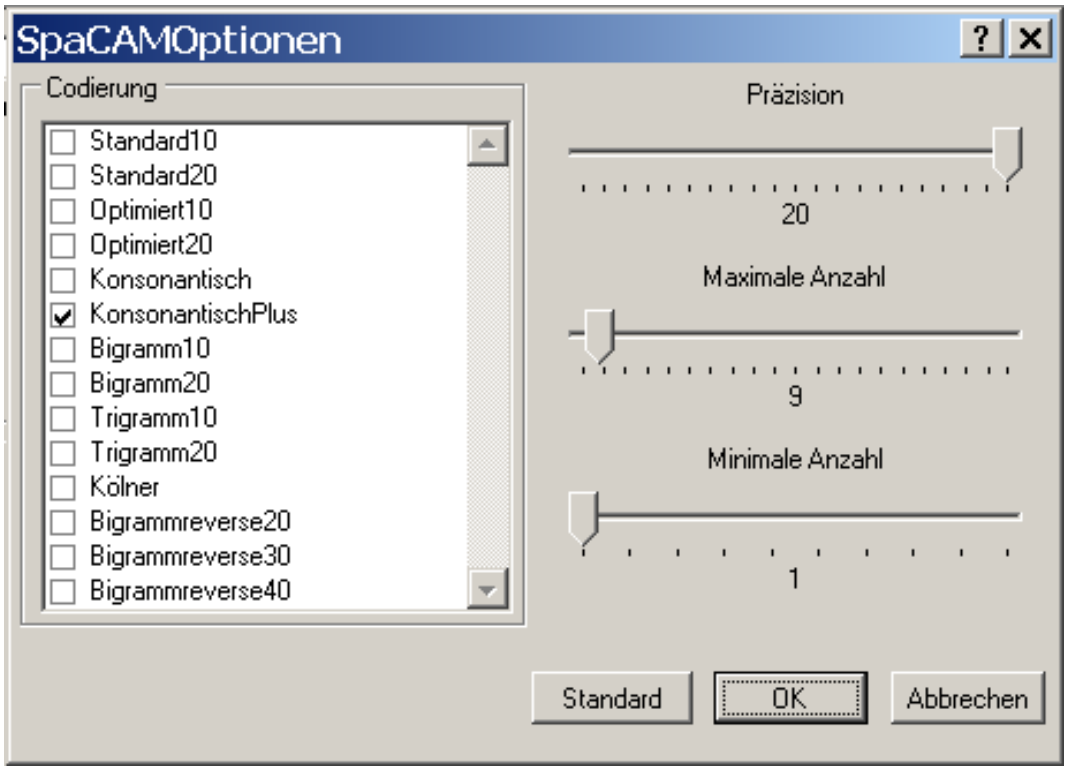


Abbildung 4.8: Optionen für das SpaCAM

Auf der rechten Seite werden drei Schieberegler angezeigt. Unter dem jeweiligen Schieberegler wird der eingestellte Wert als Zahl angezeigt. Alle Einstellungen der Schieberegler werden mit einem *fuzzy*-Algorithmus unscharf ausgewertet.

4.6.2.1 Präzision

Die Codierung legt fest, welche Merkmale eines Wortes im SpaCAM gespeichert werden. Im Sinne der Spärlichkeit werden je nach Codierung zwischen 10 und 40 Merkmale aus einer möglichen Anzahl von 320 bis 32 000 verschiedenen Merkmalen

ausgewählt. Eine Einstellung der Präzision von 20 in der graphischen Benutzeroberfläche bewirkt,[17] dass alle in der Codierung des Suchworts enthaltenen Merkmale auch in den gefundenen Wörtern aus dem Wörterverzeichnis vorkommen sollen. Wird die Präzision auf einen niedrigeren Wert eingestellt, dann werden auch Wörter zurückgegeben, die in weniger Merkmalen übereinstimmen. Dabei gilt, dass wenn ein Suchwort nicht im Wörterverzeichnis vorkommt, als Höchstmaß der Präzision unabhängig von der Einstellung die Übereinstimmung mit dem ähnlichsten Wort im Wörterverzeichnis verwendet wird, damit bei einem Suchergebnis ohne exaktem Resultat nicht zunächst die Parameter geändert werden müssen. Die beiden folgenden Schieberegler modifizieren die Wirkung dieses Reglers wie folgt:

4.6.2.2 Maximale Anzahl

Mit diesem Regler lässt sich in einem Bereich zwischen eins und 200 einstellen, wie viele verschiedene Suchwörter maximal zurückgegeben werden sollen. Diese maximale Anzahl ist jedoch nicht exakt zu verstehen. Zunächst ist sie der Angabe der Präzision untergeordnet: Ist die Präzision auf einen hohen Wert eingestellt, werden eher weniger Wörter aus dem Wörterverzeichnis zurückgegeben. Wenn die Präzision jedoch auf einen geringen Wert eingestellt wird, dann kann es sein, dass mehrere tausend Wörter zurückgegeben werden. Die Angabe der maximalen Anzahl begrenzt die Anzahl der zurückgegebenen Wörter in der Weise, dass ausgehend von den gefundenen Wörtern mit maximaler Übereinstimmung das geforderte Übereinstimmungsmaß Schritt für Schritt so weit reduziert wird, dass die maximale Anzahl überschritten wird. Da die Trefferquote mit der Verringerung der Präzision exponentiell steigt, können trotzdem sehr viele Wörter zurückgegeben werden. Das in Abbildung 4.7 (Seite 90) dargestellte Beispiel aus der Konsolenversion illustriert diesen Sachverhalt: Das Wort »Kaufrausch« wird in der Codierung *Bigramm20* mit 10 Merkmalen codiert. (Die sortierte Liste der Bigramme wird angegeben, Wortanfang und -ende sind durch das Zeichen ›@‹ markiert.) Das ähnlichste Wort im Wörterverzeichnis stimmt in 8 Merkmalen überein. Insgesamt stimmen die ausgegebenen drei Wörter in acht Merkmalen überein. Diese acht Merkmale werden mit dem numerischen Wert 20 als Maximalmaß der Präzision gleichgesetzt.

Wird die Präzision um drei reduziert, also auf siebzehn eingestellt, dann werden ohne Begrenzung durch die maximale Anzahl alle Wörter gefunden, die in mindestens $8 - 3 = 5$ Merkmalen übereinstimmen. Das sind $3063 + 409 + 49 + 3 = 3524$ verschiedene Wörterbucheinträge. Wenn die maximale Anzahl auf fünfzig gesetzt

[17] Dieser Wert wurde gewählt wegen der überwiegend codierten 20 Merkmale. Tatsächlich haben kürzere Wörter weniger Merkmale und andere Codierungen können auch mehr Merkmale codieren. Für den Nutzer sind diese Variationen unbeachtlich. Das Programm verhält sich so, wie im weiteren Text beschrieben.

wurde, dann wird die Rückgabe von Wörterbucheinträgen abgebrochen, sobald mindestens fünfzig Wörter gefunden werden. Da jede Rangstufe vollständig zurückgegeben wird, werden insgesamt 49 + 3 = 52 Wörterbucheinträge zurückgegeben. Damit entspricht diese Einstellung einer Einstellung der Präzision von neunzehn.

4.6.2.3 Minimale Anzahl

Mit der minimalen Anzahl wird eingestellt, wie weit die Präzision reduziert wird, damit mindestens die eingestellte Anzahl von Treffern erreicht wird. Weil eine geringe Präzision zu sehr ungenauen Treffern führt und außer erwünschten Wörtern mit ähnlicher Schreibweise und gleicher Bedeutung viele andere Wörter erscheinen lässt, sollte dieser Wert nicht zu hoch eingestellt werden. Im obigen Beispiel führt eine Einstellung von mehr als drei trotz Einstellung der Präzision auf ihren Maximalwert von 20 zu einer Rückgabe von 3 + 49 = 52 Wörtern, indem die Präzision auf 19 gesenkt wird.

4.7 Lemmaexpansion

Wie lautet die kürzeste Suchanfrage, mit der man alle Formen von *treffen* findet[Wie99]?

```
  "[tT]reff(e(nd?)?|s?t)?|[tT]riff(s?t)?|[tT]raf(s?t|en)?
|[tT]r\"af(en?|s?t)?|getroffen;"
```

Viele Experten werden diesen Ausdruck dekodieren können, aber nur sehr wenige werden ihn auf Anhieb richtig schreiben können. Die meisten Anwender werden dankbar sein, wenn sie reguläre Ausdrücke nicht selbst erstellen müssen. Für die Aufgabe, alle Wortformen zu einem Lemma zu finden, stellt das Projekt Wortschatz der Universität Leipzig einen Webservice bereit, der dies erledigt und programmgesteuert abgefragt werden kann.

Tabelle 4.6: Lemmaexpansion

Anfrage	**Ausgabe**
lauf, gelaufen	läuft, laufen, lief, gelaufen, liefen, laufe, laufend, liefe, lauf, läufst, lauft, laufenen[18], laufene, laufet, liefest, liefst, liefet, laufest, lieft
gelaufene	gelaufenen, gelaufene, gelaufenes, gelaufener, gelaufnen

[18] http://www.mydict.com/Wort/laufenen/ gibt zwei Fundorte in Tageszeitungen, die aber wohl eher Rechtschreibfehler beinhalten

Nicht alle gebildeten Formen werden im Korpus vorkommen, andererseits gibt es auch im Korpus Schreibweisen, die man so nicht erwartet.

Die Expansion eines Wortes auf seine verschiedenen Formen ist in der deutschen Sprache besonders schwierig. Der Webservice *wordforms* des Projekts Wortschatz der Universität Leipzig[19] soll alle anderen Wortformen zu einem Lemma liefern. Tabelle 4.6 zeigt die Ergebnisse für die Eingaben »lauf«, »gelaufen« und »gelaufene«.

Offenbar werden die verschiedenen Wortarten unterschieden. Die Erweiterung eines Suchterms auf weitere Wortformen ist damit auf bequeme Weise möglich.

Mit Hilfe des Webservices *synonyms*[20] konnten auch Synonyme in die Suche mit einbezogen werden.

4.8 Implementierung von Webservices

An dieser Stelle soll auf einige Besonderheiten hingewiesen werden, die für die Implementierung der Schnittstellen zu den Webservices des Projekts Wortschatz der Universität Leipzig wichtig sind. Die verschiedenen Internetprotokolle sind dazu da, um den Datenaustausch zwischen unterschiedlichen Rechnern mit unterschiedlichen Betriebssystemen und Programmiermodellen zu ermöglichen. In diesem Zusammenhang besonders zu erwähnen ist das Hypertext-Transfer-Protokoll (*http*) und das *SOAP*-Protokoll, dessen Name sich ursprünglich von *Simple Object Access Protocol* ableitete, jetzt aber eine geschützte Bezeichnung ist.[21]

Tabelle 4.7: Vertikale Struktur der Transportprotokolle

Anwendung	SOAP			
	HTTP		HTTPS	···
Transport	TCP			
Netzwerk	IP			
Netzzugang	Ethernet	Token Ring	FDDI	···

Das Projekt Wortschatz bietet einige Webservices an[22], mit denen fremde Benutzer auf Daten des Projektes zugreifen können.

[19] http://wortschatz.uni-leipzig.de/axis/jnlp/Wordforms.jnlp
Login: *anonymous*, Password: *anonymous*

[20] http://wortschatz.uni-leipzig.de/axis/jnlp/synonyms.jnlp
Login: *anonymous*, Password: *anonymous*

[21] http://de.wikipedia.org/wiki/SOAP

[22] http://wortschatz.uni-leipzig.de/axis/servlet/ServiceOverviewServlet

Die folgende Tabelle gibt eine Übersicht über die zur Zeit ohne Anmeldung zugänglichen Webservices:

Tabelle 4.8: Webservices der Universität Leibzig

ServiceOverview	Gibt einen Überblick über die momentan verfügbaren Services.
Cooccurrences	Liefert die als statistisch signifikant errechneten Kookkurrenzen.
Baseform	Liefert zu einem Wort die Grundform sowie die Wortklasse.
Sentences	Liefert zu einem Wort Beispielsätze.
RightNeighbours	Gibt zu einem Eingabewort die als statistisch signifikant errechneten rechten Nachbarn.
LeftNeighbours	Gibt zu einem Eingabewort die als statistisch signifikant errechneten linken Nachbarn.
Frequencies	Gibt die Frequenz sowie die Häufigkeitsklasse eines Services.
Synonyms	Gibt Synonyme zu einem Eingabewort.
Thesaurus	Wie Synonyms, ruft zuvor jedoch Baseform auf.
Wordforms	Liefert alle anderen Wortformen zu einem Lemma.
Similarity	Liefert automatisch berechnete, kontextuell ähnliche Wörter, einschließlich Antonyme, Hyperonyme, Synonyme und Kohyponyme. Verlängerte Antwortzeit.
LeftCollocationFinder	Findet linke Kollokationen, auch eingeschränkt auf Adjektive, Verben, Nomen oder Stopwörter.
RightCollocationFinder	Findet rechte Kollokationen, auch eingeschränkt auf Adjektive, Verben, Nomen oder Stopwörter.
Sachgebiet	Gibt das Sachgebiet des Eingabewortes.
Kreuzwortraetsel	Findet ein passendes Wort zum angegebenen Muster.

Um diesen Zugriff zu erleichtern, bietet das Projekt sowohl fertige Clienten wie auch JAVA-Klassen zum Download an. Im Sinne der Interoperabilität ist es aber ebenso möglich, mit anderen Programmiersprachen eine Schnittstelle zu den Daten zu erstellen. Dafür wurde die *Web Services Description Language*[23] (*WSDL*) entwickelt, die die erforderlichen Daten für den Zugriff enthält. Im Fall des Thesaurus und des Synonyms-Service enthält die zugehörige *wsdl*-Datei jedoch einen seit 2005 bekannten Fehler, der eine erfolgreiche Auswertung der zurückgelieferten Daten

[23] http://de.wikipedia.org/wiki/WSDL

verhindert. Wie Foust[24] beschreibt, fehlt in der Datei für die Type „dataVectors“ das Attribut

```
<Namespace=_
    "http://datatypes.webservice.wortschatz.uni_leipzig.de">.
```

Schon in der Anfrage sind zwei Herausforderungen zu meistern: Die Webservices sehen eine Benutzerauthentifikation vor: Der Benutzer soll sich mit Namen „anonymous“ und Passwort „anonymous“ anmelden. Das in .NET dafür vorgesehene Verfahren[25] sieht dazu entsprechend dem Standard[26] ein abgestuftes Verfahren vor, in dem der Client zunächst eine anonyme Anfrage sendet, der Server darauf eine Authentisierungsanforderung nach einem bestimmten Schema anfordert, woraufhin sich der Client in der geforderten Weise ausweisen kann. Die Webservices des Projektes unterstützen dieses Verfahren nicht. Sie verlangen bei der ersten Anfrage die Angabe von Benutzernamen und Passwort im *Basic*-Format im http-Header. Die Dokumentation der Entwicklungsumgebung gibt keine Hinweise[27], wie dies geschehen kann. Tatsächlich muss die aus der *wsdl*-Datei generierte Klasse modifiziert werden:

```
Private myusername As String
Private mypassword As String
Protected Overrides Function _
    GetWebRequest(ByVal uri As System.Uri) _
    As Net.WebRequest
Dim request As Net.WebRequest _
    = MyBase.GetWebRequest(uri)
Dim ASCII As System.Text.Encoding _
    = System.Text.Encoding.ASCII
Dim BasicEncrypt As String = myusername & ":" & _
    mypassword
Dim BasicToken As String = _
    "Basic " _
    + Convert.ToBase64String(ASCII.GetBytes(BasicEncrypt))
request.Headers.Add("Authorization", BasicToken)
'request.Headers.Add( _
    "Authorization", _
    "Basic YW5vbnltb3VzOmFub255bW91cw==")
Return request
```

[24] http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=112770&SiteId=1
[25] http://msdn2.microsoft.com/en-us/library/ms789031.aspx
[26] http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html#BasicAA
[27] http://mark.michaelis.net/Blog/CallingWebServicesUsingBasicAuthentication.aspx

```
End Function
```

Das Framework generiert die Serviceanfrage mit Hilfe der Funktion *GetWebRequest*, die vom Anwender überschrieben werden kann und so die zusätzliche Information in den Header der http-Nachricht einfügt.

Schließlich ist nicht beschrieben, wie die Anfrage formatiert sein muss. Die Analyse der Anfrage des zur Verfügung gestellten Clienten zeigte, dass abweichend von der XML-Spezifikation Überschrift und Dateneintrag in einer Tabellenstruktur übermittelt werden mussten. Im Falle der *Synonyms*-Anfrage hat die Tabelle die folgende Struktur:

Limit	**Wort**
„20“	„Haus“

Im Falle der *Wordforms*-Anfrage muss die Tabelle jedoch die folgende Struktur haben:

Word	**Limit**
„Haus“	„20“

Zu beachten sind auch die unterschiedliche Schreibungen **Wort** und **Word**. Mit **Limit** wird die Anzahl der zurückgelieferten Ergebnisse eingeschränkt.

In der *Synonyms.wsdl*-Datei liest sich diese Darstellung im XML-Format so:

```
   <xsd:complexType name="DataMatrix">
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded"
          name="dataVectors"
          type="tns1:DataVector" />
      </xsd:sequence>
   </xsd:complexType>
    .
    .
    .
<xsd:import namespace="urn:Synonyms" />
   <xsd:complexType name="DataVector">
      <xsd:sequence>
        <xsd:element maxOccurs="unbounded"
            name="dataRow"
```

```
            type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>
```

Eine Besonderheit im Zusammenhang mit UNICODE ist zu bemerken: Die Anfrage liefert die Ergebnisse *utf8*-codiert aus, was bei der Speicherung als String aber nicht berücksichtigt wird. Zeichenketten werden im Framework als *utf16*-codiert gespeichert. Die Umwandlung über einen `char`-Array führt glücklicherweise ohne großen Aufwand zum Ziel:

```
Dim stb(response.result().Length - 1) As String
For i As Integer = 0 To response.result().Length - 1
    Dim chars() As Char = _
        response.result(i)(0).ToCharArray
    stb(i) = New String(chars)
Next
```

Im folgenden Kapitel wird zunächst beschrieben, wie sich selten vorkommende Terme auf die Dokumente im Korpus verteilen. Anschließend werden die Ergebnisse dargestellt, die mit den verschiedenen untersuchten Codierungen für die unscharfe Suche erzielt wurden, und zum Schluss des Kapitels werden die Ergebnisse der Suche mit den Ergebnissen einiger anderer Suchmaschinen verglichen.

5 Ergebnisse

Zunächst besteht der Verdacht, dass selten vorkommende Terme eher gemeinsam in einer Datei vorkommen. Diese Vermutung ist sicher richtig für seltene Eigennamen, die sonst im Korpus nicht verwendet werden.

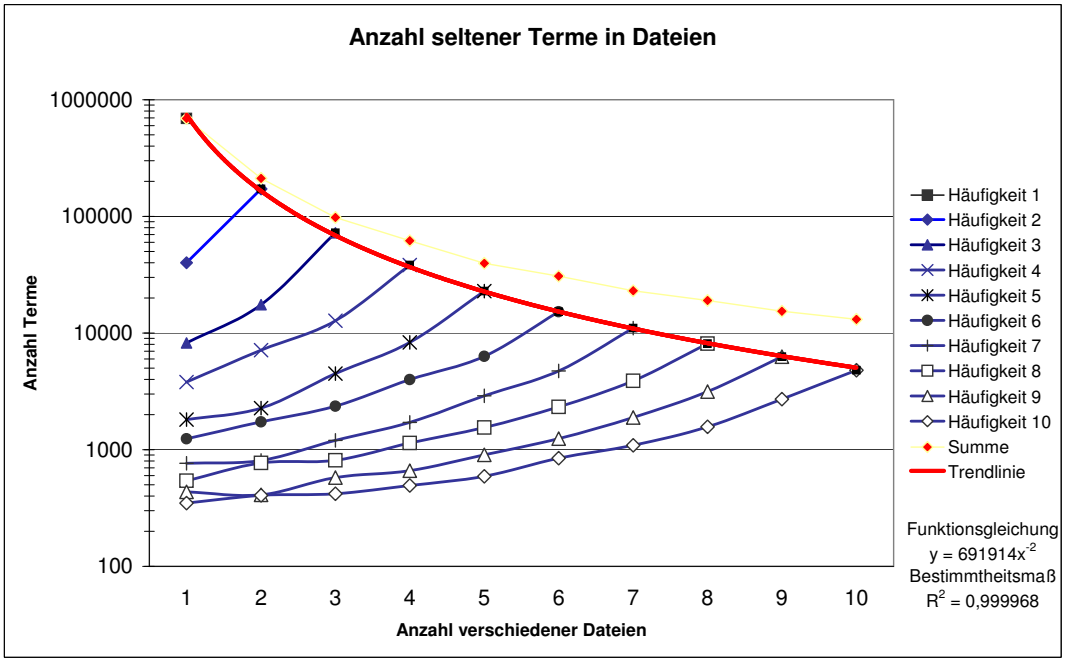


Abbildung 5.1: Mehrfachvorkommen seltener Terme in Dateien.
Lesebeispiel: Von den 98 074 Termen der Häufigkeitsklasse drei, die im Diagramm mit einem Dreieck ▲ gekennzeichnet sind, also solchen Termen, die im gesamten Korpus dreimal vorkommen, kommen 72 356 Terme in drei Dateien vor. Diese Terme kommen also in jeder Datei höchstens einmal vor. 17 500 Terme kommen in jeweils zwei Dateien vor, und 8 219 Terme kommen in nur einer Datei vor. Weitere Erläuterungen im Text.

Abbildung 5.1 zeigt, wie wenig diese Vermutung zutrifft: Für Terme mit einer Häufigkeit von ein bis zehn Vorkommen im Korpus wurde auf der Abszisse angegeben, in wie viel verschiedenen Dateien sie im Korpus erscheinen. Die Ordinate

gibt in logarithmischer Teilung die Anzahl der Terme an, auf die die Einteilung zutrifft. Zur besseren Übersicht wurden zusammengehörige Datenpunkte mit Linien verbunden. Lesebeispiel: Von den insgesamt 98 074 Termen der Häufigkeitsklasse drei, die im Diagramm mit einem Dreieck ▲ gekennzeichnet sind, also solchen Termen, die im gesamten Korpus dreimal vorkommen, kommen 72 356 Terme in drei Dateien vor. Diese Terme kommen also in jeder Datei höchstens einmal vor. 17 500 Terme kommen in jeweils zwei Dateien vor, und 8 219 Terme kommen in nur einer Datei vor.

Die Häufigkeiten seltener Terme mit einem bis zehn Vorkommen im gesamten Korpus wurden als Summe mit roten Rauten ◆ im Diagramm eingetragen. Die Simulation einer zufälligen Verteilung entsprechend den in Abbildung 3.3 auf Seite 52 ermittelten Dateilängen mit einem Median von 8 370 Zeichen je Datei ergab, dass in diesem Bereich die statistische Wahrscheinlichkeit eines gemeinsamen Vorkommens zweier Terme in einer Datei kleiner ist als 1 %, also zwei Dekaden im Diagramm. Eine nicht eingetragene Trendlinie für diese Punkte hat die beste Anpassung für eine Exponentialfunktion mit $y = hx^{-1,7}$. Dabei ist $h = 691\,914$ die Anzahl der Terme, die im Korpus genau einmal vorkommen. Tatsächlich liegen aber die ermittelten Werte dafür, dass jeder der Terme einzeln in einer Datei vorkommt, auf der rot eingetragenen Trendlinie mit $y = hx^{-2}$. Diese Trendlinie zeigt mit einem Bestimmtheitsmaß von $R^2 = 0{,}999968$ eine ausgezeichnete Anpassung für diesen Bereich.

Der Unterschied zwischen beiden Trendlinien ist also ein Maß für die Clusterbildung seltener Terme im untersuchten Korpus.

Dies gilt ebenso für Terme, die sich vom Suchterm nur gering unterscheiden oder vom Suchterm abgeleitete Wortformen.

5.1 Codierungen

5.1.1 Genauigkeit der Codierung

Die Genauigkeit einer Codierung kann nur bezüglich eines gegebenen Korpus beurteilt werden. Für das untersuchte Korpus erzeugt die Codierung nach der Kölner Phonetik mit Abstand die wenigsten unterschiedlichen Codes. So fällt fast die Hälfte aller Einträge im Wörterverzeichnis in Klassen, die mehr als 10 Wörter auf einen Code abbilden. Die TrigrammCodierung nach Rechtstrunkierung bei zehn Zeichen schneidet etwas besser ab, belegt aber unter den untersuchten Codierungen nur den vorletzten Platz, was die Anzahl der unterschiedlichen Codes und die Anzahl der eineindeutigen Codierungen betrifft. Hier fällt ein Viertel aller Einträge in Klassen mit mehr als zehn Wörtern.

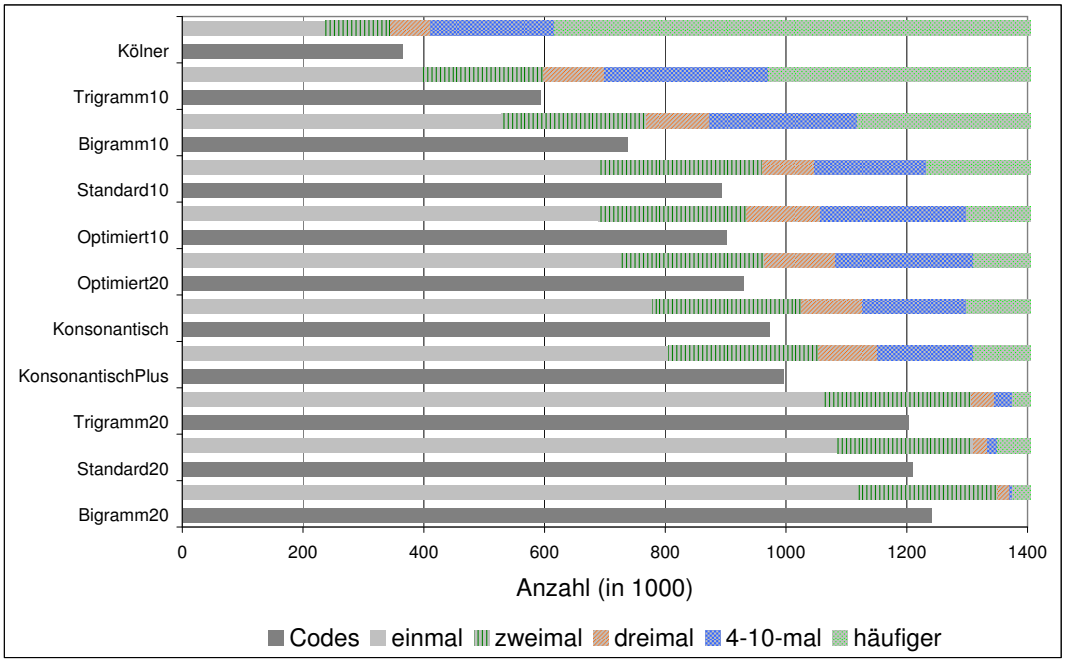


Abbildung 5.2: Genauigkeit der Codierungen.
Der untere Balken zeigt die Anzahl der von der jeweiligen Codierung für die Wörter im Korpus insgesamt erzeugten Codes, der obere Balken zeigt die Aufteilung in die Häufigkeitsklassen {1, 2, 3, 4 – 10, >10}.

