UNICO-WebRoboter:
Konzept einer spezialisierten Suchmaschine

Dipl.-Inform. Univ. Dirk Nitsche

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

genehmigten Dissertation.

Vorsitzender:	Univ.-Prof. Dr. A. Knoll
Prüfer der Dissertation:	1. Univ.-Prof. R. Bayer, PhD.
	2. Hon.-Prof. Dr. H. von Dewitz, Universität GH Duisburg

Die Dissertation wurde am 15.12.2003 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 22.03.2004 angenommen.

Kurzfassung

Suchmaschinen im WWW spielen heutzutage eine grosse Rolle, um die vorhandene Masse der Informationen besser zugänglich zu machen. Probleme sind aber weiterhin vorhanden: Ist eine gewünschte Information im WWW überhaupt vorhanden? Und wenn ja - wurde sie durch die verwendete Suchmaschine erfasst? Wie aktuell sind die gefundenen Informationen? Grosse Schwierigkeiten bereitet auch hier die Masse der Daten. Wenn ein Suchanfrage mehrere hundert oder tausende Treffer ergibt, welche Dokumente findet man unter den ersten 10-20 Treffern? Welche Kriterien werden angelegt für die Rangordnung der gefundenen Dokumente. Wie interpretiert die Suchmaschine meinen Suchbegriff: "java" als Insel oder als Programmiersprache?

In dieser Arbeit wird anhand einer selbstkonstruierten Suchmaschine demonstriert, wie spezielle Informationen aus dem WWW automatisch gesammelt und halbautomatisch nach vorgegeben Kriterien bewertet werden können. Betrachtet wird dabei die Architektur von "Web-Crawlern" sowie Ranking- und Clustering-Verfahren für Dokumente im Allgemeinen. Als Basis dient die UNICO-Datenbank, die auf unterschiedlichen Wegen Daten zu Universitäten respektive deren Lehrstühlen und Projekten aquiriert und pflegt, um Kooperationen zwischen Universitäten und der Industrie zu fördern.

Entwickelt wurde ein Lernverfahren basierend auf einem neuronalen Netz zur Verknüpfung der heterogenen Informationen von Web-Seiten (Position im Graph und Inhalt), um Dokumente nach ihrem Inhalt besser kategorisieren zu können. Der Ansatz verbessert die Bewertung von Dokumentmengen im WWW gegenüber bekannten Verfahren (wie z.B. K-Means), die zum Teil sehr schlechte Ergebnisse liefern. Die Basis für Clustering-Verfahren bilden Vergleichs-Kriterien für Dokumente, wie z.B. das Skalarprodukt für die vektorbasierte Darstellung von Dokumenten. Auch hierzu wurde ein neuer Ansatz erarbeitet, der versucht mehr Informationen in die Vergleichsoperation zu integrieren.

Die entwickelte Software wird derzeit verwendet, um Daten amerikanischer Universitäten aus dem WWW für die UNICO-Datenbank aufzubereiten.

Kapitel 1 Einleitung

1.1. Motivation und Ziele dieser Arbeit

Die in der vorliegenden Doktorarbeit beschriebenen Verfahren und entwickelten Werkzeuge finden Anwendung innerhalb des University-Industry-Cooperation-Projekts (UNICO), das sich auf Initiative von Professor Hubertus von Dewitz zur Aufgabe gesetzt hat, einen Online-Datenbankdienst ins Leben zu rufen, der es seinen Benutzern auf einfache Art und Weise ermöglicht, schnell an Forschungsdaten und Kontaktadressen von Universitäten und Fachhochschulen zu gelangen. Ziel dieser Arbeit ist es, ein funktionsfähiges System zu implementieren, um aus HTML-Daten der WebServer amerikanischer Universitäten die relevanten Informationen für die UNICO-Datenbank zu gewinnen. Die Einschränkung auf Daten amerikanischer Universitäten ist nur für das Projekt notwendig, prinzipiell lässt sich das entwickelte Werkzeug als autonomer Web-Agent einsetzen, der zuvor festgelegte Bereiche des WWW „absurft“, die erhaltenen Daten selbständig bewertet und somit die Basis für eine inhaltlich spezialisierte Suchmaschine bilden kann. Die entwickelte Software soll in Zukunft die Übernahme von Informationen aus dem WWW vereinfachen und den Inhalt der UNICO-Datenbank qualitativ und quantitativ steigern.

1.2. Überblick über die Kapitel

Kapitel 2 soll einen Überblick zum Thema „allgemeine Suchmaschinen im WWW“ geben. Dabei wird die Entwicklung dieser Hilfsmittel zu den wichtigsten Informationsdiensten im World Wide Web betrachtet. Am Beispiel von Google, das zur erfolgreichsten allgemeinen Suchmaschine des WWW aufgestiegen ist, werden Architektur und Entwicklung der abgedeckten Informationen eingehend analysiert. Kapitel 3 behandelt die Dokumentverarbeitung, vor allem die Themengebiete, die für die Entwicklung des UNICO-WebRoboters von Bedeutung sind. Von zentraler Bedeutung sind hier die Verfahren zur automatischen Zuordnung von Dokumenten in Kategorien: Clustering-Verfahren (3.4). Dargestellt werden klassische Verfahren wie der K-Means-Algorithmus (3.4.2) wie auch eigene Entwicklungen (3.4.5), sowie die Bewertung der Qualität der verwendeten Verfahren (3.4.6). Ebenfalls wichtig für die Vorverarbeitung der Dokumente ist die Bereinigung der Texte durch Stopwortbehandlung (3.2) und Wortstammbildung (3.3). Abschließend werden in diesem Kapitel noch Ranking-Verfahren (3.5) näher untersucht. Dabei wird hauptsächlich der PageRank-Algorithmus analysiert, der wohl maßgeblich zu Google’s Erfolg beigetragen hat. Kapitel 4 behandelt den zentralen Punkt dieser Arbeit: die Entwicklung des UNICO – WebRoboters. Nach einem Überblick über das UNICO – Projekt (4.1 und 4.2) und die verschiedenen Ansätze der Datenaquisition (4.3) wird das Konzept des WebRoboters erläutert (4.3.2). Die Architektur dieses Agenten teilt die Beschreibung in zwei große Teile: das Retrieval (4.3.2.3) und die Bewertung (4.3.2.4). Im ersten Teil werden notwendige Voraussetzungen, wie z.B. die Struktur einer Web-Präsenz, sowie auch Schwierigkeiten bei der Übertragung von Daten aus dem WWW beschrieben, darunter z.B. die Eindeutigkeit von Hyperlinks. Der zweite Abschnitt analysiert die Dokumentenbewertung durch ein hybrides Verfahren. Nach einer Vorverarbeitung werden hierbei klassische Methoden aus der Clustering-Thematik mit dem PageRank-Algorithmus über ein neuronales Netz verknüpft. Um das zu erreichen, ist die Bewertung von Zwischenergebnissen durch einen Experten nötig. Das auf die Art trainierte neuronale Netz bewertet abschließend den gesamten

Textkorpus. Kapitel 5 fasst die Ergebnisse zusammen. Die Daten und Resultate (5.1) spiegeln die Qualität des verwendeten Verfahrens aus Kapitel Kapitel 4 wieder. Auch eine Bewertung des gesamten Projektes (5.2) sowie dessen Perspektiven (5.3) sind hier erfasst. Kapitel 6 beinhaltet eine Referenzliste. Im Anhang (Kapitel 7) befinden sich Konfigurationsdateien der erstellten Software (7.2), ein Beispiel zum XML-Export (7.3) von bewerteten Daten und Auszüge aus dem Java-Quellcode (7.4). Die Erstellung der Software, der Test und die Parametrisierung der genutzten Verfahren stellen den Kern dieser Arbeit dar. Der verfasste Quellcode des UNICO-WebRoboters und der Verfahrenstests alleine umfasst ca. 600 Din A4 – Seiten.

Kapitel 2 Suchmaschinen im World Wide Web

2.1. Motivation / Wachstum des WWW

Aus Abbildung 2-1 kann man ersehen, dass es bei nahezu 40 Mio. Web-Servern (die Zahl der Dokumente ist noch um ein Vielfaches höher!) für den Benutzer schwierig ist, eine gewünschte Information zu finden. Es gibt viele gute Einstiegspunkte in das WWW, die bestimmte Kategorien von Informationen abdecken. Aber selbst dann muss der Anwender all diese Einstiegspunkte kennen und als Folge unter Umständen tausende von Seiten „absurfen“, bevor er die gewünschte Information erhält. Heute ist üblich, dass man die Recherche bei einer Suchmaschine beginnt und die gefundenen Dokumente als Einstiegpunkte verwendet.

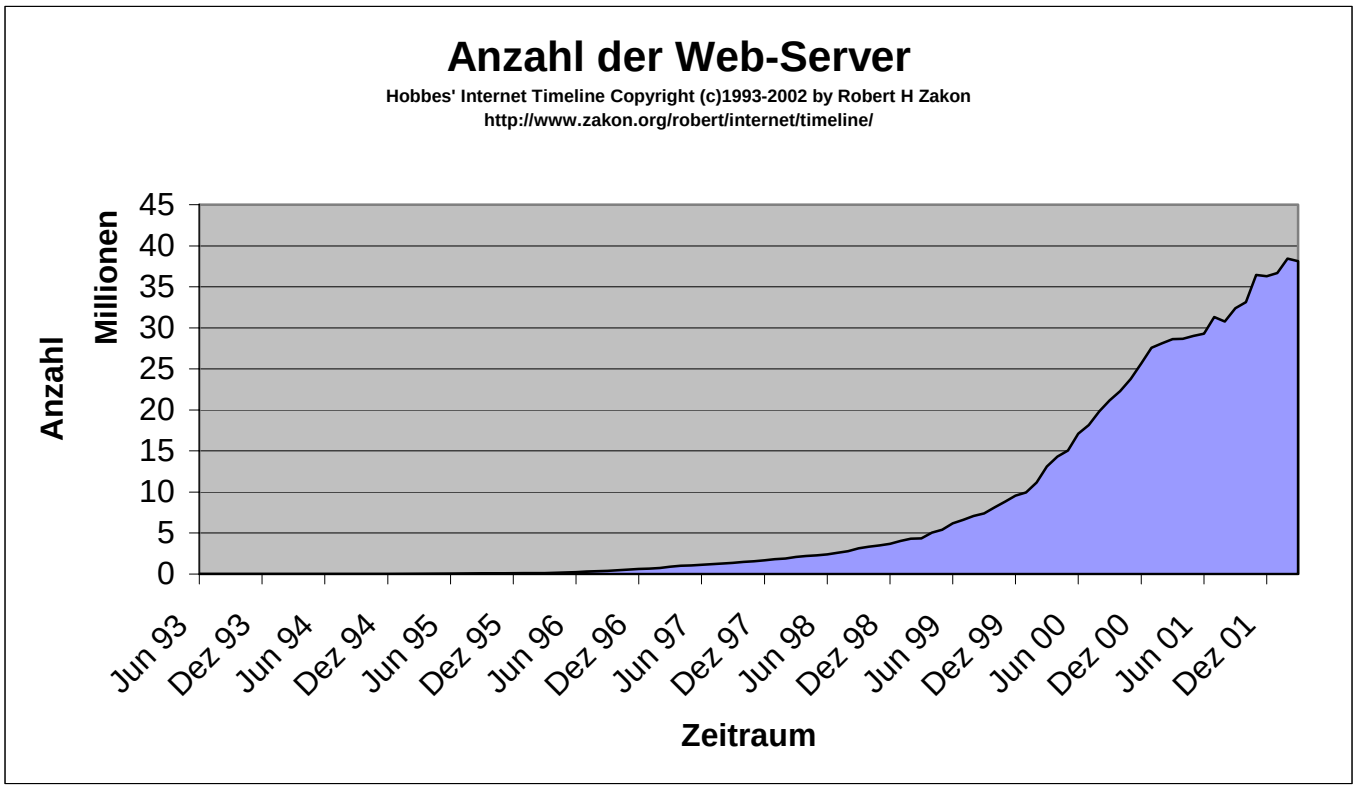

Abbildung 2-1: Anzahl der Server im WWW

2.2. Überblick

Definition: Eine *Suchmaschine* ist ein Programm, dass innerhalb einer Datenmenge zu einer gegebenen Anfrage die „passende“ Teilmenge als Ergebnis liefert. Wie Anfragen formuliert sein müssen und nach welchen Kriterien das Ergebnis generiert wird, ist sehr unterschiedlich. Im Bereich des WWW wird einem Suchbegriff, oftmals einem einzelnen Wort, eine Menge von HTML-Dokumenten als Ergebnis zugeordnet, die diesen Begriff enthalten. Alle enthaltenen Dokumente einer WWW-Suchmaschine werden durch Web-

Roboter eingesammelt. Als WWW-Suchmaschine wird oftmals auch das Eingabeformular für die Suche in einer WWW-Datenbank verstanden, z.B. http://www.google.com

Definition: Ein *WWW-Roboter* ist ein Programm, das sich entlang der Hypertext-Struktur des World Wide Web bewegt und dabei die Dokumente an den besuchten Knoten betrachtet. Es besitzt zudem die Fähigkeit, in diesen Hypertexten referenzierte Dokumente rekursiv weiter zu verfolgen. *WWW-Roboter* werden auch bezeichnet als: *Web-Robot, WebBots*, *Web Wanderer*, *Web Crawler, Web Spider* oder *Web Agent*. Gewöhnliche WebBrowser sind keine Web-Roboter, da sie von einem Menschen bedient werden und nicht automatisch Dokumente laden (Ausnahme: Bilder, Java-Applets in Dokumenten werden in der Regel automatisch geladen). Auch gibt es zusätzliche Software, so genannte „Plug-Ins", die versuchen, während der Anwender ein Dokument betrachtet, vorausschauend mögliche Folgedokumente zu laden und so die Übertragungsdauer zu vermindern. Diese Programme agieren zwar selbständig, sind aber durch das aktuelle Dokument sehr eingeschränkt in ihren Möglichkeiten und sie betrachten in der Regel nur wenige auf dem aktuellen Dokument enthaltene Hyperlinks.

Definition: Ein *WWW-Wurm* ist ein Programm, das sich im WWW verbreitet, indem es sich selbst repliziert. Das ist z.B. möglich durch die Ausnutzung von Sicherheitslöchern in Email-Clients.

Definition: *WWW-Ameisen* (*WWW-Ants*) sind verteilt und kooperativ arbeitende WWW-Roboter.

Definition: Viele Suchmaschinen verwenden ein „*Relevanzkriterium*" (*relevance ranking*), das es ihnen erlaubt, diejenigen Dokumente zuerst anzuzeigen, die der Suchanfrage am Besten entsprechen. Das *Ranking* eines Dokuments wird als Zahl dargestellt. Dieser Wert wird z.B. errechnet durch Zählen der Vorkommen des Suchbegriffs auf dem Dokument, ebenso an welcher Stelle der Begriff steht (eine Platzierung im Titel ist „mehr" wert als eine Platzierung am Ende der Seite) sowie die Größe der Seite. Es gibt viele verschiedene Methoden, um die Relevanz von Dokumenten bezüglich einer Anfrage zu bestimmen (siehe auch 3.5).

Definition: Eine *Suchmaschine* ist ein Programm, das innerhalb einer Datenmenge zu einer gegebenen Anfrage die „passende" Teilmenge als Ergebnis liefert. Wie Anfragen formuliert sein müssen und nach welchen Kriterien das Ergebnis generiert wird, ist sehr unterschiedlich. Im Bereich des WWW wird einem Suchbegriff, meistens einem einzelnen Wort, eine Menge von HTML-Dokumenten als Ergebnis zugeordnet, die diesen Begriff enthalten. Alle enthaltenen Dokumente einer WWW-Suchmaschine werden durch Web-Roboter eingesammelt. Als Web-Suchmaschine wird oftmals auch das Eingabeformular für die Suche in einer WWW-Datenbank verstanden, z.B. http://www.google.com

Es gibt mehrere tausend WWW-Suchmaschinen mit unterschiedlichen Ausprägungen:

- Allgemeine Suchmaschinen: Google, AltaVista, Yahoo, etc. sind Namen bekannter Suchmaschinen, die versuchen, das „komplette" WWW zu indexieren. Aus Abbildung 2-1 kann man ersehen, dass es sich dabei um eine sehr schwierige Aufgabe handelt. Indexierung des gesamten WWW bedeutet hier auch nur, dass es keinerlei thematische Einschränkung bei der Auswahl der zu indexierenden Seiten gibt. Allein die täglichen Veränderungen, der Wegfall und die Neueinbindung von Web-Seiten und -Servern sind derart umfangreich, dass die Kapazitäten der Web-

Roboter und Datenbanken bei Weitem nicht genügen. Hinzu kommt das so genannte „Deep“ Web, bei dem Informationen nicht statisch vom Web-Server abrufbar sind, sondern nur durch Anfrage erhalten werden. Im Gegensatz zu allgemeinen Suchmaschinen werden hier nicht bereits im WWW vorhandene Daten indexiert, die enthaltenen Daten existieren nur in der zugrunde liegenden Datenbank.

- Meta-Suchmaschinen versuchen in der Regel keinerlei Daten selbst zu indexieren, sie geben lediglich eine Anfrage an mehrere „echte“ Suchmaschinen weiter und verbinden die erhaltenen Ergebnisse zu einer einheitlichen Trefferliste. Für den Anwender ist dieser Vorgang zum Teil transparent, zum Teil lassen Meta-Suchmaschinen eine Auswahl von „echten“ Suchmaschinen zu. Vorteilhaft ist natürlich die größere Datenmenge, die recherchiert werden kann. Der Aufwand für die Betreiber einer solchen Suchmaschine liegt darin, dass ständig Anpassungen für Anfrage- und Trefferlistenformat wegen häufiger Veränderung nötig sind, da die Betreiber der „echten“ Suchmaschinen kein Interesse an derart erzeugter Last und „Datenklau“ haben. Auch die Sortierung der Treffer von verschiedenen Suchmaschinen ist schwierig, da die Relevanzkriterien in der Regel differieren.

- Spezialsuchmaschinen: z.B. Suchmaschinen für FTP-Sites, Newsgroups, Multimedia-Inhalte, wissenschaftliche Veröffentlichungen, etc.

Vergleich der 5 großen Suchmaschinen:

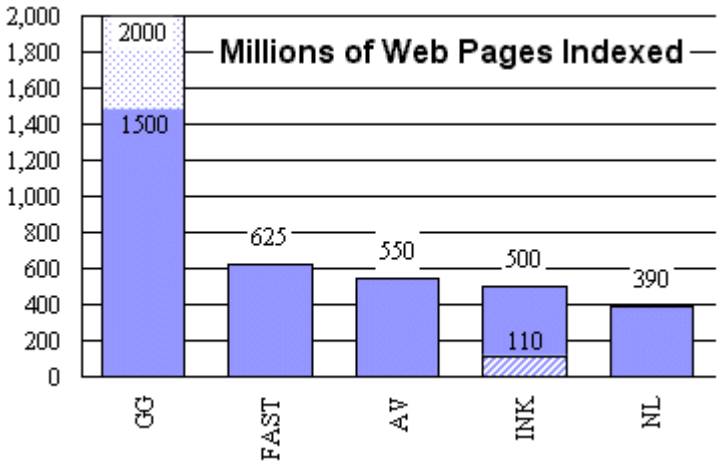

Abbildung 2-2: Indexierte Dokumente der grossen Suchmaschinen - 1

Quelle: (Mai 2002) http://www.searchenginewatch.com/reports/sizes.html

Beschreibung:

GG=Google http://www.google.com
Google ist im Moment die Topadresse für jeden, der etwas im WWW sucht. Es besitzt den größten Index einer Web-Roboter-Basierten Suchmaschine und auch sein Ranking-Algorithmus (s. 3.5.1) hat enorm zu diesem Erfolg beigetragen. Es gibt eine ganze Reihe von Partnern, die diesen Dienst in ihrem Portal zur Verfügung stellen, so unter anderen

Yahoo[1] und Netscape Search[2]. Ebenso erhält der Anwender bei Google die Möglichkeit nach Bildern oder in UseNet - Diskussionen zu suchen.

FAST=FAST Search http://www.alltheweb.com
AllTheWeb.com, (auch bekannt als FAST Search) bietet seit Mai 1999 einen der größten Web-Indexe zur Recherche an. Auch werden Web-Indexe für Multimedia und Mobile/Wireless angeboten. AllTheWeb.com ist eine Demonstration für die Suchmaschinen-Technologie von FAST. Unterschiedliche Portale nutzen diesen Dienst, darunter auch Terra Lycos[3].

AV=AltaVista http://www.altavista.com
AltaVista ist eine der ältesten Web-Roboter-Basierten Suchmaschinen im WWW. Es bietet viele Suchmöglichkeiten, auch die Suche in Newsgroups, „Shopping" Suche und Multimedia Suche. AltaVista startete 1995, zunächst unter Digital's Regie, dann unter Compaq (nach deren Übernahme von Digital 1998) und schließlich wurde es zu einer eigenen Firma, die von CMGI kontrolliert wird.

INK=Inktomi http://www.inktomi.com
Zu Beginn fand man die Suchmaschine an der UC Berkeley (http://inktomi.berkeley.edu). Es bildete sich jedoch schnell eine selbständige Firma mit selbem Namen. Der erstellte Index wurde zunächst für HotBot (http://www.hotbot.com) verwendet, später für verschiedene Partner.

NL=Northern Light http://www.northernlight.com
Start von Northern Light war im August 1997, Mitte 1999 war es Spitzenreiter bei den großen Suchmaschinen. Angeboten werden viele spezialisierte Themengebiete wie z.B. aus dem Business-Bereich.

Google ist ein Sonderfall unter den Suchmaschinen, da dort auch Linkdaten[4], d.h. die Texte, die man anklickt, um einen Hyperlink zu verfolgen, indexiert werden. So können auch Seiten als Treffer zurückgegeben werden, selbst wenn deren Inhalt nicht indexiert wurde. Das funktioniert nur dann, wenn der Suchbegriff in den Linkdaten enthalten ist. Daher steigt die Anzahl der von Google indexierten Seiten von 1,5 Billionen auf somit 2 Billionen Seiten. Zudem werden von Google ebenfalls Microsoft Office Dokumente, PDF-Daten, PS-Daten und andere textbasierte Dokumente erfasst. Nicht eingerechnet sind hingegen die „Google Groups Discussion Posts" (etwa 700 Mio. Einträge) und die ca. 330 Mio. Bilddateien.