Als in Bezug auf die Genauigkeit beste Codierung erweist sich die Bigramm-Codierung nach Rechtstrunkierung nach 20 Zeichen. Knapp 80% aller Wörter im Wörterverzeichnis werden eineindeutig codiert. Auch die einfache Standardcodierung, wie sie in Abschnitt 4.5.1.1 (Seite 81) beschrieben wurde, erzielte ein gutes Ergebnis, wenn das Eingabealphabet groß gewählt wurde und genügend Zeichen berücksichtigt wurden. Die in diesem Abschnitt beschriebenen Optimierungsversuche verschlechterten eher das Ergebnis.

Abbildung 5.2 auf Seite 103 fasst die Ergebnisse zur Genauigkeit der Codierung zusammen: Für jede untersuchte Codierung sind zwei Balken eingetragen. Der obere Balken zeigt an, wie sich die Wörter aus dem Wörterverzeichnis auf eindeutige oder mehrdeutige Codierungen verteilen. Der untere Balken zeigt die Anzahl der insgesamt gebildeten verschiedenen Codes an. So gilt für die Kölner Phonetik (oberste beide Balken), dass von den insgesamt 1 410 165 codierten Wörtern (Gesamtlänge des obersten Balkens) 237 799 eine eindeutige Codierung besitzen, 106.518 Wörter auf eine paarweise Art gleich codiert wurden, 67 365 Wörter jeweils zu dritt auf einen Code abgebildet wurden und 793 220 Wörter sich weniger als

70 000 Codes teilen mussten, weil jeweils mehr als zehn Wörter auf einen Code abgebildet wurden. Insgesamt wurden zur Codierung der 1 410 165 codierten Wörter 365 336 verschiedene Codes gebildet, abzulesen an der Länge des zweiten Balkens.

5.1.2 Modifikationen der *n*-Tupel-Codierungen

In diesem Abschnitt sollen die Auswirkungen verschiedener Modifikationen der *n*-Tupel-Codierungen aus Abschnitt 4.5.1.2 (Seite 82) näher betrachtet werden. Zunächst werden die häufigsten Bigramme ausgezählt, wenn rechtstrunkiert zehn oder zwanzig Zeichen der Wörter im Wörterverzeichnis beim Codieren berücksichtigt werden. Das Zeichen ›@‹ markiert eine Wortgrenze. Es wurde für die Codierung vor und hinter die Zeichenkette gestellt.

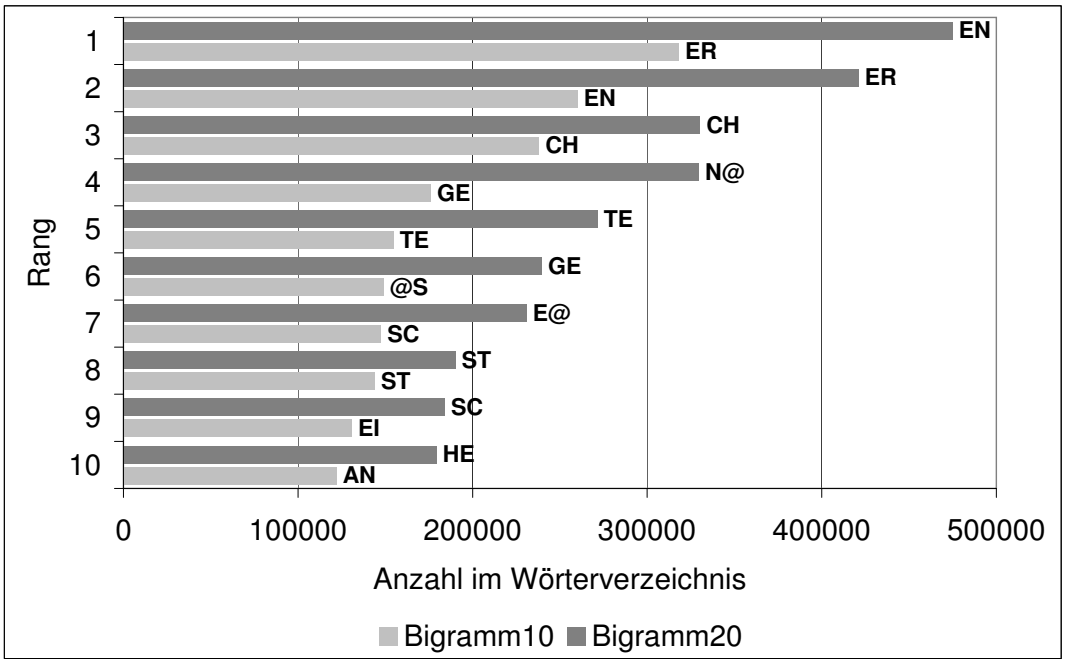


Abbildung 5.3: Vergleich Bigramm10 - Bigramm20. Erläuterungen im Text.

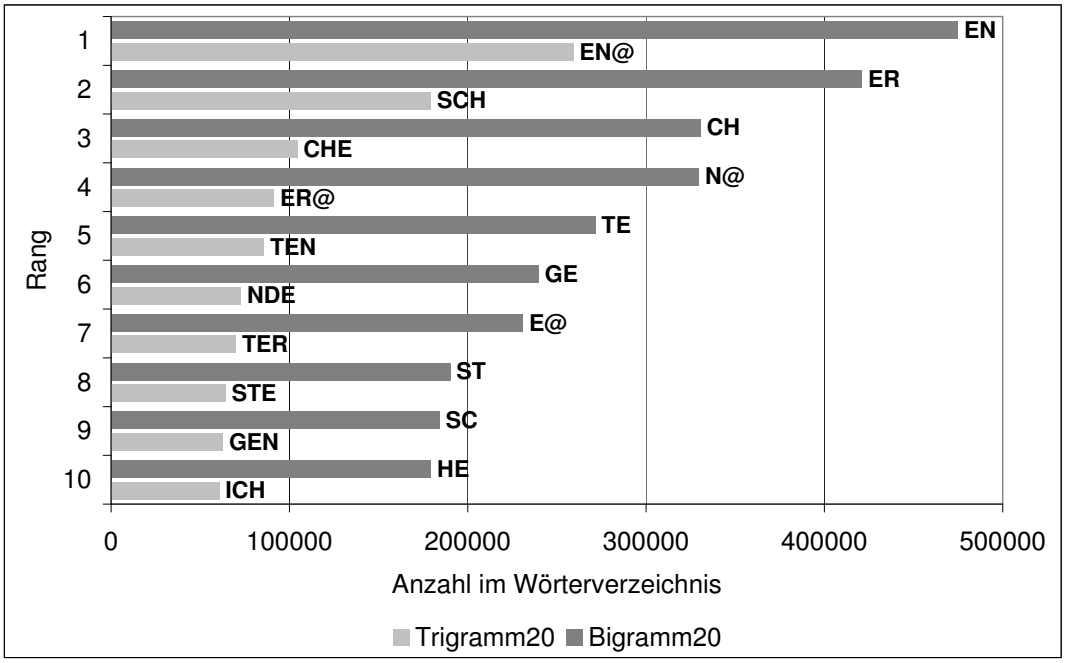


Abbildung 5.4: Vergleich Trigramm20 - Bigramm20. Erläuterungen im Text.

Abbildung 5.3 zeigt die zehn häufigsten Bigramme. Dargestellt ist die Anzahl der Bigramme im Vollformenlexikon, wenn nur die ersten zehn ■ oder die ersten zwanzig ■ Zeichen eines Wortes berücksichtigt werden. Die drei häufigsten Bigramme sind ›EN‹, ›ER‹ und ›CH‹, wobei ›EN‹ und ›ER‹ zwischen den beiden Codierungen ihre Plätze tauschen.

Das in der Codierung Bigramm20 an vierter Stelle stehende endständige ›N‹, als Bigramm ›N@‹ weist darauf hin, dass unter den Wörterbucheinträgen mit mehr als zehn Zeichen das endständige ›N‹ gehäuft vorkommt. Abbildung 5.4 vergleicht die zehn häufigsten Bigramme ■ mit den zehn häufigsten Trigrammen ■. Das Zeichen ›@‹ steht wieder für einen Worttrenner (am Anfang für einen Wortanfang, am Ende für ein Wortende). In beiden Fällen wurden nur die ersten 20 Zeichen eines Wortes berücksichtigt.

Während ›ER@‹ als Trigramm auf Rang 4 steht und ›ER‹ das zweithäufigste Bigramm ist, folgt das endständige ›R@‹ bei den Bigrammen erst auf Rang 26 mit 110 814 Vorkommen im Vollformenlexikon.

Für die Bigrammcodierung wurde untersucht, wie groß die Unterschiede sind, wenn mehr oder weniger Zeichen codiert werden und ob links- oder rechts-trunkiert wird. Ob 20, 30 oder 40 Zeichen berücksichtigt werden, spielt nur eine geringe

Rolle. Die relativen Unterschiede in der Anzahl der Bigramme im Vollformenlexikon liegen unter 1%. Die Erklärung liegt in der Verteilung der Wortlängen im Vollformenlexikon: Nur 12 928 von 1 410 165 oder 0,9% der Wörter enthalten mehr als 20 Zeichen.

Vergleicht man Links- und Rechtstrunkierung, dann behalten die Bigramme ebenfalls ihre Rangfolge bis auf eine Vertauschung von ›CH‹ und endständigem ›N@‹. Das endständige ›E@‹ ist bei den linkstrunkierten Wörtern 2% häufiger zu finden, das endständige ›N@‹ sogar 4% häufiger. Die übrigen Unterschiede liegen wieder unter 1%.

Im Anhang (Tabelle 7.6 auf Seite 175) sind viele der Wörter aufgeführt, die mehr als dreißig Zeichen lang sind. Im wesentlichen handelt es sich einerseits um ausgeschriebene Zahlwörter, andererseits finden sich aus spielerischen oder künstlerischen Gründen erfundene Bandwurmwörter. Zu einem geringeren Anteil findet man auch unbeabsichtigte Zusammenziehungen durch fehlende Worttrennerzeichen. Die Tabelle 7.7 auf Seite 176 enthält alle Wörter mit mehr als vierzig Zeichen.

5.2 Beispiele

Vor der weiteren statistischen Auswertung im Abschnitt 5.3 sollen typische Ergebnisse an ausgesuchten Beispielen verdeutlicht werden:

5.2.1 Dampfschiff

Abbildung 7.8 im Anhang auf Seite 177 listet die Ergebnisse der verschiedenen Codierungen bei der Suche nach dem Wort »Dampfschiff« detailliert auf. Die Kölner Phonetik lieferte 181 Begriffe, von denen dort nur die aufgeführt wurden, die mit einer anderen Codierung ebenfalls angezeigt wurden. Alle anderen Begriffe waren thematisch bedeutungslos und mussten den falsch gefundenen Begriffen zugeordnet werden.

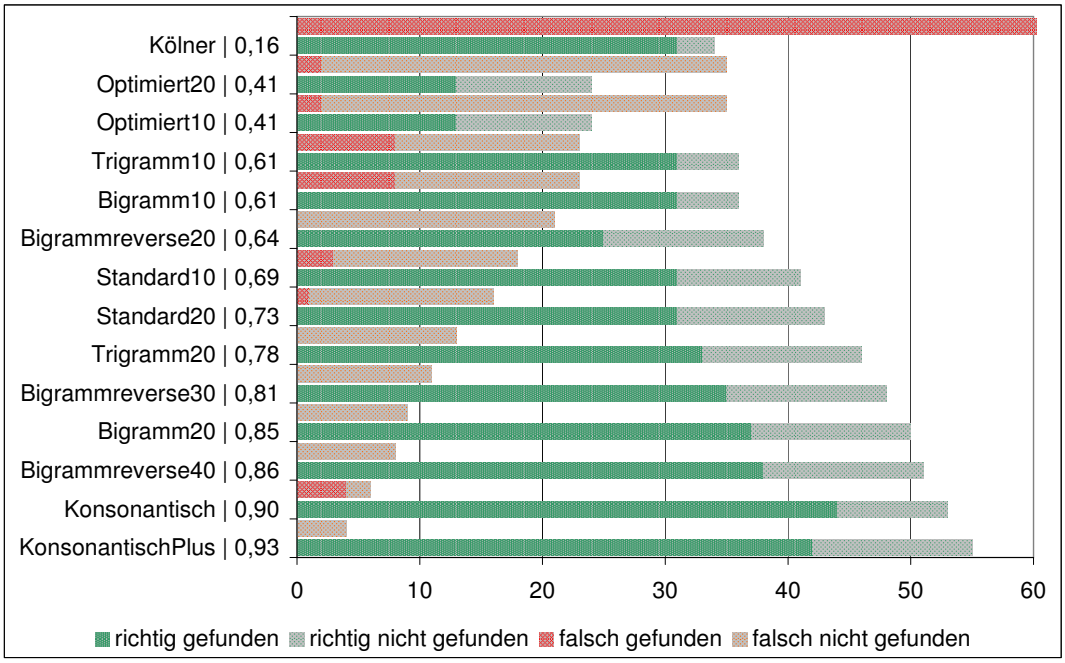


Abbildung 5.5: Suchergebnis der unscharfen Suche nach dem Wort »Dampfschiff«. Die Abbildung zeigt die Ergebnisse der unscharfen Suche mit den verschiedenen Codierungen. Zu finden waren 46 Wörterbucheinträge, die intellektuell als relevant klassifiziert worden waren (siehe Abbildung 7.8 im Anhang). Die Codierung nach der Kölner Phonetik ergab 162 falsch gefundene und 15 falsch nicht gefundene Einträge, diese Zahlen wurden im Diagramm nicht mehr dargestellt. Die Codierungen auf der Abszisse wurden nach der Korrektklassifikationsrate sortiert. Die Korrektklassifikationsrate wurde als $\frac{\text{richtigpositive + richtignegative}}{\text{Summe der Resultate}}$ berechnet.

In Abbildung 5.5 werden die Ergebnisse der unscharfen Suche nach dem Wort »Dampfschiff« mit den verschiedenen Codierungen zusammengefasst. Mit Korrektklassifikationsraten von 0,93 und 0,90 schneiden in diesem Versuch die modifizierten Bigrammcodierungen KonsonantischPlus und Konsonantisch (siehe Abschnitt 4.5.1.2) am besten ab, gefolgt von den längeren linkstrunkierten Bigrammcodierungen Bigrammreverse40 und Bigrammreverse30 mit Korrektklassifikationsraten von 0,86 und 0,81. Dazwischen liegt noch die klassische rechtstrunkierte Bigrammcodierung Bigramm20, die maximal zwanzig Zeichen erfasst und eine Korrektklassifikationsrate von 0,85 aufweist. Alle anderen Codierungen lieferten weniger als 80 % korrekte Resultate. Die Korrektklassifikationsraten wurden berechnet als

$$\frac{\text{richtigpositive + richtignegative}}{\text{Summe der Resultate}}$$

wie sie sich aus den Zahlen in den Zeilen drei bis sechs der Abbildung 7.8 im Anhang ergeben.

Die Korrektklassifikationsraten für einhundert Terme mit elf bis vierzehn Zeichen sind in Abbildung 5.9 auf Seite 114 und für zweihundert Terme mit fünfzehn bis siebzehn Zeichen in Abbildung 5.11 auf Seite 115 zusammengefasst.

Diese Ergebnisse lassen sich allein aus der Kenntnis der Genauigkeit der Codierung (vgl. Abbildung 5.2 auf Seite 103) nicht vorhersagen.

5.2.2 Beispiel purpur

Die Wortform »purpur» kommt im Korpus 22-mal in 17 Dateien vor. Die großgeschriebene Version »Purpur« kommt 1 277-mal in 882 Dateien vor. Dabei kommen beide Varianten zusammen nur in genau einer Datei vor[1]. Eine exakte Suche nach »purpur« oder »Purpur« liefert also zwei nahezu disjunkte Mengen von Treffern.

Tabelle 5.1: purpur mit verschiedenen Suchoptionen

Anzahl Dateien mit ... Vorkommen	Wortanfang, exakt	Wortanfang, Groß- und Kleinschreibung nicht unterschieden	Teilwort, exakt
1	815	1936	858
2	88	299	96
3	16	68	18
4	4	22	3
5	2	11	2
6		5	1
7		2	
9	1		1
10		2	
15		1	
23		1	

Tabelle 5.1 zeigt für drei verschiedene erweiterte Suchoptionen die Anzahl der Dateien, in denen die Wortformen vorkommen. Im ersten Fall wurde angegeben,

[1] Gryphius: Leo Armenivs,
KwiC-Ansicht:

- Das *purpur* guldne Kleidt
- Er ließ den *Purpur* fahren

dass »purpur« mit einem kleinen Anfangsbuchstaben am Wortanfang stehen sollte. Die Erweiterung des Suchbegriffs führte auf insgesamt 137 verschiedene Wortformen, die im Korpus in 926 Dateien insgesamt 1137-mal vorkommen. In 815 Dateien kam nur eine einzige Wortform aus der Liste der 137 Wortformen vor, davon war die Form »purpurnen« mit 304 Vorkommen in 275 Dateien die häufigste, gefolgt von »purpurne« mit 176 Vorkommen in 167 Dateien.

Wird die Groß- und Kleinschreibung nicht unterschieden, dann führt die Erweiterung des Suchbegriffs auf 457 unterschiedliche Wortformen, die im Korpus in 2347 Dateien insgesamt 3519-mal vorkommen.

Eine Suche nach kleingeschriebenem »purpur« als Teilwort führte zu 172 Wortformen, die in 979 Dateien insgesamt 1204-mal vorkamen. Die Suche mit einem unscharfen Verfahren erbrachte keine weiteren Variationen in der Schreibung.

5.3 Ergebnisse nach Länge des Suchterms

Die Anzahl der Zeichen im Suchterm hat einen entscheidenden Einfluß auf die Qualität des Retrievalergebnisses bei der unscharfen Suche mit einer speziellen Codierung. Für diesen Abschnitt wurden die im Abschnitt 4.5.1 auf Seite 81ff vorgestellten Codierungen miteinander verglichen. Die Codierungen „Optimiert10“, „Optimiert20“ und die „Kölner Phonetik“ wurden wegen der schlechten Performance in den Voruntersuchungen nicht einbezogen. So wurden die Suchterme mit einer zuvor definierten Anzahl von Zeichen mit Hilfe eines Zufallsgenerators aus dem Vollformenlexikon ausgewählt. Für die unscharfen Suche mit dem SpaCAM wurden die Bi- und Trigramm-Codierungen, die Konsonantischen Codierungen und die Standard-Codierungen verwendet. Im folgenden Abschnitt werden die Ergebnisse präsentiert, die mit den untersuchten Codierungen gemacht wurden.

5.3.1 Suchterme mit drei bis fünf Zeichen

Abbildung 5.6 zeigt, wie viele Wörter aus dem Vollformenlexikon gefunden wurden, wenn der Suchterm drei bis fünf Zeichen enthielt. Dazu wurden mit Hilfe eines Zufallgenerators 99 Suchterme aus dem Vollformenlexikon ausgewählt, die der genannten Bedingung entsprachen. Tabelle 5.2 nennt die Aufteilung der Terme nach Wortlänge. Ihre Häufigkeitsverteilung entspricht der Verteilung der Häufigkeiten im Vollformenlexikon, wie sie in Abbildung 3.5 auf Seite 54 angegeben ist.

Tabelle 5.2: Wortlänge und Anzahl Wörter mit drei bis fünf Zeichen

Wortlänge	3	4	5	
Suchterme	6	35	58	99

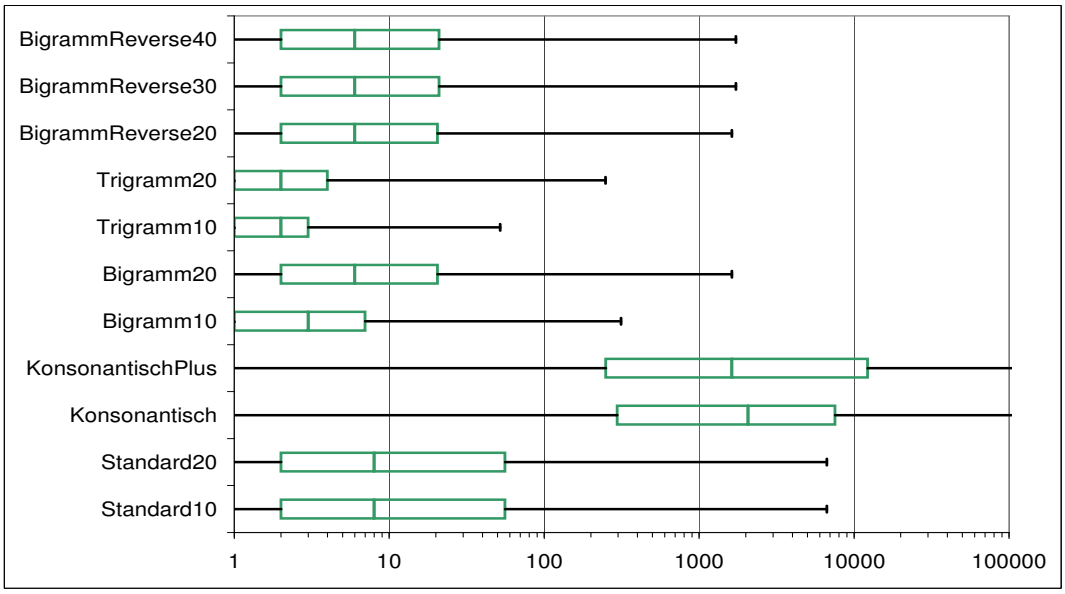


Abbildung 5.6: Anzahl der gefundenen ähnlichen Terme bei jeweils 99 Suchanfragen mit zufällig ausgewählten Termen von drei bis fünf Zeichen Länge. Die grünen Boxen bezeichnen das zweite und dritte Quartil, dazwischen liegt der Median. Die schwarzen Linien erstrecken sich vom Minimum bis zum Maximum.

In Abbildung 5.6 sind angegeben: das Minimum, das erste Quartil, der Median, das dritte Quartil und das Maximum der Anzahl der zurückgegebenen Terme je untersuchter Codierung. Je zwei Ausreißer wurden nicht mehr dargestellt: Bei den Codierungen Konsonantisch und KosonantischPlus erzeugten die Suchanfrage »Heil« mehr als 580 000 und die Suchanfrage »Neua« mehr als 230 000 zurückgegebene Begriffe. Optimal im Sinne eines übersichtlichen Ergebnisses sind die Trigrammcodierungen.

Die Unterschiede zwischen den Codierungen Bigramm10 und Trigramm10 auf der einen Seite und Bigramm20 und Trigramm20 auf der anderen Seite liegen natürlich nicht in der Länge der codierten Zeichenkette, sondern in den unterschiedlichen für die Codierung verwendeten Alphabeten A_1 und A_2, wie sie auf

Seite 82 beschrieben sind. Das Alphabet A_2 enthält Umlaute und Ziffern, und im Vollformenlexikon sind auch alle Zahlen enthalten.

In der Suchanfrage waren zufälligerweise acht vierstellige Zahlen und eine fünfstellige Zahl enthalten. Weil in den Codierungen, die das Alphabet A_1 verwendeten, Ziffern nicht codiert wurden, wurde in der Codierung des Suchterms (siehe Seite 81) ein „leerer“ Code gebildet und als Antwort nur die im Vollformenlexikon enthaltene Zahl selbst zurückgeliefert.

Die das Alphabet A_2 verwendenden n-Tupel-Codierungen gaben in der Regel nur eine Zahl zurück. In zwei Ausnahmefällen wurden drei bzw. vier gefundene „ähnliche“ Zahlen zurückgegeben.

5.3.2 Suchterme mit sechs bis zehn Zeichen

In Abbildung 5.7 wurden die Ergebnisse für die Suche nach einhundert zufallsgesteuert ausgewählten Einträge aus dem Vollformenwörterbuch mit einer Länge von sechs bis zehn Zeichen in drei Klassen aufgeteilt und zusammen dargestellt.

Wie in Tabelle 5.3 aufgelistet, bildeten 34 Wörter mit sechs oder sieben Zeichen Länge die erste Klasse, deren Verteilung in der Abbildung 5.7 jeweils als der unterste Balken in jeder Codierung eingetragen ist. Darüber bilden 37 Wörtern mit acht oder neun Zeichen die nächste Klasse. Deren Ergebnisse wurden als mittlerer Balken eingetragen. Schließlich folgt mit 29 Wörtern die Klasse der Wörter mit zehn Zeichen Länge. Deren Ergebnisse bilden den obersten Balken je Codierung.

Eingezeichnet sind die Zahl der zurückgegebenen Wörter mit der unscharfen Suche, und zwar das Minimum, das erste Quartil, der Median, das dritte Quartil und die maximale Anzahl an zurückgegebenen Wörterbucheinträgen. Außer bei den konsonantischen Codierungen sind das Minimum und das erste Quartil immer eins. Die nicht eingetragenen Ausreißer bei den Standardcodierungen liegen bei 1 462, bei den konsonantischen Codierungen bei 6 008 und 4 164. Die für diese drei Klassen in Summe wenigsten Ergebnisse liefert die Codierung Bigramm20 mit 245 zurückgelieferten Termen.

Tabelle 5.3: Wortlänge und Anzahl Wörter mit sechs bis zehn Zeichen

Wortlänge	6–7	8–9	10	
Suchterme	34	37	29	100

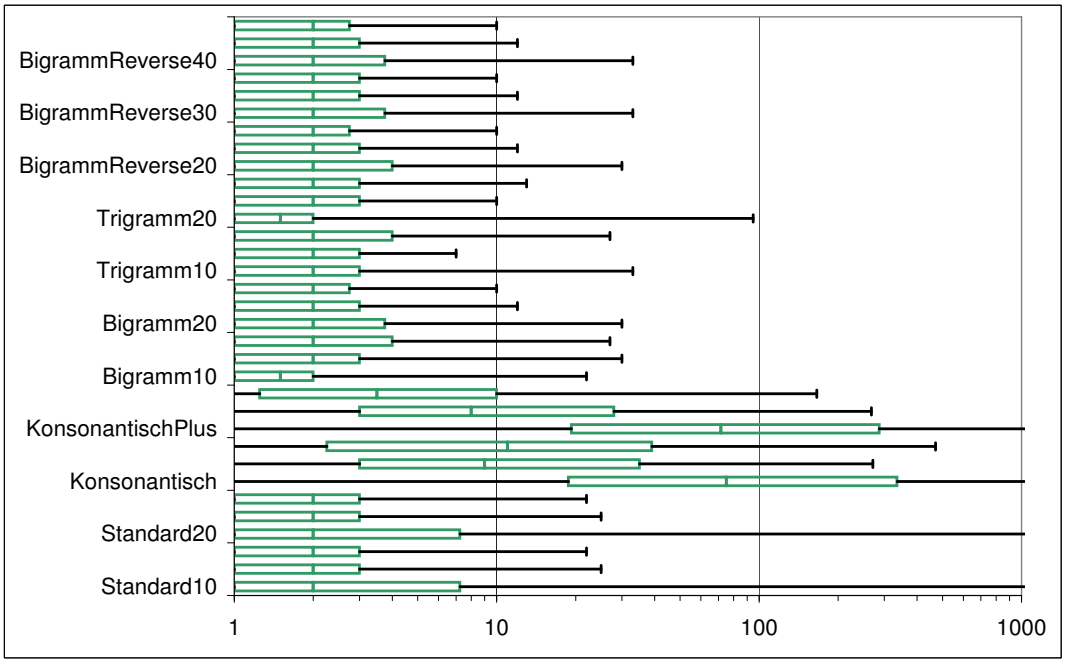


Abbildung 5.7: Einhundert Einträge aus dem Vollformenwörterbuch mit sechs bis zehn Zeichen wurden zufallsgesteuert ausgewählt. Je Codierung wurden die Suchterme entsprechend Tabelle 5.3 von unten nach oben aufgeteilt in drei Klassen.
Abgebildet sind die Zahl der zurückgegebenen Wörter mit der unscharfen Suche, und zwar das Minimum (immer 1), das erste Quartil (außer bei den konsonantischen Codierungen immer 1), der Median, das 3. Quartil und die maximale Anzahl an zurückgegebenen Wörterbucheinträgen.

5.3.3 Suchterme mit elf bis vierzehn Zeichen

Tabelle 5.4: Wortlänge und Anzahl Wörter mit elf bis vierzehn Zeichen

Wortlänge	11	12	13	14	
Suchterme	26	14	12	22	74
Gesamtzahl Ergebnis	1 540	166	314	940	2 960

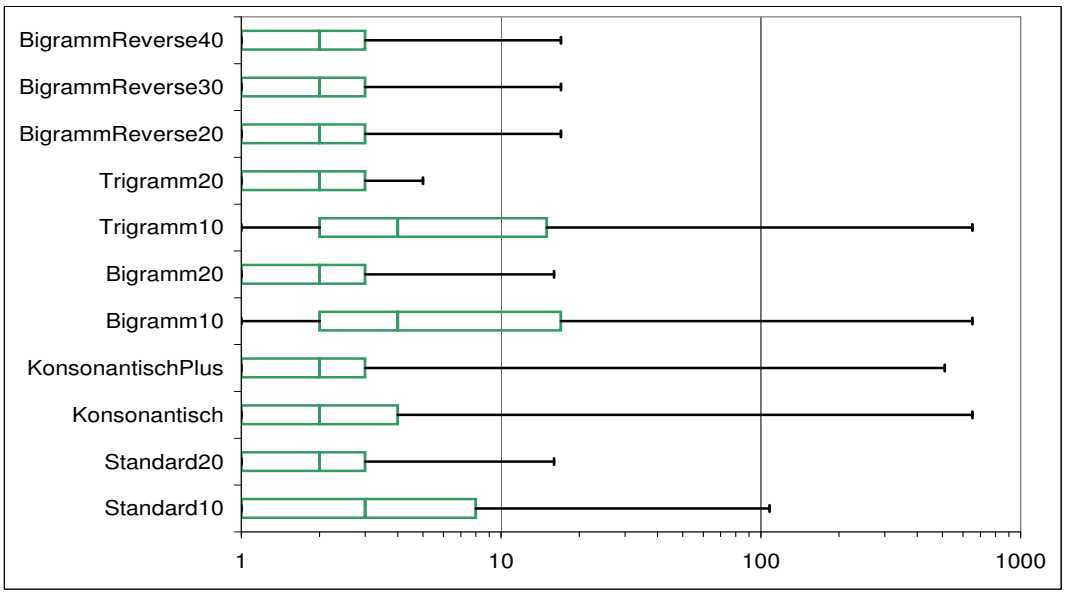


Abbildung 5.8: Aus dem Vollformenlexikon wurden 74 Suchterme mit elf bis vierzehn Zeichen zufällig ausgewählt. Die Codierungen, die nur zehn Zeichen berücksichtigen, liefern deutlich mehr Ergebnisse.

Bei Wortlängen im Bereich zwischen elf und vierzehn Zeichen liegen alle Codierungen, die zwanzig Zeichen codieren und das Alphabet A_2 verwenden gleichauf mit im Median zwei zurückgegebenen Ergebnissen. Die Trigramm20-Codierung weist die wenigsten Ausreisser bezüglich der maximalen Anzahl von Ergebnissen auf. Bei den konsonantischen Codierungen lieferte die Anfrage nach dem Wort »empfinden14« einen Ausreisser mit 514 zurückgegebenen (meist falschen) Resultaten.

Anders als im Fall der kurzen Wörter liefern die Codierungen Bigramm10 und Trigramm10 jetzt erheblich mehr meist falsche Ergebnisse.

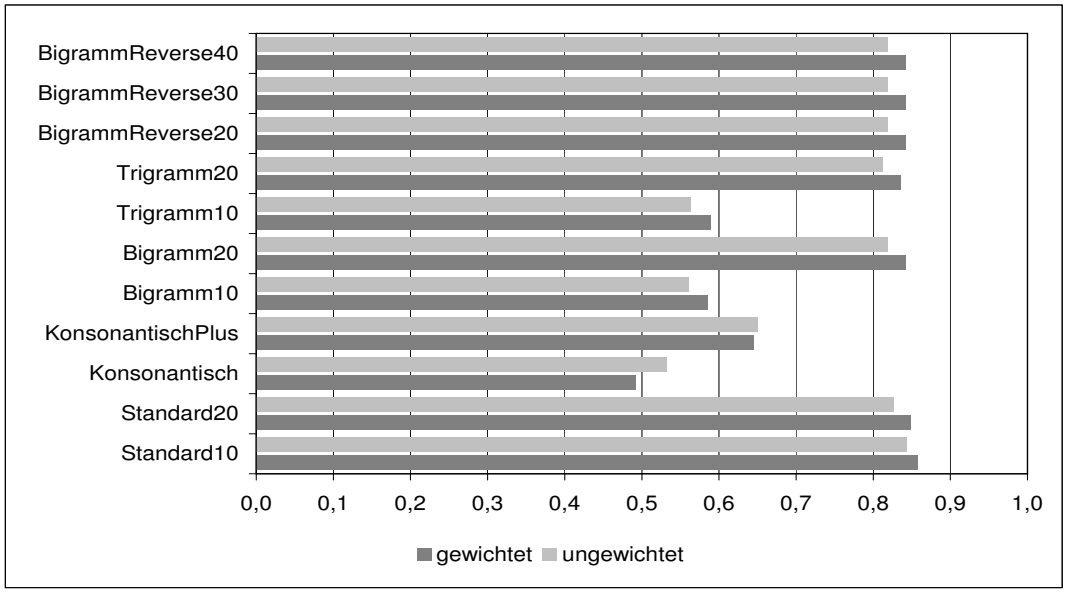


Abbildung 5.9: Korrektklassifikationsrate für 74 Suchanfragen mit Termen, die elf bis vierzehn Zeichen enthalten. Die Werte beziehen sich auf die Korrektklassifikationsrate bezüglich der Einträge im Vollformenlexikon (ungewichtet) oder wurden mit der Anzahl im Korpus gewichtet.

74 Suchanfragen mit Termen, die elf bis vierzehn Zeichen enthalten, wurden ausgewertet. Dazu wurden 2 960 Ergebnisterme intellektuell beurteilt, ob sie in Bezug auf die Suchanfrage relevant waren oder nicht. Abbildung 5.9 zeigt die erzielten Korrektklassifikationsraten. Dabei wurden die Zahlen für die Suche im Vollformenlexikon als ungewichtet dargestellt. Werden diese Zahlen mit der Häufigkeit der Terme im Korpus gewichtet, ergeben sich leichte Verschiebungen. Die Codierungen Trigramm10, Bigramm10, KonsonantischPlus und Konsonantisch liefern unterdurchschnittliche Ergebnisse, zwischen den anderen Codierungen sind keine nennenswerten Unterschiede festzustellen.

5.3.4 Suchterme mit fünfzehn bis siebzehn Zeichen

Aus allen Einträgen im Vollformenlexikon mit fünfzehn bis siebzehn Zeichen Länge wurden zweihundert Terme zufallsgesteuert ausgewählt und mit den verschiedenen Codierungen nach ähnlichen Termen gesucht. Auf diese Weise wurden insgesamt 9 283 Terme gefunden, die intellektuell auf korrekte Klassifikation überprüft wurden.

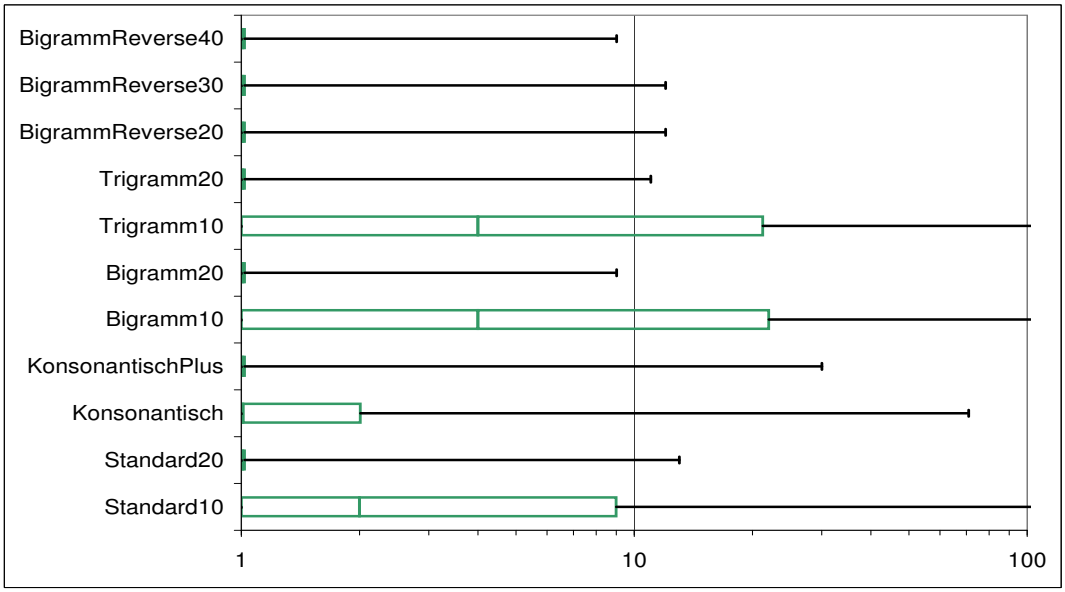


Abbildung 5.10: Anzahl der gefundenen ähnlichen Terme bei zweihundert Suchanfragen mit zufällig ausgewählten Termen von fünfzehn bis siebzehn Zeichen Länge. Bei den korrekt klassifizierenden Codierungen (siehe nächste Abbildung) wird in weniger als einem Viertel der Fälle der Suchterm erweitert.

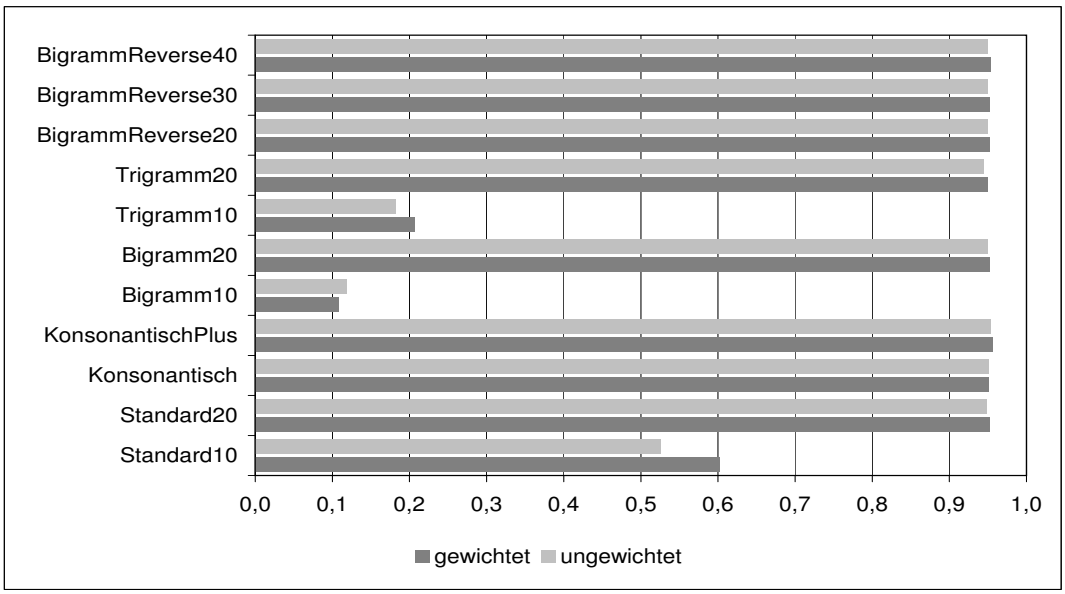


Abbildung 5.11: Korrektklassifikationsraten für 200 Terme mit fünfzehn bis siebzehn Zeichen. 9 283 Terme wurden ausgewertet.

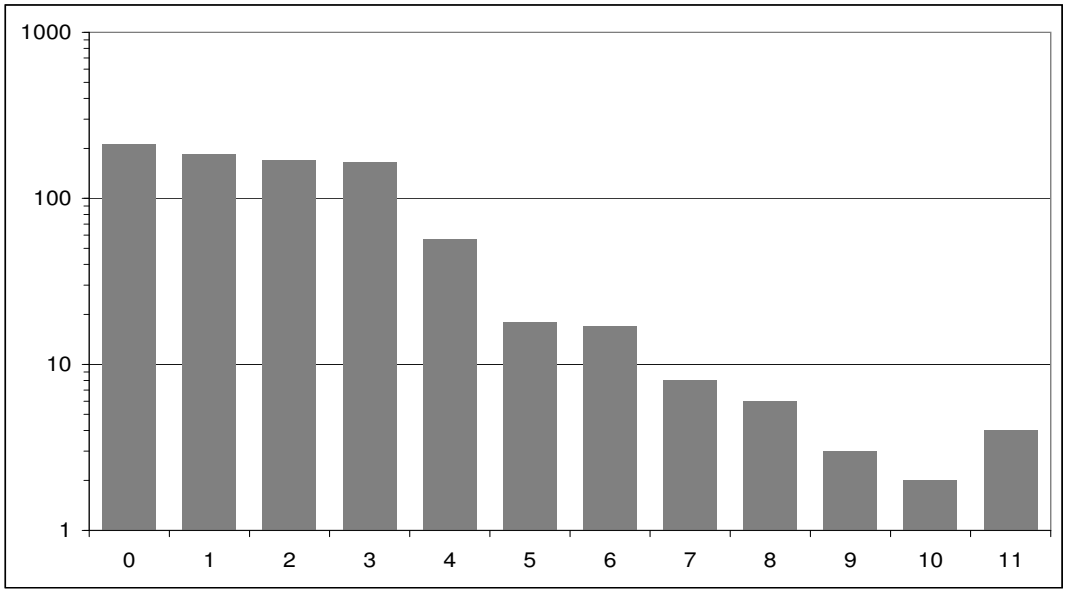


Abbildung 5.12: Levenshteindistanzen für die Ergebnisterme der unscharfen Suche über zweihundert Terme mit fünfzehn bis siebzehn Zeichen, die als korrekt klassifiziert wurden.

Wie Abbildung 5.12 belegt, ist die Levensthein-Distanz (vgl. Seite 74) kein geeignetes Maß, um „ähnliche“ Terme zu finden. Die ähnlichen Terme mit hoher Levenshtein-Distanz ergeben sich aus Teilwortbeziehungen wie z. B. »Verstandesverwandtschaft« als Ergebnisterm zum Suchterm »Verwandtschafft«.[2]

5.3.5 Suchterme mit mehr als zwanzig Zeichen

Tabelle 5.5: Wortlänge und Anzahl Wörter mit mehr als zwanzig Zeichen

Wortlänge	21	22	23	24	25	26	28	29	30	31	39	
Suchterme	38	28	12	11	3	2	1	1	1	1	1	99

[2]Dieses Ergebnis wurde mit den eingestellten Parametern ausschließlich mit den Codierungen BigrammReverse30 und BigrammReverse40 erhalten und würde sich ohne unscharfe Suche wegen der altertümlichen Schreibweise nur mit einem verkürzten Teilwort »VERWANDTSCHAF« finden lassen.

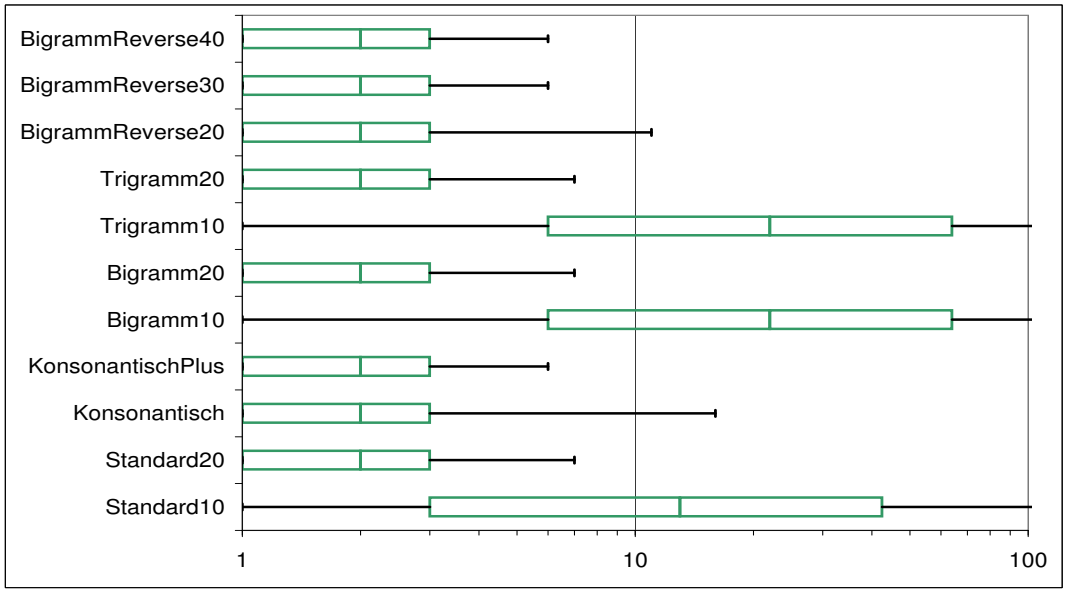


Abbildung 5.13: Wortlänge: mehr als zwanzig Zeichen. Die Codierungen, die nur zehn Zeichen berücksichtigen, liefern viele Ergebnisse.

Tabelle 5.6: Ausreisser in Abbildung 5.13

	Median	Oberes Quartil	Maximum
Standard10	13	42,5	610
Bigramm10	22	64	1928
Trigramm10	22	64	1758

Der schon in Abbildung 5.8 zu beobachtende Trend der schlechteren Performance für die Codierungen Standard10, Bigramm10 und Trigramm10 hat sich nochmals verstärkt. Mit den in Tabelle 5.6 genannten Zahlen für die Codierungen, die nur zehn Zeichen codieren, ergeben dies Codierungen in der Mehrzahl der Fälle keine sinnvollen Erweiterungen des Suchbegriffs mehr.

5.4 Seltene Substantive

Aus den Wortformen, die zwischen zwei- und zehnmal im Korpus vorkamen, wurden mit Hilfe eines Zufallsgenerators 325 Substantive ausgewählt und nach diesen Substantiven mit verschiedenen Verfahren zur Termerweiterung gesucht. Aus diesem Bestand kommen die folgenden näher erläuterten Ergebnisse.

Zunächst werden zwei positive Beispiele dargestellt, die zeigen, wie die Qualität der Suchanfrage durch die Termerweiterung steigt. Als Suchwort wurde das Wort »Alltagsgesichter« ausgewählt. Das Wort kommt im Korpus in zwei Dateien je einmal vor. Verschiedene Erweiterungsmethoden führen zu unterschiedlichen Ergebnissen. Insgesamt kommt das Wort in seinen Deklinationsformen (Alltagsgesicht | ~e | ~er | ~ern | ~es | ~s) in elf verschiedenen Dateien je einmal vor. In der Grundform findet man sechs Vorkommen, die Form »Alltagsgesichte« findet man einmal, die beiden anderen Formen je zweimal. Eine Suche nur nach der exakten Grundform liefert also nur 55% aller tatsächlichen Vorkommen.

Ein ähnliches Beispiel liefert die Suche nach dem Wort »Rittergedicht«.

Dieses Wort ist je einmal in zwei Dateien zu finden. Die Deklinationsformen (~e | ~en | ~es | ~s) führen zu sechs weiteren Dateien, die eine dieser Formen je einmal enthalten. Damit ergibt die Suche ausschließlich nach der Grundform nur 25% aller Vorkommen bezogen auf die Gesamtzahl aller deklinierten Formen.

Die Suchen mit dem Webservice *wordforms* der Universität Leipzig[3] erbrachte in beiden Fällen keine weiteren Wortformen. Je nach Einstellung der Suchoptionen war eine unscharfe Suche oder eine Teilwortsuche optimal, um die Wortformen ohne zusätzliche unerwünschte Terme zu finden.

Zu diesen vorteilhaften Beispielen finden sich allerdings auch negative Gegenbeispiele: Das Wort »Freundli« wurde in zwei Dateien je einmal gefunden. Der ersten Fall verweist auf einen Parserfehler: »Du Bild der Freundli[ch]keit!«[4], der zweite Fall bezieht sich auf eine mundartliche (oberbayerische) Zusammenziehung mit einem als Worttrenner verkannten Apostroph: »Was Schön's und Freundli's in der Musi' is,«[5].

Die Erweiterung des Wortes führt zur Inklusion aller Wörter, in denen »freundli« als Teilwort enthalten ist, insbesondere auch zur Inklusion des Wortes »freundlich«, das im Korpus 10 661-mal vorkommt. Zusammen werden zu den ursprünglichen zwei Dateien weitere 11 399 Dateien mit insgesamt 22 202 Wortformen gefunden.

5.4.1 Webservice *synonyms*

Die Erweiterung mit dem Webservice *synonyms* erbrachte nur für fünfzehn Suchbegriffe weitere Suchterme. Von diesen zusätzlichen Suchtermen kamen einige nicht im Korpus vor (»Andragogik« als Erweiterung von »Erwachsenenbildung«, »Unumstößlichsein« als Erweiterung von »Unumstößlichkeit«, »Verzichtserklärung« als

[3] http://wortschatz.uni-leipzig.de/axis/servlet/ServiceOverviewServlet

[4] http://gutenberg.spiegel.de/?id=5&xid=1009&kapitel=8&cHash=2f0e7e71912#gb_found

[5] http://gutenberg.spiegel.de/?id=5&xid=1494&kapitel=49&cHash=43591ae54bkobe048#gb_found
http://gutenberg.spiegel.de/?id=12&xid=1494&kapitel=49&cHash=43591ae54b2

Erweiterung von »Verzichterklärung«), andere führten mit mehr als 11 000 Vorkommen zu fast 6 000 weiteren Dateien (»Nichts« als Erweiterung von »Gehaltlosigkeit«) oder über 3 572 Vorkommen in 1 854 Dateien für »Gebiet« als Erweiterung von »Nationalpark«. In den fünf Fällen, in denen die gefundenen Synonyme nicht zu viele Vorkommen im Korpus hatten, erscheint die Erweiterung sinnvoll und nützlich, wie Tabelle 5.7 nahelegt.

Tabelle 5.7: nützliche Synonyme

Suchwort	Erweiterung	Häufigkeit
Bärenzucker		7
	Bärendreck	3
	Lakritze	1
Hundehalsband		3
	Halsung	2
Kartenlegen		7
	Kartenschlagen	3
Klapperkiste		6
	Klapperkasten	3
Rückfenster		2
	Heckfenster	3

5.4.2 Vergleich Bigramm-*wordforms*

In Tabelle 5.8 wird dargestellt, wie viele relevante Suchterme zusätzlich zum Term der Anfrage ermittelt wurden. 321 Anfragen wurden expandiert. *wordforms* lieferte in 71 Fällen (≙ 22 %) mehr als nur den Suchterm selbst als Ergebnis zurück. Allerdings war nur in 49 Fällen (≙ 15 %) einer der zusätzlich zurückgegebenen Terme auch im Korpus enthalten. Werden nicht im Korpus vorhandene Terme nicht berücksichtigt, dann liefert *wordforms* ausschließlich relevante Ergebnisse, jedoch nur für recht wenige Fälle.