[1] http://www.yahoo.com
[2] http://search.netscape.com
[3] http://www.terralycos.com/
[4] wird auch als Anchortext bezeichnet

Entwicklung der Index-Größe:

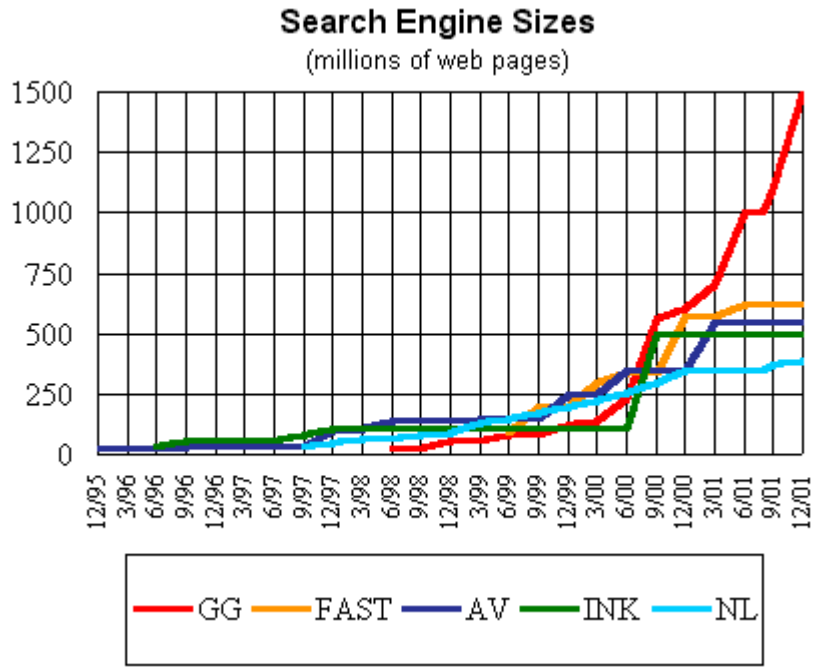

Abbildung 2-3: Indexierte Dokumente der grossen Suchmaschinen – 2

Quelle: (Mai 2002) http://www.searchenginewatch.com/reports/sizes.html

Generell kann man sagen, dass man bei Suchmaschinen mit mehr indexierten Seiten bessere Chancen hat, schwer zugängliche, seltene Informationen zu finden, da sie das WWW besser abdecken. Bei aktuellen Informationen und „Hot-Spots“ des momentanen Interesses werden die Unterschiede gegenüber Suchmaschinen mit kleinerem Index geringer sein.

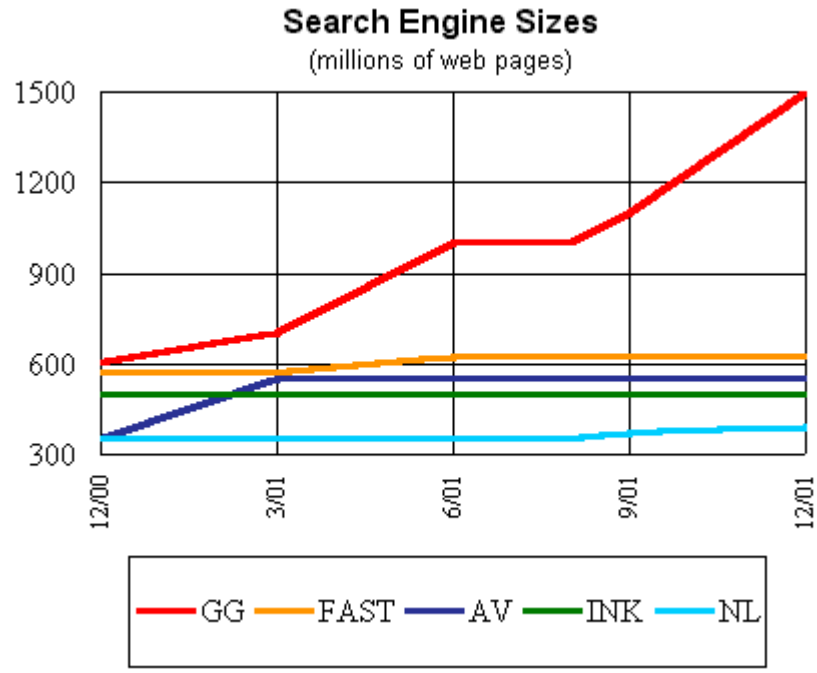

Abbildung 2-4: Indexierte Dokumente der grossen Suchmaschinen – 3

Quelle: (Mai 2002) http://www.searchenginewatch.com/reports/sizes.html

AltaVista war bei ihrem Erscheinen 1995 die Suchmaschine mit dem bei Weitem größten Index. Anfang 1996 hatten die meisten Suchmaschinen unter dem Wettbewerbsdruck bereits gleichgezogen oder hatten AltaVista sogar überboten (Inktomi). Trotz des ernormen Wachstums des WWW gab es bei den Suchmaschinen keinen erheblichen Zuwachs der indexierten Datenmenge im Zeitraum von September 1996 bis September 1997. Bis Ende 1998 lieferten sich AltaVista und Inktomi ein hartes Rennen um den größten Index. Ab 1999 gab es jedoch einen Stillstand bei Inktomi, und am Kampf um die Führung beteiligten sich jetzt AltaVista, Northern Light und FAST Search. Im Januar 2000 erreichte FAST Search die 300 Mio. Seiten Marke und war damit Branchenführer. Bald darauf erreichte auch AltaVista diesen Stand. Im Juni 2000 erreichte Google, seit Mitte 1998 im Rennen, die Rekordmarke von 500 Mio. Seiten, ein Jahr später wurden 1 Billion Seiten von Google's Index erfasst, Ende 2001 sogar 1,5 Billionen Seiten.
Die angegebenen Mengen von indexierten Seiten sind von Betreibern der Suchmaschinen selbst angeben. Es gibt jedoch Möglichkeiten, die Richtigkeit dieser Angaben mittels Testverfahren zu prüfen [1].

2.3. Architektur

Die Anatomie bekannter allgemeiner Suchmaschinen im WWW ist im Groben sehr ähnlich, so gibt es drei große Teilbereiche: Datenübertragung, Indexierung und die Suche. Am Beispiel von Google werden die Details im Folgenden näher erläutert, illustriert durch die Abbildung 2-5. Fast alle Komponenten wurden in C oder C++ implementiert und laufen auf Solaris- oder Linux-Plattformen.
Für die Datenübertragung von Dokumenten des WWW („crawling") sind mehrere Instanzen der „Crawler" zuständig. Die Adressen für die zu holenden Seiten erhalten diese vom „URL Server". Die übertragenen Dokumente werden jetzt durch den „Store Server" in das Repository überführt und dabei komprimiert.
Der „Indexer" holt sich Dokumente aus dem „Repository", entpackt und „parsed"[5] diese Daten. Dabei werden aus jedem HTML-Dokument die enthaltenen Hyperlinks extrahiert und in der „Anchors"-Datei abgelegt. Zusätzlich festgehaltene Informationen zu jedem Link sind sein Ursprungsdokument und das Dokument, auf das er verweist, sowie der Text des Hyperlinks. Der „URL Resolver" prüft und vervollständigt die Hyperlinks aus der „Anchors"-Datei. Jeder so validierte Link erhält einen eindeutigen Bezeichner, die „docID". Der Text des Hyperlinks wird im „Doc Index" mit dem Dokument, auf das er verweist, festgehalten. Nur über diesen Text ist es später bei Suche möglich, auch Seiten als Treffer zu erhalten, deren Daten noch nicht übertragen worden sind. Zusätzlich wird die Ursprung-Ziel-Information in die „Links"-Datenbank eingetragen. Sie bildet die Grundlage für das Bewertungssystem mittels „PageRank" (s. 3.5.1). Außerdem zerlegt der „Indexer" jedes Dokument in so genannte „hits". Darin wird zu jedem im Dokument auftretenden Begriff die Position im Text, ungefähre Größe des Zeichensatzes und die Groß- oder Kleinschreibung festgehalten. Die „hits" werden in den „Barrels" abgelegt. Der „Sorter" aktualisiert den bestehenden Volltextindex, so dass die neuen Dokumente über die enthaltenen Wörter bei der Suche gefunden werden können. Das „Lexicon" bildet jedes gefundene Wort auf einen eindeutigen Bezeichner ab, die so genannte „wordID". In den Barrels werden so die Wörter durch die „wordID" ersetzt und somit Platz gespart. Der „Searcher" wird bei einer Suche durch einen Anwender über einen Web-Server aufgerufen. Dabei werden die Suchbegriffe zunächst durch ihre „wordIDs" ersetzt und die

[5] „to parse" bedeutet wörtlich übersetzt analysieren, hier ist mit „parsen" die Gewinnung von bestimmten Informationen aus Textdokumenten gemeint.

entsprechenden Dokumente, die diese enthalten, über die Indexe der „Barrels“ und des „Doc Index“ herausgesucht. Zusätzlich wird zu jedem gefunden Dokument der zugehörige PageRank-Wert ermittelt. Als Ausgabe bzw. Trefferliste erhält man dann eine nach dem PageRank-Wert sortierte Liste von Dokumenten, welche die Suchbegriffe enthalten. Ausführlichere Informationen zu Google’s Interna findet man in [17].

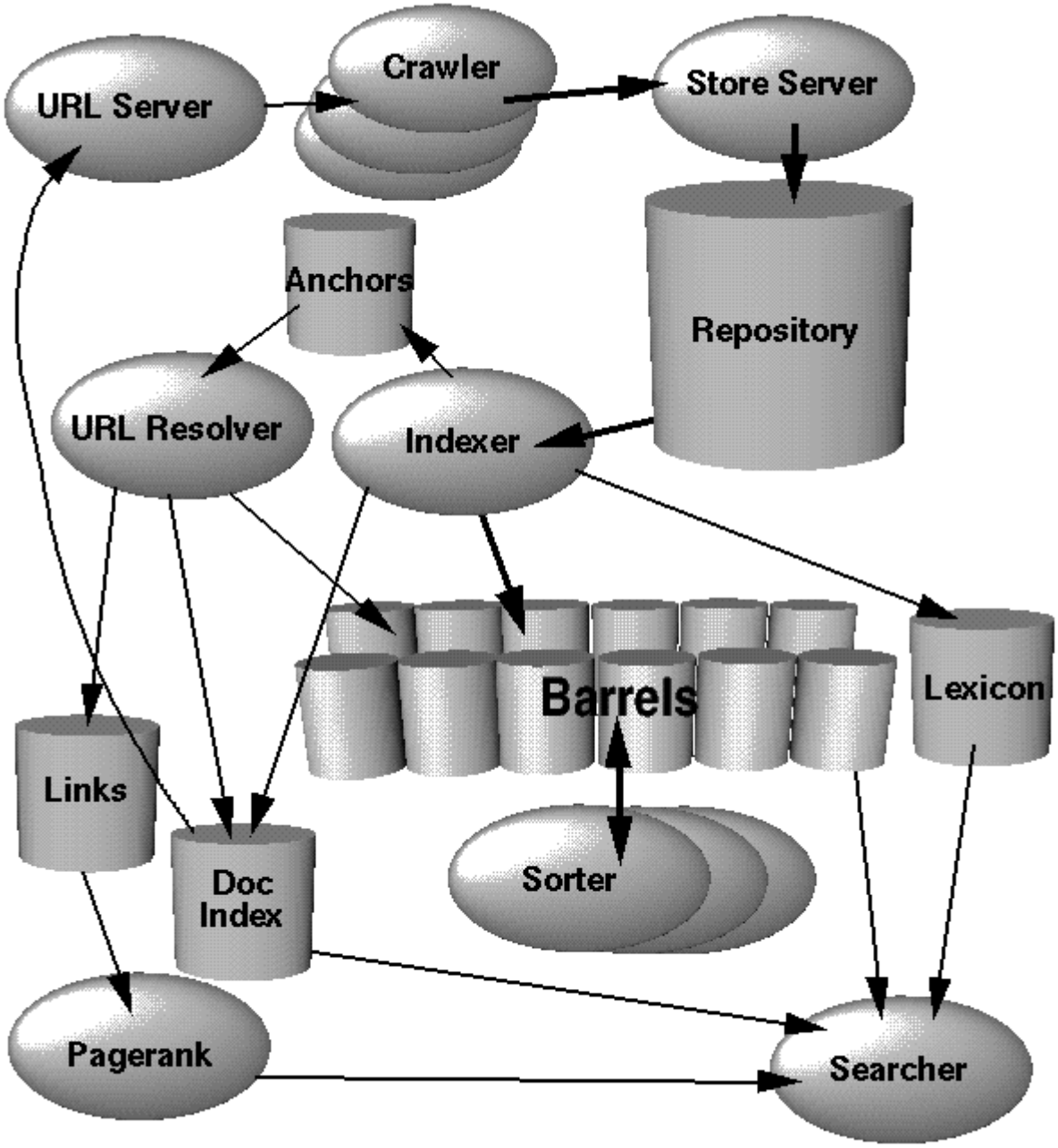

Abbildung 2-5: Architektur von Google

2.4. Einsatz und Performanz

Neben den allgemeinen Suchmaschinen, die versuchen, das gesamte WWW zu indexieren, gibt es auch noch eine Reihe anderer Ansätze. Eine größere Masse an Daten decken z.B. Meta-Suchmaschinen ab, da sie Anfragen einfach an gängige Suchmaschinen weitergeben und deren Trefferlisten zusammenführen und dem Anwender als eigene Trefferliste anbieten. Nachteilig dabei ist das Auftreten von Dubletten, die entweder aufwendig durch die Software der Meta-Suchmaschine entfernt werden müssen oder aber schlichtweg die Qualität des Ergebnisses mindern. Zudem ist ein ordentliches Ranking nicht mehr realisierbar, da jede Trefferliste einer allgemeinen Suchmaschine ein eigenes meist proprietäres Ranking-Verfahren verwendet und es an Kompatibilität zu anderen Bewertungsverfahren mangelt. Je mehr Suchmaschinen unter einer Meta-Suchmaschine vereint sind, umso größer wird auch die Abdeckung des Suchraums sein, als Beispiel kann man http://www.metager.de anführen.
Zudem gibt es eine Reihe von auf bestimmte Themen spezialisierten Suchmaschinen, darunter Musik, Bilder, Nachrichten, Newsgroups, Forschung und Online-Nachforschung (z.B. Suche nach Verwandten oder Vorfahren, Stammbaumbestimmung). Das Funktionsprinzip entspricht dem einer allgemeinen Suchmaschine, lediglich erfolgt eine Einschränkung auf einen bestimmten Bereich des WWW. Das kann durch den Typ der Daten bestimmt sein (z.B. MP3-Dateien) oder durch Restriktion auf vorgegebene Positionen im WWW (z.B. Newsgroups). Weitere Informationen zu spezialisierten Suchmaschinen kann man unter http://www.botspot.com finden.
Die Performanz einer Suchmaschine kann grob an drei Punkten feststellen:

- Antwortzeit bei einer Anfrage
- Aktualität bzw. Geschwindigkeit des „Crawling“ und der Indexierung
- Qualität des Ergebnisses

Die Antwortzeiten bei der Suchanfrage an die bekannten allgemeinen Suchmaschinen sind durchweg sehr gut, die Meta-Suchmaschinen fallen ein bisschen ab, da sie auf die Antworten der benutzten Suchmaschinen warten müssen. Die Aktualität der Daten einer Suchmaschine lässt sich nicht wirklich gut messen, die veröffentlichten Angaben der Betreiber gehen von 1 Monat bis 6 Monate. Dabei werden ausgesuchte „Hot Spots“, wie z.B. Nachrichten öfter aktualisiert als andere Daten des WWW. Schwierig zu evaluieren ist die Qualität des Ergebnisses. Es ist schon ein Problem, aus einer Suchanfrage die Intention des Benutzers zu erkennen. Grundsätzlich kann man feststellen, dass mit zunehmender Spezialisierung die Intention des Anwenders besser getroffen wird, da sie eben durch die Einschränkung auf ein Themengebiet gewissermaßen vorgegeben ist, wie z.B. Musik-Suchmaschinen. Auffällig ist, dass Google’s PageRank-Verfahren dem vom Benutzer gewünschten Ergebnis sehr nahe kommt.

Z.B. liefert Google auf die Anfrage „Gerhard Schröder“ 316.000 Treffer bei einer Antwortzeit von 0.09 Sekunden. Über die Qualität des Ergebnisses, vor allem der ersten 10-20 Treffer muss jeder Anwender selbst urteilen.

Gerhard Schröder
Auf der persönlichen Wahlkampfseite von Bundeskanzler **Gerhard Schröder** (SPD)
finden Sie Informationen über die Stationen in **Gerhard** Schröders Karriere **...**
Beschreibung: Der Bundeskanzler stellt sich, sein Leben, seine Politik und seine Tour vor.

Kategorie: World > Deutsch > ... > Bundestag > Wahl 2002 > SPD
www.gerhard-schroeder.de/ - 11k - Im Archiv - Ähnliche Seiten

Biographie: **Gerhard Schröder**, geb. 1944
1944. **Gerhard Schröder**. Photo: **Gerhard Schröder**, 1990. Politiker. **...** 22. Oktober: **Gerhard Schröder** wird für eine zweite Amtszeit zum Bundeskanzler gewählt. (iz). **...**
www.dhm.de/lemo/html/biografien/SchroederGerhardSPD/ - 20k - 10 Febr. 2003 - Im Archiv - Ähnliche Seiten

Gerhard Schröder Fotoindex
Gerhard Schröder 1991-1998. 1991. 1992. 1993. 1994.
1995. 1996. 1997. 1998. DIE FOTOS. BIOGRAPHIE. INHALT.
www.dhm.de/ausstellungen/spuren_der_macht/ schroeder_index.htm - 5k - Im Archiv - Ähnliche Seiten
[Weitere Resultate von www.dhm.de]

Biografie
&132;Ich habe am eigenen Leibe erfahren, was es bedeutet, sich Chancen erkämpfen zu müssen&147;, sagt **Gerhard Schröder** immer wieder. **...**
www.bundeskanzler.de/Biografie-.7737.htm - 50k - Im Archiv - Ähnliche Seiten

CNN - Germany Votes - **Gerhard Schroeder**: Upstart at the **...** - [Diese Seite übersetzen]
... The Parties. **Gerhard Schroeder**: Upstart at the chancellor's gate.
In this story **...**
www.cnn.com/SPECIALS/1998/09/germany/ candidates/schroeder/ - 36k - Im Archiv - Ähnliche Seiten

BBC NEWS | Europe | Profile: **Gerhard Schroeder** - [Diese Seite übersetzen]
Gerhard Schroeder rose to the top from the very bottom,
via night school and the Young Socialists. **...**
news.bbc.co.uk/2/hi/europe/2242899.stm - 47k - Im Archiv - Ähnliche Seiten

REGIERUNGonline
... Bundeskanzler **Gerhard Schröder** hat am 4. Februar die weiteren Schritte bei der Reform des Arbeitsmarktes und der sozialen Sicherungssysteme erläutert. **...**
Beschreibung: Es wird über die Regierung, ihre Funktion und die verfassungsrechtlichen Grundlagen, die Themen ihrer...
Kategorie: World > Deutsch > ... > Oberste Bundesbehörden
www.bundesregierung.de/ - 101k - 10 Febr. 2003 - Im Archiv - Ähnliche Seiten

Gerhard Schröder - [Diese Seite übersetzen]
... peopleBiography—Noteworthy People—1998 People in the News **Gerhard Schröder**. unseated 16-year chancellor Helmut Kohl to become Germany's next leader. **...**
Beschreibung: (Infoplease.com)
Kategorie: Reference > Encyclopedias > Infoplease.com > Biographies > G
www.infoplease.com/ipa/A0767288.html - 27k - Im Archiv - Ähnliche Seiten

Ohne Titel
http://aktionletzteshemd.de/.
www.aktionletzteshemd.de/ - 1k - Im Archiv - Ähnliche Seiten

SPD.de - spd.de - **Gerhard Schröder**
... Ottmar Schreiner. **Gerhard Schröder**. Martin Schulz. Heide Simonis. **...** Impressum. spd.de > Partei > Köpfe > **Gerhard Schröder**. **Gerhard Schröder**, Parteivorsitzender. **...**
www.spd.de/servlet/PB/menu/1009838/ - 70k - Im Archiv - Ähnliche Seiten

Kapitel 3 Dokumentverarbeitung

3.1. Motivation

Sucht ein Anwender über eine Suchmaschine nach bestimmten Begriffen, so wird in der Regel diese Suche über die Volltextkomponente eines Datenbanksystems realisiert. Ein Volltextindex gibt dabei zu jedem Suchbegriff die Menge von Dokumenten zurück, die den Begriff enthalten. Beispiel für eine invertierte Liste (vereinfachte Darstellung):

Suchbegriff / Term	Dokumente
am	1,2,3,4,5
benutzerkennung	1,3
benutzerkennungen	1,3,4
bereich	2,3,5
betriebsgruppe	1,2,3,5
das	1,3,4,5
der	1,3,5
des	1,2,3
die	1,2,3,4,5
eigenem	2,5
fuer	1,2,4,5
im	3,5
information	2,4,5
informationen	1,2,4
institut	1,2,3,4,5
lan	3,4
lokalen	1,2,4
mit	3,4,5
netzwerk	1,2
netzwerke	2
rechner	3
teilnahme	3,5
test	2,3,5
testbetrieb	1,2,3,4,5
testbetriebsgruppe	1,2,3
testseite	1
testseiten	4
testservice	1,2,3
testzwecke	1
wan	5
zugang	4,5
zum	2,3,4,5
…	…

In der Liste ist zu jedem vorkommenden Begriff eine Liste von Dokumenten angegeben, die diesen Begriff enthält. Zu den Suchbegriffen „teilnahme“ und „information“ würde diese Liste die Dokumente 3 und 5, sowie die Dokumente 2, 4 und 5 zurückliefern. Waren die Begriffe durch ein logisches UND verknüpft, so wird die Schnittmenge aller Dokumente

gebildet, d.h. Ergebnis ist dann nur Dokument 5, da dort beide Suchbegriffe zugleich vertreten sind. Bei einer Verknüpfung durch ein logisches ODER erhält der Anwender die Vereinigungsmenge aller durch die Suchbegriffe gefundenen Dokumente: Ergebnis wäre Dokument 2, 3, 4 und 5.

Bei großen Dokumentsammlungen kann der Umfang einer solchen Liste enorm sein, so finden sich auf den ca. 115.000 Dokumenten der WebPräsenz der Informatik und Mathematik der TU München etwa 850.000 verschiedene Begriffe. Mit steigender Anzahl der Dokumente ist ein absteigendes Wachstum der Menge der Terme zu erwarten, da es endliche Anzahl von Begriffen / Termen gibt. Grundsätzlich ist es jedoch wünschenswert, die Anzahl der suchbaren Terme in der Liste so klein wie möglich zu halten, ohne Information zu verlieren. Je weniger Suchbegriffe in der Liste enthalten sind, desto schneller kann bei einer Anfrage eine Antwort generiert werden.
Verschiedene Möglichkeiten einer geeigneten Reduktion werden in den folgenden Unterpunkten 3.2 und 3.3 näher beschrieben.
Zusätzlich ist es für den Anwender einer Suchmaschine vorteilhaft, wenn die Ergebnisdokumente sortiert sind nach Relevanz, vor allem bei großen Treffermengen. Jedoch ist es ein kaum zu bewältigendes Problem, die Intention des Anwenders nur durch die Betrachtungen der Suchbegriffe zu erahnen. Dazu kommt die Schwierigkeit, der Software einer Suchmaschine „beizubringen“, die Semantik eines Dokuments einem Thema zuzuordnen. Lösungsansätze hierzu sind in den Unterpunkten 3.4 und 3.5 erläutert.

3.2. Bestimmung von Stopworten

3.2.1.Definition

Ein Stopwort ist ein Begriff, der keine eigene inhaltliche Bedeutung bezüglich der Dokumentenmenge besitzt, in der er vorkommt.

3.2.2.Statische Stopwortbestimmung

In Beispielliste aus 3.1 kann man auf den ersten Blick sehen, dass viele Begriffe ohne eigene inhaltliche Bedeutung sind: „am“, „das“, „der“, „des“, „die“, „eigenem“, „fuer“, „im“, „mit“, zum. Es gibt für jede Sprache von Hand generierte Stopwortlisten, die man hier einsetzen kann, um die Liste davon zu bereinigen. Im UNICO – Web-Agenten kommt für die deutsche Sprache eine Liste mit etwa 2.500 Einträgen zum Einsatz:

a	ab	abends	aber	abermals
abhanden	absatzweise	abschnittsweise	abschnittweise	abseits
abwaerts	abzueglich	acht	achte	achtem
achten	achtens	achter	achteraus	achtes
achtmal	achttausendmal	achtzig	achtzigmal	achtzigst
achtzigste	achtzigstem	achtzigsten	achtzigstens	achtzigster
achtzigstes	adj	aequatorwaerts	aerztlicherseits	aeusserst
aeusserstenfalls	all	allabends	allda	alldieweil
alle	alledem	allein	allem	allemal
allen	allenfalls	allenthalben	aller	allerart

allerdings	allerenden	allererst	allerfruehestens	allerhand
allerlei	allerorten	allerorts	allerseits	allerspaetestens
allerwaerts	allerwege	allerwegen	allerwegs	allerweil
alles	allesamt	allewege	alleweil	allezeit
allgemach	allseits	alltags	allueberall	allweil
allzeit	allzu	allzumal	allzusammen	als
alsbald	alsdann	also	am	amtshalber
an	anbei	anbetrachts	andere	anderem
anderen	anderenfalls	anderenorts	anderentags	anderenteils
anderer	andererseits	anderes	anderlei	andernfalls
andernorts	anderntags	andernteils	anderorts	anders
anderseits	andersherum	andersrum	anderswie	anderswo
anderswoher	anderswohin	anderthalbmal	anderwaerts	anderweit
andeutungsweise	andrerseits	aneinander	anerkanntermassen	anfangs
angesichts	angriffsweise	anhand	anhangsweise	anhin
anlaesslich	annaehernd	annaeherungsweise	anno	another
ans	anscheinend	anschliessend	ansonst	ansonsten
anstandshalber	anstandslos	anstatt	anstelle	antwortlich
auch	auf	aufeinander	aufgrund	aufm
aufn	aufs	aufseite	aufseiten	aufwaerts
…				

Durch diese feste Stopwortliste wird der Volltextindex jeder deutschen Dokumentensammlung um maximal 2.500 Einträge reduziert. Bei den 843.000 Begriffen der WebPräsenz der Informatik und Mathematik der TU München erscheint der Gewinn gering, jedoch ist zu beachten, dass Wörter wie „der“, „die“, „das“ in nahezu jedem Dokument vorhanden sind. Daraus folgt, dass jedes dieser Wörter in 115.000 Dokumenten auftritt und das auch in der zweiten Spalte der invertierten Liste stehen würde. Somit ist die Speicherplatzersparnis wesentlich größer als man zunächst annehmen könnte.

3.2.3.Dynamische Stopwortbestimmung

Zusätzlich zu den festen Stopwörtern kann man in jeder Dokumentenmenge Begriffe ausmachen, die speziell für diese Menge keine zusätzliche Information beinhalten. So kommt die Begriffe „institut“ und „testbetrieb“ auf allen 5 Dokumenten im Beispiel aus 3.1 vor. Ebenso gibt es Begriffe, die lediglich auf einem einzigen Dokument auftreten, so z.B. „rechner“ oder „wan“. In beiden Fällen ist durch diese Begriffe kein zusätzlicher Informationsgewinn für den Benutzer zu erwarten, da die Information entweder für alle Dokumente pauschal gilt oder vernachlässigbar ist wegen geringem Vorkommen. Das Beispiel gibt dabei nur schlecht den gewünschten Effekt wieder, da hier ein Dokument 20% der Gesamtmenge entspricht. Bei den 115.000 Dokumenten der WebPräsenz der Informatik und Mathematik der TU München entspricht ein Dokument jedoch nur 0,0009% aller Dokumente. Durch Entfernen dieser Begriffe reduziert sich die zugehörige Wortliste von etwa 843.000 Einträgen um 321.000 auf 522.000 Einträge. Wegen der großen Anzahl war es zu erwarten, das es keinen Begriff gibt, der auf allen Dokumenten vorkommt. Man kann für beide Extreme die Grenzen nach der Anzahl setzen, z.B. Stopwort ist jeder Begriff, der auf mehr als 90% oder auf weniger als 1% aller Dokumente vorkommt. Mit dieser Einstellung

reduziert man die Liste im Beispiel der WebPräsenz der Informatik und Mathematik der TU München etwa auf die Hälfte. Der Verlust an wichtiger Information bleibt dabei sehr gering.

3.3. Methoden der Wortstammbildung

3.3.1.Einführung

Das Auffinden relevanter Informationen in einem Textkorpus[6] wird erschwert unter Anderem durch das Vorhandensein unterschiedlicher morphologischer Varianten eines bestimmten Begriffs. So ist es bei einer Suchanfrage über den gesamten Textkorpus wünschenswert, dass z.B. zum Suchbegriff „Datenbank“ auch Dokumente gefunden werden, die diesen Begriff nicht exakt enthalten, sondern auch Abwandlungen davon wie etwa „Datenbanken“.
Das Finden einer Grundform bzw. des Wortstamms lässt sich erreichen durch Konflation der vorhandenen Wörter.

Definition: Kürzen, Kombinieren oder auch Zusammenfügen von Wörtern (Sequenzen von Buchstaben) und/oder "Wortelementen", um unterschiedliche morphologische Varianten eines Wortes auf eine Grundform zu reduzieren, nennt man *Konflation*.

In der Praxis lassen sich die Wörter eines Textkorpus auf ihren Wortstamm reduzieren, in dem man entweder manuell die Konflationsregeln (reguläre Ausdrücke) anwendet oder das Verfahren mittels Rechnerunterstützung automatisiert. Programme, die zu einem gegebenen Wort den Wortstamm bilden, werden auch „Stemmer“ genannt. Ein positiver Seiteneffekt der Wortstammbildung ist die Reduktion der Wortmengen, da mehrere Begriffe durch ihren Wortstamm ersetzt werden. In Abhängigkeit von der Qualität des „Stemmers“ ist eine Reduktion der Wortmenge auf bis zu 50% möglich. Die Wortstammbildung kann man entweder während der Indexierung des Textkorpus oder während der Suche vornehmen. Ein wichtiger Vorteil des „Stemming“ während der Indexierung ist einerseits die Effizienz des Verfahrens - die Terme müssen dann nicht mehr während der Suche gestemmt werden - und andererseits eine Komprimierung der Index-Datei (was beim „Stemming“ während der Suche entfällt). Der Nachteil ist, dass die Information über den vollständigen Begriff verloren geht, oder man braucht zusätzlichen Speicherplatz, wo sowohl der Begriff selbst als auch der entsprechende Wortstamm gespeichert werden. Unter diesem Aspekt wäre das „Stemming“ während der Suche geeigneter.

Nach der Methode der Konflation können Algorithmen bzw. „Stemmer“ in mehrere Gruppen eingeteilt werden:

1. ***Affix Removal - Algorithmen*** entfernen die Präfixe und/oder die Suffixe
2. ***Successor variety – „Stemmer“*** verwenden als Basis für das „Stemming“ Frequenzen von Buchstabensequenzen im gesamten Textkorpus.
3. ***Wortstammbildung nach der n-Gramm Methode*** – Wörter werden konflatiert aufgrund gemeinsamer n-Gramme.

[6] Eine Sammlung von Dokumenten, Texten

4. ***Table lookup – „Stemmer“*** verwenden eine möglichst vollständige Liste von Wörtern in jedweder Abwandlung und deren Wortstamm.

Es gibt folgende Bewertungskriterien, die bei der Beurteilung der „Stemmer“ beachtet werden müssen:

- **Korrektheit** - also weder „*overstemming“* (wenn zu viel von dem Term entfernt wurde – was dazu führt, dass auch nicht verwandte Begriffe den gleichen Wortstamm haben, weswegen dann bei der Suche auch nichtrelevante Dokumente durchsucht werden und in der Antwortmenge enthalten sind) noch „*understemming“* (wenn zu wenig von dem Term entfernt wurde - führt dazu, dass verwandte Begriffe nicht konflatiert werden, weswegen relevante Dokumente vielleicht nicht durchsucht werden und nicht in die Antwortmenge kommen)

- **Retrieval-Effizienz** - gemessen nach „Precision“ und „Recall“, sowie der Geschwindigkeit des Verfahrens

- **Kompressionsrate** – ist nur bei der Wortstammbildung vor der Indexierung des Textkorpus definiert

3.3.2.Affix Removal – Algorithmen

Dieses Verfahren entfernt nach fest vorgegebenen Regeln Präfixe und Suffixe von Wörtern. Ein einfacher Algorithmus könnte so aussehen:

If a word ends in "ies" but not "eies" or "aies"
then "ies" -> "y"
If a word ends in "es" but not "aes", "ees" or "oes"
then "es" -> "e"
If a word ends in "s", but not "us" or "ss"
then "s" -> NULL

policies	→	policy
dates	→	date
statistics	→	statistic
baroness	→	baroness

Üblicherweise kommen „***iterative longest match“*** Methoden zum Einsatz in der Praxis. Es wird dabei immer die längste mögliche Sequenz entfernt. Das wird solange wiederholt, bis keine Zeichen mehr entfernt werden. Ist der so entstandene Wortstamm nicht korrekt konflatiert, gibt es zwei Verfahren zur Korrektur:

- *RECODING:* Verwendung einer Kontext-sensitive Transformation der Form AxC → AyC (z.B. wenn der entstandene Wortstamm mit "i" endet und auf ein "k" folgt, dann i → y, z.B. skies/sky)

- *PARTIAL MATCHING:* beim Vergleich werden nur n Anfangszeichen des Wortstamms betrachtet; zwei Wortstämme sind dann gleich, wenn die ersten n Zeichen übereinstimmen.

Es gibt mehrere Verfahren/Autoren zur Wortstammbildung nach dieser Methode:

- Salton (1968)
- Lovins (1968)
- Dawson (1974)
- Porter (1980)
- Paice (1990)

Die populärste Methode stammt von Porter und wird ausführlicher behandelt, da sie in der für diese Arbeit entwickelten Software zum Einsatz kommt. Der Algorithmus besteht aus einer Reihe von Regeln, die die Bedingungen und die Aktionen festlegen:

Das **Maß *m*** eines Wortstamms bezeichnet die Zahl seiner alternierenden Vokal-Konsonant-Sequenzen. Zu den Vokalen gehören a, e, i, o, u und y wenn es nach einem Konsonant kommt.

Das Maß *m* ist definiert als:

$$[C](VC)^m[V]$$

(mit **C** für Konsonant und **V** für Vokal)

Beispiel:

m	Beispiele
0	TR, EE, TREE, Y, BY
1	TROUBLE, OATS, TREES, IVY
2	TROUBLES, PRIVATE, OATEN

1. ***<X>** Der Wortstamm endet mit dem gegebenen Buchstaben
2. ***v*** Der Wortstamm enthält einen Vokal.
3. ***d** Der Wortstamm endet mit einem Doppelkonsonant.
4. ***o** Der Wortstamm endet mit einer Sequenz Konsonant-Vokal-Konsonant, wobei der letzte Konsonant w, x oder y ist.

Der folgende Pseudo-Code soll das Verfahren verdeutlichen. Die angegebenen Regeln werden nacheinander abgearbeitet. Sie sind so angeordnet, dass immer das längste Suffix entfernt wird.

```
step1a(word);
step1b(stem);
if (the second or third rule of step 1b was used)
   step1b1(stem);
step1c(stem);
step2(stem);
```

step3(stem);
step4(stem);
step5a(stem);
step5b(stem).

Step 1a Regeln

Bedingung	Suffix	Ersatz	Beispiel
NULL	sses	ss	caresses → caress
NULL	ies	i	ponies → poni
			ties → tie
NULL	ss	ss	carress → carress
NULL	s	NULL	cats → cat

Step 1b Regeln

Bedingung	Suffix	Ersatz	Beispiel
(m>0)	eed	ee	feed → feed agreed → agree
(*v*)	ed	NULL	plastered → plaster bled → bled
(*v*)	ing	NULL	motoring → motor sing → sing

Step 1b1 Regeln

Bedingung	Suffix	Ersatz	Beispiel
NULL	at	ate	conflat(ed) → conflate
NULL	bl	ble	troubl(ing) →trouble
NULL	iz	ize	siz(ed) → size
(*d and not (*<L> or *<S> or *<Z>))	NULL	single letter	hopp(ing) → hop
			tann(ed) → tan
			fall(ing) → fall
			hiss(ing) → hiss
			fizz(ing) → fizz
(m=1 and *o)	NULL	e	fail(ing) → fail
			fil(ing) → file

Step 1c Regeln

Bedingung	Suffix	Ersatz	Beispiel

Bedingung	Suffix	Ersatz	Beispiel
(*v*)	y	i	happy → happi
			sky → sky

Step 2 Regeln

Bedingung	Suffix	Ersatz	Beispiel
(m>0)	ational	ate	relational → relate
(m>0)	tional	tion	conditional → condition
			rational → rational
(m>0)	enci	ence	valenci → valence
(m>0)	anci	ance	hesitanci → hesitance
(m>0)	izer	ize	digitizer → digitize
(m>0)	abli	able	conformabli → conformable
(m>0)	alli	al	radicalli → radical
(m>0)	entli	ent	differentli → different
(m>0)	eli	e	vileli → vile
(m>0)	ousli	ous	analogousli → analogous
(m>0)	ization	ize	vietnamization → vietnamize
(m>0)	ation	ate	predication → predicate
(m>0)	ator	ate	operator → operate
(m>0)	alism	al	feudalism → feudal
(m>0)	iveness	ive	decisiveness → decisive
(m>0)	fulness	ful	hopefulness → hopeful
(m>0)	ousness	ous	callousness → callous
(m>0)	aliti	al	formaliti → formal
(m>0)	iviti	ive	sensitiviti → sensitive
(m>0)	biliti	ble	sensibiliti → sensible

Step 3 Regeln

Bedingung	Suffix	Ersatz	Beispiel
(m>0)	icate	ic	triplicate → triplic
(m>0)	ative	NULL	formative → form
(m>0)	alize	al	formalize → formal
(m>0)	iciti	ic	electriciti → electric
(m>0)	ical	ic	electrical → electric
(m>0)	ful	NULL	hopeful → hope
(m>0)	ness	NULL	goodness → good

Step 4 Regeln

Bedingung	Suffix	Ersatz	Beispiel
(m>1)	al	NULL	revival → reviv
(m>1)	ance	NULL	allowance → allow
(m>1)	ence	NULL	inference → infer
(m>1)	er	NULL	airliner → airlin
(m>1)	ic	NULL	gyroscopic → gyroscop
(m>1)	able	NULL	adjustable → adjust
(m>1)	ible	NULL	defensible → defens
(m>1)	ant	NULL	irritant → irrit
(m>1)	ement	NULL	replacement → replac
(m>1)	ment	NULL	adjustment → adjust
(m>1)	ent	NULL	dependent → depend
(m>1)	ion	NULL	adoption → adopt
(m>1)	ou	NULL	homologou → homolog
(m>1)	ism	NULL	communism → commun
(m>1)	ate	NULL	activate → activ
(m>1)	iti	NULL	angulariti → angular
(m>1)	ous	NULL	homologous → homolog
(m>1)	ive	NULL	effective → effect
(m>1)	ize	NULL	bowdlerize → bowdler

Step 5a Regeln

Bedingung	Suffix	Ersatz	Beispiel
(m>1)	e	NULL	probate → probat
			rate → rate
(m=1 and not *o)	e	NULL	cease → ceas

Step 5b Regeln

Bedingung	Suffix	Ersatz	Beispiel
(m>1 and *d and *<L>)	NULL	single letter	controll → control roll → roll

Die hier angegebenen Regeln gelten für die englische Sprache, es gibt jedoch ebenso für deutsch, französisch, spanisch, holländisch und andere Sprachen Regeln und Praxiserfahrungen. Nachteilig bei diesem Verfahren ist, dass es immer Ausnahmen zu diesen Regeln gibt, so dass ein falscher Wortstamm produziert wird. Beispiele hierfür sind:

organisation	→	organ
policy	→	police
execute	→	executive
arm	→	army
european	→	europe
cylinder	→	cylindrical
create	→	creation
search	→	search

3.3.3.Successor Variety – „Stemmer“

Diese Stemmer basieren auf den Arbeiten aus dem Bereich der strukturalen Linguistik, die versucht, Wort- und Morphemgrenzen aufgrund der Distribution der Phoneme in einem großen Korpus zu determinieren. Die Wortstammbildung verwendet Buchstaben statt Phonemen und den Textkorpus statt der phonologisch transkribierten Ausdrücke.

Definition der Begriffe nach Hafer und Weis:

a	ein Wort der Länge n;
a_i	Präfix von a mit der Länge i;
D	Korpus;
D_{ai}	Teilmenge von D, die diejenigen Terme enthält, dessen ersten i Buchstaben genau dem a_i entsprechen;
S_{ai}	die Zahl der unterschiedlichen Buchstaben, die auf der Stelle i+1 vorkommen (unter Verwendung der Teilmenge D_{ai})

Die Nachfolger-Verschiedenheit (successor variety) eines Strings ist definiert als die Zahl der unterschiedlichen Buchstaben, die diesem String in den Wörtern eines Textkorpus folgen.

Beispiel:

Für folgenden Textkorpus

able, axle, accident, ape, about

ist also die Nachfolger-Verschiedenheit für das Wort "apple":

"**a**" wird im Textkorpus von vier unterschiedlichen Buchstaben gefolgt → **4**
"**ap**" wird im Textkorpus von einem Buchstaben gefolgt → **1** ….. usw.

In einem großen Korpus (ca. 2000 Terme) wird sich anfangs mit jedem Buchstaben, der einem "Teilstring" hinzugefügt wird, die Nachfolger-Verschiedenheit erhöhen bis die Grenze eines Segments erreicht wird; in dem Moment fällt sie wieder stark zurück. Diese Information wird benutzt, um die Wortstämme zu identifizieren.
Aufgrund von so gewonnenen Nachfolger-Verschiedenheiten wird das gegebene Wort segmentiert; hier gibt es wieder mehrere Methoden:

„Cutoff" Methode: Es wird ein Schwellwert für die Nachfolger-Verschiedenheit bestimmt. Bei Überschreiten der Grenze wird der Suffix abgeschnitten. In der Praxis ist es jedoch schwierig, einen geeigneten Schwellwert zu Beginn festzulegen.
„Complete word" Methode: Es werden diejenigen Segmente verwendet, die als vollständiges Wort im Textkorpus vorkommen. Problematisch ist hier, dass eventuell wichtige Wortstämme nicht Element des Textkorpus sein können, z.B. „reader" und „readable" sind enthalten, der Begriff „read" jedoch nicht.
„Peak and plateau" Methode: Wenn in einem Segment die Nachfolger-Verschiedenheit eines Buchstabens größer als die seines Vorgängers und seines Nachfolgers ist, so wird an dieser Stelle das Segment abgeschnitten. Das entspricht der „Cutoff" Methode, wobei hier der passende Schwellwert errechnet werden kann.
Beispiel:

Prefix	**Successor Variety**	**Letters**
R	3	E, I, O
RE	2	A, D
REA	1	D
READ	3	A, I, S
READA	1	B
READAB	1	L
READABL	1	E
READABLE	1	*BLANK*

Hier hat bei dem Prefix „READ" die Nachfolger-Verschiedenheit den Wert 3, Vorgänger und Nachfolger haben jeweils den Wert 1, d.h. „READ" wird als Wortstamm verwendet.

In der Praxis zeigt sich, dass keine der drei Methoden optimale Ergebnisse liefert. Verbesserung bietet die Kombination der „Complete Word" Methode und der „Peak and Plateau" Methode. Zusätzlich haben umfangreiche Tests folgende Regelmässigkeit für die englische Sprache ergeben: Ist ein Segment Präfix von mindestens zwölf Wörtern, so ist es ein Wortstamm. Ist das nicht der Fall, so kann man das zweite Segment als Wortstamm verwenden, da im Englischen mehrfache Präfixe sehr selten sind.

Algorithmus:

if (first segment occurs in ≥ 12 words in corpus)
first segment is stem
else (second segment is stem)

3.3.4.Wortstammbildung nach n-Gramm-Methode

In diesem Verfahren (Adamson, Boreham) werden Paare von Wörtern gebildet, die eine gemeinsame Anzahl von n-Grammen besitzen. Für n = 2 (Digramme) ergibt sich folgendes Beispiel für die Wörter „statistics“ und „statistical“ mit sechs gemeinsamen Digrammen:

statistics → st ta at ti is st ti ic cs
Digramme: s1 = { at, cs, ic, is, st, ta, ti }

statistical → st ta at ti is st ti ic ca al
Digramme: s2 = { al, at, ca, ic, is, st, ta, ti }

s1 ⋂ s2 = { at, ic, is, st, ta, ti }

Das Ähnlichkeitsmaß S zwischen den beiden Begriffen wird berechnet nach der Formel:

$$S = \frac{2C}{A + B}$$

Dabei ist:

A → Zahl der einzelnen Digrammen aus dem ersten Wort;
B → Zahl der einzelnen Digrammen aus dem zweiten Wort;
C → Zahl der gemeinsamen Digrammen.