Im Gegensatz dazu ergibt die Termerweiterung mit der unscharfen SpaCAM-Suche mittels Bigrammen in 220 Fällen (≙ 69 %) zusätzliche relevante Terme, jedoch auf Kosten der Genauigkeit, wie Abbildung 5.14 zeigt.

Abbildung 5.14 zeigt für die Erweiterung des Suchterms mit der unscharfen Suche mit Bigrammen das Verhältnis $\frac{\text{relevante Suchterme}}{\text{Gesamtzahl zurückgegebener Suchterme}}$. Auf der Abszisse ist in Klassen gruppiert angegeben, wie viele Suchterme insgesamt zurückgegeben wurden. Diese Klassen enthalten im Unterschied zur ersten Spalte in Tabelle 5.8 ein Element mehr, weil der Suchterm selbst stets zurückgegeben wird.

Tabelle 5.8: Vergleich zusätzlich gefundener relevanter Suchterme

Zusätzliche Suchterme	*wordforms*	Bigramm
0	272	101
1	28	126
2	14	36
3	5	23
4	2	9
5	0	6
6	0	8
7	0	1
8	0	3
9	0	2
10	0	0
20	0	3
>20	0	3

Die Relevanz wurde intellektuell bestimmt. Nicht relevante Ergebnisse wurden als solche markiert. Die Ordinate bezeichnet den Anteil der relevanten Suchterme der Stichprobe in Prozent bezogen auf die Gesamtzahl der Suchterme der Stichprobe. Zusätzlich wurde das Konfidenzintervall für $\alpha = 0{,}05$ eingetragen. Das Konfidenzintervall bezeichnet den Bereich innerhalb dessen eine Wahrscheinlichkeit von $1 - \alpha$ besteht, dass der Mittelwert der Gesamtheit tatsächlich innerhalb dieses Bereichs liegt. Das Konfidenzintervall ist um so größer, je mehr die Messwerte streuen und je weniger Messungen durchgeführt wurden. Für den Bereich bis fünf gefundene Terme oder vier zusätzlich gefundene Terme sind diese Terme zu mehr als 50 % relevant. Diese Aussage bezieht sich auf 194 Fälle, entsprechend 60 % der Suchanfragen.

5.5 Andere Suchmaschinen

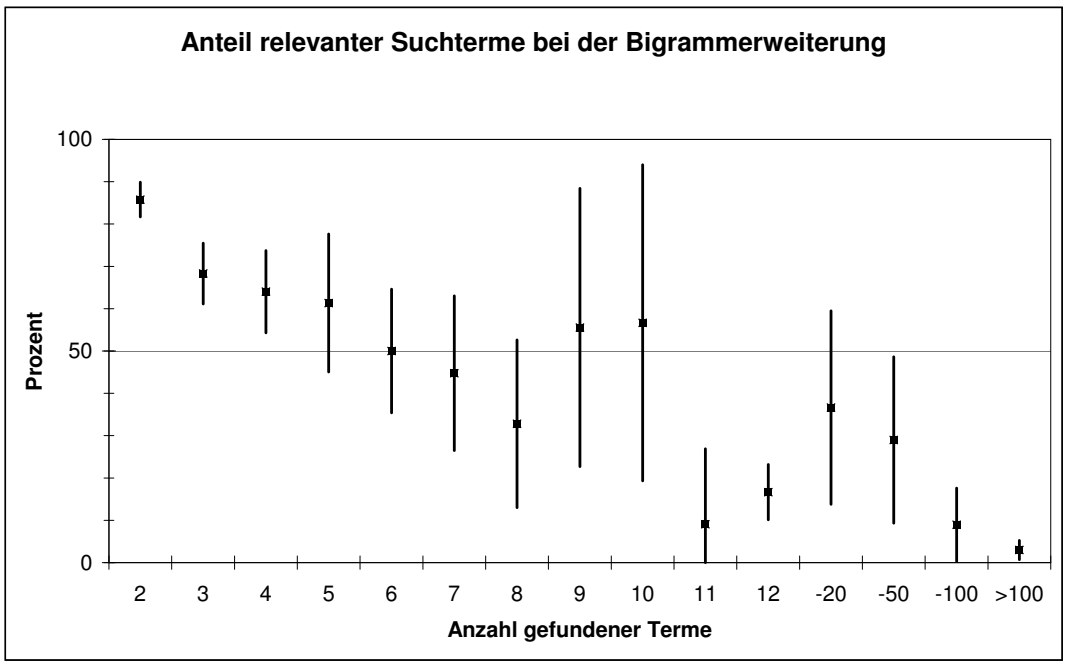


Abbildung 5.14: Anteil relevanter Suchterme bei der Bigrammerweiterung. Eingetragen sind der Mittelwert des Anteils relevanter Suchterme von der Gesamtzahl zurückgegebener Suchterme bei der Erweiterung mittels Bigrammsuche und das Konfidenzintervall für $\alpha = 0{,}05$.

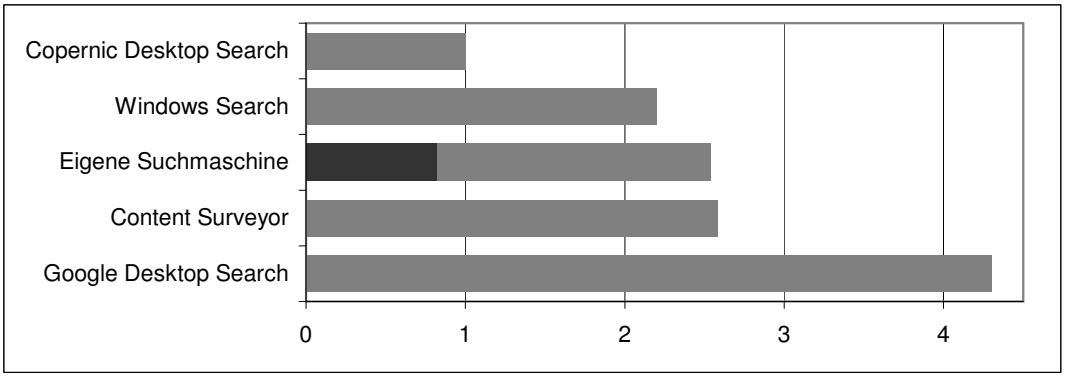


Abbildung 5.15: Speicherbedarf für die Indexierung in GB. Der Speicherbedarf für die eigene Suchmaschine bezieht sich auf die experimentelle Version ohne Datenkompression und mit einer Vielzahl von Suchalgorithmen. Der schwarze Balken bezeichnet den Platzbedarf für eine komprimierte Variante. Der Speicherbedarf des Content Surveyors wurde hochgerechnet.

Zum Vergleich wurden in einige andere Suchmaschinen eine Auswahl von einhundert der oben verwendeten seltenen Substantive eingegeben. Diese Substantive kamen zwei- bis zehnmal im Korpus vor. Abbildung 5.15 nennt den Speicherbedarf der Suchmaschinen in Gigabyte für den Index. Die eigene Suchmaschine arbeitete in dieser experimentellen Version ohne Datenkompression und mit einer Vielzahl von Indices für die verschiedenen Suchalgorithmen. Der Wert für die komprimierte Version wurde als schwarzer Balken davor eingetragen. Der Wert für den Content Surveyor wurde zum Vergleich hochgerechnet, weil das Programm nur einen Bruchteil der Dateien verarbeiten konnte. Diese Daten werden in Abschnitt 6.6 diskutiert.

5.5.1 Windows Desktopsearch

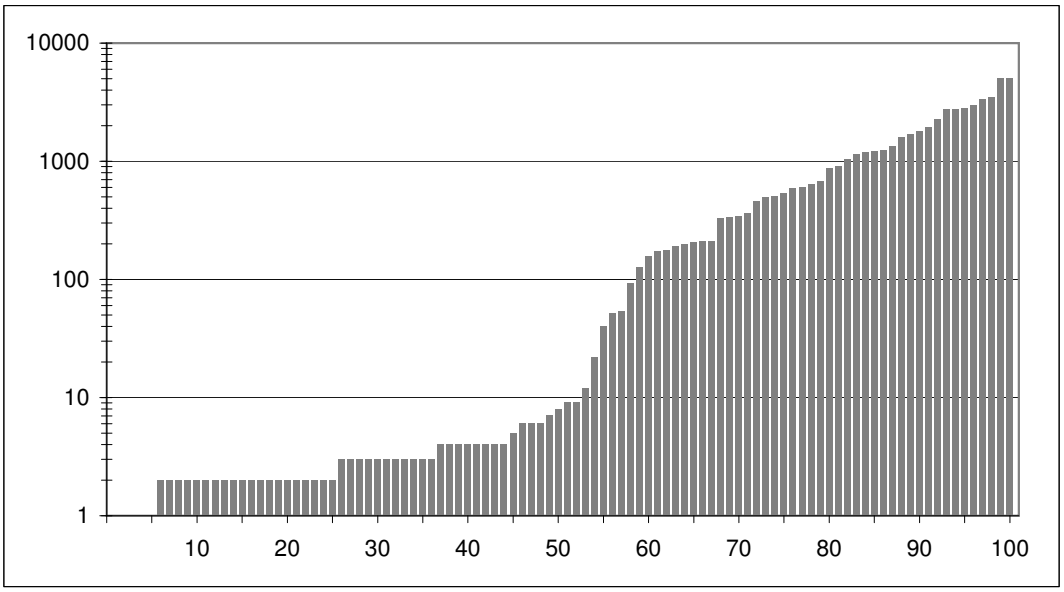


Abbildung 5.16: Anzahl von MS-Search zurückgegebener Dateien auf einhundert Surchterme

Für die erstmalige Indexierung der Dateien des Projekts Gutenberg-DE benötigte Windows® Search Version 3.01[6] etwa zwölf Stunden.

Abgesehen von Antwortzeiten, die bei längeren Rückgabelisten im Sekundenbereich lagen, war nur die Hälfte der Ergebnisse verwendbar. Abbildung 5.16 zeigt die Anzahl der je Suchterm zurückgegebenen Dateien. In vierzig Prozent der Fälle wurden mehr als einhundert Dateien zurückgegeben, die zum Suchbegriff passen sollten.

Leider wird nicht explizit angegeben, um welche Begriffe es sich handeln könnte. Auch die Referenz in der Hilfestellung gibt keine Auskunft darüber, nach welchem Algorithmus die Vorverarbeitung oder das Retrieval ablaufen. Deshalb können hier keine näheren Aussagen darüber getroffen werden, wie es um die Relevanz der Terme in der unscharf erweiterten Termmenge bestellt ist.

Für die Terme »Erdenalter« und »Erdwelt« wurden 5 017 bzw. 5 018 Dateien als ergebnisrelevant zurückgeliefert.

[6] http://www.microsoft.com/downloads/details.aspx?familyid=738FC2DE-49B9-4E69-9227-2206277AB7C9&displaylang=de

Tabelle 5.9: MS-Search für Wortanfänge von »Gebetserhörungen«

Suchterm	Anzahl Dateien
Ge	65 658
Geb	40 785
Gebe	24 450
Gebet	6 527
Gebets	216
Gebetse	4
Gebetser	603
Gebetserh	4
Gebetserhö	4
Gebetserhör	4
Gebetserhöru	4
Gebetserhörun	4
Gebetserhörung	4
Gebetserhörunge	2
Gebetserhörungen	2

Neben anderen Kriterien werden anscheinend Komposita in ihre Bestandteile zerlegt und dann der erste Teil des Kompositums als Suchterm verwendet. Dies erklärt das Verhalten jedoch nicht vollständig, wie Tabelle 5.9 zeigt: Für »Gebetser« liegt mit der Anzahl der gefundenen Dateien ein Anomalie vor. Im Korpus kommen »Gebetserhörung« und »Gebetserhörungen« je einmal in je zwei Dateien vor. Für »Gebetse« werden korrekt diese vier Dateien angegeben, ebenso für die nachfolgenden Wortanfänge. Für »Gebetser« werden jedoch auch Dateien angegeben, die z. B. als einziges ähnliches Wort den Term »Gebetsgemachs« enthalten.

5.5.2 Google Desktopsearch

Die Desktopsuche[7] von Google benötigte etwa acht Stunden für die erstmalige Indexierung der Dateien im Leerlauf. Anders als auf der Seite angegeben darf der Rechner dafür nicht im Ruhezustand sein (in dem der Prozessor abgeschaltet ist),

[7] Version Google Desktop 5.9.0911.03589-de-pb

sondern muss sich im Leerlauf befinden, d. h. es dürfen keinerlei Benutzeraktivitäten stattfinden, sonst wird die Indexierung für eine Stunde unterbrochen.[8]

Nachdem die Google-Desktopsuche prinzipiell alle Festplattenlaufwerke durchsucht, die Dateien des Projekts Gutenberg aber in einem Unterordner auf einer Festplatte gespeichert waren, wurden alle anderen Festplatten und die anderen Ordner im Hauptverzeichnis der Festplatte von der Suche ausgenommen.

Weil einige Textdateien mit den Auswertungsergebnissen zur unscharfen Suche in einem Ordner parallel zum Ordner mit den Dateien des Projekts Gutenberg lagen, wurde dieser Ordner mit indexiert. Diese Dateien wurden bei der Desktopsuche bevorzugt dargestellt, weil die Standardeinstellung „Nach Datum sortiert“ ist.

Die Suche nach dem Wort »Goethe« dauerte beim ersten Mal 16,5 Sekunden, bei den folgenden Suchvorgängen 0,3 und 0,15 Sekunden. Wurde die Standardeinstellung belassen und nach Datum sortiert, dann wurden 217 Dateien gefunden. Sollte nach Relevanz sortiert werden, dann wurden nur 198 Dateien gefunden, darunter viele JavaScript-Dateien mit der Typkennung ».js«, die sich im Browser unmittelbar nicht darstellen lassen.

Im Korpus kommt das Wort »Goethe« 5 023-mal in insgesamt 1 423 HTML-Dateien vor und zusätzlich in 23 JavaScript-Dateien.

Blätterte man durch die Seiten mit den Fundlisten, vergrößerte sich die Zahl der Fundstellen: Auf der zweiten Seite wurde von 379 gefundene Dateien berichtet, auf den folgenden Seiten steigerte sich diese Zahl bis hin zu 1 950 angeblichen Fundstellen. Tabelle 5.10 gibt einen Eindruck von der Zunahme der Zahl der gefundenen Dateien.

Tabelle 5.10: Google-Desktopsuche nach »Goethe«

Ergebnisseite	1	2	3	4	5	6	7	8	9
gefundene Dateien	198	379	544	695	835	964	1 083	1 194	1 298

Bei weiter hinten liegenden Seiten schwankte die Zahl der Fundstellen. So wurden beim Aufruf der Seite 37 1 375 Fundstellen berichtet, auf Seite 187 auch wieder 2 013 Fundstellen. Als letzte Seite wurde Seite 214 mit einer Fundstelle und insgesamt 2 131 gefundenen Dateien angezeigt, während Seite 213 noch 2 151 Fundstellen berichtete.

[8] Unter der Voraussetzung, dass Ihr Computer eingeschaltet bleibt, sollte die Erstindizierung durch Google Desktop innerhalb weniger Tage abgeschlossen sein. Dieser Erstindizierungsvorgang wird nur durchgeführt, wenn sich Ihr Computer im Ruhezustand befindet. Deshalb empfiehlt es sich, Ihren Computer nach der Installation von Google Desktop für ein oder zwei Abende eingeschaltet zu lassen. http://desktop.google.com/support/bin/answer.py?hl=de&answer=12403

Dieser Sachverhalt scheint allgemein zu gelten: bis zur vorletzten Seite wird die Anzahl der Ergebnisse grob geschätzt, erst auf der letzten Seite wird die genaue Zahl genannt.

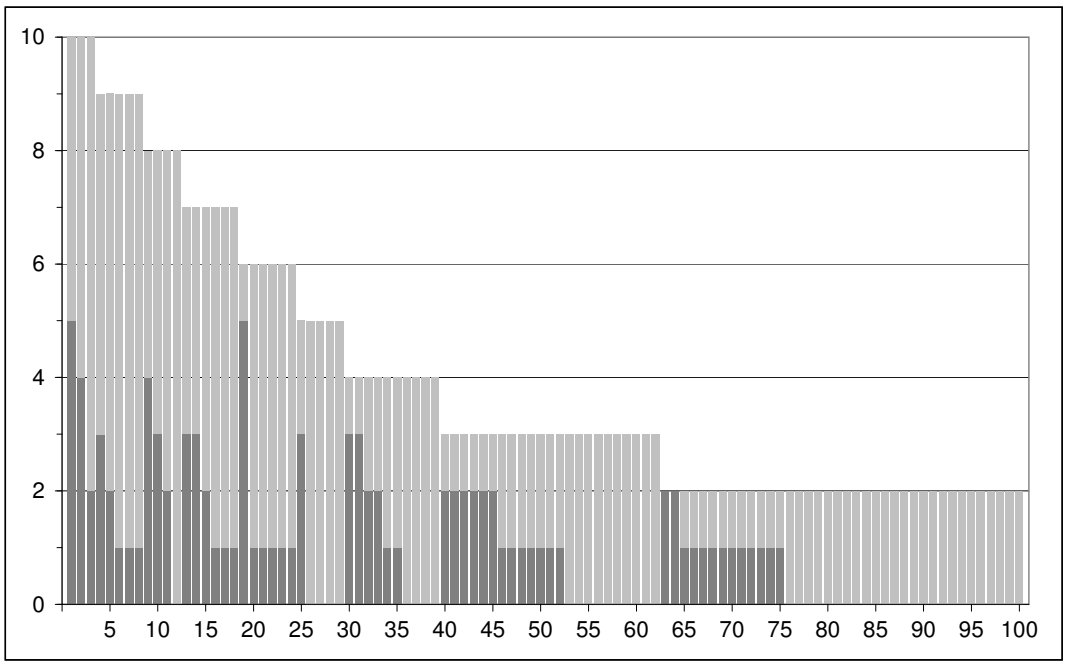


Abbildung 5.17: Google-Desktopsuche: von einhundert Substantiven, die höchstens zehnmal im Korpus vorkommen (hintere Balkenreihe) wird weniger als ein Viertel von der Google-Desktopsuche gefunden (vordere Balkenreihe).

Bei der Suche nach seltenen Substantiven wurden nur recht wenige Ergebnisse überhaupt gefunden. In 44 von einhundert Fällen lautete die Ausgabe „Ihre Suche - ... - hat zu keinen Ergebnissen geführt.“ Von den im Korpus vorhandenen Termen wurden weniger als ein Viertel gefunden. Abbildung 5.17 stellt, nach Häufigkeit geordnet, die Ergebnisse des Versuchs dar. Die Säulen sind als hintereinander liegend zu betrachten. Die dunkelgraue Säule zeigt die Anzahl der von Google Desktop Search gefundenen Terme an, beide Säulen zusammen die im Korpus vorhandenen Terme. Anders als bei der Web-Suche wurden „ähnliche“ Terme in keinem Fall vorgeschlagen.

5.5.3 Copernic Search

Copernic Search[9] benötigte für die erste Indexierung nur der Dateien des Projekts Gutenberg etwa vierundzwanzig Stunden. Danach erwies sich die Suchfunktion als schnell und restringiert. Erweiterungen des Suchterms wurden nur in fünf von hundert Fällen vorgenommen. So wurden zum Term »Handelsreise« alle Vorkommen gefunden, in denen der Term Wortanfang ist (Handelsreise, ~n, ~nde, ~nden, ~nder, und auch Handelsreiseleben). Während der Indexierung wird Groß- und Kleinschreibung nicht unterschieden. Diakritische Zeichen werden offenbar weggelassen und ›ss‹ und ›ß‹ zu einem Zeichen zusammengefasst. So werden zum Term »Getummel» mit vier Vorkommen auch alle Formen gefunden, in denen »getümmel» oder »Getümmel» am Wortanfang stehen. In zwölf von hundert Fällen wurde ein Vorschlag für einen veränderten Suchterm gemacht: Zu »Fihrer« wurde »fuhrer« vorgeschlagen, zu »Granittafel« »granatapfel« und zu »Hasenmüller« wurde »havelmuller« vorgeschlagen. Dabei gab es zum Suchterm »Fußpfund« als Vorschlag »fusspfund«, das jedoch nicht im Korpus vorkommt und als »Fußpfund« gefunden wurde. In den meisten Fällen wurden Endungen, die durch Deklination zum Stamm gefügt wurden, jedoch nicht erkannt und keine Vorschläge zur Suche der Stammform gemacht.

5.5.4 Content Surveyor

Content Surveyor 1.0 [10] stürzte zuverlässig ab, wenn das gesamte Korpus indiziert werden sollte. Für einen kleinen Ausschnitt mit weniger als eintausend Dateien war eine Indizierung möglich. Die Suche nach den Term »Gebet« führte allerdings zu einer Liste von 714 Dokumenten, von denen die ersten fünfzig angezeigt wurden. In diesen fünfzig Dateien waren Terme wie »geht«, »gebot«, »ging« oder auch »Lebensumstände« als Funde hervorgehoben, so dass nicht klar ist, nach welchen Kriterien das nicht angegebene Viertel der indexierten Dateien ausgesondert wurde.

Die Indexdateien benötigten 39 MB Speicherplatz. Da im Korpus 66 190 Dateien (vgl. Abschnitt 3.2.1) indexiert werden sollten, wurde für Vergleichszwecke die Zahl 2,58 Gigabytes, die in Abbildung 5.15 genannt wurde, aus diesen Daten extrapoliert.

[9] Copernic Search (Version 3.2.1 (Build 8) vom 6.8.2009)http://www.copernic.com

[10] Neuropower AG, http://neuropower.com/products_n_solutions/business-center/content_surveyor/

6 Diskussion

6.1 HTML und Unicode

Indexieren bedeutet, in einen Index aufnehmen. Der Indexeintrag soll wiederum auf einen bestimmten Ort verweisen. Zeichensätze, die für verschiedene Zeichen eine unterschiedliche Anzahl von Bytes benötigen, führen dazu, dass in einer Zeichenkette die Abbildungen *Zeichen* → *Byte* und *Byte* → *Zeichen* nicht mehr linear sind. Solche Zeichensätze sind HTML und Unicode.[1]

Die Dateien des Korpus selbst sind HTML-codiert. Die Tabelle 7.1 im Anhang listet alle HTML-*entities* auf, die im Korpus vorkommen. Unter *entity*[2] sind alle diejenigen Zeichenketten in den Originaldateien zu verstehen, die Sonderzeichen beschreiben. Sie wurden mit dem regulären Ausdruck »`&.{2,10}?;`« aus dem *body* der Originaldateien ausgefiltert. Diese *entities* gibt es in drei verschiedenen Formaten: Gemeinsam ist ihnen der äußere Rahmen, bestehend aus einem einleitenden Et-Zeichen (*engl. ampersand*, `&`), und einem schließenden Semikolon ›`;`‹. Dazwischen können beliebige andere Ziffern und Buchstaben aus dem ASCII-Zeichensatz stehen. Dafür steht der Punkt im regulären Ausdruck. Für die Filterung wurde die Anzahl auf den Bereich von zwei bis zehn Zeichen eingegrenzt »`{2,10}`«. Das folgende Fragezeichen setzt den *lazy*-Modus, d. h. erkannt wird die kürzeste Zeichenfolge, die die Bedingungen erfüllt.

Am bekanntesten und entsprechend am häufigsten vorkommend sind die *named entities*, bei denen der Zwischentext eine mnemonische Zeichenkette ist wie `szlig` für ›ß‹. Außerdem kann der Zwischentext einen Unicode-Codepoint darstellen. Dafür gibt es die Varianten der Darstellung als Dezimalzahl mit einem vorangestellten Doppelkreuz ›`#`‹ oder der Darstellung als Hexadezimalzahl mit einem vorangestellten ›`#x`‹. Während bei den *named entities* Groß- und Kleinschreibung unterschieden wird, wird diese bei den Buchstaben A bis F der Hexadezimalzahlen nicht unterschieden. Dafür ist die Anzahl der Ziffern innerhalb des Unicode-Rahmens von bis zu 5 Stellen beliebig, insbesondere können Nullen den signifikanten Stellen

[1] Eine absolute Kurzfassung des Konzepts Unicode findet sich in http://www.joelonsoftware.com/articles/Unicode.html

[2] http://www.w3.org/TR/html4/sgml/entities.html

vorangestellt werden oder auch nicht. Surrogat-Paare werden als ein Zeichen dargestellt, auch wenn die vorhandene Zeichentabelle dafür nur das Ersatzzeichen `[]` für „nicht vorhandenes Zeichen“ zur Verfügung stellt.

HTML weist die bereits beschriebene Mehrdeutigkeit der Codierung von Sonderzeichen auf. ›ä‹ kann dargestellt werden als »`&auml;`« oder »`&#228;`« oder »`&#xE4;`« oder »`&#x00E4;`« oder »`&#x00e4;`« usw.

Im Konzept von Unicode steht ein Codepoint für jedes darstellbare Zeichen, wobei Glyphen, die dargestellten Zeichen, auch durch die Überlagerung mehrerer Codepoints dargestellt werden können. Die Glyphe ›ä‹ kann also nicht nur dargestellt werden durch den Codepoint U+00E4 ›ä‹ sondern auch durch die Kombination U+0061 ›a‹, gefolgt von U+00A8 ›¨‹[3].