Aus diesen Daten wird eine Ähnlichkeitsmatrix erstellt:

	$word_1$	$word_2$	$word_3$	...	$word_{n-1}$
$word_1$		S_{12}	S_{13}	$S_{1...}$	$S_{1(n-1)}$
$word_2$	S_{21}		S_{23}	$S_{2...}$	$S_{2(n-1)}$
$word_3$	S_{31}	S_{32}		$S_{3...}$	$S_{3(n-1)}$
...	$S_{...1}$	$S_{...2}$	$S_{...3}$		$S_{...(n-1)}$
$word_n$	S_{n1}	S_{n2}	S_{n3}	$S_{n...}$	$S_{n(n-1)}$

In der Praxis werden die meisten Wörter, wenn man sie paarweise vergleicht, ein Ähnlichkeitsmaß gleich null haben, so dass die Matrix dünn besetzt ist.

3.3.5.Tabellenbasierte Wortstammbildung

Alle Indexterme und ihre Stämme sind in einer Tabelle gespeichert:

Term	**Wortstamm**
engineering engineered	engineer engineer

engineer	engineer

Schwierig ist generell die Erstellung einer solchen Tabelle für eine Sprache, sowie der Pflegeaufwand. Ständige Aktualisierungen sind nötig, da der Sprachschatz generell sich verändert (neue Rechtschreibung, neue Begriffe, Modewörter). Ebenso ist zu beachten, dass es z.B. im wissenschaftlichen Bereich eine Vermischung von deutscher und englischer Sprache gibt. Hinzu kommt, dass allein für die deutsche Sprache der Umfang einer solchen Tabelle mehrere 100.000 Einträge enthalten würde.
Wenn man jedoch eine vollständige Liste als Grundlage für eine Wortstammbildung verwendet, so erhielte man im Gegensatz zu allen anderen Methoden ein optimales Ergebnis.

3.4. Clustering – Verfahren

3.4.1.Vektorquantisierung

3.4.1.1.Definition

Vektorraum: Eine nicht leere Menge V, für deren Elemente Addition und skalare Multiplikation definiert sind, heisst *reeller Vektorraum* oder *linearer Raum*, wenn folgende Bedingungen für $x, y, z \in V$, $\alpha, \beta \in \mathbf{R}$ erfüllt sind:

Kommutativität: $x + y = y + x$
Distributivität: $\alpha (x + y) = \alpha x + \alpha y$, $(\alpha + \beta) x = \alpha x + \beta x$
Assoziativität : $(x + y) + z = x + (y + z)$, $(\alpha ß)x = \alpha (\beta x)$
Nullvektor : $\exists 0 \in V: x + 0 = x, \forall x \in V$
Für $0, 1 \in \mathbf{R}$ gilt: $0x = 0$, $1x = x, \forall x \in V$

Vektor: Alle Elemente eines Vektorraums V heissen *Vektoren*. Ein Beispiel für einen Vektorraum ist der Raum $\mathbf{R}^n$, $n \in \mathbf{N}$. Die Vektoren $x \in \mathbf{R}^n$ sind definiert als $x = (x_1, x_2, x_3, \ldots, x_n)^T$, $x_i \in \mathbf{R}$, $i = 1, 2, 3, \ldots, n$.

Muster: Vektoren als Eingabedaten für Vektorquantisierer werden auch als *Muster* bezeichnet. Sie sind in der Regel als reellwertige Vektoren dargestellt.

Trainingsmenge: Eine endliche Menge von Mustern resp. Vektoren heisst *Trainingsmenge*, wenn ihre Elemente als Eingabe für einen Vektorquantisierer dienen.

Lernen: Als Lernen wird ein Prozess bezeichnet, bei dem ein System selbständig seine Leistung steigern kann. Die Aktualisierung der Kenntnisse des Systems erfolgt durch Auswertung aller bisher erhaltenen Eingabedaten. Es werden zwei Arten von Lernen unterschieden:

Überwachtes Lernen: Lernen von Eingabe-Ausgabe-Relationen, die durch zuvor generierte Trainings-Sets vorgegeben werden.

Unüberwachtes Lernen: Selbständiges Erkennen und Lernen von Zusammenhängen zwischen Eingabedaten und Erzeugen entsprechender Ausgaben.

Eine Methode zur Partitionierung (Deskretisierung, Quantisierung) eines Vektorraumes nennt man *Vektorquantisierung*. Partitionieren bedeutet hierbei, den Vektorraum in Klassen zu zerlegen, d.h. jeder Vektor des Eingaberaumes wird einer dieser Klassen zugeordnet. Da es keine Differenzierung der Vektoren innerhalb der Klassen gibt, entspricht die Partitionierung einer Kodierung des Eingaberaumes mit Informationsverlust. Diese Verfahren sind gut geeignet, relevante Informationen aus verrauschten Daten zu erlangen.

Der Raum wird durch die Quantisierung in die Klassen k = 1, 2, ..., N zerlegt. Jede Klasse k wird durch einen Vektor ω_k repräsentiert. Dieser Vektor wird in der Literatur auch als *Klassenprototyp*, *Repräsentant* oder *Codebuchvektor*[7] bezeichnet.

Die folgende Einteilung der Vektoren in Klassen wird auch als *Voronoipartitionierung* bezeichnet. Hierbei werden die Klassen *Voronoizellen* genannt, die den Eingaberaum in eine Menge $\boldsymbol{V} = \{ V_k \mid k = 1, 2, ... N \}$ zerlegt. Die Zuordnung der Vektoren des Eingaberaumes X erfolgt nach einem festgelegten Abstands- oder Korrelationsmass d. Dabei wird ein Vektor $x \in X$ einer Klasse / Voronoizelle V_k zugeordnet, wenn der Abstand zum Repräsentanten ω_k von V_k minimal bzw. das Korrelationsmass zwischen x und dem Repräsentanten ω_k von V_k maximal ist gegenüber allen anderen Repräsentanten. Ein Beispiel für ein Abstandsmass ist der euklidische Abstand. Hier sind zwei Vektoren um so „ähnlicher", je kleiner der Abstand ist. Im Gegensatz dazu steht das Skalarprodukt zweier Vektoren für ein Korrelationsmass. Hier sind sich zwei Vektoren um so „ähnlicher", je größer der Wert des Skalarproduktes ist.

Die Auswahl des Repräsentanten, dem ein Vektor $x \in X$ zugeordnet werden soll, erfolgt also nach folgender Regel:

Gegeben:

X	Eingaberaum, Menge von Vektoren
$x \in X$	beliebiger Vektor des Eingaberaums X
$\boldsymbol{V} = \{ V_k \mid k = 1, 2, ... N \}$	Partitionierung/Zerlegung von X in N Klassen
$V_i \in \boldsymbol{V}$	Klasse/Voronoizelle der Partitionierung/Zerlegung $\boldsymbol{V}$
ω_i	Repräsentant der Klasse V_i, Vektor der gleichen Dimension wie die Vektoren des Eingaberaums X
d(x,y)	Abstands- bzw. Korrelationsmass für zwei Vektoren x, y

Regel für die Zerlegung des Eingaberaums X in Abhängigkeit vom gewählten Korrelationsmaß d:

1.Minimaler Abstand als beste Korrelation zweier Vektoren:

$$x \in V_j \leftrightarrow j = \min_i \{ d(x, \omega_i) \} \text{ mit } V_j \in \boldsymbol{V}, \forall i \in \{1,2, \ldots, N\}, j \in \{1,2, \ldots, N\},$$

[7] Der Begriff Codebuchvektor entstammt der Kodierungstheorie. Als Codebuch wird hier die Menge aller Repräsentanten (Symbole für die Eingabedaten) bezeichnet.

falls $V_m = \{x \mid d(x, \omega_m) \leq d(x, \omega_n), \forall m \neq n\}$ mit $m,n \in \{1, 2, \ldots, N\}$

Beispiel für d: ***euklidischer Abstand***

bzw.

2.Maximaler Abstand als beste Korrelation zweier Vektoren:

$x \in V_j \leftrightarrow j = \max_i \{ d(x, \omega_i) \}$ mit $V_j \in \boldsymbol{V}$, $\forall i \in \{1,2, \ldots, N\}, j \in \{1,2, \ldots, N\}$,

falls $V_m = \{x \mid d(x, \omega_m) \geq d(x, \omega_n), \forall m \neq n\}$ mit $m,n \in \{1, 2, \ldots, N\}$

Beispiel für d: ***Skalarprodukt***

Zusatz:
Da die max- bzw. min-Funktion hier mehrere Werte als Ergebnis liefern können, kann in einem solchen Fall **zufällig** ein Wert ausgewählt werden. Dieser Fall tritt immer dann ein, wenn z.B. mehrere Vektoren des Eingaberaums genau auf einer bestimmten Trennlinie der Partitionierung liegen. Wenn man mehrere Werte zuläßt, so bedeutet dies ein Mehrfachzuordnung von bestimmten Vektoren zu verschiedenen Klassen. Betrachtet man Vektoren als Repräsentanten von Dokumenten und Klassen als Repräsentanten inhaltlicher Themen, so erhält man eine Mehrfachzuordnung von Dokumenten zu verschiedenen Themengebieten, was durchaus sinnvoll sein kann.

Die Grenzen zweier benachbarter Voronoizellen sind Hyperebenen senkrecht zur Verbindungslinie der Repräsentanten der Zellen. Ein Partitionierung ist dann optimal, wenn alle Repräsentanten dem Schwerpunkt der jeweils ihnen zugeordneten Vektoren entsprechen.

3.4.1.2.Vergleichskriterien für Vektoren

Es gibt viele verschiedene Messmethoden, um die Ähnlichkeit von zwei Vektoren als reelle Zahl auszudrücken. Grundsätzlich entsteht beim Testen mit unterschiedlichen Verfahren der Eindruck, dass die Aussagekraft eines reellen Ähnlichkeitswertes mit zunehmender Größe der Vektordimension abnimmt. Die Ursache ist darin zu sehen, dass beim Vergleich von hochdimensionalen Daten auf einen 1-dimensionalen Wert viel Information verloren geht. Je niedriger die Dimension der Daten, desto geringer ist der Informationsverlust durch diese Reduktion.

Einschränkung:

Grundsätzlich gilt für alle hier behandelten Verfahren, dass jeder Vektor nur Werte ≥ 0 beinhaltet.

3.4.1.2.1.Korrelation / Skalarprodukt

$$\text{corr}(x, y) = \langle x, y \rangle = \sum_{i=1}^{n} x_i y_i$$

Diese Methode beinhaltet für das Ergebnis sowohl eine Abhängigkeit von der Länge wie auch vom eingeschlossenen Winkel. Es gilt: je größer corr(x, y), desto ähnlicher sind x und y. Ist corr(x, y) = 0, so bedeutet dies geringste Ähnlichkeit zwischen x und y.

Beispiel: x = (5, 0, 6)
y = (10, 0, 12)

corr(x, x) = 5*5 + 0*0 + 6 * 6 = 25 + 36 = 61

corr(x, y) = 5*10 + 0*0 + 6*12 = 50 + 72 = 122

Hier wird eine Schwäche dieses Ähnlichkeitsmasses sichtbar: der Vektor z ist x ähnlicher als x sich selbst. Das lässt sich durch Normierung z.B. auf Einheitslänge korrigieren.

3.4.1.2.2.Richtungskosinus

$$\cos(x, y) = \langle x, y \rangle / ||x|| \, ||y||$$

Die Ähnlichkeit von zwei Vektoren wird hier basierend auf dem eingeschlossenen Winkel bestimmt. Werden nur auf Einheitslänge normierte Vektoren verwendet (||x|| = ||y|| = 1), so entspricht der Richtungskosinus der Korrelation.
Der Funktionswert cos(x, y) = 1 kennzeichnet die genaue Übereinstimmung (größte Ähnlichkeit), d.h. y = ax, a$\in$**R**. Falls cos(x, y) = 0, so sind x und y orthogonal (geringste Ähnlichkeit). Wegen der oben genannten Einschränkung kann der Richtungscosinus keine Werte < 0 annehmen.

3.4.1.2.3.Euklidischer Abstand

Hierzu wird ein n-dimensionaler Vektor als Punkt im n-dimensionalen Raum betrachtet. Der euklidische Abstand entspricht für den ein- bis dreidimensionalen Raum dem intuitiven Abstandsbegriff.

$$d(x, y) = || x - y || = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Bei diesem Ähnlichkeitsmass ist eine größere Sensitivität bezüglich der Länge der Vektoren festzustellen im Vergleich zu den vorhergehenden. Das wird durch folgende Abhängigkeit klar: $\| x - y \|^2 = \| x \|^2 + \| y \|^2 - 2\langle x, y \rangle$. Werden die Vektoren zuvor auf Einheitslänge normiert, so liefern darauf basierende Vektorquantisierer vergleichbare Ergebnisse zu den vorangegangenen Methoden:
$\|x\| = \|y\| = 1 \rightarrow$
$\| x - y \|^2 = \| x \|^2 + \| y \|^2 - 2\langle x, y \rangle = 1 + 1 - 2\langle x, y \rangle = \underline{2 (1 - \langle x, y \rangle)}$.

3.4.1.2.4.p-Korrelation

Dieses Mass entspricht dem Skalarprodukt mit dem Unterschied, dass 0-Werte innerhalb der Vektoren gesondert behandelt werden. Sobald einer der beiden Werte 0 ist, der andere Wert jedoch positiv, so ergäbe sich im Skalarprodukt kein Beitrag zum Ergebnis und somit kommt diese Differenz der beiden Vektoren nicht zum Ausdruck. Das kann man durch geeignete Wahl eines zusätzlichen Faktors p korrigieren.

$$\text{corr2}(x, y) = \sum_{i=1}^{n} nm \text{ mit } n = x_i, m = y_i, \text{ falls } x_i>0, y_i>0$$

$$n = p, m = y_i, \text{ falls } x_i = 0, y_i >0$$

$$n = x_i, m = p, \text{ falls } x_i > 0, y_i =0$$

$$n = 0, m = 0, \text{ sonst, wobei } p \in \mathbf{R}$$

Beispiel: x = (5, 0, 6)
y = (5, 10, 6)

corr(x, y) = 5*5 + 0*10 + 6 * 6 = 25 + 36 = 61

corr(x, x) = 5*5 + 0*0 + 6 * 6 = 25 + 36 = 61

Zum Vergleich der euklidische Abstand:

d(x, y) = 10

d(x, x) = 0

Daraus kann man ersehen, dass x und y zueinander genauso ähnlich sind, wie x zu sich selbst, wenn man das Skalarprodukt als Mass verwendet. Verwendet man die p-Korrelation mit p = -1, so ergibt sich:

corr2(x, y) = 5*5 + -1*10 + 6 * 6 = 25 - 10 + 36 = 51

corr2(x, x) = 5*5 + 0*0 + 6 * 6 = 25 + 36 = 61

Hier wird aus dem Ergebnis jetzt ein Unterschied zwischen den Vektoren x und y sichtbar. In der Regel ist für p ein Wert kleiner Null sinnvoll, man kann den negativen Einfluss von p abschwächen, in dem man z.B. p = -0.5 wählt. Wenn p = 0, so entspricht das dem Skalarprodukt. Vergleicht man Vektoren sehr hoher Dimension mit vielen Nullstellen, so kann das Ergebnis stark negativ sein.

3.4.1.2.5.UNICO-Korrelation

Um die Schwierigkeiten bei der Bewertung der p-Korrelation zu vermeiden, wurde die Umsetzung der dort verwendeten Idee verfeinert. Problematisch sind dabei vor allem die stark negativen Werte beim Vergleich von hoch-dimensionalen Vektoren, sowie die geeignete Wahl eines p-Werts. Ursächlich für das Problem der negativen Werte ist das fehlende Wissen über deren Gewichtung. Folgender Ansatz soll positive und negative Werte beim Vektorvergleich in ein besseres Verhältnis setzen.

pa(x,y)	Skalarprodukt von Vektoren x und y, hier positiver Aspekt für ucorr(x,y)
p(x,y)	Anzahl der Summanden von pa(x,y), deren Wert größer 0 ist
na(x,y)	$(\sum_i x_i \mid x_i > 0, y_i = 0) + (\sum_j y_j \mid x_j = 0, y_j > 0)$
n(x,y)	$\lvert\{ x_i \mid x_i>0, y_i=0 \}\rvert + \lvert\{ y_i \mid x_i=0, y_i>0 \}\rvert$
	mit $x_i, x_i \in x$ und $y_i, y_i \in y$

$$ucorr(x,y) = \frac{pa(x,y)}{p(x,y)} - \frac{na(x,y)}{n(x,y)}, \text{ falls } \frac{pa(x,y)}{p(x,y)} > \frac{na(x,y)}{n(x,y)}$$

$$ucorr(x,y) = 0, \text{ sonst}$$

Durch die Faktoren 1/p(x, y) und 1/n(x, y) werden die positiven und negativen Aspekte in Relation zu der Anzahl der jeweils beteiligten Summanden gesetzt.

Beispiel:

Gegeben sind zwei Vektoren x und y mit den Werten:

x = (2, 0, 0, 3, 7, 1)
y = (1, 3, 0, 6, 0, 1)

Basiswert für die Berechnung der UNICO-Korrelation ist das Skalarprodukt, d.h. wenn mind. einer der beiden Faktoren 0 ist, so ist der Beitrag zum Gesamtwert auch gleich 0. Zusätzlich werden die Übereinstimmungen gezählt, d.h. die Anzahl Vorkommen, wenn beide Faktoren größer 0 sind.

Positiver Aspekt pa = 2*1 + 3*6 + 1*1 = 21 bei 3 Übereinstimmungen (p)

Als Gegengewicht dazu werden die Werte der Vektoren aufsummiert, die selbst nicht 0 sind, aber deren Gegenwert des jeweils anderen Vektors 0 sind. Analog wird auch hier die Anzahl dieser Fälle gezählt.

Negativer Aspekt na = 3 + 7 = 10 bei 2 einseitigen 0-Werten (n)

$$ucorr(x,y) = \frac{pa}{p} - \frac{na}{n} = \frac{21}{3} - \frac{10}{2} = 7 - 2 = 5$$

Negative Werte ergeben sich in der Praxis nur sehr selten, sind jedoch möglich. Somit wird zusätzlich der Wert von ucorr nach unten hin auf 0 begrenzt. Diese Massnahme hat keinen Einfluss auf die Qualität der Ergebnisse, jedoch lassen sich die Werte so besser verarbeiten.

3.4.2.K – Means (klassisch)

Der K-Means-Algorithmus ist eines der Standardverfahren zur Bestimmung von Klassenrepräsentanten für eine Menge von Vektoren gleicher Dimension. Nach der Initialisierung werden die Klassenrepräsentanten iterativ verbessert, bis sie mit den Schwerpunkten der von ihnen definierten Voronoizellen zusammenfallen. Es handelt sich um ein Verfahren mit unüberwachtem Lernen. Die Klassenrepräsentanten werden zu Beginn zufällig aus der Menge der Dokumente ausgewählt oder es werden Zufallsvektoren (-dokumente) generiert.

Definition:

t	Zeitkomponente, Iterationsschritt
$\omega_i(t)$	Repräsentant der Klasse i zur Zeit t (nach t Iterationsschritten)
$V_i(t)$	Voronoizelle i zur Zeit t (nach t Iterationsschritten) für i=1, 2, …, k wobei k die zuvor festgelegte Anzahl von Klassen/Themen/Clustern beschreibt

Initialisierung:

t = 0	
$\omega_i(0)$	Zufallsvektor oder zufällig aus der Trainingsmenge ausgewählt
$V_i(0)$	alle Voronoizellen enthalten zu Beginn keine Vektoren

Iteration:

Alle Vektoren der Trainingsmenge werden der Voronoizelle zugeteilt, zu dessen Repräsentanten die verwendete Ähnlichkeitsfunktion den besten Wert liefert.

Für jede Voronoizelle wird aus allen ihr zugeordneten Vektoren der Schwerpunkt berechnet. Der entstehende Vektor ist dann der neue Repräsentant dieser Klasse / Voronoizelle.

$$\omega_i(t) = \frac{1}{\#V_i(t-1)} \sum_{x \in V_i(t-1)} x$$

$\#V_i(t\text{-}1)$ ist der Wert für die Anzahl der Vektoren in $V_i(t\text{-}1)$. Wird der Richtungskosinus als Ähnlichkeitsmass verwendet, kann der Normierungsfaktor $\frac{1}{\#V_i(t-1)}$ entfallen.

Iterationsende:

Sobald sich die Klassenrepräsentanten nicht mehr verändern, ist Konvergenz erreicht. Ein Beweis für die Konvergenz von K-Means-basierten Algorithmen ist in [4] beschrieben.

Bewertung:

Die Praxis zeigt, dass das Ergebnis nicht nur von dem gewählten Ähnlichkeitsmass abhängt, sondern auch stark durch die Wahl der Startrepräsentanten $\omega_i(0)$ bei festem k beeinflusst wird. Die gefundene Lösung ist oftmals unbefriedigend.

3.4.3.K-Means (reverse)

Diese Variante des K-Means-Algorithmus dreht den Ablauf nach dem teile-und-herrsche-Prinzip um: die Menge der Dokumente wird als Startcluster betrachtet. Iterativ wird nun ein bestimmter Cluster ausgewählt (z.B. der Cluster, der die meisten Dokumente enthält) und mit dem klassischen K-Means-Algorithmus in zwei neue Cluster geteilt. Dieser Vorgang wird solange wiederholt bis die gewünschte Anzahl an Clustern entstanden ist. Die Auswahl der zwei Repräsentanten pro Teilungsvorgang erfolgt ebenso wie beim klassischen K-Means-Verfahren: zwei Vektoren werden aus dem zu teilenden Cluster zufällig ausgewählt.

Definition:
Dokumentmenge D
Clustermenge C enthält alle Clusterrepräsentanten und die zugehörigen Dokumente
Cluster $c \in C$: der zu teilende Cluster
Cluster c_1, c_2: neu zu bildende Cluster

Split-Bedingung *b*:

z.B. Minimale Anzahl von Dokumenten in c_1: $|c_1| \geq 0{,}2 * |c|$ oder in c_2: $|c_2| \geq 0{,}2 * |c|$
Maximale Anzahl von Splitversuchen: *s*

Initialisierung:
Alle Dokumente von D werden dem Cluster c zugeteilt

Iteration:
Split:

- Teile *c* in zwei Subcluster c_1, c_2 unter Verwendung des klassischen K-Means-Algorithmus.
- Wiederhole die Teilung solange, bis die Split-Bedingung *b* erfüllt ist.
- Wenn nach *s* Versuchen die Split-Bedingung *b* nicht erfüllt ist, wird der bis dahin beste Versuch genommen.