Darüber hinaus verwendet Unicode Bytefolgen unterschiedlicher Längen zur Darstellung eines Codepoints. Bei UTF-8 sind dies zwischen einem und sechs Bytes, bei UTF-16 zwei oder vier Bytes. Zeichen, die in UTF-16 vier Bytes zur Darstellung benötigen, werden als *supplementary characters* bezeichnet. Mit sechzehn Bit kann man 2^{16} = 65 536 Werte darstellen. Die Codepoints U+FFFE und U+FEFF werden als *Byte Order Mark, BOM* bezeichnet. Sie stehen am Anfang einer UTF-16 codierten Datei und dienen dazu, dem Parser mitzuteilen, ob die Datei *Little-Endian, LE* (auch Intel- oder Windows-Format, U+FFFE) oder *Big-Endian, BE* (auch Motorola-Format, U+FEFF) -codiert ist. [Con07]

Dieser Adressraum für Codepoints wird *Basic Multilingual Plane (BMP)* genannt. 16-bit Codepoints aus dem Bereich U+D800 bis U+DFFF werden *Surrogates* genannt. Werte aus dem Bereich U+D800 bis U+DBFF heißen *High-Surrogate Code Units*, Werte aus dem Bereich U+DC00 bis U+DFFF heißen *Low-Surrogate Code Units*.[4]

Insgesamt umfasst dieser Adressraum 1 114 112 Werte. Allerdings kommen *Surrogates* mit vier Bytes selbst in fernöstlichen Texten mit 1 % Vorkommen eher selten vor.[5]

Im untersuchten Korpus kamen *Surrogates* nicht vor. Die Normalisierung von Unicode-Text ist komplex[DH08], die Anforderungen an die Realisierung von Algorithmen zur Verarbeitung regulärer Ausdrücke sind hoch [DD08]. Eine eineindeutige Abbildung *Zeichen* ↔ *Byte* erleichtert die Indexierung und beschleunigt die Suche in Zeichenketten. Deshalb wurde an Stelle von UTF-16 die Untermenge UCS-2 (*Universal Character Set 2-Bytes*) verwendet. Diese Codierung ist identisch mit der *Basic Multilingual Plane* und entspricht auch der internen Zeichendarstellung

[3] in *pdf*-Dokumenten oder in TEX muss diese Reihenfolge umgekehrt werden: \¨a

[4] Definitionen D71 bis D74 in http://www.unicode.org/versions/Unicode5.0.0/ch03.pdf#79

[5] Zahlenangaben zitiert nach Mark Davis, Präsident des Unicode Consortiums http://www.unicode.org/faq/utf_bom.html#utf16-5

im .NET-Framework.[6] Hierbei werden nur die Codepoints verwendet, die sich mit zwei Bytes darstellen lassen.

Auch innerhalb dieses Bereichs gibt es zahlreiche Codepoints, deren Glyphen sich auf verschiedene Weise darstellen lassen. So werden *singletons* (»einelementige Mengen«) beschrieben, die bei einer Normalisierung aus dem Text entfernt werden. Als Beispiele für solche *singletons* werden die Zeichen ›Å‹ für die Längeneinheit Ångström (= 10^{-10} m) und ›Ω‹ für die Einheit Ohm des elektrischen Widerstands als Codepoints U+212B und U+2126 definiert. Nach einer Normalisierung werden sie in der *Normalization Form C*[7] NFC mit den gleichen Glyphen als U+00C5 (*A-with-ring*) und U+03A9 (*Omega*) dargestellt. In der *Normalization Form D*[8] (NFD), die keine zusammengesetzten Zeichen mehr kennt, wird das Zeichen Å in seinen Komponenten ›A‹ (U+0041) und folgendem ›˚‹ (U+030A) dargestellt.

In der untersuchten Implementierung wurden zunächst die *HTML-entities* entsprechend einer Tabelle in Unicode umgesetzt. Anschließend wurden akzentuierte Zeichen innerhalb der Codierung mittels einer Verweistabelle in Zeichen aus dem Zeichensatz der verwendeten Codierung umsetzen. Hier erscheint eine andere Datenstruktur vorteilhafter, die ohne Umwege eine bidirektionale Umsetzung ermöglicht. Dies ist möglich, sobald die verwendeten Codierungen und ihre Zeichensätze festliegen, was in dieser experimentellen Implementierung noch nicht der Fall war.

Für den binären Vergleich von Unicode-Zeichenketten ist eine vorherige Normalisierung erforderlich.[9] Die Normalisierung ermöglicht lediglich die binäre Aussage gleich oder nicht gleich. Ist darüber hinaus eine Sortierung erforderlich, z. B. bei der alphabetischen Präsentation von Ergebnissen, dann gibt der *Unicode Collation Algorithm* [DW09] eine Handreichung zur Umsetzung. Diese Unterscheidung ist wichtig, weil die Sortierfolge (*collation*) länderspezifisch ist und auch die Behandlung von Ziffern in Zeichenfolgen besondere Überlegungen erfordern. Deswegen ist die Sortierfolge im Gegensatz zur Normalisierung individuell an die gewünschte Umgebung anpassbar.

6.1.1 Bedingter Trennstrich

Eine besondere Behandlung benötigt das Unicodezeichen U+00AD (SOFT HYPHEN,). Das Zeichen bezeichnet einen bedingten Trennstrich, der nur falls erforderlich am Zeilenende sichtbar wird. Das Verständnis um die Semantik des

[6] http://msdn.microsoft.com/en-us/library/system.string.aspx
[7] *Canonical Decomposition, followed by Canonical Composition*
[8] *Canonical Decomposition*
[9] Section 3.11 Normalization Forms in http://www.unicode.org/versions/Unicode5.2.0/ch03.pdf

Zeichens hat sich im Laufe der letzten Jahre verändert.[10] Ausgangspunkt ist der Standard ESMA-94 [ECM85] aus den Jahr 1984. Hier wird zunächst definiert:

> 5.5 Graphic Character
> A character, other than a control function, that has a visual representation normally handwritten, printed or displayed, and that has a coded representation consisting of one or more bit combinations.

Später wird zum bedingten Trennstrich festgelegt:

> 6.3.3 SOFT HYPHEN (SHY)
> A graphic character that is imaged by a graphic symbol identical with, or similar to, that representing HYPHEN, and which may be inserted or removed by operations determing the layout of text.

Während das Leerzeichen `SPACE (SP)` sowohl als *graphic character*, als auch als *control character* oder als beides charakterisiert wird, gilt dies für das bedingte Trennzeichen zu der Zeit noch nicht. In der zweiten Ausgabe von 1986 [ECM86] liest sich die Definition wie folgt:

> 6.3.3 SOFT HYPHEN (SHY)
> A graphic character that is imaged by a graphical symbol identical with, or similar to, that representing HYPHEN, for use when a line break is permitted in the text as presented.

In der Adaptation durch die ISO [ISO98] liest sich der Anwendungsbereich so:

> ... that representing HYPHEN, for use when a line break has been established within a word.

Wegen der unterschiedlichen Interpretation des bedingten Trennstrichs in verschiedenen Browsern wurde in der Spezifikation von HTML-2.0 noch vom Gebrauch abgeraten [BLC95]:

> Use of the non-breaking space and soft hyphen indicator characters is discouraged because support for them is not widely deployed.

In der aktuellen Spezifikation HTML-4.01 wird nun geklärt [W3C99]:

> In HTML, there are two types of hyphens: the plain hyphen and the soft hyphen. The plain hyphen should be interpreted by a user agent as just another character. The soft hyphen tells the user agent where a line

[10] http://www.cs.tut.fi/~jkorpela/shy.html

> break can occur.
> Those browsers that interpret soft hyphens must observe the following semantics: If a line is broken at a soft hyphen, a hyphen character must be displayed at the end of the first line. If a line is not broken at a soft hyphen, the user agent must not display a hyphen character. For operations such as searching and sorting, the soft hyphen should always be ignored.
> In HTML, the plain hyphen is represented by the "-" character (- or -). The soft hyphen is represented by the character entity reference (or)

Die Spezifikation spricht deshalb auch von einem *invisible hyphenation hint*.

Dieser Regel folgend wurde diese Entität zur Indexierung deshalb ersatzlos entfernt. Problematisch wird das Zeichen bei der Darstellung der Ergebnisse: Da es im Originaltext enthalten ist, im indexierten Token jedoch nicht, wird eine Wortform, die bedingte Trennstriche enthält, bei einer Suche mit regulären Ausdrücken im Originaltext nicht mehr gefunden. Dieses Problem kann dadurch gelöst werden, dass der Originaltext vor der Darstellung im Browser dahingehend verändert wird, dass bedingte Trennstriche entfernt werden.

Allerdings kamen bedingte Trennstriche im gesamten Korpus nur in drei Dateien vor, dabei in zwei Dateien als falsche Codierung: Einmal werden sie an Stelle eines Gedankenstrichs ›–‹ verwendet:[11]

```
Am Abend begann im Speisesaal &#173; es war der Tag
des Kabeljaus mit brauner So&szlig;e &#173; der
Streik mit wahrhaft sch&ouml;ner Einhelligkeit
```

Im richtig interpretierenden Browser wird dieses Zeichen dann einfach ausgelassen. In der zweiten Datei wird versucht, ein Zeichen aus einem speziellen Font zu benutzen[12]:

```
<font face="Symbol">&shy;</font>und eben so viel<b> </b>
<font face="Symbol">&macr;</font>
```

Im Font „Symbol“ sind unter den Codepoints 173 und 175, bzw. AD_{16} und AF_{16} die Symbole ↑ (`&uarr;`, U+8593) und ↓ (`&darr;`, U+8595) zu finden. Im Internet Explorer 8 wird an dieser Stelle für `&shy;` ein Leerzeichen und für `&macr;`, eigentlich ein Überstrich ¯, tatsächlich der Pfeil nach unten gezeigt. In seiner vorgesehenen Bedeutung kommt es deshalb nur in einer Fußnote vor[13]:

[11] zola/novellen/michu.htm
[12] fechner/epsypht1/epsyph09/epsyph09.htm
[13] riesbeck/briefe/r_430.htm

```
allerernsthafteste Vor&shy;&uuml;bung und Technik
```

6.2 Indexerstellung

Die optimale Art, wie ein invertierter Index erstellt wird, hängt wesentlich von der Aktualisierungsfrequenz des zugrunde liegenden Korpus ab. In unseren Fall war das Korpus festgelegt. Der Index musste deshalb nur einmal erstellt werden. Häufig soll auch weiter hinzukommendes Material in den Index aufgenommen werden. Ist die hinzukommende Menge klein in Bezug auf das Ausgangskorpus, dann ist es sinnvoll, einen weiteren Index für die Aktualisierungen parallel zum Hauptindex zu pflegen. Neue Texte werden in einem aktuellen Korpus gesammelt und dieses Korpus nach einer Aktualisierung komplett neu indexiert.

Für häufige und umfangreichere Aktualisierungen schlagen [TGMS93] ein Verfahren mit kurzen und langen Listen vor. Lange Listen enthalten die etwa 5 Prozent häufigen Wörter, die knapp 95 Prozent der insgesamt vorkommenden Wörter enthalten. Diese Zahlen werden durch das vorliegende Korpus bestätigt (siehe Abbildung 3.2 auf Seite 51). Hier wird für jedes Wort eine Liste geführt. Hinzukommende Textstellen müssen dann nur angehängt werden. Das hier vorliegende Korpus enthält 1 410 643 Wörter im Lexikon. 5 % davon sind immer noch über 70 000 Wörter, so dass auch bei diesem Verfahren die Datenstruktur nicht einfach auf der Basis des Dateisystems organisiert werden kann. Die kurzen Listen sind als Körbe organisiert, die viele alphabetisch sortierte Wörter enthalten. Wenn ein seltenes Wort in den Index aufgenommen werden soll, wird die kurze Liste in den Arbeitsspeicher geladen, neu organisiert und wieder gespeichert.

6.3 Erstellen von Suchanfragen

Werden Texte durch die Eingabe eines Suchwortes gesucht, dann entstammt dieses Suchwort zunächst dem aktiven Wortschatz des Suchenden. Im weiteren Verlauf der Suche können weitere Wörter für die Suche verwendet werden, die zuvor nicht zum aktiven Wortschatz gehörten. Der aktive Wortschatz eines Durchschnittssprechers wird auf 10 000 Lexeme geschätzt [Bußß02]. Für Goethe wurde aktuell ein aktiver Wortschatz von 90 000 Lexemen ermittelt [Wis98, Benutzerhinweise Band III], wobei diese Zahl aufgrund neuer Quellenfunde in den vergangenen Jahren vervierfacht wurde.

Neben die Erweiterung der Menge der Suchwörter durch Flexion tritt die unscharfe Suche, die auch Schreibfehler erfassen lässt. Teilwortsuche und Synonymsuche erweitern die Menge der Suchwörter nochmals erheblich.

Die automatische Erzeugung von Suchtermen (*query terms*) wird in nur wenigen Arbeiten behandelt. Die vielzitierte Arbeit von [Gra93] beschäftigt sich mit dem Problem, wie Suchanfragen für große Datenbanken optimiert werden können. TREC vergleicht mit dem Query track in TREC-9 [Buc00], welche Art der Suchanfrage zu einem gegebenen Thema mit den teilnehmenden Systemen die besten Lösungen liefert. Die Suchanfragen wurden intellektuell aus den vorhandenen Dokumenten erzeugt. Zur Verfeinerung und Verbesserung des Rankings werden mit Methoden des interaktiven Relevance Feedbacks und des automatischen Pseudo Relevance Feedbacks [XC96, ACD+03, XJW09, CGR08] weitere Suchterme zur Suchanfrage zugefügt und die Trefferliste mit verschiedenen, manuell getunten Parametern in ihrer Reihenfolge sortiert (siehe Abschnitt 2.5 ab Seite 47).

[GJM01] beschreiben ihr Verfahren, aus wenigen vorgegebenen Dokumenten Suchterme für weitere Anfragen zu extrahieren, mit denen weitere Dokumente im Web gesucht werden, um auf diese Weise automatisch ein Korpus für Sprachen mit geringer Verbreitung zu erstellen. [San08] artikuliert das Problem mehrdeutiger Suchanfragen.

Mihov und Schulz [MS04] bevorzugen den von ihnen entwickelten schnellen Algorithmus zur Berechnung der Levenshtein-Distanz (siehe Abschnitt 4.4.1 auf Seite 74), um ähnliche Wörter in umfangreichen Wörterbüchern zu finden. Das Vorgehen dort war allerdings völlig anders: Aus den vorhandenen Wörtern wurden zufällig fehlerhafte neue erzeugt, die wiederum gegen die vorhandenen getestet wurden. Dies erlaubt zwar ein vollautomatisches Verfahren, entspricht aber wohl kaum den Nutzererwartungen, insbesondere im Hinblick auf die Teilwortsuche. Außerdem ist die Levensthein-Distanz kein besonders geeignetes Maß, um Ähnlichkeit zwischen Termen zu definieren, wie in Tabelle 7.8 auf Seite 177 abzulesen ist.

In dieser Arbeit wurden die Suchanfragen aus dem vorhandenen Vollformenlexikon generiert. Nach einstellbaren Parametern wurden die Suchterme mit einem Zufallsgenerator ausgewählt. Als Parameter konnten eingestellt werden: die Häufigkeit im Korpus, die Länge der Zeichenkette, der Beginn der Zeichenkette mit einem Großbuchstaben, ein alphabetischer Auszug und die Anzahl der zurückzugebenden Suchwörter in der erzeugten Liste. Außerdem konnte bestimmt werden, ob die Auswahl die Häufigkeit des Terms im Korpus berücksichtigen sollte.

Für eine intellektuelle Kontrolle sind überschaubare Datenmengen notwendig. Werden bei einer Suche sehr viele Texte gefunden, dann muss eine Auswahl getroffen werden. Um die Zahl der Fundstellen in einem überschaubaren Rahmen zu halten, wurden deshalb aus den im Korpus vorhandenen Termen solche Terme ausgewählt, die zwischen zwei- und zehnmal vorkamen. Von den über 500 000 Termen (vgl. Abbildung 3.4), die dieser Bedingung entsprachen, begannen 308 015

mit einer Majuskel, davon wurden, beginnend mit dem 440sten Term jeder 880ste Term ausgesucht, so dass 350 Terme extrahiert wurden. Unter diesen Termen waren 25, die keine Substantive waren und deshalb ausgesondert wurden. In den automatisierten Verfahren wurden alle verbliebenen 325 Terme ausgewertet, für die händische Eingabe in die anderen Suchmaschinen wurden aus dieser Liste die ersten einhundert Terme verwendet.

6.4 Auswahl der Indexterme

Während vor zehn Jahren noch die in Abschnitt 2.1 zitierte Aussage von [Man00], dass für die Indexierung einer großen Textmenge eine Auswahl der Indexterme erforderlich sei, Gültigkeit beanspruchen konnte, so läßt sich diese Aussage heute nicht mehr bestätigen. Mit den vorgestellten Algorithmen kann ein Vollformenwörterbuch in Sekundenbruchteilen auch mit einer unscharfen Suche durchforscht werden.

Der Fokus dieser Untersuchung liegt in der Analyse selten vorkommender Terme. Überraschenderweise kamen selten vorkommende Wörter in den Dateien des Projekts Gutenberg-DE in jeder Datei meist nur in einer laufenden Wortform vor, während andere Wortformen und ähnliche Terme in anderen Dateien erschienen (Abbildung 5.1 auf Seite 101). Um eine Übersicht über die verschiedenen Verwendungen eines Wortes zu erhalten, genügt es daher nicht, nur eine Wortform zu suchen, sondern es sollten möglichst alle Wortformen erfasst werden. Alle Wortformen zu erfassen ist wiederum eine recht komplexe Aufgabe. Verschiedene Verfahren liefern unterschiedlich gute Resultate, wobei Verfahren zur unscharfen Suche unverzichtbar erscheinen und für deutsche Texte generell besser als Stemming-Verfahren abschnitten, wie das schlechte Abschneiden der Codierungen Optimiert10 und Optimiert20 zeigt (Tabelle 7.8 auf Seite 177).

6.5 Bewertung des Retrievalergebnisses

Für die Bewertung eines Retrievalergebnisses stehen eine Vielzahl von Maßen zur Verfügung. Das Programm trec_eval[14] berechnet 135 verschiedene Maße, von denen in einer Untersuchung von [BDKM08] die 27 in der Literatur gebräuchlichsten miteinander verglichen wurden. Nach dieser Untersuchung genügen sieben verschiedene Maße, um die untersuchten 27 Maße zu repräsentieren.

[14] http://www-nlpir.nist.gov/projects/trecvid/trecvid.tools/trec_eval_video/README

Die Bewertung der Qualität eines Retrievalverfahrens ist nicht trivial. Es gibt eine Vielzahl von Bewertungskriterien und -verfahren, die auch von den Zielvorstellungen des Nutzers abhängen. Hawking und Craswell [HC05] sprechen einige der Probleme der Beurteilung an. Thom und Scholer [TS07] haben eine Auswahl von fünf Bewertungsfunktionen auf ihre Korrelation untereinander untersucht. Untersucht wurden die

precision at 1	die über alle Suchläufe gemittelte *precision* des ersten gefundenen Dokuments,
precision at 10	die über alle Suchläufe gemittelte *precision* der ersten zehn gefundenen Dokumente,
mean average precision	durchschnittliche, über alle Suchläufe gemittelte *precision*, wobei die gemittelte *precision* eines Suchlaufs aus der *precision* aller im Suchlauf gefundenen relevanten Dokumente gebildet wird,
mean reciprocal rank	Mittelwert über alle Suchläufe des Kehrwertes des Rangs des ersten relevanten Dokuments
	und die
R-precision	die über alle Suchläufe gemittelte *precision*, nachdem R_t Dokumente zum Thema t gefunden wurden, wobei R_t die Zahl aller zum Thema t verfügbaren Dokumente bezeichnet.

Die Verfahren wurden verglichen mit Kendalls τ Korrelationskoeffizienten für ordinal skalierte Daten [Ken38]. Die Verfahren konnten in zwei Gruppen eingeteilt werden, nämlich solche, die insbesondere ein einziges relevantes Dokument in der Spitzengruppe favorisierten und in solche, die eine breiter angelegte Fundmenge favorisierten. Entsprechend war die Korrelation zwischen dem Maßen *precision at* 1 und *mean reciprocal rank* als Vertreter der ersten Gruppe sehr hoch und andererseits die Korrelation zwischen *mean average precision* und *R-precision*.

Dabei stellten Thom und Scholer fest, dass das zur Zeit vorwiegend verwendete Verfahren *mean average precision* nicht unbedingt den Wünschen der Nutzer entspricht, was von [TS06] und [AMSC07] bestätigt wird.

6.5.1 Bewertung einer Einzelabfrage

Zunächst sollen die gebräuchlichen Maße und Wahrscheinlichkeiten definiert werden.

richtig positiv (TP)	falsch positiv (FP)	**Gefunden** $P = TP + FP$
falsch negativ (FN)	richtig negativ (TN)	**Nicht gefunden** $FN+TN$
Relevant $R = TP + FN$	**Nicht relevant** $N = FP + TN$	

Betrachtet man zunächst den Ausfall einer Abfrage: Aus der Gesamtheit der Dokumente ($TP + FP + FN + TN$) werden durch die Abfrage zurückgegeben ($TP + FP$). Nicht zurückgegeben werden ($FN + TN$). Gesucht werden jedoch die relevanten Dokumente ($TP + FN$).

Die Qualität des Retrievalsystems ist offenbar umso besser, je geringer der Anteil ($FP + FN$) ist und umso höher der Anteil ($TP + TN$) ist. Das Problem besteht einerseits in der Bewertung, welche Dokumente für eine gegebene Anfrage als relevant gelten sollen, andererseits darin, wie die Anzahl der falsch negativen Ergebnisse bestimmt werden soll, wenn die Dokumentenmenge wie in unserem Fall mit 66 190 Dokumenten sehr groß ist.

Zur Beurteilung werden häufig die Maße *recall* (Trefferquote) und *precision* (Genauigkeit) benutzt. Mit $R = TP + FN$ und $P = TP + FP$ ist *recall* als

$$recall = \frac{|R \cap P|}{|R|} \tag{6.1}$$

definiert. Dazu muss jedoch die Anzahl der relevanten Dokumente bekannt sein oder aber geschätzt werden. Mit *precision* wird das Verhältnis der gefundenen relevanten Dokumente zur Gesamtzahl der gefundenen Dokumente bezeichnet. Die *precision* bestimmt sich wegen $TP = |R \cap P|$ zu

$$precision = \frac{|R \cap P|}{|P|}. \tag{6.2}$$

Die Zahlenwerte beider Maße bewegen sich definitionsgemäß zwischen 0 und 1. Eine Anfrage, die alle vorhandenen Dokumente liefert, hat ein *recall* von 1, wohingegen eine Anfrage, die genau ein relevantes Dokument liefert, stets eine *precision* von 1 hat. Um die Unzulänglichkeiten dieser Maße zu verbessern, wurde deshalb als Bewertungskriterium das harmonische Mittel beider Zahlen vorgeschlagen. Das *F-score* ist so definiert:

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{6.3}$$

Als Verfeinerung lassen sich *precision* und *recall* unterschiedlich gewichten:

$$F_\alpha = (1+\alpha) \cdot \frac{precision \cdot recall}{\alpha \cdot precision + recall}. \tag{6.4}$$

Im Kapitel 5 wird die Korrektklassifikationsrate P (*accuracy)* als Maß verwendet:

$$P_{\text{richtig klassifiziert}} = \frac{TP + TN}{TP + FP + TN + FN}. \tag{6.5}$$

All diesen Maßen gemeinsam ist die vorher verlangte binäre Entscheidung, ob ein Dokument relevant ist (T) oder nicht (N). Tatsächlich sind Dokumente jedoch eher mehr oder eher weniger relevant. Deswegen wird häufig ein Ranking vorgenommen, um die gefundenen Dokumente in eine Reihenfolge von für den Nutzer vermutlich wichtiger zu für den Nutzer vermutlich unwichtiger zu bringen.

6.5.2 Bewertung eines Verfahrens

Eine einzelne Suche hat wenig Aussagekraft, wenn man Verfahren vergleichen will. Um die unterschiedliche Qualität verschiedener Suchalgorithmen beurteilen zu können, kann man den p-Wert heranziehen: Ein Zufallsexperiment wird durchgeführt, z.B. werden mit den verschiedenen zu vergleichenden Verfahren je einhundert zufällig bestimmte Begriffe gesucht. Jede Suche ergibt nach intellektueller Bewertung eine Korrektklassifikationsrate. Die Nullhypothese des statistischen Tests lautet: Verfahren A und Verfahren B haben die gleiche Korrektklassifikationsrate. Die Nullhypothese wird verworfen, wenn die Verfahren eine unterschiedliche Korrektklassifikationsrate aufweisen (zweiseitiger Test).[15]

Als Voraussetzung für diesen Test wird angegeben, dass der Ausfall der durchgeführten Experimente einer bestimmten Verteilung, nämlich einer Gauß-Verteilung folgen müssen. Diese Voraussetzung ist hier nicht gegeben. Alternative Testverfahren findet man unter der Bezeichnung parameterfreie Hypothesentests, z. B. den Wilcoxon-Mann-Whitney-Test.[16]

Bei der Bestimmung der Korrektklassifikationsrate wird als Grundgesamtheit die Menge aller zurückgegebenen Ergebnisse verwendet. Dies bedeutet, dass die Ergebnisse abhängig sind von jeder einzelnen Codierung in der Liste. Werden Codierungen mit schlechter Korrektklassifikationsrate entfernt, dann kann die Anzahl der falsch positiven Ergebnisse erheblich sinken. Damit ändert sich das

[15] http://qed.econ.queensu.ca/walras/custom/300/351B/notes/glo_07.htm#pvalue

[16] Eine Website, in der man direkt die Rohdaten zur Berechnung des Tests eingeben kann, ist unter http://faculty.vassar.edu/lowry/utest.html zu finden.

Ergebnis auch der Codierungen mit der besseren Korrektklassifikationsrate hin zu einer schlechteren Korrektklassifikationsrate.

Ob eine unscharfe Suche vorteilhaft oder nicht ist, hängt wesentlich von der Wortlänge des Suchterms ab. Aufgrund der Tatsache, dass sinnvolle Terme mit drei Buchstaben in der Regel Stoppwörter sind, d.h. sehr häufig vorkommen (vgl. Abbildung 5.6), führt eine Veränderung eines Buchstabens zu einer erheblich vergrößerten Treffermenge unerwünschter Ergebnisse, also einer Verschlechterung der Qualität des Suchergebnisses. Im Sinne einer besseren Präzision aufgrund einer geringeren Treffermenge ist die Codierung Trigramm10 für die kurzen Wörter mit bis zu fünf Zeichen optimal (Abschnitt 5.3.1). Von den untersuchten Codierungen sind bei diesen Wortlängen die konsonantischen Codierungen unbrauchbar. Dieses Verhältnis kehrt sich für Terme mit mehr als vierzehn Zeichen um: Die Codierung Trigramm10 liefert in vielen Fällen sehr viele Ergebnisse, andere Codierungen, die zwanzig oder mehr Zeichen berücksichtigen, liefern wenige (Abbildung 5.10 S. 115), dafür aber richtige Ergebnisterme (Abbildung 5.11 S. 115).