Entferne *c* aus C

Füge c_1, c_2 in C ein
Auswahl *c* aus C nach unterschiedlichen Kriterien möglich, z.B. größte Anzahl der zugeordneten Dokumente

Iterationsende:
Sobald die gewünschte Anzahl an Cluster in C entstanden ist.

Bewertung:
Dieses Verfahren ist dem klassischen K-Means überlegen. Gründe hierfür sind in der Wiederholung des Splits zu sehen. Auch andere Kriterien für die Wiederholung der Teilung sind erfolgversprechend, z.B. die Qualität der entstandenen Cluster (s. 3.4.6).

3.4.4.WebSOM

3.4.4.1.Allgemeine Eigenschaften

Bei WebSOM werden selbstorganisierende Karten (SOM: self organizing maps) zur Vektorquantisierung verwendet. Diese Methode wurde von Kohonen ([35], [36]) entwickelt und ist ein neuronales Netz, das unüberwacht lernt, d.h. es existiert kein Trainingsset, keine Vorgabe der korrekten Ausgabe. Die selbstorganisierende Karte kann als ein- oder zweidimensionales Feld von Neuronen angesehen werden. Diese Neuronen entsprechen den Klassenrepräsentanten, die sich an die gegebene Menge von Eingangsdaten (Mustern) anpassen und somit den Eingaberaum topographisch in den Ausgaberaum abbilden (kartografieren). Die Karte enthält nach dem Lernvorgang Informationen über die Positionierung der Klassenrepräsentanten im Eingaberaum, so dass die Strukturierung des Eingaberaumes visualisiert werden kann. Die selbstorganisierende Karte repräsentiert eine nicht lineare Projektion eines hoch dimensionalen Eingaberaumes auf einen niedrig dimensionalen Ausgaberaum [37].

3.4.4.2.Algorithmus

Gegeben:

X	-	Eingaberaum, Menge von Vektoren aus R^n
x	-	Vektor aus X
N	-	Anzahl der Neuronen / Zellen
d	-	Dimension der Karte, in der Regel ist d = 1 oder d = 2
A	-	Ausgaberaum, d-dimensionale Karte aus N Neuronen
t	-	diskreter Zeitpunkt, Lernschritt
x(t)	-	für den Lernschritt t ausgewählter Eingabevektor
$w_i(t)$	-	Zustandsvektor des i-ten Neurons zum Zeitpunkt t, entspricht einem Cluster-Repräsentanten

Schritt 1: Initialisieren der Karte

Die Ausgangswerte der Klassenrepräsentanten $w_i(t=0)$ können zufällig gewählt werden. Da sich zufällige Zustandsvektoren ungünstig auf die Entfaltung und Ordnung der Karte auswirken können, ist eine geordnete Initialisierung vorzuziehen (s. hierzu [36]).

Schritt 2: Bestimmung der Gewinnerzelle

Der Eingabevektor x(t) aus dem Eingaberaum X wird mit jedem Klassenrepräsentanten $w_i(t)$ verglichen. Das Neuron, dessen Zustandvektor $w_i(t)$ nach der verwendeten Vektorvergleichsfunktion (s. 3.4.1.2) dem Eingabevektor x(t) am nächsten liegt (am ähnlichsten ist), heisst Gewinnerzelle.

Schritt 3: Anpassen der Karte / Lernschritt

Alle Zellen / Neuronen der Karte „bewegen" sich geometrisch betrachtet in Richtung des Eingabevektors. Die Bewegung der Gewinnerzelle ist dabei am größten, je weiter eine Zelle von Gewinnerzelle auf der Karte entfernt ist, umso geringer ist deren Veränderung.

Lernregel: $w_i(t+1) = w_i(t) + \varepsilon(t)\, h(t+1, i,j)\, (x(t) - w_i(t))$, für alle i aus $\{1, \ldots, N\}$

$\varepsilon(t)$ - Lernrate der Karte:
je größer dieser Wert umso stärker ist die Veränderung der Karte bei jedem Lernschritt

$\sigma(t)$ - Nachbarschaftsradius zum Zeitpunkt t:
Hierdurch wird die Größe des Gebietes um den Gewinner beschrieben, in dem Zellen verändert werden dürfen.

h(t,i,j) - Nachbarschaftsfunktion:
Je weiter eine Zelle von der Gewinnerzelle auf der Karte entfernt ist, umso kleiner ist dieser Wert. Verwendet werden kann:

Diskrete Nachbarschaftsfunktion:

$h(t,i,j) = 1$, wenn $d(i,j) \leq \sigma(t)$ (a)
$h(t,i,j) = 0$, sonst

Exponentialfunktion:

$$h(t,i,j) = \exp\left(-\frac{d(i,j)^2}{2\acute{o}(t)^2}\right) \quad \text{(b)}$$

Im Laufe des Lernvorgangs, also mit zunehmendem t, werden die Lernrate und der Nachbarschaftsradius immer mehr reduziert („Abkühlung" des Lernens). Zu Beginn sind diese Werte jedoch groß und es ergibt sich damit eine schnelle Ausbreitung der Karte über den Eingaberaum. Später werden nur noch geringfügige Anpassungen gemacht. Damit zerfällt der Lernprozess in zwei Phasen: die initiale Ordnung der Karte und die Konvergenzphase. Erfahrungswerte zeigen, dass für die zweite Phase mit kleiner werdender Lernrate ca. die 10 – 100-fache Menge an Lernschritten im Vergleich zur ersten Phase nötig ist, um gute Ergebnisse zu erzielen. Ausführliche Informationen zu selbstorganisierenden Karten sind in [35] und [36] zu finden.

3.4.5.WordCon

3.4.5.1.Allgemeine Eigenschaften

Der hier beschriebene Algorithmus basiert nicht auf einer zuvor festgelegten Dokumentabstandsfunktion (Vergleich von Vektoren s. 3.4.1.2) wie es z.B. beim K-Means-Algorithmus oder beim WebSOM-Algorithmus der Fall ist. Auch ist es möglich, sogar sehr wahrscheinlich, dass Dokumente mehreren Clustern zugeordnet werden können. Die Anzahl der Cluster ist nicht festgelegt, sondern entwickelt sich aus den gegebenen Dokumenten. Dazu erhält jeder Cluster eine ihn beschreibende Wortmenge. Die Qualität der Cluster bezüglich „Precision“ und „Recall“ ist in einem gewissen Rahmen parametrisierbar, d.h. bessere Qualität der Cluster bedeutet eine größere Anzahl von Clustern und durch „Auflockerung“ der Clustervereinigungsregeln erhält man eine geringere Anzahl von Clustern, aber in der Regel auch schlechtere Qualität.

3.4.5.2.Algorithmus

Gegeben:

T	-	Menge von Dokumenten
$\|T\|$	-	Anzahl der Dokumente
t_i	-	Dokument i von T, $1 \leq i \leq \|T\|$
Z	-	Wortmenge aller Dokumente
$\|Z\|$	-	Anzahl aller Wörter von Z
z_j	-	Wort j von Z, $1 \leq j \leq \|Z\|$
h_k	-	Häufigkeit eines Wortes k auf unterschiedlichen Dokumenten, $0 < k \leq \|T\|$
C	-	Menge von Dokumenthaufen (Cluster)
c_l	-	Cluster l von C mit $c_l = (W, D)$, $W = \{ w_1, w_2, w_3, \ldots \in Z \}$, $D = \{ t_1, t_2, t_3, \ldots \in T \}$ mit $W \subseteq Z$ und $D \subseteq T$

Ein Dokument kann in mehreren Clustern vorhanden sein, was dadurch gerechtfertigt ist, dass jeder Cluster am Ende der Berechnung einem inhaltlich festgelegten Thema entspricht und Dokumente durchaus mehrere Themengebiete abdecken können. Die meisten anderen Clusteringverfahren erzwingen jedoch eine Zuordnung zu einem Thema, was man als nachteilig erachten kann.

Schritt 1:

Zunächst werden aus allen Dokumenten alle nicht benötigten Zeichen entfernt, wie z.B. Interpunktion. Die verbleibenden Wörter werden so konvertiert, dass sie keine großen Buchstaben mehr enthalten. Jetzt wird eine Wortliste erstellt und zu jedem Wort wird dabei angegeben, auf wie vielen verschiedenen Dokumenten es vorkommt. Es wird dabei lediglich ein Vorkommen eines Wortes auf einem Dokument festgehalten, selbst wenn es mehrfach darin auftritt. Als Ergebnis erhält man eine Liste von Worten und die dazugehörige Anzahl von Dokumenten (H_j), auf denen es tatsächlich vorkommt $\{(w_j, H_j)\}$.

Schritt 2:

Die Wortliste wird bereinigt durch das Entfernen von Stopwörtern. Dabei wird zum einen eine feste Liste von Stopwörtern verwendet.
Zusätzlich werden Wörter entfernt, deren Häufigkeit h_j einen Schwellwert, überschreitet (z.B. $h_j > n/2$) oder unterschreitet (z.B. $h_j < 2$). Falls $h_j = |T|$, so bedeutet dies, dass das zugehörige Wort auf jedem Dokument der Menge vorkommt und somit in diesem Fall keine zusätzliche Information über ein Dokument beisteuern kann, um es von anderen Dokumenten unterscheiden zu können. Das Gleiche gilt für das andere Extrem: $h_j = 1$ bedeutet, dass ein Wort nur auf einem einzigen Dokument vorkommt und somit keine Vergleichsbasis zu anderen Dokumenten bildet.
Die verbliebene Wortmenge wird durch Reduktion auf den Wortstamm weiter verkleinert (s. 3.3).

Schritt 3:

Bildung der Cluster: zu jedem Eintrag der Wortmenge werden jetzt die zugehörigen Dokumente gesucht und in den Cluster eingetragen. Beispiel: Das Wort „database“ ist enthalten in den Dokumenten 5 und 10, dann ist der zugehörige Cluster
c = { W = {„database“}, D = {5, 10}}. Cluster c wird ein neues Element von C.

Schritt 4:

Zwei Cluster c1 = {W1, D1}, c2 = {W2, D2} mit $c1, c2 \in C$ werden vereinigt, wenn gilt: D1 = D2. Es entsteht der Cluster c3 = { W1 $\cup$ W2, D1}. Cluster c3 wird neues Element von C, c1 und c2 werden entfernt.

Beispiel:
c1 = { W1 = {„database“}, D1 = {5, 10}}
c2 = { W2 = {„sql“}, D2 = {5, 10}}

Hier ist die Vereinigung
c3 = c1 $\cup$ c2 = { W3 = {„database“, „sql“}, D3 = {5, 10}}
erlaubt, da D1 = D2.

Schritt 5:

In Schritt 4 wurden nur Cluster vereinigt, wenn die Mächtigkeit der Dokumentmengen gleich ist und identisch. Jetzt sollen auch Cluster vereinigt werden, bei denen die Dokumentmenge eines Clusters eine echte Teilmenge der Dokumentmenge des anderen Clusters ist:
c1 = {W1, D1}, c2 = {W2, D2} mit $c1, c2 \in C$
Dann ist c3 = c1 $\cup$ c2 = c3 = { W1 $\cup$ W2, D1} genau dann, wenn $D2 \subset D1$.
Beispiel:
c1 = { W1 = {„computer“, „science“}, D1 = {1, 5, 10}}
c2 = { W2 = {„database“ , „sql“}, D2 = {5, 10}}

Hier ist die Vereinigung:

c3 = c1 $\cup$ c2 = { W3 = {„computer“, „science“ , „database“, „sql“}, D3 = {1, 5, 10}}
erlaubt, da D2 $\subset$ D1.

Es ist zu beachten, dass der jeweils „kleinere“ Cluster, also dessen Dokumentmenge die geringere Mächtigkeit besitzt (im Beispiel: c2), mehreren „größeren“ Clustern zugeordnet werden kann.
Beispiel:
c4 = { W4 = {„bayer“}, D4 = {3, 4, 5, 10, 11}}

Dann gibt es zusätzlich folgende Vereinigung:
c5 = c4 $\cup$ c2 = { W5 = {„bayer“ , „database“, „sql“}, D5 = {3, 4, 5, 10, 11}}
da D2 $\subset$ D4.

Sobald alle Vereinigungen durchgeführt sind, werden die neuen Cluster (im Beispiel c3, c5) in C eingefügt, die darin aufgegangen Cluster (im Beispiel c1, c2, c4) aus C gelöscht.

Daraus folgt auch, dass Dokumente im Regelfall mehreren Clustern und damit auch unterschiedlichen Wortmengen zugeordnet werden. So ist z.B. das Dokument 5 aus dem Beispiel sowohl im neu gebildeten Cluster c3 also auch im ebenfalls neuen Cluster c5 vorhanden. Wenn man in der Wortmenge jedes Clusters eine Beschreibung desselben sieht, dann folgt daraus, dass ein Dokument, das mehreren Clustern zugeordnet ist, einfach mehr als ein Thema inhaltlich behandelt.

Schritt 6:

Da bei großen Dokumentmengen die Anzahl der Cluster nach Schritt 5 immer noch zu groß ist, muss man die Vereinigungsbedingung schrittweise abschwächen, bis man die Clustermenge auf ein gewünschtes Niveau reduziert hat.

Neu eingeführt wird ein Vereinigungsfaktor f mit $0 \leq f \leq 1$.
c1 = {W1, D1}, c2 = {W2, D2} mit c1, c2 $\in$ C
O.B.d.A. sei D1 $\supseteq$ D2
Dann ist c3 = c1 $\cup$ c2 = c3 = { W1 $\cup$ W2, D1 $\cup$ D2 } genau dann, wenn $D_s = D2 \cap D1$ und $|D_s| / |D2| \geq f$

Beispiel:
c1 = { W1 = {„computer“, „science“}, D1 = {1, 5, 10, 15}}
c2 = { W2 = {„database“ , „sql“}, D2 = {5, 10, 11}}

$D_s = D2 \cap D1 = \{5, 10\}$, $|D_s| / |D2| = 2 / 3$

Hier ist die Vereinigung:
c3 = c1 $\cup$ c2 = { W3 = {„computer“, „science“ , „database“, „sql“}, D3 = {1, 5, 10, 11, 15}}
nur dann erlaubt, wenn f $\leq$ 2/3 ist. Auch hier sind Mehrfachzuordnungen möglich. Dieser Schritt ist auch mehrfach wiederholbar, da durch neu gebildete Cluster neue Vereinigungsmöglichkeiten entstehen.

Wenn f = 1 gewählt wird, entspricht das Schritt 4 und 5. Falls f = 0, wird ein Cluster entstehen, der alle Wörter und alle Dokumente enthält. Das heisst, man kann die Qualität der Cluster mit diesem Faktor f steuern: je kleiner f, umso größer ist die Wahrscheinlichkeit,

dass Cluster vereinigt werden, die inhaltlich nicht zusammengehören. Andererseits wird mit kleinerem f auch die Anzahl der Cluster geringer.

3.4.6.Qualitätsbeurteilung von Clustering – Verfahren

3.4.6.1.Kriterien für Clusterqualität

3.4.6.1.1.Voraussetzungen

Gegeben:	C_r	– der r-te Cluster
	n_r	– Größe des r-ten Clusters
	$E(C_r)$	– Entropy des r-ten Clusters
	$P(C_r)$	– Purity des r-ten Clusters
	$Rc(C_r, i)$	– Recall für Cluster r zum Thema i
	$Pr(C_r, i)$	– Precision für Cluster r zum Thema i
	$F(C_r, i)$	– F-Mass für Cluster r zum Thema i
	q	– Anzahl der vorhandenen Themen aller Dokumente
	n	– Anzahl aller Dokumente
	n_r^i	– Anzahl der Dokumente des i-ten Themas im r-ten Cluster
	n_r	– Anzahl der Dokumente r-ten Cluster
	n_i	– Anzahl der Dokumente des i-ten Themas

Themen sind dabei zuvor von einem Experten festgelegte Teilmengen von allen betrachteten Dokumenten. In der Regel werden für den Test von Clustering-Verfahren von Hand generierte Test-Sets verwendet, die in Themen eingeteilt sind. Ab hier werden die Begriffe Themen, Klassen und Kategorien synonym verwendet. Cluster hingegen sind von einem Algorithmus automatisch generiert. Die Begriffe sind grundsätzlich orthogonal, idealerweise soll durch ein Clustering-Verfahren jeder Cluster einem Thema entsprechen, also dieselben Dokumente repräsentoeren.

3.4.6.1.2.Entropy (Informationsgehalt)

Die Entropie bezeichnet den Informationsgehalt eines Clusters, d.h. die Verteilung von Themen innerhalb jedes Clusters.

Entropy des Clusters C_r: $$E(C_r) = -\frac{1}{\log q}\sum_{i=1}^{q}\frac{n_r^i}{n_r}\log\frac{n_r^i}{n_r}$$

Die Entropie aller Cluster kann man auch in einem Wert zum Ausdruck bringen:

$$Entropy = \frac{1}{n}\sum_{r=1}^{k} n_r E(C_r)$$

Die perfekte Lösung durch ein Clusteringverfahren wird also ausgedrückt durch Cluster, die jeweils genau ein gegebenes Themengebiet beinhalten. In diesem Fall ist das Ergebnis der

Cluster-Entropien und damit auch der Gesamt-Entropy gleich Null. Im Allgemeinen gilt: je kleiner der Entropie-Wert, umso besser ist die gefundene Lösung.

3.4.6.1.3.Purity (Reinheitsgrad)

Der Reinheitsgrad eines Clusters wird bestimmt durch die größte Anzahl an Dokumenten gleichen Themas im Verhältnis zur Gesamtzahl an Dokumenten, die diesem Cluster zugeordnet sind.

Reinheitsgrad des Cluster C_r = $P(C_r) = \frac{1}{n_r} \max_i (n_r^i)$

Die Reinheit aller Cluster ist analog zur Gesamt-Entropy definiert:

$$Purity = \frac{1}{n} \sum_{r=1}^{k} n_r P(C_r)$$

Im Allgemeinen gilt: je größer der Reinheitsgrad, umso besser ist die gefundene Lösung.

3.4.6.1.4.F-Mass

Das F-Mass [43] beinhaltet Ideen aus dem Information Retrieval: die Trefferquote (Precision) und die Rückgewinnungsquote (Recall) [41][42]. Dazu wird jeder Cluster als das Ergebnis einer Anfrage betrachtet und jedes Themengebiet als das gewünschte Ergebnis zu einer Anfrage. Nach diesen Voraussetzungen werden die Treffer- und die Rückgewinnungsquote für jeden Cluster und jedes Themengebiet berechnet:

Recall (Rückgewinnungsquote): $Rc(C_r, i) = \frac{n_r^i}{n_i}$

Precicion (Trefferquote): $\Pr(C_r, i) = \frac{n_r^i}{n_r}$

Das F-Mass für einen einzelnen Cluster zu einem bestimmten Thema wird wie folgt definiert:

$$F(C_r, i) = \frac{2 * Rc(C_r, i) * \Pr(C_r, i)}{Rc(C_r, i) + \Pr(C_r, i)}$$

Je grösser die die Werte Rc und Pr, umso größer der F-Wert. Für die Bewertung der Gesamtlösung wird der maximale Wert aus allen F-Massen jedes Clusters bestimmt:

$$F = \sum_i \frac{n_i}{n} \max\{F(C_r, i)$$

3.4.7.Test

3.4.7.1.Umgebung und Parameter

Im Rahmen der Arbeit sind hauptsächlich die K-Means Algorithmen getestet worden, vornehmlich die verwendeten Vergleichskriterien (s. 3.4.1.2) Skalarprodukt und UNICO-Korrelation.

Verwendete Hardware: Dual Pentium 3 1GHz, 1GB RAM

Testsets:
Spiegel Online A: 50 Dokumente von http://www.spiegel.de aus den Kategorien Auto, Kultur, Politik, Sport, Wissenschaft. Keine Dubletten, exakt 10 Dokumente pro Kategorie.
Spiegel Online B: 1000 Dokumente von http://www.spiegel.de aus den Kategorien Auto, Kultur, Netzwelt, Panorama, Politik, Reise, Sport, Unispiegel, Wirtschaft, Wissenschaft. Keine Dubletten, exakt 100 Dokumente pro Kategorie.

20 Newsgroups: 20000 Dokumente aus folgenden 20 Newsgroups (Quelle: s. [44] und [45]). Dubletten sind vorhanden, da einige Dokumente in verschiedene Newsgroups „gepostet“ worden sind. 1000 Dokumente pro Kategorie.

Alle Dokumente sind vor der Verarbeitung durch die Clustering-Verfahren frei von Stopwörtern, Interpunktion und sonstigen überflüssigen Zeichen. Zusätzlich sind alle verbleibenden Begriffe auf ihre Stammform reduziert (s. 3.3), kleingeschrieben und die Häufigkeit ihres Auftretens im Text festgehalten.

Beispiel:

Dokument 21, Politk aus Spiegel Online B

Hohe Haftstrafen für Vergewaltiger im Bosnienkrieg

Drei bosnische Serben hat das Uno-Kriegsverbrechertribunal in Den Haag zu hohen Haftstrafen verurteilt. Sie wurden in der Berufungsverhandlung erneut für schuldig befunden, muslimische Frauen vergewaltigt zu haben …

Dokument 21 als Vektor nach der Vorverarbeitung:

Begriff	**Häufigkeit**
uno-kriegsverbrechertribunal	1
wurd	1
verurteilt	1
erneut	1
hoh	2
serb	1
haftstraf	2

haag	1
berufungsverhandl	1
vergewaltigt	1
schuldig	1
bosnisch	1
bosnienkrieg	1
befund	1
vergewalt	1
muslim	1
frau	1
…	…

Um die Bearbeitung so effizient wie möglich zu gestalten wurden zusätzlich alle Begriffe entfernt, die auf weniger als 1% der Dokumente oder die auf nahezu jedem Dokument vorkommen. Das Ergebnis ist hiervon nur geringfügig beeinflusst. Da das K-Means-Verfahren keine reproduzierbaren Ergebnisse liefert, wurden mehrere Testläufe durchgeführt und deren Durchschnittswert als Ergebnis angegeben.

3.4.7.2.Ergebnisse

Der Parameter **k** bezeichnet Anzahl von Clustern, die von K-Means zu erstellen waren. In der letzten Spalte ist die durchschnittliche Laufzeit eines Clustering-Laufs festgehalten.

Testset	**k=5**	**k=10**	**k=15**	**k=20**	**k=50**	**k=100**	**Dauer**
Spiegel Online A (Skalarprodukt)	56,0	59,4	60,8	70,8	-	-	~ 30 s
Spiegel Online A (UNICO-Korrelation)	64,4	66,0	67,6	71,4	-	-	~ 30 s
Spiegel Online B (Skalarprodukt)	-	40,8	44,8	45,54	46,79	-	~ 4 min
Spiegel Online B (UNICO-Korrelation)	-	41,0	46,54	48,86	47,32	-	~ 4 min
20 Newsgroups (Skalarprodukt)	-	-	-	37,72	41,01	45,50	~ 1 Std.
20 Newsgroups (UNICO-Korrelation)	-	-	-	39,41	44,72	49,69	~ 1 Std.