Unabhängig von der Korrektklassifikationsrate im Allgemeinen interessiert den Nutzer bei einer speziellen Suchanfrage auch nur die Korrektklassifikationsrate und die Anzahl der zurückgegebenen Ergebnisterme für diese spezielle Suchanfrage. Im Abschnitt 5.4 wurde dieser Frage nachgegangen. Wie Abbildung 5.14 auf Seite 121 dokumentiert, sinkt die Korrektklassifikationsrate, wenn mehr Terme zurückgegeben werden. Nach dieser Untersuchung liegt ein optimaler Kompromiss zwischen fünf und zehn Ergebnistermen.

Ein intelligenter Suchalgorithmus prüft deshalb die Menge der Ergebnisterme und wählt daraufhin zeitgleich oder anschließend ein anderes Suchverfahren, wenn die Ergebnismenge nicht den Erwartungen entspricht.

6.6 Andere Suchmaschinen

Zu dem in Abbildung 5.15 auf Seite 121 dokumentierten Speicherbedarf für die Indexdateien sind an dieser Stelle einige Anmerkungen zu machen.

Der für die eigene Suchmaschine ermittelte Speicherbedarf umfasst sämtliche Indexdateien, d.h. er enthält sowohl die Sammeldatei, die das gesamte Korpus enthält, wie auch ein Suffixarray für die Teilwortsuche, wie auch Dateien für alle beschriebenen Codierungen zur unscharfen Suche. Die Sammeldatei mit 1,3 GB ist im Format UCS-2 codiert. Das bedeutet, fast jedes zweite Byte hat den Wert 0. Diese Datei lässt sich gut komprimieren. Die Komprimierung mit dem Programm WinZip®[17] führte zu einer Dateigrösse von 322 MB, also einer Kompression auf

[17] WinZip® 8.1 SR-1 (5266) compiled: Feb 11 2003 http://www.winzip.com/index.htm

weniger als 25 %. Wurden alle für den kaskadierten Index erforderlichen Dateien mit allen vorteilhaften Codierungen komprimiert, ergab sich ein Platzbedarf von 845 MB entsprechend weniger als 40 % der ursprünglichen Größe. Das Zip-Format ist frei verfügbar[18,19] und auch auf *Pipes* und *Streams* anwendbar. Nach einem nicht überprüften Bericht kann die Verarbeitung wegen der geringeren Anzahl von I/O-Operationen trotz höherer Prozessorlast durch die Komprimierung und Dekomprimierung beschleunigt sein.

Außer den verglichenen gibt es natürlich noch zahlreiche andere Suchmaschinen, die eine unscharfe Suche versprechen. Als Open Source Projekte wären an dieser Stelle insbesondere *Lemur*[20] und *Swish-e*[21] zu nennen. Diese Projekte bieten jedoch keine fertige Benutzerschnittstelle, sondern sind eher als Werkzeugkästen zur Integration in andere Projekte konzipiert, die eine passende Benutzerschnittstelle bereitstellen. Außerdem bieten sie in ihrer vorliegenden Form (bislang) keine wirkliche unscharfe Suche. Die zu Lemur gehörende Suchmaschine *INDRI*[22] bietet nur eine suffixbasierte *wildcard*-Suche, Swish-e nennt *wildcard*-Suche, *Stemming*, *Soundex* und *Metaphone* als Algorithmen für die unscharfe Suche.

6.6.1 Windows

Die Desktopsuche von Windows erwies sich als besonders langsam. In vierzig Prozent der Fälle lieferte die Suche nach einem seltenen Term mehr als einhundert und bis zu 5 000 Dateien zurück. Die Antwortzeiten korrelierten mit der Anzahl der „gefundenen“ Terme und erreichten mehrere Sekunden. Dabei war weder ersichtlich, warum Terme als Funde betrachtet wurden, noch, welche Terme dies sein sollten. Insbesondere erscheint bei Komposita der erste Bestandteil als Grundlage der Suche verwendet zu werden, obwohl im Deutschen der Kern des Kompositums in der Regel rechts sitzt. Außerdem wurden Teilwörter auf eine verblüffende Weise erweitert, die den Nutzererwartungen nicht gerecht wird. Wie in Tabelle 5.9 notiert, konnte eine Verlängerung des (Teilwort-)Suchterms zu einer eklatanten Vergrößerung der Treffermenge führen.

6.6.2 Google

In der Selbstdarstellung von Google® heißt es:

[18] http://www.info-zip.org/
[19] http://zlib.net/
[20] http://www.lemurproject.org/
[21] http://swish-e.org/
[22] http://www.lemurproject.org/indri/

> Google ist das einzige Unternehmen, das sich darauf konzentriert, die „perfekte Suchmaschine“ zu entwickeln. Dazu Larry Page, Mitbegründer von Google: „Die perfekte Suchmaschine würde genau erkennen, was der Nutzer meint, und genau die Ergebnisse ausgeben, die er sich wünscht“. ... Google kombiniert die Gesamtwichtigkeit mit der Relevanz für eine spezifische Suchanfrage und ist deshalb in der Lage, die relevantesten und verlässlichsten Ergebnisse zuerst aufzuführen. ... Dabei wird eine Gleichung mit über 500 Millionen Variablen und zwei Milliarden Ausdrücken berechnet.[23]

Die Millionen Variablen und Milliarden Ausdrücke ergeben sich auch durch manuelle Eingriffe zur Optimierung des Rankings. Die Ergebnislisten werden durch bezahlte Kräfte daraufhin durchgesehen, ob sie wohl in ihrem Ranking der Intention des Nutzers entsprechen. Dazu äußerte sich der Leiter des *evaluation teams* von Google, Scott Huffman, wie folgt:

> When I type “movies” into Google, I expect Google to know where I am and bring back the times of the movies that are playing, not bring me just Web links that happen to match the word movies. When I type “pizza in san francisco”, I expect Google to bring me back a map and some links to good pizza places with reviews underneath.[24]

Mit angegebenen 6 000 Experimenten pro Jahr und einer Anzahl von Teilzeitevaluatoren, die im vierstelligen Bereich liegt, spielt das *User Feedback* für die Weiterentwicklung dieser Suchmaschine eine entscheidende Rolle. Leider werden diese Ergebnisse als Betriebsgeheimnisse betrachtet und entsprechend nicht veröffentlicht.

Die in der Desktop-Suche erzielten Ergebnisse lassen vermuten, dass dieses kostenlose Programm der Firma Google Inc. nicht im Focus ihrer Aufmerksamkeit steht. Von den einhundert gesuchten Termen wurden nur 56 überhaupt gefunden. Auch die Zahl der Fundstellen lag mit nur einem Viertel der tatsächlichen Vorkommen im Korpus weit unter den Nutzererwartungen.

Ähnliche Terme oder erweiterte Terme im Sinne der Teilwortsuche wurden nicht angeboten. Dabei benötigten die Datendateien für den Index mit 4,3 Gigabyte den weitaus meisten Platz. Gründe für dieses Versagen konnten nicht gefunden

[23] http://www.google.com/corporate/tech.html

[24] http://www.businessweek.com/the_thread/techbeat/archives/2009/10/googles_scott_h.html

werden. Nach der Hilfestellung zum Programm werden bei der ersten Indexierung je Laufwerk 100 000 Dateien indiziert.[25]

Da im Korpus 66 000 Dateien zu indexieren waren, sollte diese Bedingung erfüllt sein. Auch der Verdacht, dass nur die ersten n Bytes einer Datei indiziert wurden, konnte nicht bestätigt werden.

6.6.3 Copernic Search

Copernic Search[26] war die einzige ernst zu nehmende Alternative unter den untersuchten Suchmaschinen für die unscharfe Suche auf dem eigenen PC. Die Algorithmen scheinen sich jedoch im Wesentlichen darauf zu beschränken, statt des ganzen Wortes nach Wortanfängen zu suchen, Groß- und Kleinschreibung nicht zu beachten, diakritische Zeichen wegzulassen und ›ss‹ und ›ß‹ zusammenzufassen. Insbesondere wurden keine Versuche einer beliebigen Form des Stemmings gemacht. Sobald ein Suchterm eine deklinierte Form mit einer Endung ist, wird die Stammform nicht mehr gefunden. Umgekehrt werden deklinierten Formen aufgrund der Suche nach Wortanfängen gefunden. Der Nutzer hat jedoch keine Möglichkeiten, seine Suche durch Eingabe möglicher Parameter zu steuern.

6.6.4 Content Surveyor

Nachdem Content Surveyor 1.0[27] von der Größe des Korpus überfordert war, wurde auch für einen kleineren Ausschnitt aus den Dateien auf eine zufällige Anfrage hin Dokumente mit wahllosen oder beliebigen Termen gefunden, die keinen Bezug zum Suchterm hatten. Schließlich wurden drei Viertel der indexierten Dateien zurückgegeben, so dass nicht klar ist, nach welchen Kriterien das nicht angegebene Viertel der indexierten Dateien ausgesondert wurde.

[25] In der Erstindizierungsphase indiziert Google Desktop nur 100 000 Dateien pro Laufwerk. Bei einem Laufwerk mit mehr als 100 000 Dateien fügt Google Desktop während der Indizierung Ihrem Index in Echtzeit Dateien hinzu, wenn Sie diese verschieben oder öffnen. http://desktop.google.com/support/bin/answer.py?hl=de&answer=12374

[26] Copernic Search (Version 3.2.1 (Build 8) vom 6.8.2009)http://www.copernic.com

[27] Neuropower AG, http://neuropower.com/products_n_solutions/business-center/content_surveyor/

7 Folgerungen und Ausblick

Vor zehn Jahren noch stellte Mandl [Man00] fest, dass für die Indexierung einer großen Textmenge eine Auswahl der Indexterme erforderlich sei. Diese in Abschnitt 2.1 zitierte Aussage lässt sich heute nicht mehr bestätigen.

Mit den in dieser Arbeit vorgestellten Algorithmen kann ein Vollformenwörterbuch in Sekundenbruchteilen auch mit einer unscharfen Suche durchforscht werden. Der zusätzlich erforderliche Speicherbedarf ist gering und die zur erstmaligen Indexierung aufzubringende zusätzliche Zeit lag für das untersuchte sehr große Korpus im Bereich weniger Minuten.

Dazu wurden aus bekannten Verfahren neue Algorithmen und Datenstrukturen entwickelt, um die gewünschte Performance zu erzielen. Bei der Implementierung dieser Algorithmen und Datenstrukturen wurde auf maschinenabhängige Programmiertricks verzichtet. Die Prinzipien gelten allgemein und sollten sich problemlos auf andere Betriebssysteme und Programmiersprachen übertragen lassen.

Aufgrund der unerschöpflichen Produktivität der deutschen Sprache in der Erzeugung neuer Wörter enthalten auch beliebig große Korpora einen hohen Anteil selten vorkommender Terme. Um diese selten vorkommenden Terme finden zu können, sind unscharfe Suchverfahren erforderlich. Bei unscharfen Suchverfahren kann die Menge der gefundenen ähnlichen Terme stark schwanken.

Es wurde gezeigt, dass größere Treffermengen eine deutlich schlechtere Qualität der Ergebnisse bedeuten. Eine optimale Größe der Treffermenge liegt bei fünf bis zehn gefundenen Termen.

Um solche optimalen Treffermengen zu erreichen, müssen die Suchverfahren für die unscharfe Suche gesteuert werden.

Je nach Häufigkeit des Suchbegriffs im Korpus und gewünschter Trefferanzahl sind unterschiedliche Algorithmen für die unscharfe Suche optimal. Welcher Algorithmus optimal ist, hängt bei der unscharfen Suche insbesondere von der Länge des Suchbegriffs ab.

Für sehr kurze Wörter mit nicht mehr als fünf Zeichen entspricht die Trigrammsuche am besten den Nutzererwartungen nach überschaubarer Trefferzahl. Für alle anderen Wörter ist eine Bigrammsuche vorteilhafter. Variationen der Bigrammsuche haben das Potential, deren Ergebnisqualität weiter zu verbessern.

Ein optimales Suchverfahren wählt den zur Wortlänge passenden Algorithmus und prüft die Treffermenge automatisch. Bei einer zu kleinen oder zu großen Treffermenge verändert es die Parameter für die Suche oder wählt einen anderen Algorithmus. Je nach Maschinenausstattung kann das auch parallel erfolgen.

Die vorgestellte Suchmaschine ist schnell, effizient und sehr flexibel. Sie benötigt bei voller Kontrolle über die Art der Suche vergleichsweise wenig zusätzlichen Speicherplatz für die Indexdateien.

Bibliotheken halten die von ihnen archivierten Dokumente in der Regel als Bücher und Zeitschriftenbände vor. Auch dieses Dokument ist mit seinen Querverweisen unter Angabe der Seitenzahl und dem Register am Schluss für ein Lesen als Hardcopy vorbereitet. Obwohl diese Form des Lesens häufig die bequemere ist, wird die Suche nach Textstellen durch eine elektronische Erschließung wesentlich erleichtert.

Mit den hier vorgestellten Verfahren ist eine manuelle Indexierung nicht mehr erforderlich, weil die unscharfe Suche unter Einbezug eines Thesaurus das Auffinden von Dokumenten mit Termen ermöglicht, die der Benutzer gemeint, aber nicht eingegeben hat.

Ebenso erscheint es angebracht, dass Dokumentenserver eine solche unscharfe Suchfunktion implementieren, damit der Nutzer nicht raten muss, in welcher Form ein gesuchter Term im gesuchten Dokument erscheint.

In der Hardwareentwicklung geht der Trend weg von immer schneller getakteten Prozessoren in einer von-Neumann-Architektur und hin zu einer Architektur mit assoziativen Speichermodellen. Mit einem solchen Rechner wäre noch einmal eine erhebliche Leistungssteigerung möglich.

Literaturverzeichnis

Angegebene Internetadressen entsprechen dem Stand von Anfang 2010 und können sich erfahrungsgemäß ändern.

[ACD+03] Amitay, E.; Carmel, D.; Darlow, A.; Herscovici, M.; Lempel, R. ; Soffer, A.: Juru at TREC 2003 - Topic Distillation using Query-Sensitive Tuning and Cohesiveness Filtering. In: *NIST Special Publication 500-255: The 12th Text REtrieval Conference (TREC 2003)*, 2003, 276-282

[ACR01] Amati, G.; Carpineto, C. ; Romano, G.: FUB at TREC-10 Web Track: A probabilistic framework for topic relevance term weighting. In: *Proceedings of the Tenth Text Retrieval Conference (TREC-10)*. Gaithersburg, Md. : NIST Special Publication 500-250, 2001, 182-191

[AMSC07] Al-Maskari, A.; Sanderson, M. ; Clough, P.: The relationship between IR effectiveness measures and user satisfaction. In: *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA : ACM, 2007, 773-774

[AVR02] Amati, G.; Van Rijsbergen, C.: Probabilistic models of information retrieval based on measuring the divergence from randomness. In: *ACM Transactions on Information Systems* 20 (2002), Nr. 4, 357-389. http://eprints.gla.ac.uk/3798/1/3798.pdf

[Bar97] Barry, R. K. (Hrsg.): *ALA-LC Romanization Tables: Transliteration Schemes for Non-Roman Scripts*. Library of Congress, 1997 http://www.loc.gov/catdir/cpso/roman.html

[Bat86] Bates, M.: Subject access in online catalogs: A design model. In: *JASIS* 37 (1986), Nr. 6, 357-376. http://www3.interscience.wiley.com/cgi-bin/fulltext/10016883/PDFSTART

[Bau94] Bauer, F. L.: *Kryptologie*. Berlin : Springer, 1994

[BDKM08] Baccini, A.; Déjean, S.; Kompaoré, D. ; Mothe, J.: *Analyse des critères d'évaluation des systèmes de recherche d'information*. ftp://ftp.irit.fr/IRIT/SIG/2008_ISI_BDKM.pdf. Version: 2008

[Beu05] Beutelspacher, A.: *Kryptologie*. Vieweg Verlagsgesellschaft, 2005

[Bla09] Black, P. E.: *Dictionary of Algorithms and Data Structures*. http://www.itl.nist.gov/div897/sqg/dads/HTML/trie.html. Version: 2009

[BLC95] Berners-Lee, T.; Connolly, D.: *Hypertext Markup Language - 2.0*. http://www.ietf.org/rfc/rfc1866.txt. Version: 1995

[BLJ04] Bach, F. R.; Lanckriet, G. R. G. ; Jordan, M. I.: Fast Kernel Learning using Sequential Minimal Optimization / Division of Computer Science & Department of Statistics, University of California. Version: 2004. http://www.eecs.berkeley.edu/Pubs/TechRpts/2004/CSD-04-1307.pdf. 2004 (CSD-04-1307). – Forschungsbericht

[BM72] Bayer, R.; McCreight, E. M.: Organization and maintenance of large ordered indexes. In: *Acta Informatica* 1 (1972), Nr. 3, S. 173–189

[BM04] Bansal, S.; Modha, D. S.: CAR: Clock with Adaptive Replacement. In: *Proc. Third USENIX Conf. on File and Storage Technologies (FAST 04*, 2004, 187–200

[Bußß02] Bussmann, H.; Bussmann, H. (Hrsg.): *Lexikon der Sprachwissenschaft*. Kröner, 2002

[Buc00] Buckley, C.: The TREC-9 Query Track. In: *NIST Special Publication 500-249: The Ninth Text REtrieval Conference*, 2000, 81-86

[Bur98] Burges, C. J. C.: A Tutorial on Support Vector Machines for Pattern Recognition. In: *Data Mining and Knowledge Discovery* 2 (1998), 121-167. http://research.microsoft.com/en-us/um/people/cburges/papers/SVMTutorial.pdf

[BYRN99] Baeza-Yates, R.; Ribeiro-Neto, B.; Baeza-Yates, R. (Hrsg.); Ribeiro-Neto, B. (Hrsg.): *Modern Information Retrieval*. New York : ACM Press, Addison-Wesley, 1999

[CC03] Caid, W.; Carleton, J.: *Context Vector-Based Text Retrieval*. http://www.fairisaac.com/nr/rdonlyres/e26f07ed-0bac-4987-8610-d98053f2960e/0/contextvectorwhitepaper.pdf.

http://ksuseer1.ist.psu.edu/viewdoc/download;jsessionid=62F37926BAE373EFA7C3A3598656B619?doi=10.1.1.87.7893&rep=rep1&type=pdf. Version: 2003. – URL nicht erreichbar

[CDJB77] *Kapitel* Access to the Internal Lexicon. In: Coltheart, M.; Davelaar, E.; Jonasson, J. ; Besner, D.: *Attention and Performance VI*. Hillsdale, NJ : Erlbaum, 1977, S. 535–555

[CGR08] Cao, G.; Gao, J.-Y. N. a. ; Robertson, S.: Selecting good expansion terms for pseudo-relevance feedback. In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ACM Press, 2008, 243-250

[Cle07] Clematide, S.: Vorlesungsscript Einführung in die Computerlinguistik I / Institut für Computerlinguistik Universität Zürich. Version: 2007. http://www.cl.uzh.ch/siclemat/lehre/hs07/ecl1/script/script.pdf. 2007. – Forschungsbericht

[Con07] Consortium, T. U.: *The Unicode Standard, Version 5.0*. Addison-Wesley, 2007 http://www.unicode.org/versions/Unicode5.0.0/ch03.pdf

[CRP+01] Coltheart, M.; Rastle, K.; Perry, C.; Langdon, R. ; Ziegler, J.: The DRC model: A model of visual word recognition and reading aloud. In: *Psychological Review* 108 (2001), Nr. 01, S. 204–258

[CSK08] Catanzaro, B.; Sundaram, N. ; Keutzer, K.: Fast Support Vector Machine Training and Classification on Graphics Processors. In: *Proceedings of the 25 th International Conference on Machine Learning*, 2008, 104–111

[CST00] Cristianini, N.; Shawe-Taylor, J.: *AN INTRODUCTION TO SUPPORT VECTOR MACHINES*. Cambridge University Press, 2000 http://www.support-vector.net/chapter_6.html

[CT06] Christy, A.; Thambidurai, P.: Feature Selection for Efficient Text Categorization and Knowledge Discovery Using Classification Techniques. In: *Asian Journal of Information Technology* 5 (2006), Nr. 8, 872-876. http://www.medwelljournals.com/fulltext/ajit/2006/872-876.pdf

[CZR05] Craswell, N.; Zaragoza, H. ; Robertson, S.: Microsoft Cambridge at TREC-14: Enterprise Track. In: *Proceedings of the Fourteenth Text REtrieval Conference (TREC 2005)*. Gaithersburg, USA : NIST, November 2005. – Describes application and tuning of Okapi BM25F

[Dav06] Davis, T.: *Direct Methods for Sparse Linear Systems*. http://www.cise.ufl.edu/research/sparse/CSparse/. http://www.cise.ufl.edu/research/sparse/CSparse/. Version: 2006

[DD08] Davis, M.; Dürst, M.: *Unicode Normalization Forms*. http://www.unicode.org/reports/tr15. Version: 2008

[DDF+88] Deerwester, S.; Dumais, S.; Furnas, G.; Harshman, R.; Landauer, T.; Lochbaum, K. ; Streeter, L.: *Computer information retrieval using latent semantic indexing*. http://www.freepatentsonline.com/4839853.html. http://www.freepatentsonline.com/4839853.html. Version: 1988

[DDF+90] Deerwester, S.; Dumais, S.; Furnas, G.; Landauer, T. ; Harshman, R.: Indexing by latent semantic analysis. In: *Journal of the American Society for Information Science* 41 (1990), 391-407. http://lsa.colorado.edu/papers/JASIS.lsi.90.pdf

[Dev88] Development, D. o. T. C.-o. f.: Report on the Conference. In: *Fifth United Nations Conference on the Standardization of Geographical Names* Bd. 1. Montreal : United Nations, 1988, 42-43

[DH08] Davis, M.; Heninger, A.: *Unicode Regular Expressions*. http://www.unicode.org/reports/tr18/. Version: 2008

[DL90] DuRoss Liddy, E.: Anaphora in natural language processing and information retrieval. In: *Information Processing and Management* 26 (1990), S. 39–52

[Dos96] Dosdrowski, G. (Hrsg.): *Duden – Die deutsche Rechtschreibung*. Bd. 1. 21. Dudenverlag, 1996

[DW09] Davis, M.; Whistler, K.: *Unicode Collation Algorithm*. http://www.unicode.org/reports/tr10/. Version: 2009

[ECM85] ECMA: *8-bit Single-byte Coded Graphic Character Set*. http://www.ecma-international.org/publications/files/ECMA-ST-WITHDRAWN/ECMA-94,1stEdition,March1985.pdf. Version: 1985

[ECM86] ECMA: *8-bit Single-byte Coded Graphic Character Sets Latin Alphabets No. 1 to No. 4*. http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-094.pdf. Version: 1986

[Ede07] Edelkamp, S.: *Suffix tree*. http://www.nist.gov/dads/HTML/suffixtree.html. Version: 2 2007. – Dictionary of Algorithms and Data Structures [online]

[Eid02] Eiden, W.: *Präzise Unschärfe. Informationsmodellierung durch Fuzzy-Mengen*. Ibidem Verlag, 2002

[Ess04] Esser, W.: Fault-tolerant Fulltext Information Retrieval in Digital Multilingual Encyclopedias with Weighted Pattern Morphing. In: S., M. (Hrsg.); Tait, J. (Hrsg.): *Advances in Information Retrieval, Procedings of the 26th ECIR* Bd. 2997, Springer, 2004 (LNCS), S. 338–351

[Ess05] Esser, W.: Fault-tolerant fulltext search for large multilingual scientific text corpora. In: *Journal of Digital Information* 6 (2005), Nr. 1, S. 1–9

[Fid85] Fidel, R.: Individual variability in online searching behavior. In: Parkhurst, C. (Hrsg.); ASIS (Veranst.): *ASIS'85: Proceedings of the ASIS 48th Annual Meeting* Bd. 22 ASIS, 1985, 69-72

[FLGD83] Furnas, G.; Landauer, T.; Gomez, L. ; Dumais, S.: Statistical semantics: Analysis of the potential performance of key-word information systems. In: *Bell System Technical Journal* 62 (1983), Nr. 6, S. 1753–1806

[Fol91] Foltz, P.: Models of Human Memory and Computer Information Retrieval: Similar Approaches to Simiar [Druckfehler im Original] Problems / University of Colorado. Version: 1991. http://ics.colorado.edu/techpubs/pdf/91-03.pdf. Boulder, Colorado, 1991 (91-03). – ICS Tech Report

[For03] Forman, G.: An Extensive Empirical Study of Feature Selection Metrics for Text Classification. In: *Journal of Machine Learning Research* 3 (2003), 1289-1305. http://jmlr.csail.mit.edu/papers/volume3/forman03a/forman03a.pdf

[Fri96] Friedrich, M.: *Entwurf und Implementierung von Verfahren zur Bestimmung von Textinhalten aus Verkehrsmeldungen für die Überführung in das digitale RDS-TMC-Format.*, TH Darmstadt, Diplomarbeit, 1996. http://www.smile-datentechnik.de/projekte/dipl/

[Fri08] Friedl, J.; Friedl, J. (Hrsg.): *Reguläre Ausdrücke*. Köln : O'Reilly, 2008 http://www.regular-expressions.info/tutorial.html

[FRWH98] Fuhr, N.; Rittberger, M. ; Womser-Hacker, C.; Fuhr, N. (Hrsg.); Rittberger, M. (Hrsg.) ; Womser-Hacker, C. (Hrsg.): *Information Retrieval. Materialien zur Herbstschule*. Bonn : Gesellschaft für Informatik, 1998

[Fuh95] Fuhr, N.: *Information Retrieval*. 1995. – Universität Dortmund (Skriptum zur Vorlesung im SS 93)

[Gal91] Gallant, S.: Context Vector Representation for Document Retrieval. In: *Natural language text Retreival Workshop*. Anaheim, CA, 1991 (AAAI-91)

[Gil90] Giloi, W.: Konrad Zuses Plankalkül als Vorläufer moderner Programmiermodelle / Konrad-Zuse-Zentrum für Informationstechnik. Version: 1990. http://www.zib.de/Publications/Reports/TR-90-13.pdf. Berlin, 1990. – Forschungsbericht