In der Spalte **Dauer** ist die durchschnittliche Laufzeit eines Clustering-Laufs angegeben. Die erarbeiteten Zahlen innerhalb der Tabelle sind die Reinheitsgrade (s. 3.4.6.1.3) der jeweiligen Gesamtlösung in Prozent. Je höher also dieser Wert, umso besser ist das zugehörige Resultat bewertet. Der Reinheitsgrad wird hier als Maßstab verwendet, da er am Besten einem intuitiven Qualitätmaß entspricht.

3.4.7.3.Beurteilung

Anhand der Tabelle aus 3.4.7.2 kann man feststellen, dass die UNICO-Korrelation gegenüber dem einfachen Skalarprodukt zu einer leichten Verbesserung im Gesamtergebnis führt. Auch muss bemerkt werden, dass die Qualität der erzielten Lösung abnimmt, je größer die Anzahl der Dokumente ist. Je kleiner der Textkorpus, umso größer ist der Vorteil der UNICO-Korrelation im Vergleich zum Skalarprodukt.
Die Clustering-Methode „WordCon“ (s. 3.4.5) ist für größere Dokumentmengen nicht geeignet aufgrund des hohen Speicherplatzbedarfs. So ist zu jedem Wort, das mindestens zwei Mal im Textkorpus auftritt, eine Liste von Dokumentnummern anzufügen, die den jeweiligen Begriff enthalten. Diese Kombination sind die Basis-Cluster für die folgende Berechnung. Im schlechtesten Fall erhält man für n Begriffe auf m Dokumenten eine Matrix der Dimension n*m. Um die Clusteranzahl zu verringern (s. Algorithmus 3.4.5.2), sind wiederum im schlechtesten Fall $(n*m)^2$ Vergleiche nötig. Einsatzmöglichkeiten für dieses Verfahren sind neben kleineren Dokumentmengen auch die Bestimmung von geeigneten Cluster-Repräsentanten z.B. für K-Means. So liefert z.B. ein Testlauf des WordCon-Verfahrens mit dem Testset „Spiegel Online A“ folgende Start-Cluster (Auszug): Zu Beachten sind die Mehrfachzuordnungen von Dokumenten zu einer Menge von Begriffen.

Begriffe	Dokumentnummern
dortmund, leverkus	32, 36, 31
zusammenfass, mitgeteilt, ubertrag	34, 46, 27
effenberg, fc, spanisch, madrid	41, 30, 33
dialog, menschenrechtssituation, kanzl, menschenrecht	15, 23, 22
amt, zweck, entstand	4, 23, 20
manag, matthias, zerfress, klaus	31, 29, 44
ministerium, uberpruf	23, 25, 27
dat, mitgeteilt, forscherteam, ergebniss	40, 46, 27
orch, gebet, verstand, komponist	15, 18, 37
eur, hubraum, lehn, saub, fassungsvermog, kombi, ccm, typ, airbag, ec, offerraumvolum, nm	2, 8, 18
gebet, arg, jenufa, staatsop, vitalitat, carlos, freiheit, ablauf, sturm	15, 14, 37
luft, illegal, amerika, martin	18, 14, 39
bestreit, kad, turni, bevorsteh, argentini, wm-endrund	36, 37, 39
sex, sexuell, besitz, sicht	32, 10, 49
illustr, crow, russell, auffall, trio, vergleichbar, stehend, schauspiel	45, 13, 14
toll, fand, begann, schult, nero, streit	18, 7, 44
arm, trug, schutz, dau	28, 18, 48
verhind, getotet	23, 3, 7
identifizi, geschlecht, mannch, gefund, bann, weibch	41, 49, 44
horizont, abschuss, licht, planet, himmel	45, 15, 43
national, zweifel, beweis	47, 28, 44

streck, firma, ergebnis	29, 24, 44
beginn, entschuld, schneid	26, 35, 37
gekleidet, entscharf, versucht, trug, jon, schutz, trag	43, 18, 48
umstand, gescheh, offnet, sozial, erbracht	26, 24, 10
frankreich	9, 37, 39
flucht, erkenntnis, schlicht, theurich, eindeut, charakt	15, 3, 10
werd, mannschaft, entschuld, fordert, gefehlt	30, 35, 37
umstand, offnet, spur	18, 24, 10
karlsruh, pruf, gegeb, unzulass, scheitert, klag	21, 25, 27
op, kommentiert	17, 15, 39
experiment, nachricht, dust	17, 47, 15
optimist, abhang, schatz	15, 9, 48
beobachtet, aufzeichn, big	19, 40, 18
anfangsverdacht, zulass, verdacht, einstig, falsch, eingestellt	28, 20, 27
fuhrung, definitiv	38, 28, 39
schlecht, geschaftsfuhr, bilanz	13, 48, 29
elektron	17, 5, 27
esp, stossfang, ausbau, serienausstatt, technisch	8, 37, 5

Heute gängige Clustering-Verfahren liefern nur mittelmäßige Ergebnisse, da aufgrund mangelnden Wissens über die Bedeutung von Worten, vor allem in einem bestimmten Kontext, eine gute Klassifikation nicht möglich ist. Auch statistische Verfahren wie die Bayes'sche Klassifikation liefern keine guten Lösungen, da es auch für Menschen schwierig ist, die Bedeutung von Begriffen in Zahlen auszudrücken. Zudem ist der Aufwand bei diesen Methoden ungleich höher.

3.4.8.Visualisierung

Ein Clustering-Verfahren teilt eine Menge von Dokumenten in bestimmte Klassen ein. Auch diese Klassen stehen zueinander in Beziehung, da es sich um für den jeweiligen Cluster repräsentative Dokumente (ausgedrückt durch einen Wort-Vektor) handelt. Daher kann man anhand einer Abstandsmatrix zu einer gegebenen Klasse z.B. seine nächsten Nachbarn bestimmen. So ist zum Beispiel bei den Themengebieten Mathematik, Physik und Informatik zu erwarten, dass die Überschneidungen durch gemeinsame Begriffe die „Mathematik-Klasse" einen geringeren Abstand zur „Physik-Klasse" besitzt als zur „Informatik-Klasse". Um das Ergebnis eines Clustering-Prozesses graphisch darzustellen, ist es nötig, mittels der Abstände der Klassen eine 2-dimensionale Karte zu erzeugen. In [43] wird ein Algorithmus vorgestellt, der diese Abbildung erzeugt.
Ein ähnliches Problem ist die graphische Darstellung aller Dokumente resp. Ihrer Wort-Vektoren. Hier dient die Visualisierung vor allem zur Erkennung von Clustern. Ausführliche Informationen zur Visualisierung sind z.B. in [36 / S. 117] zu finden.

3.5. Ranking

3.5.1.PageRank

Unter diesem Namen ist ein Bewertungsalgorithmus bekannt geworden, der 1998 für die Internetsuchmaschine Google zum Einsatz gekommen ist und wesentlich zu dessen Erfolg beigetragen hat. Dabei werden Dokumente des WWW bewertet, ohne dass deren Inhalt betrachtet wird. Lediglich die Verbindungen der Dokumente durch Hyperlinks bilden die Grundlage für den Algorithmus. Motiviert wird dieser Ansatz vor allem durch die Effizienz des Verfahrens und einer enormen Datenmenge von mehreren 10 Mio. Dokumenten. Die meisten inhaltsabhängigen Bewertungsmethoden sind ungleich aufwendiger, da jedes Dokument im Allgemeinen mehr Wörter enthält als Hyperlinks. Viele dieser Verfahren liefern gute und sogar effiziente Lösungen für Daten im Bereich von bis zu mehreren 10.000 Dokumenten. Übersteigt die Menge an Daten jedoch diese Marke, verschlechtert sich die Performance drastisch. Die Frage ist, ob also der PageRank – Algorithmus außer durch Geschwindigkeit auch durch die Qualität der Ergebnisse überzeugen kann. Google's Erfolg scheint das zu bestätigen, im Folgenden soll der Algorithmus näher beschrieben werden. Wichtig ist, dass im Gegensatz zu vielen anderen Bewertungsverfahren hier versucht wird, den gesamten Textkorpus, also alle vorhandene Dokumente, zu bewerten. Der errechnete Wert für jedes Dokument ist damit also nicht abhängig von einer Anfrage.

Das Gewicht (PageRank) R eines Dokuments p wird berechnet durch:

$$R(p) = c * \sum_{q:q \to p} \frac{R(q)}{N_q} \qquad \text{(PR)}$$

Ein Dokument q gibt einen Teil seines Gewichtes an alle Dokumente weiter, auf die es verweist

N_q ist Anzahl der Hyperlinks, die von einem Dokument q auf andere Dokumente verweisen

c ist eine Normalisierungskonstante, sie ist so gewählt, dass die Summe der Gewichte aller Dokumente gleich 1 ist

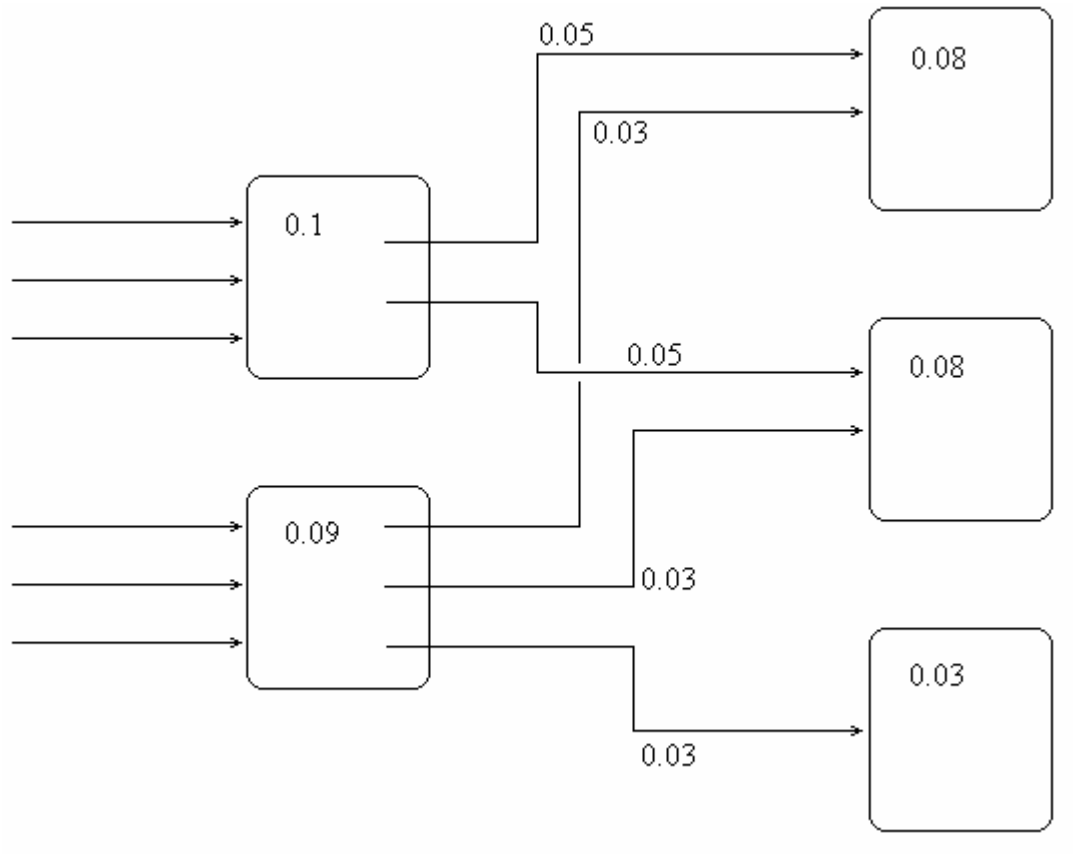

Abbildung 3-1: PageRank - Gewichtsfluss durch den Hyperlinkgraphen

Das Gewicht eines Dokuments „fließt“ also über die Hyperlinks zu den Dokumenten, auf die es verweist. Um den gesamten Textkorpus T zu bewerten, ist es nötig, für jedes Dokument obige Gleichung zu berechnen. D.h. dass ein Gleichungssystem mit so vielen Gleichungen entsteht, wie Dokumente vorhanden sind: |T|. Pro Gleichung gibt es maximal |T| unbekannte Gewichte. In der Praxis ist es jedoch so, dass nicht jedes Dokument auf jedes andere verweist, die Anzahl der Referenzen pro Dokument ist eher gering: z.B. Web-Präsenz TU München, Mathematik und Informatik: |T| = 115.710 Dokumente, jedes Dokument enthielt durchschnittlich 25 Verweise auf Dokumente der gleichen Web-Präsenz. Je größer der Textkorpus ist, umso mehr „Möglichkeiten“ hat ein Dokument auf andere zu verweisen, jedoch zeigen viele Tests, dass auch für wesentlich größere Web-Präsenzen ein Wert von 25 Referenzen pro Dokument zu erwarten ist. Die Abbildung 3-1 zeigt den Einfluss von Dokumenten auf ihre Nachfolger.

Um das Gleichungssystem zu lösen, existiert ein iteratives Verfahren. Die Berechnung ist beendet, wenn die berechneten Werte für jedes Dokument stabil sind (Konvergenz).

PageRank – Algorithmus:

Gegeben:
Textkorpus T, Anzahl aller Dokumente |T|, Dokumente p,q, N_q bezeichnet die Anzahl der Hyperlinks, die von einem Dokument q ausgeht.

Initialisierung: $$\forall p \in T : R(p) = \frac{1}{|T|} \qquad \text{(a)}$$

Wiederhole bis sich die errechneten Werte nicht mehr verändern (Iteration):

$$\forall p \in T : R(p) = \sum_{q : q \to p} \frac{R(q)}{N_q} \qquad \text{(b)}$$

$$c = \frac{1}{\sum_{p \in T} R(p)} \qquad \text{(c)}$$

$$\forall p \in T : R(p) = cR(p) \qquad \text{(d)}$$

Im ersten Iterationsteilschritt (b) wird die Gleichung für ein Dokument p berechnet, zu Beginn unterscheidet sich der Einfluss aller Dokumente, die auf p verweisen, nur durch die Anzahl der Hyperlinks, die sie besitzen: N_p. Im Teilschritt (c) wird die Normalisierungskonstante berechnet und anschließend (d) werden damit alle Dokumente normalisiert. Eine Variante, die im UNICO – Web-Roboter zum Einsatz kommt berechnet die Konstante nach jeder Berechnung des Gewichts eines Dokuments und normalisiert diesen Wert sofort. Das Ergebnis vieler Experimente mit dieser Methode ist die Reduktion der Iterationen, die nötig sind, um Konvergenz zu erreichen. Die Ergebnisse unterscheiden dabei nur geringfügig.

Beschleunigter PageRank – Algorithmus:

Gegeben: Textkorpus T, Anzahl aller Dokumente |T|,

Initialisierung: $\forall p \in T : R(p) = \frac{1}{|T|}$

$$c = \sum_{p \in T} R(p)$$

Wiederhole bis sich die errechneten Werte nicht mehr verändern:

$$\forall p \in T :$$

$$\mathrm{c} = \mathrm{c} - \mathrm{R(p)} \qquad \text{(a)}$$

$$R(p) = \sum_{q : q \to p} \frac{R(q)}{N_q} \qquad \text{(b)}$$

$$\mathrm{c} = \mathrm{c} + \mathrm{R(p)} \qquad \text{(c)}$$

$$R(p) = \frac{R(p)}{c} \qquad \text{(d)}$$

In der Praxis genügt es, die Differenz von der durchschnittlichen Abweichung der errechneten Werte zu den Werten der vorhergehenden Iteration zu berechnen. Ist die Differenz genügend klein, so kann man die Berechnung abbrechen. Die erreichten Werte stellen eine angenäherte Lösung für das lineare Gleichungssystem (PR) dar. Der Konvergenzfall wird in der folgenden Abbildung 3-2 illustriert:

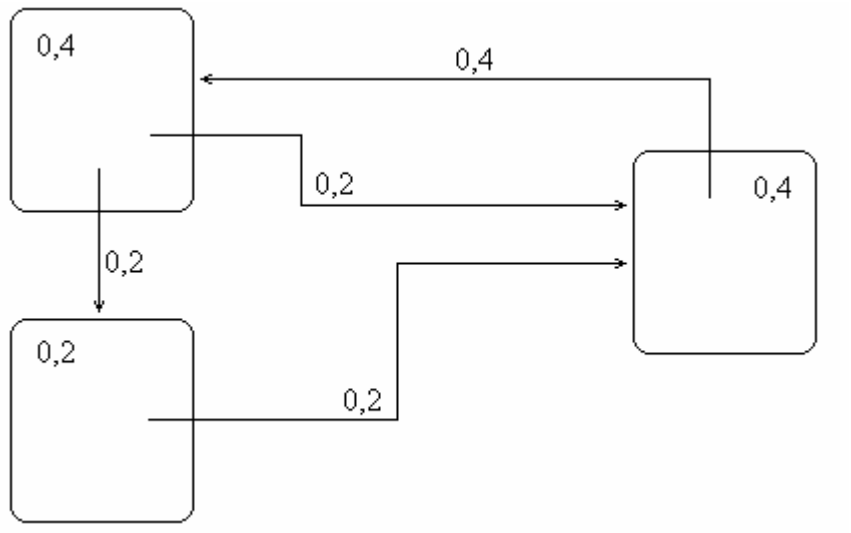

Abbildung 3-2: PageRank - Stabiler Zustand im Graphen

Ein Problem für den PageRank – Algorithmus (auch für die beschleunigte Variante) stellt folgender Fall dar: eine Teilmenge des Textkorpus besitzt nur Referenzen auf Dokumente dieser Menge. Diese Dokumente fungieren dann als „Ablauf“ und absorbieren das gesamte vorhandene Gewicht.

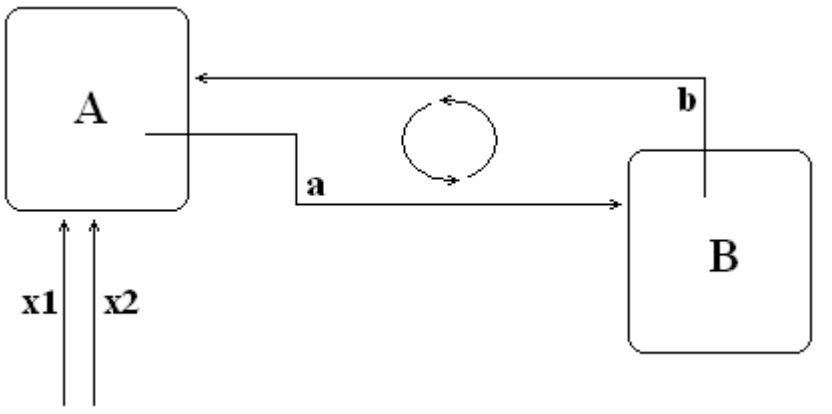

Abbildung 3-3: PageRank - Das Gewicht fliest im Kreis ohne Abflussmöglichkeit

Wenn Dokument A und B zu Beginn das Gewicht 0,1 besitzen und von anderen Dokumenten die Gewichte x1 = 0,1 und x2= 0,1 konstant zufließen, so ergibt sich für die

ersten Iterationsschritte unter der Annahme, dass zunächst A berechnet wird (ohne Normalisierung):

Iteration	A	a	B	b
1	0,3	0,3	0,3	0,3
2	0,5	0,5	0,5	0,5
3	0,7	0,7	0,7	0,7
...	...	...	...	...

Das Problem lässt sich lösen, in dem man eine zusätzliche Konstante E einführt, die bei jedem Iterationsschritt das Gewicht jedes Dokuments „regeneriert“.

$$R(p) = c\left(\sum_{q:q\to p} \frac{R(q)}{N_q} + E(p)\right)$$

Verbesserter PageRank – Algorithmus:

Gegeben: Textkorpus T, Anzahl aller Dokumente |T|, wähle α = 0,15

Initialisierung: $\forall p \in T : R(p) = \frac{1}{|T|}$

$$E(p) = \frac{\alpha}{|T|} \text{ mit } 0 < \alpha < 1$$

Wiederhole bis sich die errechneten Werte nicht mehr verändern:

$$\forall p \in T : R(p) = \sum_{q:q\to p} \frac{R(q)}{N_q} + E(p) \quad \text{(a)}$$

$$c = \frac{1}{\sum_{p\in T} R(p)} \quad \text{(b)}$$

$$\forall p \in T : R(p) = cR(p) \quad \text{(c)}$$

In dieser Form kann man PageRank als Modellierung des Zufallssurfers auf diesem Textkorpus ansehen, der bei einer zufälligen Seite beginnt und an jedem Dokument folgende Optionen hat:

Springe mit der Wahrscheinlichkeit von E(p) zu einem beliebigen Dokument p
Verfolge einen Hyperlink auf dem aktuellen Dokument

R(p) entspricht der Wahrscheinlichkeit, dass der Zufallssurfer an einem beliebigen Zeitpunkt auf dem Dokument p angelangt ist. Die „E-Sprünge" sind notwendig, um zu verhindern, dass der Zufallssurfer in einer Menge von Dokumenten „gefangen" ist, die nur Verweise in diese Menge, also keinen Ausgang besitzen. Zusätzlich können die „E-Werte" in Abhängigkeit von bestimmten Dokumenten gesetzt (personalisiert) werden. Wenn für bestimmte Dokumente E(p) = 0 gesetzt wird, so begrenzt man damit auch die Möglichkeiten der E-Sprünge. Als Resultat werden die dadurch bevorzugten Dokumente mit E(p) > 0 zusammen mit deren näherer Umgebung bessere Gewichte erhalten.
Experimente von Google mit 322 Mio. Dokumenten ergaben die Konvergenz des Algorithmus bei ungefähr 52 Iterationen. Empirisch ermittelt ist die Anzahl der Iterationen bis Erreichen der Konvergenz: bei *n* Hyperlinks sind etwa *log n* Iteration notwendig. Dieser Wert ist durch viele Tests mit dem UNICO – Web-Roboter bestätigt, tendenziell liegt der Wert für die beschleunigte Variante etwa 10% darunter. Google setzt den PageRank – Wert der Dokumente nicht nur für das Ranking der Treffer ein, sondern auch im Retrieval-Prozess wird der Wert aller vorhandenen Dokumente ständig aktuell gehalten. Damit ist es möglich, die Liste der zu holenden Dokumente nach ihrem geschätzten PageRank – Wert zu sortieren und so die „wichtigsten" zuerst zu holen.