[GJM01] Ghani, R.; Jones, R. ; Mladenic, D.: Automatic Web Search Query Generation to Create Minority Language Corpora. In: *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2001, 432 - 433

[GM04] Gabrilovich, E.; Markovitch, S.: Text Categorization with Many Redundant Features: Using Aggressive Feature Selection to Make SVMs Competitive with C4.5. In: *ICML'04*, 2004

[GMS01] Greco, S.; Matarazzo, B. ; Slowinski, R.: Rough sets theory for multicriteria decision analysis. In: *European Journal of Operational Research* 129 (2001), Nr. 1, S. 1–47

[Gra93] Graefe, G.: Query Evaluation Techniques for Large Databases. In: *ACM Computing Surveys* 25 (1993), Nr. 2, 73-170. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.108.3178&rep=rep1&type=pdf

[Gra04] Graecae, T. L.: *The TLG Beta Code Manual 2004*. http://stephanus.tlg.uci.edu/encoding/BCM2004.pdf. Version: 2004

[Hag94] Hagström, M.: Error tolerant Retrieval in Large Text Files / Universität Hildesheim. 1994 (12). – Hildesheimer Informatikberichte

[Hag96] Hagström, M.: *Textrecherche in großen Datenmengen auf der Basis spärlich codierter Assoziativmatrizen*, Universität Hildesheim, Diss., 1996

[Ham50] Hamming, R. W.: Error-detecting and error-correcting codes. In: *Bell System Technical Journal* 29 (1950), Nr. 2, S. 147–160

[Har74] Harter, S.: *A probabilistic approach to automatic keyword indexing*, The University of Chicago, Diss., 1974

[Har75] Harter, S. P.: A Probabilistic Approach to Automatic Keyword Indexing. Part I. On the Distribution of Specialty Words in a Technical Literature. In: *Journal of the American Society for Information Science* 26 (1975), Nr. 4, 197-206. http://dx.doi.org/10.1002/asi.4630260402. – DOI 10.1002/asi.4630260402

[Har88] Harman, D.: Towards interactive query expansion. In: *Proceedings of the Eleventh Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Grenoble, 1988, S. 321–331

[Har92] Harman, D.: Relevance feedback revisited. In: *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Kopenhagen, 1992, S. 1–10

[HC05] Hawking, D.; Craswell, N.: The Very Large Collection and Web Tracks. In: Voorhees, E. (Hrsg.); Harman, D. (Hrsg.): *TREC: Experiment and Evaluation in Information Retrieval*, MIT Press, 2005

[Heb49] Hebb, D.; Hebb, D. (Hrsg.): *The Organization of Behavior: a neuropsychological approach*. Wiley, 1949

[Hei94] Heitland, M.: *Einsatz der SpaCAM-Technik für ausgewählte Grundaufgaben der Informatik*, University of Hildesheim, Diss., 1994

[Hob78] Hobbs, J.: Resolving pronoun references. In: *Lingua* 44 (1978), 311-338. http://www.isi.edu/~hobbs/ResolvingPronounReferences.pdf

[HQW06] Heyer, G.; Quasthoff, U. ; Wittig, T.; Heyer, G. (Hrsg.); Quasthoff, U. (Hrsg.) ; Wittig, T. (Hrsg.): *Text Mining: Wissensrohstoff Text*. Herdecke : W3L-Verlag,, 2006

[HZ07] Hawking, D.; Zobel, J.: Does Topic Metadata Help With Web Search? In: *Journal of the American Society for Information Science and Technology* 58 (2007), Nr. 5, 613-628. http://goanna.cs.rmit.edu.au/~jz/fulltext/jasisthz.pdf

[Ide71] *Kapitel* New experiments in relevance feedback. In: Ide, E.: *The Smart system – experiments in automatic document processing*. Englewood Cliffs, NJ : Prentice Hall Inc., 1971, S. 337–354

[IEE04] IEEE: *The Open Group Base Specifications Issue 6 IEEE Std 1003.1, 2004 Edition*. http://www.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap09.html. Version: 2004

[ISO98] ISO/IEC: *Information technology – 8-bit single-byte coded graphic character sets*. 1998

[Joa98] Joachims, T.: Text Categorization with Support Vector Machines: Learning with Many Relevant Features. In: *Proceedings of the Tenth European Conference on Machine Learning*. Berlin : Springer, 1998, 137-142

[Kec01] Kecman, V.: *Learning and Soft Computing*. Cambridge, MA, : The MIT Press, 2001

[Ken38] Kendall, M. G.: A new measure of rank correlationa. In: *Biometrika* 30 (1938), 81-93. https://vpn.serv.uni-osnabrueck.de/cgi/reprint/30/1-2/,DanaInfo=biomet.oxfordjournals.org+81

[KF88] Klir, G. J.; Folger, T. A.: *Fuzzy Sets, Uncertainty and Information*. Upper Saddle River, NJ, USA : Prentice Hall, 1988. – 368 S.

[Knu73] Knuth, D. E.: *The Art of Computer Programming*. Addison-Wesley, 1973

[KPPS98] Komorowski, J.; Pawlak, Z.; Polkowski, L. ; Skowron, A.: *Rough Sets: A Tutorial*. http://www.uit.edu.vn/forum/index.php?act=Attach&type=post&id=19757. Version: 1998

[KS03] Kärkkäinen, J.; Sanders, P.: Simple linear work suffix array construction. In: Baeten, J. e. a. (Hrsg.): *Proc. 30th International Colloquium on Automata, Languages and Programming (ICALP '03)* Bd. 2719, 2003 (LNCS), 943-955

[KY95] Klir, G. J.; Youan, B.: *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Prentice Hall PTR, 1995. – 592 S.

[Lau01] Laurikari, V.: *Efficient submatch addressing for regular expressions.*, Helsinki University of Technology, Diplomarbeit, 2001. http://laurikari.net/ville/regex-submatch.pdf

[Lau06] Laurikari, V.: *TRE home page*. http://laurikari.net/tre/index.html. http://laurikari.net/tre/index.html. Version: 2006

[Lev65] Levenshtein, V.: Binary codes capable of correcting deletions, insertions, and reversals. In: *Doklady Akademii Nauk SSSR* 163 (1965),

S. 845–848. – (Russisch). Englische Übersetzung in: Soviet Physics Doklady, 10(8) S. 707-710, 1966

[Lew05] Lewandowski, D.; Lewandowski, D. (Hrsg.): *Web Information Retrieval: Technologien zur Informationssuche im Internet*. Bd. 7. Deutsche Gesellschaft f. Informationswissenschaft u. Informationspraxis, 2005 http://www.durchdenken.de/lewandowski/web-ir/download/Web-IR-Buch.pdf

[LFL98] Landauer, T.; Foltz, P. ; Laham, D.: An Introduction to Latent Semantic Analysis. In: *Discourse Processes* 25 (1998), 259-284. http://lsa.colorado.edu/papers/dp1.LSAintro.pdf

[LGF97] Lundquist, C.; Grossman, D. A. ; Frieder, O.: Improving relevance feedback in the vector space model. In: *Proceedings of the sixth international conference on Information and knowledge management (CIKM '97)*. Las Vegas, 1997, S. 16–23

[Lil54] Liley, O.: Evaluation of the subject catalog. In: *American Documentation* 5 (1954), Nr. 2, 41–60. http://www3.interscience.wiley.com/cgi-bin/fulltext/114212804/PDFSTART

[LPSLY99] Liddy, E.; Paik, W. ; Szu-Li Yu, E.: *Natural language processing system for semantic vector representation which accounts for lexical ambiguity*. http://www.patentstorm.us/patents/5873056-fulltext.html. http://www.patentstorm.us/patents/5873056-fulltext.html. Version: 1999

[Lub01] Luba, T.: *Konkordanzen in langen Zeichenketten*, Universität Hildesheim, Diss., 2001

[LWQ09] Liang, J.; Wang, J. ; Qian, Y.: A new measure of uncertainty based on knowledge granulation for rough sets. In: *Inf. Sci.* 179 (2009), Nr. 4, S. 458–470. http://dx.doi.org/http://dx.doi.org/10.1016/j.ins.2008.10.010. – DOI http://dx.doi.org/10.1016/j.ins.2008.10.010. – ISSN 0020–0255

[Mak00] Mak, G.: *The Implementation of Support Vector Machines using the Sequential*. Montreal, Canada, School of Computer Science McGill University, Diplomarbeit, 2000. http://www.cs.mcgill.ca/~hv/publications/99.04.McGill.thesis.gmak.pdf

[Man00] Mandl, T.: Einsatz neuronaler Netze als Transferkomponenten beim Retrieval in heterogenen Dokumentbeständen / InformationsZentrum Sozialwissenschaften. Version: 2000. http://www.gesis.org/

publikationen/Berichte/IZ_Arbeitsberichte/pdf/ab_20.pdf. Lennéstraße 30, D-53113 Bonn, 2000. – Forschungsbericht

[Man01] Mandl, T.: *Tolerantes Information Retrieval: Neuronale Netze zur Erhöhung der Adaptivität und Flexibilität bei der Informationssuche.*, Konstanz, Diss., 2001. http://eprints.rclis.org/archive/00003832/01/cc_diss_mandl.pdf

[Man05] Mandl, T.: Tolerantes Information Retrieval. Neuronale Netze zur Erhöhung der Adaptivität and Flexibilität bei der Informationssuche. / Hochschulverband für Informationswissenschaften e.V. Konstanz. Version: 2005. http://web1.bib.uni-hildesheim.de/edocs/2005/480465738/doc/480465738.pdf. Konstanz, 2005 (39). – Forschungsbericht

[Min11] Minkowski, H.: *Geometrie der Zahlen*. Leipzig : Teubner, 1911 http://gallica.bnf.fr/ark:/12148/bpt6k99643x.image.f1

[MK60] Maron, M.; Kuhns, J.: On relevance, probabilistic indexing and information retrieval. In: *Journal of the ACM* 7 (1960), S. 216–244

[Mor68] Morrison, D. R.: PATRICIA - Practical Algorithm to Retrieve Information Coded in Alphanumeric. In: *Journal of the ACM* 15 (1968), Nr. 4, S. 514–534

[MP43] McCulloch, W.; Pitts, W.: A logical calculus of the ideas immanent in nervous activity. In: *Bulletin of Mathematical Biophysics* 5 (1943), S. 115–133

[MRS08] Manning, C. D.; Raghavan, P. ; Schütze, H.: *Introduction to Information Retrieval*. Cambridge University Press, 2008 http://nlp.stanford.edu/IR-book/information-retrieval-book.html

[MS04] Mihov, S.; Schulz, K.: Fast Approximate Search in Large Dictionaries. In: *Computational Linguistics* 30 (2004), Nr. 4, 451-477. http://www.cis.uni-muenchen.de/people/Schulz/Pub/fastapproxsearch.pdf

[NR02] Navarro, G.; Raffinot, M.; Navarro, G. (Hrsg.); Raffinot, M. (Hrsg.): *Flexible Pattern Matching in Strings: practical on-line search algorithms for texts and biological sequences*. Cambridge Univ. Press, 2002

[OMY06] Onoda, T.; Murata, H. ; Yamada, S.: Non-Relevance Feedback Document Retrieval based on One Class SVM and SVDD. In: *Proceedings of the International Joint Conference on Neural Networks, IJCNN 2006*, 2006, S. 1212 – 1219

[Paw91] Pawlak, Z.: *Rough Sets: Theoretical Aspects of Reasoning About Data*. Dordrecht : Kluwer Academic Publishing, 1991

[Ped05] Pedersen, T. T.: *Transliteration of Greek*. http://transliteration.eki.ee/pdf/Greek.pdf. Version: 2005

[PJ96] Pirkola, A.; Järvelin, K.: The effect of anaphor and ellipsis resolution on proximity searching in a text database. In: *Information Processing and Management* 32 (1996), 199-216. http://www.info.uta.fi/tutkimus/fire/archive/AP0b.pdf

[Pla99] *Kapitel* Fast training of support vector machines using sequential minimal optimization. In: Platt, J. C.: *Fast training of support vector machines using sequential minimal optimization Advances in kernel methods: support vector learning*. MIT Press, 1999, 185-208

[PS06] Pagiamtzis, K.; Sheikholeslami, A.: Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey. In: *IEEE journal of solid-state circuits* 41 (2006), 3, Nr. 3, 712-727. http://www.pagiamtzis.com/pubs/pagiamtzis-jssc2006.pdf

[PTRV98] Papadimitriou, C. H.; Tamaki, H.; Raghavan, P. ; Vempala, S.: Latent semantic indexing: a probabilistic analysis. In: *PODS '98: Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*. New York, NY, USA : ACM, 1998, 159-168

[PWZ88] Pawlak, Z.; Wong, S. K. M. ; Ziarko, W.: Rough sets: probabilistic versus deterministic approach. In: *Int. J. Man-Mach. Stud.* 29 (1988), Nr. 1, S. 81–95. http://dx.doi.org/http://dx.doi.org/10.1016/S0020-7373(88)80032-4. – DOI http://dx.doi.org/10.1016/S0020–7373(88)80032–4. – ISSN 0020–7373

[Rap97] Rapp, R.: Text-Detektor. Fehlertolerantes Retrieval ganz einfach. In: *c't-Archiv* 4 (1997), S. 386

[Rij79] Rijsbergen, C. J. V.: *Information Retrieval*. London : Butterworths, 1979 http://www.dcs.gla.ac.uk/Keith/Preface.html

[RL03] Ruthven, I.; Lalmas, M.: A survey on the use of relevance feedback for information access systems. In: *The Knowledge Engineering Review* 18 (2003), 95-145. http://www.dcs.qmul.ac.uk/~mounia/CV/Papers/ker_ruthven_lalmas.pdf

[Roc66] Rocchio, J.: *Document retrieval systems-optimization and evaluation*. Cambridge, MA., Harvard Computational Laboratory, Diss., 1966

[Roc71] *Kapitel* Relevance feedback in information retrieval. In: Rocchio, J.: *The SMART retrieval system*. Englewood Cliffs. NJ : Prentice Hall, 1971, S. 313–323

[Rom05] Romberger, D.: *Bildmustererkennung mit Hilfe spärlich codierter Assoziativmatrizen (SpaCAM) am Beispiel von Kfz-Kennzeichen*, Hildesheim, Diss., 2005. http://web1.bib.uni-hildesheim.de/edocs/2006/511041160/doc/511041160.pdf

[Rus18] Russell, R. C.: *SoundEx*. http://www.pat2pdf.org/pat2pdf/foo.pl?number=1261167. Version: 1918

[Rus22] Russell, R. C.: *Soundex Patent*. http://www.pat2pdf.org/pat2pdf/foo.pl?number=1435663. Version: 1922

[RWHB98] Robertson, S. E.; Walker, S. ; Hancock-Beaulieu, M.: Okapi at TREC-7. In: *Proceedings of the Seventh Text REtrieval Conference*. Gaithersburg, USA, November 1998, 253-264

[RWJ+94] Robertson, S. E.; Walker, S.; Jones, S.; Hancock-Beaulieu, M. ; Gatford, M.: Okapi at TREC-3. In: *Proceedings of the Third Text REtrieval Conference (TREC 1994)*. Gaithersburg, USA : NIST, November 1994

[RWJ+95] Robertson, S. E.; Walker, S.; Jones, S.; Hancock-Beaulieu, M. M. ; Gatford, M.: Okapi at TREC-3. In: *Overview of the Third Text REtrieval Conference (TREC-3)*, 1995, S. 109–126

[Sal71] Salton, G.; Salton, G. (Hrsg.): *The SMART Retrieval System - Experiments in Automatic Document Processing*. Prentice-Hall, 1971

[Sal86] Salton, G.: Another Look at Automatic Text Retrieval Systems. In: *Communications of the ACM* 20 (1986), S. 648 – 656

[Sal89] Salton, G.; Salton, G. (Hrsg.): *Automatic Text Processing*. Addison-Wesley, 1989

[Sal90] Salton, G.: Improving Retrieval Performance by Relevance Feedback. In: *Jounal of the American Society for Information Science* 41 (1990), Nr. 4, 288-297. http://users.cs.fiu.edu/~vagelis/classes/COP6776/publications/jasistSalton1990.pdf

[San08] Sanderson, M.: Ambiguous queries: test collections need more sense. In: *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, ACM, 2008, 499-506

[SB88] Salton, G.; Buckley, C.: Parallel text search methods. In: *Communications of the ACM* 31 (1988), Nr. 2, 202-215. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.87.7443&rep=rep1&type=pdf

[SGM05] *Kapitel* Rough set based decision support. In: Slowinski, R.; Greco, S. ; Matarazzo, B.: *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer-Verlag, 2005, S. 475–527

[Sin01] Singhal, A.: Modern Information Retrieval: A Brief Overview. In: *IEEE Data Engineering Bulletin* 24 (2001), 35-43. http://singhal.info/ieee2001.pdf

[SJWR00a] Sparck Jones, K.; Walker, S. ; Robertson, S.: A probabilistic model of information retrieval: development and comparative experiments. Part 1. In: *Information Processing and Management* 36 (2000), 779-808. http://www.soi.city.ac.uk/~ser/blockbuster/pmir-pt1-reprint.pdf

[SJWR00b] Sparck Jones, K.; Walker, S. ; Robertson, S.: A probabilistic model of information retrieval: development and comparative experiments. Part 2. In: *Information Processing and Management* 36 (2000), 809-840. http://www.soi.city.ac.uk/~ser/blockbuster/pmir-pt2-reprint.pdf

[SK03] Strey, A.; Kaiser, J.: *Vorlesung technische Informatik II*. http://www.informatik.uni-ulm.de/ni/Lehre/WS03/TechInf2/2003w-TI2-B3-4.pdf. Version: 2003

[SM83] Salton, G.; McGill, M.; Salton, G. (Hrsg.); McGill, M. (Hrsg.): *Introduction to Modern Information Retrieval*. New York. : McGraw-Hill, 1983

[STC00] Shawe-Taylor, J.; Cristianini, N.: *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000

[Sti05] Stiebe, R.: *Textalgorithmen. Vorlesung im Wintersemester 2005/06*. http://theo.cs.uni-magdeburg.de/lehre06w/textalg/skript/skript2005.pdf. Version: 2005

[Sto07] Stock, W.; Stock, W. (Hrsg.): *Information Retrieval*. München : Oldenbourg, 2007

[SV97] *Kapitel* Similarity relation as a basis for rough approximations. In: Slowinski, R.; Vanderpooten, D.: *Advances in Machine Intelligence & Soft Computing*. Raleigh, 1997, S. 17–33

[SV00] Slowinski, R.; Vanderpooten, D.: A Generalized Definition of Rough Approximations Based on Similarity. In: *IEEE Transactions on Knowledge and Data Engineering* 12 (2000), Nr. 2, 331-336. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.5620&rep=rep1&type=pdf

[TB74] Tarr, D.; Borko, H.: Factors influencing inter-indexer consistency. In: *Proceedings of the ASIS 37th Annual Meeting* Bd. 11, 1974, S. 50–55

[TD04] Tang, C.; Dwarkadas, S.: Hybrid global-local indexing for effcient peer-to-peer information retrieval. In: *NSDI'04: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation*. Berkeley, CA, USA : USENIX Association, 2004, S. 16–16

[TGMS93] Tomasic, A.; Garcia-Molina, H. ; Shoens, K.: Incremental Updates of Inverted Lists for Text Document Retrieval / Stanford University Computer Science. Version: 1993. http://infolab.stanford.edu/pub/tomasic/1993/stan.cs.tn.93.1.ps. 1993. – Forschungsbericht

[TS06] Turpin, A.; Scholer, F.: User performance versus precision measures for simple search tasks. In: *Annual ACM Conference on Research and Development in Information Retrieval archive. Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval.* Seattle, Washington, USA : ACM Press, 2006, 11-18

[TS07] Thom, J.; Scholer, F.: A Comparison of Evaluation Measures Given How Users Perform on Search Tasks. In: *Proceedings of Twelfth Australasian Document Computing Symposium (ADCS 2007)*. Melbourne, Australia, : RMIT University, 12 2007, 100-103

[Vap95] Vapnik, V.: *The Nature of Statistical Learning Theory*. New York : Springer-Verlag, 1995

[Vap98] Vapnik, V.: *Statistical Learning Theory*. New York : John Wiley and Sons, Inc., 1998

[W3C99] W3C: *HTML 4.01 Specification*. http://www.w3.org/TR/html401/struct/text.html. Version: 1999

[WAD07] Wartena, C.; Anjewierden, A. ; Dieten, W. v.: Using patterns over ontology terms to detect inconsistencies in large text collections. In: Mönnich, U. (Hrsg.); Kühnberger, K.-U. (Hrsg.): *OTT'06 Ontologies in Text Technology: Approaches to Extract Semantic Knowledge from Structured Information.* Osnabrück, 2007 (Publications of the Institute of Cognitive Science (PICS) 1)

[Wah85] Wahrig, G.; Wahrig, G. (Hrsg.): *Deutsches Wörterbuch*. Mosaik Verlag, 1985

[Wei73] Weiner, P.: Linear pattern matching algorithm. In: *Proc. 14th Symposium on Switching and Automata Theory* IEEE, 1973, S. 1–11. – zit. nach : Kärkkäinen and Sanders, 2003

[WFXF04] Wang, L.; Fan, W.; Xi, W. ; Fox, E. A.: Can we get a better retrieval function from machine? In: *Proceedings of the Text Retrieval Conference (TREC)*. Gaithersburg, MD, 2004

[Wie99] Wiebel, D.: *CQP / XKWIC (Corpus Query Processor) / (X Key Word In Context)*. http://www.wiebel.de/pdf/xkwic.pdf. Version: 1999

[Wis98] Wissenschaften, A. d. W. i. G. u. d. H. A. d. W. B.-B. A. d.; Wissenschaften, A. d. W. i. G. u. d. H. A. d. W. B.-B. A. d. (Hrsg.): *Goethe-Wörterbuch*. Kohlhammer, 1998 http://germazope.uni-trier.de/Projects/WBB/woerterbuecher/gwb/benutzerhinweise

[WM91] Wu, S.; Manber, U.: Fast Text Searching With Errors / University of Arizona, Department of Computer Science. Version: 1991. www.cs.utk.edu/~cs494/reading-list/agrep.1.ps. Tucson, AZ, 1991. – Tech. Report TR-9111

[WM92a] Wu, S.; Manber, U.: Agrep – A Fast Approximate Pattern-Matching Tool. In: *Usenix Winter 1992 Technical Conference*, 1992, S. 153–162

[WM92b] Wu, S.; Manber, U.: Fast Text Searching Allowing Errors. In: *Communications of the ACM* 35 (1992), S. 83–91

[WS92] Wilbur, W.; Sirotkin, K.: The Automatic Identification of Stop Words. In: *Journal of Information Science* 18 (1992), 45-55. http://intl-jis.sagepub.com/cgi/reprint/18/1/45

[XC96] Xu, J.; Croft, W. B.: Query expansion using local and global document analysis. In: *SIGIR '96: Proceedings of the 19th annual international ACM*

SIGIR conference on Research and development in information retrieval. New York, NY, USA : ACM, 1996. – ISBN 0–89791–792–8, S. 4–11

[XJW09] Xu, Y.; Jones, G. J. F. ; Wang, B.: Query dependent pseudo-relevance feedback based on wikipedia. In: *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, 2009, 59-66

[XZZ09] Xu, W.-h.; Zhang, X.-y. ; Zhang, W.-x.: Knowledge granulation, knowledge entropy and knowledge uncertainty measure in ordered information systems. In: *Appl. Soft Comput.* 9 (2009), Nr. 4, S. 1244–1251. http://dx.doi.org/http://dx.doi.org/10.1016/j.asoc.2009.03.007. – DOI http://dx.doi.org/10.1016/j.asoc.2009.03.007. – ISSN 1568–4946

[YP97] Yang, Y.; Pedersen, J. O.: A Comparative Study on Feature Selection in Text Categorization. In: *ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1997, 412-420

[Zad65] Zadeh, L.: Fuzzy Sets. In: *Information and Control* 8 (1965), 338-353. http://www-bisc.cs.berkeley.edu/zadeh/papers/Fuzzy%20Sets-1965.pdf

[Zad94] Zadeh, L. A.: Fuzzy logic, neural networks, and soft computing. In: *Commun. ACM* 37 (1994), Nr. 3, 77–84. http://dx.doi.org/http://doi.acm.org/10.1145/175247.175255. – DOI http://doi.acm.org/10.1145/175247.175255. – ISSN 0001–0782

[Zad01] Zadeh, L. A.: A New Direction in AI: Toward a Computational Theory of Perceptions. In: *AI Magazine* 22 (2001), S. 73–84

[Zad02] Zadeh, L.: *What is BISC?* http://www-bisc.cs.berkeley.edu/BISCProgram/default.htm. Version: 2002

[Zel97] Zell, A.: *Simulation neuronaler Netze*. R. Oldenbourg Verlag, 1997

[Zip49] Zipf, G.: *Human behavior and the principle of least effort*. Cambridge, MA. : Addison Wesley, 1949

[ZPC00] Ziegler, J.; Perry, C. ; Coltheart, M.: The DRC model in visual word recognition and reading aloud: An extension to German. In: *European Journal of Cognitive Psychology* 12 (2000), Nr. 3, 413–430. http://www.univ-provence.fr/gsite/Local/lpc/dir/ziegler/article/2000.DRC.ECP.ziegler.perry.coltheart.pdf

[Zus45] Zuse: *The Plankalkül*. http://www.zib.de/zuse/English_Version/Inhalt/Texte/Chrono/40er/Pdf/0233.pdf. Version: 1945

[ZYG08] *Kapitel* Axiomatic Definition of Knowledge Granularity and its Constructive Method. In: Zhao, M.-q.; Yang, Q. ; Gao, D. z.: *Rough Sets and Knowledge Technology*. Berlin / Heidelberg : Springer, 2008, 348-354