3.5.2. HITS

Der HITS Algorithmus basiert auf einer Idee von Kleinberg (1997). Dabei wird das WWW als gerichteter Graph betrachtet, in dem die Knoten Dokumente im WWW und die Hyperlinks die Kanten repräsentieren. Wenn $G = (V, E)$ dieser Graph ist mit V als Dokumentmenge und E als Menge der Hyperlinks, so gilt für ein Dokument p: falls $(p, q) \in E$, so verweist Dokument p durch einen Hyperlink (p,q) auf Dokument q.
„Hubs" und „Authorities" wurden schon in 4.3.2.4 kurz vorgestellt. Die Definition für den HITS – Algorithmus ist etwas restriktiver:

Ein Dokument p ist ein „Hub" für eine Anfrage Q, falls p viele Verweise auf Dokumente enthält, die für Q relevant sind.

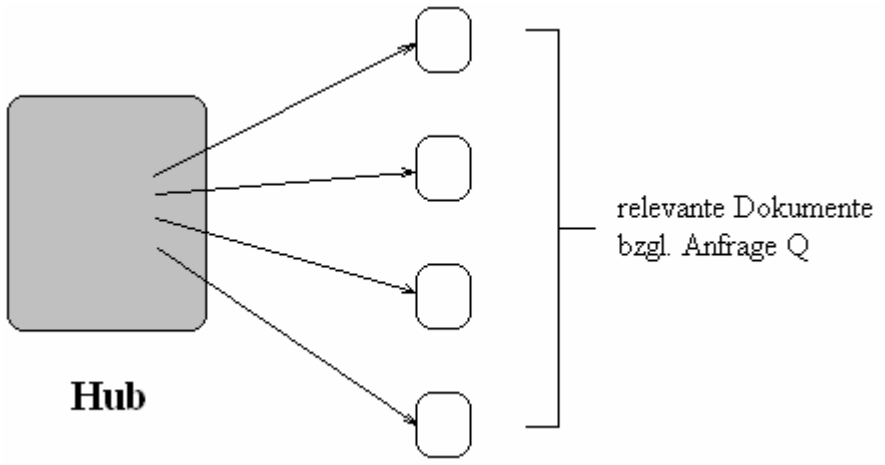

Abbildung 3-4: HITS-Algorithmus – ein Hub

Ein Dokument p ist dagegen eine „Authority“ für eine Anfrage Q, falls p für Q relevant ist und viele Verweise durch andere Dokumente vorhanden sind. Abbildung 3-4 und Abbildung 3-5 deuten an, dass Hubs und Authorites einer Anfrage Q aus der Struktur des Graphen erkennbar sind. Gewünscht ist außerdem, dass „gute“ Hubs auf „gute“ Authorites verweisen, ebenso wird eine „gute“ Authority von „guten“ Hubs referenziert.

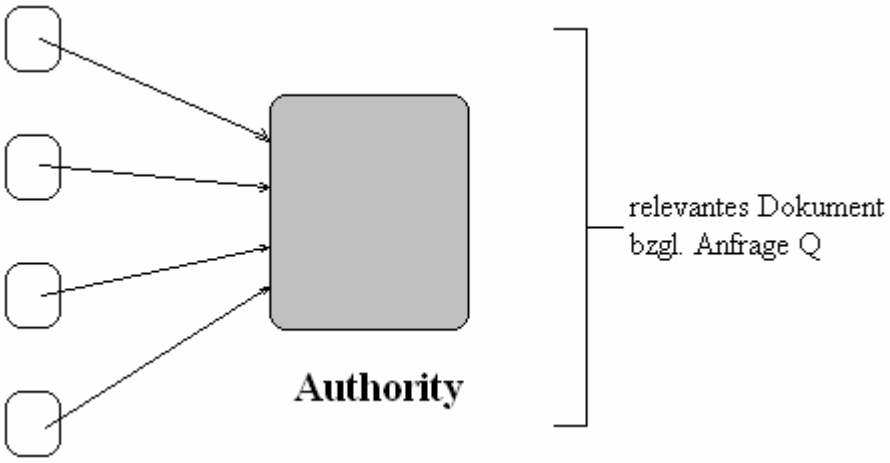

Abbildung 3-5: HITS-Algorithmus – eine Authority

Algorithmus:

Schritt 1: Für eine Anfrage Q werden die ersten t (z.B. 100) Dokumente durch Google oder Altavista bestimmt. Diese Menge wird als „root set“ bezeichnet und enthält im Allgemeinen viele relevante Dokumente, aber nicht alle guten Hubs oder Authorities.

Schritt 2: Das „root set“ wird erweitert um Dokumente, auf welche im „root set“ verwiesen wird oder welche Dokumente aus dem „root set“ referenzieren. Diese Menge wird „base set“ genannt.

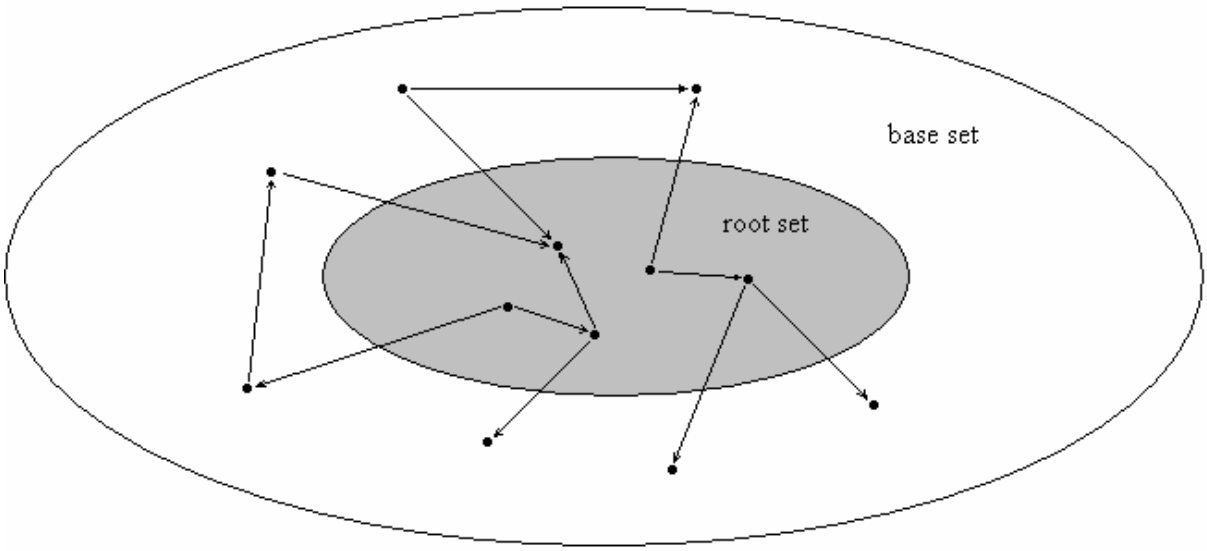

Abbildung 3-6: HITS-Algorithmus – Bestimmung des "base sets"

Damit das „base set“ nicht zu groß wird, beschränkt man die Anzahl der Dokumente, die pro Dokument des „root sets“ maximal hinzugefügt wird, z.B. 50. Manche Dokumente werden von tausenden Dokumenten referenziert. Um zu verhindern, das Hyperlinks, die hauptsächlich der Navigation dienen, aufgenommen werden, sind Hyperlinks in die gleiche Domäne zu ignorieren.

Schritt 3: Jetzt werden die Gewichte h(p) (Hub) und a(p) (Authority) für jedes Dokument p des „base sets“ berechnet. Entscheidend dabei ist die Anzahl der ein- und ausgehenden Referenzen. Würde man als Lösung einfach die Summen

$$a(p) = \sum_{(q,p)\in E} 1 \quad \text{und} \quad h(p) = \sum_{(p,q)\in E} 1$$ mit den Dokumenten p,q aus dem „base set“

wählen, so würden populäre Seiten wie z.B. Yahoo! oder Amazon **immer** Authorities darstellen, da sie sehr oft referenziert werden. Andererseits wäre jede Liste von Hyperlinks ein guter Hub, so z.B. Bookmarks, unter der Bedingung, dass sich diese Dokumente im „base set“ befinden. Besser ist eine rekursive Lösung des Problems:

Initialisierung: für alle Dokumente gilt: a(p), h(p) werden auf einen festgelegten Startwert gesetzt.

Normalisierung: a(p), h(p) sind stets normalisiert:

$$\sum_{p\in V} a(p)^2 = 1 \quad \text{und} \quad \sum_{p\in V} h(p)^2 = 1$$

Iteration: Bestimmung der neuen Gewichte aus den alten

$$a(p) = \sum_{(q,p)\in E} h(q) \quad \text{und} \quad h(p) = \sum_{(p,q)\in E} a(q)$$

mit anschließender Normierung.

Es wird solange iteriert, bis die Werte stabil bleiben = Konvergenz.

Schritt 4: Nach Berechnung des Resultats, kann je nach Anwenderwunsch eine Liste von Übersichtsseiten (Hubs) oder eine Liste von Inhaltsseiten (Authorities) zurückgegeben werden.

Bewertung:

Interessant ist die Unterscheidung zwischen Hubs und Authorities für den Benutzer. In der Regel werden für die Berechnung zwischen 10 und 20 Iterationen benötigt. Trotzdem ist diese Methode im Vergleich zum klassischen Web-Retrieval wesentlich langsamer. Antwortzeiten in der Größenordnung von mehreren Minuten sind die Regel.

Einsatz:

Testeinsatz für die Suchschnittstelle des UNICO – Web-Roboters ist geplant. Dabei werden nur Dokumente innerhalb der UNICO – Datenbank einbezogen, d.h. das „root set" wird nur erweitert durch Dokumente, die sich in der Datenbank befinden.

Bekannte Probleme:

Falls alle Dokumente einer Anfrage ein einzelnes externes Dokument referenzieren, so wird dieses zu stark als Authority bewertet. Ebenso falls ein externes Dokument auf viele Dokumente des „root sets" verweist, so wird dieses zu stark als Hub bewertet.
Automatisch erzeugte Hyperlinks (z.B. Werbebanner des Webhost-Providers) führen zu falschen Authorities.
Anfragen wie z.B. „bmw auto" führen dazu, dass allgemeine Dokumente über Autos und Linklisten über verschiedene Automarken im Ergebnis bevorzugt sind: Der Suchbegriff „auto" dominiert den Begriff „bmw".

Verbesserungen:

Der gleiche Autor (= gleiche Domäne) kann maximal nur eine „Stimme" abgeben für eine externe Seite. Zugleich kann ein Dokument insgesamt nur eine „Stimme" für Seiten einer Domäne abgeben:
Falls k Dokumente p_i einer Domäne auf ein Dokument q verweisen, so wird das Gewicht $aw(p_i, q)=1/k$ für jeden Hyperlink (p_i, q) gesetzt.
Falls von einem Dokument p Hyperlinks auf m Dokumente q_i einer anderen Domäne verweisen, so wird das Gewicht $hw(p, q_i) = 1/m$ gesetzt.
Veränderter Iterationsschritt:

$$a(p)=\sum_{(q,p)\in E} aw(q,p)*h(q) \text{ und } h(p)=\sum_{(p,q)\in E} hw(p,q)*a(q)$$

Um die Probleme 2. und 3. zu lösen, kann versucht werden, Dokumente aus dem „base set" zu entfernen, die inhaltlich zu weit von denen aus dem „root set" entfernt sind. Realisiert

wird diese Idee durch eine künstliche Anfrage geformt aus den Dokumenten des „root sets“, um die Ähnlichkeit zu den Dokumenten des „base sets“ zu bestimmen:

- Eine feste Menge von Wörtern *W*, z.B. 1.000, aus Dokumenten des „root sets“ soll die Vergleichsmenge bilden
- Die Wortmenge *W* und alle Dokumente innerhalb des „base sets“, die nicht Teil des „root sets“ sind, werden mittels Vektorquantisierung über das Kosinusmass verglichen (s. 3.4.1), bezeichnet durch *s(p)*.
- Bei einem gegebenen Schwellwert t wird jedes Dokument aus dem „base set“ entfernt, dessen Ähnlichkeit zur Wortmenge *W* kleiner als *t* ist. Der Schwellwert kann durch nachstehende Funktionen bestimmt werden:
  t = Median aller Werte *s(p)*
  t = Median aller Werte *s(p)* im „root set“
  1/10 *max(s(p))*

Dieser Auswahlschritt erfolgt zwischen Schritt 2 und Schritt 3 des HITS – Algorithmus. Zusätzlich können die Werte von *s(p)* bei der Berechnung der Gewichte *a(p)* und *h(p)* verwendet werden:

$$a(p) = \sum_{(q,p)\in E} aw(q,p) * s(q) * h(q) \quad h(p) = \sum_{(p,q)\in E} aw(p,q) * s(q) * a(q)$$

3.5.3.Vergleich

HITS verbessert die Qualität der Ergebnisse einer Suchmaschine, jedoch zu dem Preis eines deutlich höheren Aufwandes, den vor allem der Anwender durch eine erhöhte Wartezeit (30 Sekunden bis mehrere Minuten) zu spüren bekommt. Diese Verzögerung ist der Hauptgrund dafür, dass diese Technik in den großen kommerziellen Suchmaschinen nicht verwendet wird. Entscheidender Vorteil von HITS ist jedoch, dass die Suche auf Dokumente ausgedehnt wird, welche die Anfrageterme nicht enthalten. Der PageRank – Algorithmus zusammen mit einer Erweiterung der Suche durch die Verknüpfung von Ankertexten mit Dokumenten dürfte aber ähnlich erfolgreich sein. Die Erweiterung des HITS – Algorithmus bringt eine weitere deutliche Verbesserung der Qualität des Ergebnisses gegenüber der Urform. Auch Goggle’s PageRank-Variante dürfte diese Ergebnisse kaum übertreffen. Wenn man die Arbeitsweise der Algorithmen betrachtet, so ist ein gravierender Unterschied, dass HITS abhängig von einer Benutzeranfrage ist und deshalb die Ordnung der Dokumente erst zu diesem Zeitpunkt berechnet werden kann. Die Ordnung der Dokumente ist bei PageRank jedoch statisch und beschreibt die „objektive“ Wichtigkeit eines Dokuments für einen durchschnittlichen Surfer. Der Aufwand zur Berechnung der beiden Algorithmen ist ähnlich, jedoch fallen diese Kosten bei PageRank nur einmal bei der Indexierung der Dokumente an.

Kapitel 4 UNICO

4.1. Das Projekt

UNICO (**Un**iversity **I**ndustry **Co**operation) ist ein Online-Informationsdienst des Fachinformationszentrums der Firma Siemens, der Kooperationen zwischen der Siemens AG[8] und den deutschen, aber auch internationalen Hochschulen erleichtern soll. Es wird eine Datenbank aufgebaut, die sowohl eine vielschichtige Beschreibung der Forschungsaktivitäten der Hochschulen, als auch Wege zur gezielten Kontaktaufnahme enthält.

Technologie - Transfer heute:

Der Technologie - Transfer zwischen den Universitäten und der Industrie gestaltet sich auch heute alles andere als einfach. Ein Unternehmen, das zu einem bestimmten Themengebiet Informationen benötigt, hat im Wesentlichen nur drei verschiedene Möglichkeiten mit den Universitäten in Kontakt zu kommen:

Datensammlungen in Buchform oder auf CD:
Es gibt verschiedene Produkte auf dem Markt, die Informationen über die Forschungstätigkeiten an deutschen Universitäten in Buchform oder auf CD bereitstellen. Siehe dazu [3]. Der größte Nachteil dieser Publikationen ist, dass sie nur jährlich erscheinen und somit nur eine begrenzte Möglichkeit bieten, aktuelle Forschungsdaten einzusehen. Auch die Recherche solcher Quellen gestaltet sich vor allem bei Büchern mühsam und zeitaufwendig.

Websites der Universitäten:
Eine Recherche im Internet allein an den deutschen Hochschulen ist eine langwierige und mühsame Angelegenheit. Die Daten sind unterschiedlich granular, die Suche - so sie überhaupt als Dienst bereitsteht - gestaltet sich schwierig, da die Schnittstelle sich von Hochschule zu Hochschule unterscheidet.

Technologie - Transferstellen:
Die Technologie - Transferstellen bieten eine Anlaufadresse für Firmen oder Forschungseinrichtungen. Allerdings bieten nur wenige dieser Stellen einen Online - Dienst an, der eine schnelle Recherche ermöglichen würde. Weiterhin besitzen die Technologie - Transferstellen nur die Möglichkeit, über Forschungstätigkeiten eines regional sehr begrenzten Bereichs Auskunft zu geben. Eine zentrale Anlaufstelle für alle deutschen oder gar europäischen und internationalen Hochschulen gibt es derzeit nicht.

[8] http://www.siemens.com

4.2.Entstehung der UNICO – Datenbank

Seit Mitte 1997 gibt es die UNICO – Datenbank, deren Urversion im Rahmen einer Diplomarbeit[3] entstanden ist. Die enthaltenen Daten werden in zwei Hierarchiestufen eingeteilt: Institutionen und Projekte. Institutionen sind z.B. Forschungseinrichtungen oder Lehrstühle, die Projekte werden einer Institution untergeordnet: z.B. Diplomarbeiten, Dissertation, Forschungsprojekte. Das verwendete Datenmodell besteht daher aus zwei Relationen:

Dabei kann eine Institution mehrere Projekte unterhalten, diese Verbindung ist über eine CHILD – PARENT – Beziehung realisiert: s. Abbildung 4-1

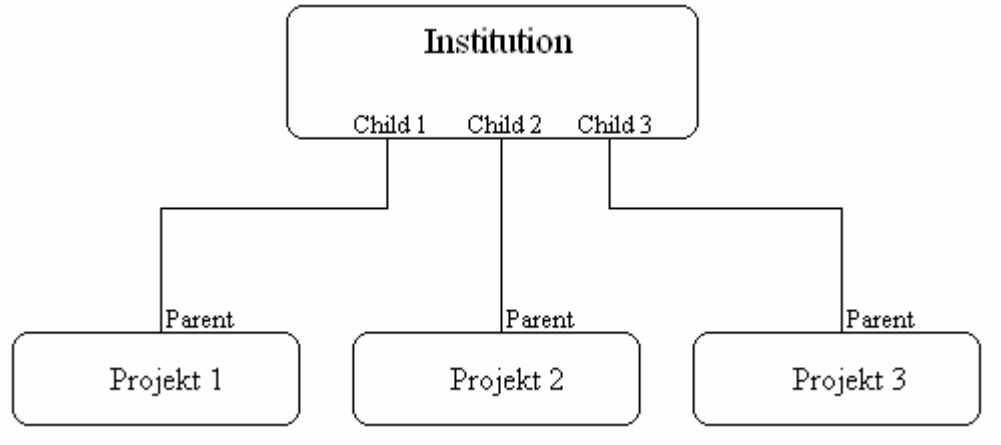

Abbildung 4-1: UNICO-Datenbank-Schema: Instituts - Projekt – Beziehung

Institutionen und Projekte sind dabei eigenständige Dokumente.

Die Institutionen - Relation umfasst die folgenden Felder:

Schlüssel des Datensatzes (SCHLUESSEL), automatisch generiert
Schlüssel der zur Institution gehörigen Projekte (CHILD)
Name der Institution (NAME)
Informationen über die Kontaktperson (PERSON)
Möglichkeiten der Kontaktaufnahme (KONTAKT)
Netzadresse der Institution (NETZADRESSE)
Erstellungsdatum des Dokuments (ERSTELLT)
Erfassungsdatum des Datensatzes (ERFASST)
Charakterisierende Schlagworte (SCHLAGWORT)
Beschreibung der aktuellen Forschungstätigkeit (FORSCHUNG)
Publikationen (PUBLIKATION)
Bestehende Kooperationen (KOOPERATION)
Vorhandene Ausstattung (AUSSTATTUNG)
Kooperationsangebote (ANGEBOT)
Daten zur Lehre (LEHRE)

Die Projekt - Relation umfasst die folgenden Felder:

Schlüssel des Datensatzes (SCHLUESSEL) , automatisch generiert

Schlüssel der zum Projekt gehörigen Institution (PARENT)
Name des Projekts (NAME)
Name der Institution (INSTITUT)
Informationen über die Kontaktperson (PERSON)
Möglichkeiten der Kontaktaufnahme (KONTAKT)
Netzadresse des Projekts (NETZADRESSE)
Erstellungsdatum des Dokuments (ERSTELLT)
Erfassungsdatum des Datensatzes (ERFASST)
Charakterisierende Schlagworte (SCHLAGWORT)
Beschreibung der aktuellen Forschungstätigkeit (FORSCHUNG)
Publikationen (PUBLIKATION)

Das Technologietransferangebot folgender Hochschulen wurde bereits vollständig in UNICO integriert:

Technische Universität Clausthal
Universität Darmstadt
Universität Dresden
Heinrich-Heine-Universität Düsseldorf
Universität Hannover
Technische Universität Ilmenau
Technische Universität München
Freie Universität Berlin
Humboldt-Universität Berlin
Technische Universität Berlin
Ruhr-Universität Bochum
Universität Duisburg

Eine Reihe weiterer Hochschulen sind mit weiteren Datensätzen vertreten.

Die Gewinnung von Daten gestaltete sich von Anfang an schwierig, da die betroffenen Institutionen die Möglichkeit nicht nutzten, die Daten selbst einzubringen. Der Vorteil für die Universitäten wäre eine höhere Wahrscheinlichkeit einer Kooperation mit der Industrie resp. der Siemens AG gewesen. Es gab eine Eingabemaske im WWW, mit der jedes Institut seine Daten selbst eingeben konnte. Die erhaltenen Daten wären dann nach der Übersetzung in das UNICO – Datenmodell und nach manueller Kontrolle direkt in die Datenbank gewandert. Weitere gelegentlich verwendete Formen des Datenaustauschs waren und sind immer noch Forschungsberichte in elektronischer Form, also z.B. MS-Access-Datenbanken, Word-Dokumente oder auch einfache Textdateien. Die Daten wurden dann durch studentische Hilfskräfte mit verschiedenen Hilfsmitteln, wie z.B. „lex“ und „yacc“[9], weiterverarbeitet und die für UNICO relevanten Informationen extrahiert. Ein solcher Verarbeitungsprozess ist nur dann sinnvoll, wenn die Darstellung der elektronischen Daten homogen ist, da dann nach einmaliger Erstellung der Hilfsprogramme der Datenextraktionsprozess automatisch ablaufen kann. Trotzdem ist diese Art der Datengewinnung zeitaufwendig und unsicher: die Einarbeitung studentischer Hilfskräfte in die Verarbeitungswerkzeuge benötigt Zeit und verlangt ein Grundwissen über Programmierung, zudem hängt das Ergebnis stark vom Bearbeiter ab. Hinzu kommt, dass viele Universitäten keine Forschungsberichte in elektronischer Form anbieten und auf Anfrage schlicht auf die universitären Web-Seiten verweisen. Im Gegensatz zu Forschungsberichten sind HTML[10]-Seiten in der Regel inhomogen, d.h. an jedem Lehrstuhl für jedes Projekt erstellt ein anderer Mitarbeiter die HTML-Dokumente nach seinen Vorstellungen. Selbst wenn die Daten homogen innerhalb der Web-Server einer Universität vorlägen, wäre es immer noch schwierig, diese aus ihrer HTML-Struktur herauszulösen und die Verlinkung

[9] Werkzeuge für die lexikalische Analyse und Verarbeitung von elektronischen Texten auf UNIX-Systemen
[10] HTML: **H**yper**t**ext **M**arkup **L**anguage – s. http://www.w3.org/MarkUp/

aufzulösen (z.B. wenn die Beschreibung eines Lehrstuhls auf drei miteinander durch Hyperlinks[11] verbundenen HTML-Seiten steht).
All diese Schwierigkeiten, die bis heute bestehen, führten zu unterschiedlichen Methoden der Datengewinnung, die im Rahmen des UNICO – Projekts mit wechselndem Erfolg getestet wurden und werden. Diese Verfahren sollen im Folgenden beschrieben werden.