Anhang

Tabelle 7.1: Liste der Sonderzeichen

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman❶
&#26;	2	□	□	□
&#157;	1	▯	▯	▯
&#39;	22	'	'	'
&#45;	7	-	-	-
&shy;	3			
&#173;	2			
&#8209;	14		-	-
&ndash;	186882			–
&#8211;	2985		–	–
&#150;	358	–	–	–
&mdash;	2896			—
&#151;	733	—	—	—
&#8212;	69		—	—
&#8213;	2		―	―
&#x30d;	7		□	❷
&#717;	1		―	―
	4			
&quot;	64220			"
&amp;	884			&
&&zlig;	1			&&zlig;
&‹	3			&‹
&139;	1			&139;
&AUML;	1			&AUML;
&EACUTE;	1			&EACUTE;
&ö	3			&ö
&OUML;	2			&OUML;
&öuml;	1			&öuml;
&uum;	1			&uum;
&UUML;	3			&UUML;
&üuml;	1			&üuml;
&Üuml;	1			&Üuml;
&Uuuml;	1			&Uuuml;
&#91;	2	[	[	[
&#93;	2	]	]	]
&#136;	1	ˆ	ˆ	ˆ
&iexcl;	3			¡
&#168;	10	¨	¨	¨
&uml;	1			¨

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman❶
&macr;	3			¯
&acute;	2412			´
&#180;	71	´	´	´
&#184;	1	¸	¸	¸
&#x2d8;	4		˘	˘
&#900;	7		΄	΄
&#145;	939	‘	▯	‘
&lsquo;	520			‘
&#8216;	431		‘	‘
&rsquo;	14333			’
&#146;	1691	’	▯	’
&#8217;	1187		’	’
&#130;	61	,	▯	,
&sbquo;	52			,
&#8218;	27		,	,
&ldquo;	4802			“
&#147;	3312	“	▯	“
&#8220;	101		“	“
&#147</P>;	1			“¶;❸
&rdquo;	3327			”
&#148;	3133	”	▯	”
&#8221;	11		”	”
&#132;	4311	„	▯	„
&bdquo;	1540			„
&#8222;	410		„	„
&prime;	91			′
&#8242;	21		′	′
&lsaquo;	18684			‹
&#8249;	914		‹	‹
&rsaquo;	18796			›
&#8250;	916		›	›
&#x2212;	1		−	−
&minus;	1			−
&#x2215;	8		∕	∕
&#8729;	2		∙	∙
&lt;	6951			<
&gt;	6760			>
&#062;	11	>	>	>

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman
&#177;	1	±	±	±
&laquo;	660542			«
&raquo;	665490			»
&times;	48			×
&divide;	1			÷
&#x221a;	8		√	√
&#x2323;	14		⌣	⌣
&loz;	4			◊
&#9678;	1		◎	◎
&pound;	22			£
&#163;	4	£	£	£
&curren;	18			¤
&sect;	1849			§
&#167;	184	§	§	§
&copy;	124			©
&#169;	1	©	©	©
&reg;	7			®
&deg;	291			°
&#176;	29	°	°	°
&#181;	1	µ	µ	µ
&para;	16			¶
&middot;	180			·
&#134;	166	†	†	†
&dagger;	36			†
&#8224;	1		†	†
&bull;	4			•
&#149;	3	•	•	•
&#8226;	1		•	•
&#133;	870	…	…	…
&hellip;	383			…
&#128;	2	€	▯	€
&euro;	1			€
&frac14;	88			¼
&#8540;	1		⅜	⅜
&#x215c;	1		⅜	⅜
&frac12;	294			½
&#189;	1	½	½	½
&#8541;	1		⅝	⅝
&frac34;	45			¾

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman
&#190;	6	¾	¾	¾
&#8542;	1		⅞	⅞
&sup1;	1			¹
&sup2;	39			²
&#178;	1	²	²	²
&sup3;	4			³
&infin;	3			∞
&#97;	4	a	a	a
&ordf;	3			ª
&aacute;	8003			á
&Aacute;	290			Á
&agrave;	9374			à
&#224;	3	à	à	à
&Agrave;	38			À
&acirc;	3988			â
&Acirc;	10			Â
&auml;	1901639			ä
&#228;	54	ä	ä	ä
&Auml;	33007			Ä
&#196;	2	Ä	Ä	Ä
&#259;	2		ă	ă
&#x103;	1		ă	ă
&#257;	6		ā	ā
&atilde;	30			ã
&Atilde;	67			Ã
&aring;	37			å
&#229;	11	å	å	å
&Aring;	43			Å
&#197;	13	Å	Å	Å
&aelig;	2015			æ
&AElig;	94			Æ
&#98;	2	b	b	b
&#99;	2	c	c	c
&#269;	2		č	č
&ccedil;	1538			ç
&#231;	4	ç	ç	ç
&Ccedil;	66			Ç
&#272;	2		Đ	Đ
&eth;	17			ð

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman❶
&#101;	2	e	e	e
&eacute;	27243			é
&#233;	2	é	é	é
&Eacute;	154			É
&egrave;	5632			è
&#232;	1	è	è	è
&Egrave;	15			È
&#200;	5	È	È	È
&ecirc;	5432			ê
&#234;	6	ê	ê	ê
&Ecirc;	20			Ê
&euml;	1410			ë
&Euml;	2			Ë
&#277;	4		ě	ě
&#275;	5		ē	ē
&#9;	588			❹
&#73;	2	I	I	I
&iacute;	358			í
&Iacute;	3			Í
&igrave;	98			ì
&Igrave;	1			Ì
&icirc;	5274			î
&Icirc;	5			Î
&iuml;	2599			ï
&Iuml;	1			Ï
&#301;	2		ǐ	ǐ
&#299;	2		ī	ī
&#74;	64	J	J	J
&#107;	2	k	k	k
&#108;	2	l	l	l
&#110;	2	n	n	n
&#328;	1		ň	ň
&ntilde;	1091			ñ
&Ntilde;	8			Ñ
	659478			❺
	172			❺
&#111;	6	o	o	o
&oacute;	823			ó
&Oacute;	6			Ó
&ograve;	315			ò
&Ograve;	6			Ò
&ocirc;	5862			ô
&Ocirc;	3			Ô
&ouml;	1179367			ö
&#246;	25	ö	ö	ö
&Ouml;	10353			Ö
&#335;	1		ŏ	ŏ
&#333;	2		ō	ō
&otilde;	19			õ
&Otilde;	17			Õ
&oslash;	23			ø
&Oslash;	25			Ø
&#216;	23	Ø	Ø	Ø
&oelig;	220			œ
&#156;	109	œ	□	œ
&#339;	14		œ	œ
&#338;	2		Œ	Œ
&OElig;	2			Œ
&#80;	2	P	P	P
&#114;	4	r	r	r
&#x159;	6		ř	ř
&#345;	1		ř	ř
&#115;	4	s	s	s
&#x15b;	3		ś	ś
&#154;	402	š	□	š
&#138;	1	Š	□	Š
&#352;	1		Š	Š
&szlig;	1631881			ß
&#223;	39	ß	ß	ß
&szligt;	3	ßt;		ßt;
&#116;	4	t	t	t
&thorn;	48			þ
&#117;	3	u	u	u
&uacute;	131			ú
&Uacute;	4			Ú
&ugrave;	353			ù
&Ugrave;	5			Ù
&ucirc;	1992			û

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman
&#251;	1	û	û	û
&Ucirc;	3			Û
&uuml;	2352481			ü
&#252;	40	ü	ü	ü
&Uuml;	41347			Ü
&#220;	4	Ü	Ü	Ü
&#365;	2		ŭ	ŭ
&#363;	2		ū	ū
&yacute;	9			ý
&Yacute;	2			Ý
&Yuml;	5			Ÿ
&#122;	5	z	z	z
&#142;	2	Ž		Ž
&alpha;	3706			α
&#945;	1247		α	α
&#x3b1;	1		α	α
&Alpha;	249			A
&#913;	101		A	A
&#x391;	2		A	A
&#940;	366		ά	ά
&#x3ac;	8		ά	ά
&#902;	3		Ά	Ά
&beta;	251			β
&#946;	139		β	β
&Beta;	27			B
&#914;	18		B	B
&gamma;	575			γ
&#947;	219		γ	γ
&Gamma;	22			Γ
&#915;	14		Γ	Γ
&delta;	899			δ
&#948;	448		δ	δ
&#x3b4;	3		δ	δ
&Delta;	76			Δ
&#916;	27		Δ	Δ
&epsilon;	3407			ε
&#949;	856		ε	ε
&#x3b5;	3		ε	ε
&Epsilon;	123			E

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman
&#917;	38		E	E
&#x395;	2		E	E
&#941;	271		έ	έ
&#x3ad;	3		έ	έ
&#904;	11		Έ	Έ
&zeta;	83			ζ
&#950;	18		ζ	ζ
&#x3b6;	1		ζ	ζ
&Zeta;	10			Z
&#918;	4		Z	Z
&eta;	1196			η
&#951;	321		η	η
&#x3b7;	1		η	η
&Eta;	43			H
&#919;	9		H	H
&#x397;	1		H	H
&#942;	283		ή	ή
&#x3ae;	4		ή	ή
&#905;	1		Ή	Ή
&theta;	494			θ
&#952;	163		θ	θ
&#x3b8;	1		θ	θ
&Theta;	38			Θ
&#920;	9		Θ	Θ
&#977;	1		ϑ	ϑ
&iota;	3451			ι
&#953;	871		ι	ι
&Iota;	81			I
&#921;	20		I	I
&#x399;	1		I	I
&#943;	698		ί	ί
&#x3af;	2		ί	ί
&#906;	3		Ί	Ί
&kappa;	1178			κ
&#954;	488		κ	κ
&Kappa;	110			K
&#922;	39		K	K
&lambda;	1159			λ
&#955;	527		λ	λ

Erscheinen im Klartext	Anzahl	Charset ISO-8859-1	Unicode	Interpretation von IE8 Eingestellte Schriftart: Times New Roman
&#x3bb;	1		λ	λ
&Lambda;	29			Λ
&#923;	9		Λ	Λ
&mu;	1050			μ
&#956;	367		μ	μ
&Mu;	66			M
&#924;	25		M	M
&#x39c;	1		M	M
&nu;	2816			ν
&#957;	1316		ν	ν
&#x3bd;	4		ν	ν
&Nu;	63			N
&#925;	14		N	N
&xi;	110			ξ
&#958;	75		ξ	ξ
&Xi;	3			Ξ
&#926;	1		Ξ	Ξ
&omicron;	3351			ο
&#959;	1121		ο	ο
&#x3bf;	1		ο	ο
&Omicron;	115			O
&#927;	26		O	O
&#972;	518		ό	ό
&#x3cc;	4		ό	ό
&pi;	1121			π
&#960;	460		π	π
&Pi;	96			Π
&#928;	42		Π	Π
&rho;	1548			ρ
&#961;	730		ρ	ρ
&Rho;	37			P
&#929;	19		P	P
&#x3a1;	1		P	P
&sigma;	1246			σ
&#963;	556		σ	σ
&sigmaf;	1377			ς
&#962;	758		ς	ς
&Sigma;	88			Σ
&#931;	38		Σ	Σ
&#x3a3;	2		Σ	Σ
&tau;	2269			τ
&#964;	1061		τ	τ
&Tau;	96			T
&#932;	26		T	T
&#x3a4;	1		T	T
&upsilon;	1506			υ
&#965;	434		υ	υ
&Upsilon;	29			Y
&#933;	2		Y	Y
&#973;	262		ύ	ύ
&#x3cd;	2		ύ	ύ
&phi;	347			φ
&#966;	201		φ	φ
&Phi;	20			Φ
&#934;	14		Φ	Φ
&chi;	391			χ
&#967;	183		χ	χ
&#x3c7;	1		χ	χ
&Chi;	24			X
&#935;	7		X	X
&psi;	52			ψ
&#968;	20		ψ	ψ
&Psi;	7			Ψ
&#936;	2		Ψ	Ψ
&omega;	1023			ω
&#969;	322		ω	ω
&#x3c9;	2		ω	ω
&Omega;	33			Ω
&#937;	2		Ω	Ω
&#974;	242		ώ	ώ
&#x3ce;	2		ώ	ώ
&#1076;	1		д	д
&#1030;	2		I	I
&#1094;	1		ц	ц
&#1100;	1		ь	ь
&#63743;	1			

Anmerkungen:
❶ soweit in der Schriftart verfügbar, sonst Palatino oder Lucida Sans Unicode, andere Browser können die Zeichen anders interpretieren.
❷'COMBINING VERTICAL LINE ABOVE'
❸Zeichenkette, bestehend aus Anführungszeichen, Absatzmarke und Semikolon
❹'CHARACTER TABULATION', Horizontaler Tabulator
❺'NO-BREAK SPACE'

Tabelle 7.2: Liste der Worttrenner

Dezimal	Hexadezimal	Glyphe	Unicode-Bezeichnung
0	0		NULL
9	9		HORIZONTAL TABULATION
10	A		LINE FEED
11	B		VERTICAL TABULATION
12	C		FORM FEED
13	D		CARRIAGE RETURN
32	20		SPACE
33	21	!	EXCLAMATION MARK
34	22	"	QUOTATION MARK
35	23	#	NUMBER SIGN
37	25	%	PERCENT SIGN
38	26	&	AMPERSAND
39	27	'	APOSTROPHE
40	28	(	LEFT PARENTHESIS
41	29	)	RIGHT PARENTHESIS
42	2A	*	ASTERISK
43	2B	+	PLUS SIGN
44	2C	,	COMMA
45	2D	-	HYPHEN-MINUS
46	2E	.	FULL STOP
47	2F	/	SOLIDUS
58	3A	:	COLON
59	3B	;	SEMICOLON
60	3C	<	LESS-THAN SIGN
61	3D	=	EQUALS SIGN
62	3E	>	GREATER-THAN SIGN
63	3F	?	QUESTION MARK
91	5B	[	LEFT SQUARE BRACKET
92	5C	\	REVERSE SOLIDUS
93	5D	]	RIGHT SQUARE BRACKET
95	5F	_	LOW LINE
96	60	`	GRAVE ACCENT
123	7B	{	LEFT CURLY BRACKET

Dezimal	Hexadezimal	Glyphe	Unicode-Bezeichnung
124	7C	|	VERTICAL LINE
125	7D	}	RIGHT CURLY BRACKET
126	7E	~	TILDE
130	82	‚	SINGLE LOW-9 QUOTATION MARK
132	84	„	DOUBLE LOW-9 QUOTATION MARK
133	85	…	HORIZONTAL ELLIPSIS
134	86	†	DAGGER
139	8B	‹	SINGLE LEFT-POINTING ANGLE QUOTATION MARK
145	91	‘	LEFT SINGLE QUOTATION MARK
146	92	’	RIGHT SINGLE QUOTATION MARK
147	93	“	LEFT DOUBLE QUOTATION MARK
148	94	”	RIGHT DOUBLE QUOTATION MARK
149	95	•	BULLET
150	96	–	EN DASH
151	97	—	EM DASH
155	9B	›	SINGLE RIGHT-POINTING ANGLE QUOTATION MARK
160	A0		NO-BREAK SPACE
161	A1	¡	INVERTED EXCLAMATION MARK
163	A3	£	POUND SIGN
167	A7	§	SECTION SIGN
171	AB	«	LEFT-POINTING DOUBLE ANGLE QUOTATION MARK
175	AF	¯	MACRON
176	B0	°	DEGREE SIGN
178	B2	²	SUPERSCRIPT TWO
180	B4	´	ACUTE ACCENT
183	B7	·	MIDDLE DOT
187	BB	»	RIGHT-POINTING DOUBLE ANGLE QUOTATION MARK
8195	2003		EM SPACE

Anmerkungen zu Tabelle 7.2:

Die Zahlen im grau unterlegten Bereich sind die Codepoints der Windows-Codepage 1252, wie sie in Deutschland standardmäßig eingestellt ist. Die Zeichencodes werden erzeugt durch die Basic-Funktion Chr(). Für alle anderen Zahlen sind Unicode- und Codepage 1252-Codepoints identisch und wahlweise durch Chr() oder ChrW() zu erzeugen. Eine Übersetzungstabelle von cp1252 nach Unicode ist zu finden unter http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP1252.TXT

Das Zeichen für Codepoint U+2003 (EM SPACE) ist in der Codepage 1252 nicht enthalten und kann nur mit Hilfe der Funktion ChrW(8195) oder ChrW(&H2003) erzeugt werden.

Die Liste enthält alle Sonderzeichen, die als Worttrenner betrachtet wurden und die im untersuchten Korpus auch vorkamen. Unicode beschreibt viele weitere

Codepoints, die als Worttrenner betrachtet werden müssen. Diese kamen im Korpus jedoch nicht vor (vgl. die Liste aller Sonderzeichen).

Tabelle 7.3: Absatzbildende HTML-Tags

Name	Description
BLOCKQUOTE	long quotation
BR	forced line break
CAPTION	table caption
CENTER	text-align property
CITE	citation
CODE	computer code fragment
COL	table column
COLGROUP	table column group
DIR	multicolumn directory list
DD	definition description
DIV	division, block-level content
DL	definition list
DT	definition term
FIELDSET	form control group
FRAME	subwindow
FRAMESET	window subdivision
H1 , H2 , H3	heading
H4 , H5 , H6	heading
HR	horizontal rule
IFRAME	inline subwindow
ISINDEX	single line prompt
LI	list item
MAP	client-side image map
MENU	menu list
OL	ordered list
P	paragraph
SAMP	sample program output, scripts, etc.
TABLE	table
TBODY	table body
TD	table data cell
TFOOT	table footer
TH	table header cell
THEAD	table header
TITLE	document title
TR	table row
UL	unordered list

Tabelle 7.4: Schriftformatierende HTML-Tags

Name	**Description**
B	bold text style
BIG	large text style
EM	emphasis
I	italic text style
S	strike-through text style
SMALL	small text style
STRIKE	strike-through text
STRONG	strong emphasis
SUB	subscript
SUP	superscript
TT	teletype or monospaced text style
U	underlined text style

Tabelle 7.5: Beta-Code

Alpha	A	Nu	N
Beta	B	Xi	C
Gamma	G	Omicron	O
Delta	D	Pi	P
Epsilon	E	Rho	R
Zeta	Z	Sigma	S
Eta	H	Tau	T
Theta	Q	Upsilon	U
Iota	I	Phi	F
Kappa	K	Chi	X
Lambda	L	Psi	Y
Mu	M	Omega	W
Smooth breathing	)	Period	.
Rough breathing	(	Apostrophe	,
Acute	/	Colon	:
Grave	\	Question Mark	;
Circumflex	=	Hyphen	-
Iota subscript	|	Dash	_

Uppercase Letters

1. To type uppercase letters, prefix each letter with an asterisk, *. E.g. *P*L*A*T*W*N

Diacritics

1. On lowercase letters these are keyed in the order: (1) letter, (2) breathing, (3) accent, (4) iota subscript. E.g. W(=|

2. On uppercase letters these are keyed in the order: (1) asterisk, (2) breathing, (3) accent, (4) letter, (5) iota subscript. E.g. *(=W|

Die folgenden Tabellen geben eine Übersicht über die im Korpus vorhandenen langen Wörter mit mehr als dreißig Zeichen.

Tabelle 7.6: Auswahl aus der Liste der Wörter, die mehr als 30 und weniger als 40 Zeichen enthalten. (Die vollständige Liste umfasst 118 Wörter)

internationalerVerschwörungspolitik
Apapurincasiquinitschchiquisaquaner
Donaudampfschifffahrtsgesellschaft
Gegenständlichkeitsempfindungen
Himmiherrgottssaggerementsschlamp
Hühneraugenessenzbereitungsversuche
Karamalomilapitschipatschiwatschi
Menschenrasseveredelungsinstitut
Oberappellationsgerichtsvizepräsidenten
Schneebergerschnupftabaksgroßhändler
Sternocleidobronchocricothyrioideus
Wahrscheinlichkeitsberechnungstheorien
allenPrivateigentumsverhältnissen
emporblühendeSchwindelspekulation
menschlichenVervollkommnungsfähigkeit
wieersievordieserlangenZeitbeiihrem

Tabelle 7.7: Vollständige Liste aller Wörter, die vierzig und mehr Zeichen enthalten und keine Zahlwörter sind.

Abecedeeefgehaikaelemenopequeresesthetheuvauweixypsilonzet
Auseinanderplatzungsattentätermordbombeninstrument
Eintrittsbillettenpreisermäßigungkommission
Eintrittsbillettenpreisermäßigungskommission
Erdenlappenlumpenundfetzenreinigungsinstitut
Gegenseitigengeldbeitragendenverhältnismäßigkeiten
Konstantinopolitanischerdudelsackpfeifergesell
Kreuzhimmelbataillongranatenbombenstiefelknecht
Mammonpossessunegalitätsapplanierungsexperimente
Militärtaxkatasterrentensteuergebührenbemessungsämter
Modewarenverlagsniederlagverschleißhändlerin
Oberlandessporteleinzahlungskassenrevidirungsfeldwebel
Oberlandessporteleinzahlungskassenvisitirungsfeldwebel
Reichskammergerichtssupernumerarakzessistbote
Rummeldebummeldefimmeldepippeldehusseldebusseldekimmeldelümmelde
Stadtgerichtsamtswachtmeistersobersubstituten
Taginshauszutragenersparungskunsterfindung
Wawkawampanoosucwinnebayowallazvsagamoresa
Weltauffasseraumwortkindundkunstanschauung
Wortbandwurmstockabtreibmittellehrbuchstempelkostenersatzberechnung

seinBefehlwurdevondreienseinerLeuteausgeführt
θumamimaθumaramlisiaeipurenaieθeeraisieepanamineθunastavhelefu

Tabelle 7.8: (folgende Seite) Ergebnisse für die Suche nach Dampfschiff.
Farbig unterlegt sind falsch positive und falsch negative Resultate. Die Ziffern unter den Codierungen bezeichnen den Rang nach erlaubten Abweichungen (1 oder 2). Werden keine Abweichungen zugelassen (nur Rang 1), dann nimmt die Codierung Konsonantisch die Werte der Codierung KonsonantischPlus an und die Codierungen unterscheiden sich im Ergebnis nicht von einander.

	Häufigkeit	Levenshteindistanz	KonsonantischPlus	Konsonantisch	Bigrammreverse40	Bigramm20	Bigrammreverse30	Trigramm20	Standard20	Standard10	Bigrammreverse20	Bigramm10	Trigramm10	Optimiert10	Optimiert20	Kölner
gefunden			42	48	38	37	35	33	32	34	25	39	39	15	15	181
richtig gefunden			42	44	38	37	35	33	31	31	25	31	31	13	13	31
richtig nicht gefunden			13	9	13	13	13	13	12	10	13	5	5	11	11	3
falsch gefunden			0	4	0	0	0	0	1	3	0	8	8	2	2	162
falsch nicht gefunden			4	2	8	9	11	13	15	15	21	15	15	33	33	15
Korrektklassifikationsrate			0,93	0,90	0,86	0,85	0,81	0,78	0,73	0,69	0,64	0,61	0,61	0,41	0,41	0,16
Dampfschiff	238	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Dampfschiff2	1	1	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffe	96	1	1	1	2	2	2	2	1	1	2	1	1	1	1	1
Dampfschiffs	6	1	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffen	29	2	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffes	24	2	1	1	2	2	2	2	1	1	2	1	1	1	1	1
Kampfschiffe	1	2		2					2	2				2	2	2
Dampfschiffahrt	21	4	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffbüro	1	4	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffchen	1	4	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschwall	1	4										2	2			1
Dampfspeier	2	4														2
Abenddampfschiff	1	5	1	1	2	2	2	2			2			1	1	
Dampfschiebers	1	5								2		1	1			2
Dampfschiffahrts	3	5	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschifffahrt	4	5	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschwaden	4	5										2	2			1
Dampsschiffmann	1	5							2	2						
Luftdampfschiffe	1	5	1	1										1	1	
Dampfkriegsschiff	1	6		2	2	2	2				2					
Dampfschifffahrts	1	6	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschifflinien	1	6	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffstelle	1	6	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffwellen	2	6	1	1	2	2	2	2	1	1	2	1	1			1
Donaudampfschiffe	1	6	1	1	2	2	2				2			1	1	
Großkampfschiffe	1	6		2										2	2	
Rheindampfschiffe	1	6	1	1										1	1	
Staatsdampfschiff	1	6	1	1	2	2	2	2			2			1	1	
Dampfschiffsbrücke	2	7	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffslinien	1	7	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffstation	2	7	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschiffverkehr	1	7	1	1	2	2	2	2	1	1	2	1	1			1
Dampfschnaubend	1	7										2	2			2
Schleppdampfschiff	1	7	1	1	2	2	2	2			2			1	1	
Dampfschienenbahn	1	8								2		1	1			2
Dampfschiffskapitän	2	8	1	1	2	2	2	2	1	1		1	1			1
Dampfschornstein	1	8										2	2			2
Dampfschornsteine	7	8										2	2			2
Kriegsdampfschiffes	1	8	1	1										1	1	
Schleppdampfschiffe	1	8	1	1										1	1	
Dampfschiffsneckerei	1	9	1	1	2	2	2	2	1	1		1	1			1
Dampfschleppboote	1	9										2	2			2
Schiffdampf	1	9		2												
Schiffskampf	1	9		2												
Dampfschiffskompagnie	1	10	1	1	2	2	2	2	1	1		1	1			1
Dampfschiffverbindung	1	10	1	1	2	2	2	2	1	1		1	1			1
Dreimasterdampfschiff	1	10	1	1	1		1				2			1	1	
Schiffskampfe	1	10		2												
Dampfschiffahrtslinien	1	11	1	1	2	2	2	2	1	1		1	1			1
Dampfschiffschornstein	1	11	1	1	2	2	2	2	1	1		1	1			1
dampfschlotgekrönte	1	11										2	2			2
Unterweserdampfschiffe	1	11	1	1										1	1	
Dampfschiffahrtsstation	1	12	1	1	2	2	2	2	1	1		1	1			1
Dampfschiffanlegestelle	1	12	1	1	2	2	2	2	1	1		1	1			1
Dampfschiffsverbindungen	1	13	1	1	2	2	2	2	1	1		1	1			1
Dampfschiffahrtsgesellschaft	3	17	1	1	2	2	2	2	1	1		1	1			1
Dampfschifffahrtsgesellschaft	1	18	1	1	2	2		2	1	1		1	1			1
Donaudampfschiffahrtsgesellschaft	1	22	1	1	2	2										
Donaudampfschifffahrtsgesellschaft	1	23	1	1	2	2										

Register