4.3.Ansätze zur Datengewinnung

4.3.1.Wrapper – Tools

Viele Forschungsberichte und ähnliche für UNICO wichtige Datensammlungen sind sehr homogen aufgebaut, so dass man diese (Semi-) Struktur ausnutzen kann, um Daten automatisiert zu extrahieren. Ein Ansatz hierzu sind Wrapper – Tools. Dabei werden bestimmte Muster in den zu bearbeitenden Daten benutzt um die dort enthaltenen Informationen zu extrahieren, z.B.

Auszug aus http://www.informatik.tu-muenchen.de/fakultaet/lehrstuehle.html

Informatik I - **Prof. B. Brügge, Prof. G. Klinker**

^ **Angewandte Softwaretechnik**

Empirical Software Engineering; Reusable Object-Oriented Architectures and Applications; Distance Learning; Multi-Project Management and Organizational Knowledge; Groupware; Communication Metrics; Software Engineering Education; Maintenance of Complex Systems; Augmented Reality; Wearable Computing.

Lehre: Software Engineering; Einführung in Augmented Reality; Praktika: Software Engineering, Augmented Reality.

Informatik II - **Prof. J. Eickel**

^ **Formale Sprachen, Übersetzerbau, Software-Konstruktion**

Generierung von Programmierumgebungen; Generierung von Benutzungsoberflächen; Dokumentarchitektur; Graphgrammatiken.

Lehre: Übersetzung von Programmiersprachen; Generierung von Dokumentenarchitektur; Übersetzung objektorientierter Programmiersprachen; Generierung von Bedienoberflächen; Attributierte Grammatiken; Syntaxanalyse; Praktikum: Übersetzerbau.

Informatik III - **Prof. R. Bayer**, **Prof. D. Kossmann**

^ **Datenbanksysteme, Wissensbasen**

[11] Hyperlink: Querverweis in einem Dokument. Er führt zu einer anderen Textstelle des Dokumentes oder zu einem anderen Dokument, eventuell auf einem anderen Web-Server.

Datenstrukturen und effiziente Algorithmen zur Datenorganisation; Parallele, verteilte und vernetzte Datenbanksysteme; Multimedia Datenbanksysteme; Information Retrieval; Wissensrepräsentation und Wissensbanken; Grosse Multimedia-Archive mit assoziativer Suche; Digitale Bibliotheken in Netzen; Geographische Informationssysteme; Mehrdimensionale Zugriffsstrukturen; Data Warehouses; XML Datenbanken und semi-strukturierte Daten; Web Services und Plattformen für Web Services; Datenbankanwendungen (z.B. Enterprise Resource Planning (ERP), eBusiness, eEntertainment, eLearning); Datenbanken für mobile Anwendungen und Dienste.

Lehre: Datenbanksysteme; Architektur und Implementierung von Datenbanksystemen; Parallele und verteilte Datenbanksysteme; Datenstrukturen und Datenorganisation; Data Warehousing; XML Informationssysteme; Praktikum: Datenbanksysteme.

Informatik IV - **Prof. M. Broy, Prof. T. Nipkow, Prof. E. Denert, Prof. K-R. Moll, Prof. H. Schwärtzel**

^ Software und Systems Engineering

Software and Systems Development; Software Engineering; Programming Methods; Property Oriented Specification Techniques; Modelling Distributed Systems; Semantics; Development and Verification Techniques; Interactive Tools for Specification and Verification; Rewriting and Unification Theory; Functional Programming; Process Modelling.

Lehre: Modellierung verteilter Systeme; Projektorganisation und Management in der Software-Entwicklung; Logik; Parallele und Verteilte Programmierung; Temporale Logik; Softwareengineerung; Termersetzungssysteme; Praktika: Softwaretechnik, Spezifikation und Verifikation.

Vier Lehrstühle sind in diesem Beispieldokument beschrieben, die Information ist jeweils gleich strukturiert. So erwartet man nach dem Schlüsselwort „Informatik“ die Nummer des Lehrstuhls, sowie die Namen der Professoren. Nach „^ “ folgt der Name des Lehrstuhls, im folgenden Absatz Schlüsselwörter über die Themengebiete. Abschließend erfolgt nach dem Bezeichner „Lehre“ die Information über die Lehrinhalte. Da es sich um ein HTML-Dokument handelt, kann man die im HTML-Code enthaltenen Strukturinformationen zusätzlich zur Erkennung der gewünschten Information heranziehen. Diese Methode der Informationsgewinnung ist vor allem dann sehr erfolgreich, wenn die HTML-Dokumente automatisch generiert worden sind, da der „Rahmen“ aller Dokumente dann sehr homogen ist, lediglich die enthaltene Information verändert sich. So lässt sich z.B. eine Meta-Suchmaschine generieren, die eine Suchanfrage an mehrere allgemeine Suchmaschinen im WWW weiterleitet, deren (automatisch) generierte Ergebnisseiten „wrappt“, die jeweiligen Treffer erkennt und als eigene Trefferliste darstellt. Da die Betreiber der kommerziellen Suchmaschinen an solchem „Datenklau“ kein Interesse haben, verändern sie die Struktur ihrer Seiten von Zeit zu Zeit. Die Meta-Suchmaschine müssen dann den jeweiligen Wrapper an die neue Struktur anpassen, um die Ergebnisse weiterhin verwenden zu können. Ein Beispiel für eine Meta-Suchmaschine ist z.B. MetaGer: http://www.metager.de. Weitergehende Informationen über Wrapper-Tools sind in [18] bis [34] zu finden.

4.3.2.Web – Roboter

4.3.2.1.Motivation

Um die UNICO - Datenbank auch mit internationalen Daten zu füllen und sie damit attraktiver zu machen, wurde Anfang 2000 beschlossen, zunächst die amerikanischen Universitäten einzubringen. Da im UNICO - Team die „Manpower“ fehlt, um die Daten in einer derartigen Formatierung und Qualität zu gewinnen, wie es bisher bei den deutschen Universitäten möglich war, wurde entschieden, einen intelligenten Agenten zu konstruieren, der die Web-Präsenz der jeweiligen Universität "absurft", um die für UNICO relevanten Daten in eine volltext-recherchierbare Datenbank einzufügen. Eine Einordnung der Web-Daten in das UNICO - Datenmodell ist jedoch nicht ohne immensen Aufwand möglich. Diese *neue* Datenbank, sowie die dazugehörige Suchschnittstelle, sind daher für den recherchierenden Anwender deutlich getrennt. Das ist notwendig, um die vorhandenen qualitativ hochwertigen Daten nicht durch Daten mit Suchmaschinenqualität zu „verwässern“. Der "Trade-Off" hierbei ist, dass die Web-Daten zwar von minderer Qualität als die der originalen UNICO - Datenbank sind, aber dafür wesentlich schneller erfasst werden können. Zudem ist auch die Aktualisierung problemlos. Als weiterer Pluspunkt ist zu vermerken, dass der Operator für die Verwendung eines solchen Systems keinerlei spezifische Fachkenntnis aus dem Bereich der Programmierung benötigt.

Aufgabenstellung:

Entwicklung eines Agenten mit folgenden Maßgaben:

- *Retrieval-Komponente*: Bei Eingabe einer Anfangsadresse durch einen UNICO-Administrator wird die komplette Web-Site "abgesurft" und die Web-Seiten werden unter Einhaltung vorgegebener Einschränkungen in eine Datenbank eingefügt

- *Evaluationskomponente*: Sind alle Web-Seiten einer Institution in der Datenbank erfasst, erfolgt die Bewertung derselben nach vorgegebenen Kriterien

- *Exportkomponente*: Es existiert eine Exportmöglichkeit der enthaltenen Daten mitsamt der jeweiligen Bewertung. Exportformat ist XML (s. 7.3: Export DTD und Beispiel - Ausgabe)

- *GUI*: Entwicklung einer graphischen Bedienoberfläche

- Der Agent ist Thread[12]-basiert, d.h. es werden mehrere URLs gleichzeitig bearbeitet, die Datenarchivierung erfolgt im Hintergrund

4.3.2.2.Architektur

Der Programmablauf lässt sich in vier Teilschritte zerlegen (s. Abbildung 4-2: Architektur des UNICO – WebAgenten):

[12] „Programmfaden“: einzelnes sequentielles Programmstück, mehrere Threads sind parallel ausführbar

1. Eine Komponente, die für das Übertragen der HTML-Dokumente von dem entsprechenden Web – Server zuständig ist, fügt diese in eine angeschlossene Datenbank ein.

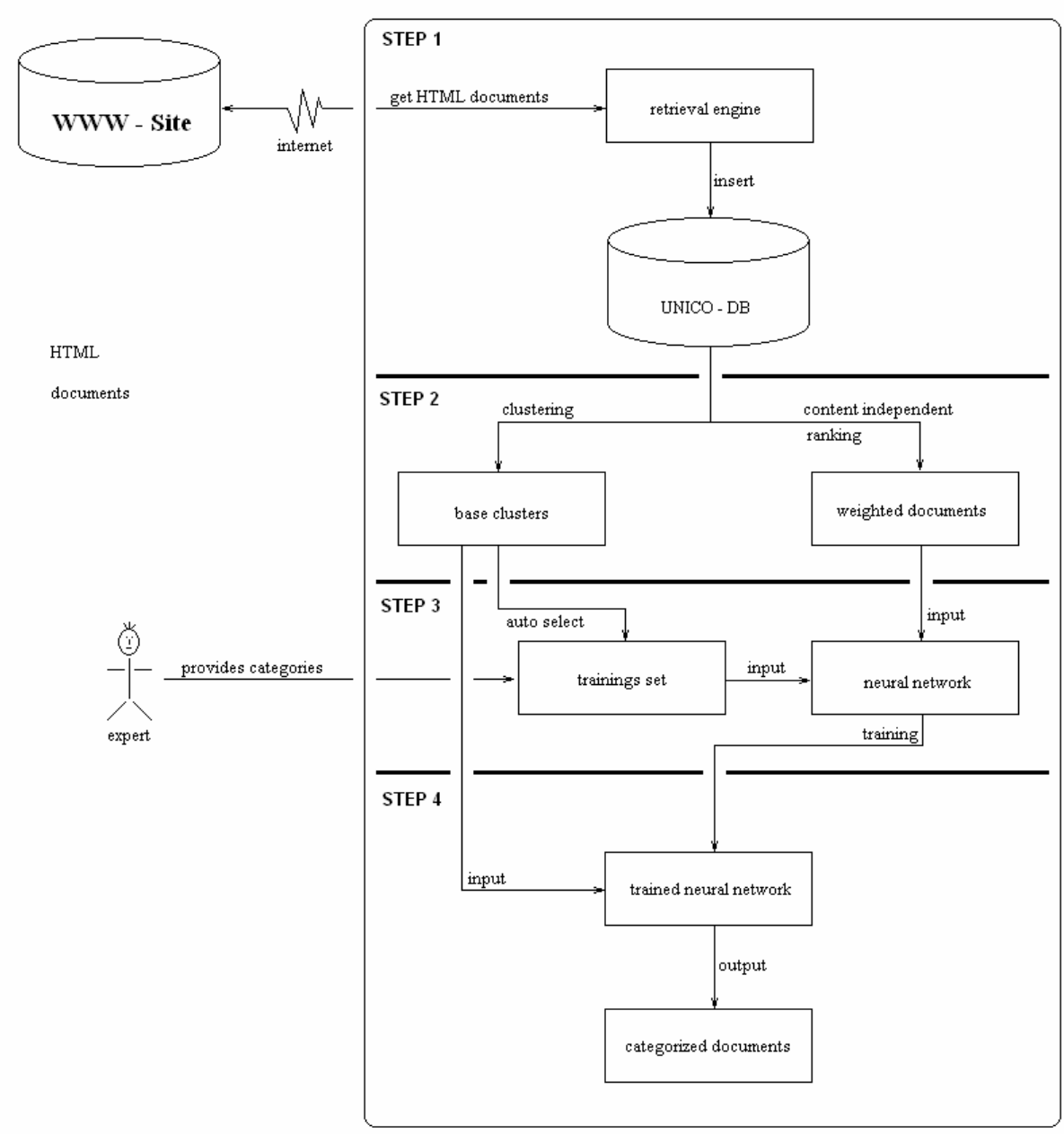

Abbildung 4-2: Architektur des UNICO – WebAgenten

2. Sobald alle Dokumente des Servers in die Datenbank übertragen wurden, werden diese zunächst durch Entfernen von Stopwörtern[13] und durch ein Verfahren zur Wortstammbildung (s. 3.3) „gereinigt“. Anschließend werden die Basis – Cluster gebildet, d.h. es werden Dokumente zusammengeschlossen, die inhaltliche

[13] Stopwort: Ein Wort ohne inhaltliche Bedeutung für den Gesamtkontext

Gemeinsamkeiten besitzen (durch das K-Means-Verfahren). Außerdem werden die Dokumente bezüglich ihrer Lage innerhalb des Hyperlink-Graphen bewertet. Das erfolgt durch den PageRank – Algorithmus (s. 3.5.1).

3. Aus den Basis – Clustern wird im dritten Schritt ein Trainingsset generiert, das aus jedem Cluster mindestens zwei Dokumente enthalten muss. Ein Experte muss jetzt zu dem Element des Trainingssets eine oder mehrere Kategorien zuweisen, zu denen er das Dokument nach seinem Wissen zuordnet und gewichtet. Die Anzahl der Kategorien ist frei wählbar, als Minimum wäre schlicht „Relevanz" als einzige Kategorie möglich. Zu jedem ausgewählten Dokument wird zusätzlich zu der Information, aus welchem Cluster das Dokument stammt, der zugehörige PageRank-Wert und die vom Experten bestimmte Kategorie mit zugehörigem Gewicht für das Training des neuronales Netzes übergeben.

4. Das vollständig trainierte neuronale Netz bestimmt jetzt anhand seiner erlernten Bewertung die Zugehörigkeit aller Dokumente zu den vom Experten festgelegten Kategorien.

Die beteiligten Komponenten werden im folgenden Abschnitt genauer beschrieben.

4.3.2.3.Retrieval

4.3.2.3.1.Überblick

Die Aufgabe dieser Komponente besteht darin, sämtliche verfügbaren WWW-Daten einer Institution in eine Datenbank zu übertragen. Dazu wird zuerst eine Startadresse angeben, z.B. für das Institut für Informatik der TU München: http://www.in.tum.de. Ausgehend von dieser Adresse werden rekursiv alle Hyperlinks verfolgt und die Datenbank eingefügt, die auf dieser Seite und ihrer Nachfolger vorkommen. Um zu vermeiden, dass Seiten verfolgt werden, die nicht zur gewünschten Institution gehören, muss zu Anfang eine Beschränkung auf festgelegte Bereiche des WWW erfolgen, deren Daten von Interesse sind. So ist z.B. die Webpräsenz des Instituts für Informatik der TU München auf die Domäne 131.159.*.* beschränkt („allow"). Möglich ist zusätzlich eine Einschränkung des erlaubten Bereichs („deny"): das ist von Vorteil, wenn bestimmte Bereiche der Webpräsenz einer Institution nicht von Interesse sind, wie z.B. bestimmte Fakultäten einer Universität, studentische Homepages, Softwaremanuale, usw.. Dazu ist es erforderlich, die WWW-Seiten der betreffenden Einrichtung nach „uninteressanten" Bereichen zu durchsuchen. Um diesen Aufwand gering zu halten, ist es sinnvoll, nur Subdomänen, Webserver oder ganze Verzeichnisse anzugeben und nicht etwa einzelne Dokumente. Beispiele: Subdomäne: http://131.159.12.*, Webserver: http://131.159.12.20, Verzeichnis: http://131.159.0.51/rechenbetrieb. Einschränkungen dieser Art sind im Gegensatz zu den „allow"-Beschränkungen nicht zwingend notwendig, verringern jedoch die Menge an Daten, da derart gekennzeichnete Bereiche der Web-Präsenz vom Retrieval ausgeschlossen werden. Die Startadresse, die Einschränkungen „allow" und „deny" werden zusammen mit einer Beschreibung als neue Sitzung in die Datenbank eingetragen. Diese Informationen können interaktiv eingegeben werden oder durch Einlesen einer Konfigurationsdatei (s. Anhang 7.2.2). Grob ist die Retrieval – Komponente in drei Abschnitte geteilt: Der *Datenbankteil* ist zuständig für die Ausgabe von Hyperlinks, deren HTML-Daten vom *Retrievalteil* geholt werden sollen und für das Einfügen von fertig bearbeiteten Daten des *Parsingteils*. Dieser extrahiert aus HTML-Daten wichtige Informationen zur Weiterverarbeitung (Hyperlinks!). Alle Teile werden im

Programm durch Threads repräsentiert. Für die Datenbankaktionen gibt es einen Eingabe- und einen Ausgabe-Thread. Mehr Parallelität ist hier nicht möglich, da es sonst zu Inkonsistenzen innerhalb der Datenbank kommen könnte oder die Threads sich gegenseitig blockieren, da z.B. Daten in denselben Tabellen modifiziert werden sollen. Für das Retrieval und das Parsing sind üblicherweise jeweils zehn Threads zuständig. Die Menge ist jedoch vor Programmstart (s. Anhang 7.2.1) und auch zur Laufzeit veränderbar. Der Vorteil des Einsatzes von Threads wird vor Allem beim Retrieval sichtbar: wäre nur ein einzelner Thread beim Holen der HTML-Daten im Betrieb, so wäre das komplette Programm blockiert, falls es Verzögerungen beim Übertragen der Daten gibt (z.B. durch geringe Übertragungsrate). Im Extremfall können das mehrere Minuten sein (z.B. bei einer Zeitüberschreitung der Anfrage). Sind mehrere Threads beteiligt, so steigt die Wahrscheinlichkeit, dass zumindest einer Daten zur weiteren Verarbeitung liefern kann und somit der Programmfluss erhalten bleibt.

4.3.2.3.2.Architektur der Retrieval-Komponente

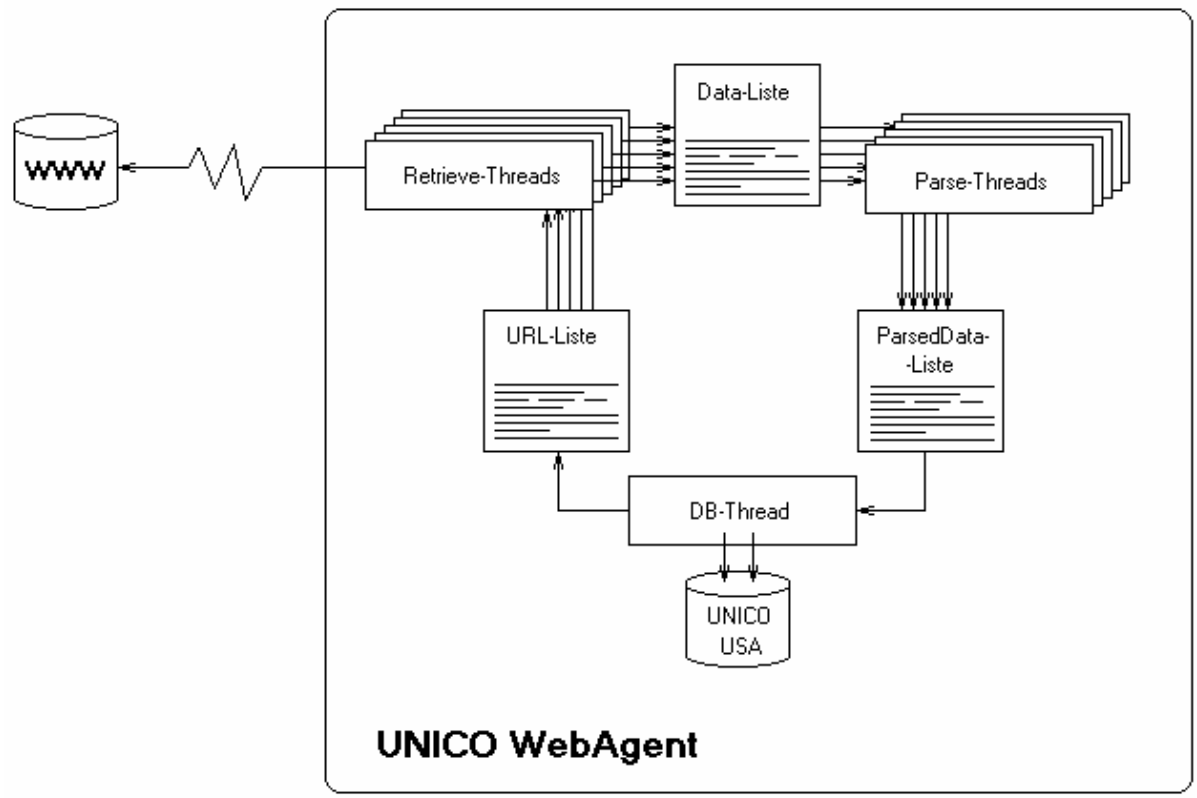

Abbildung 4-3: Architektur der Retrieval - Komponente

Der Arbeitsvorgang der Übertragung einer Web-Präsenz in eine Datenbank entspricht einem Kreislauf. Hyperlinks werden aus einer Datenbanktabelle in eine Liste eingefügt, mehrere Threads versuchen die dazu gehörigen Daten aus dem WWW zu übertragen. Diese nun vorhandenen vollständigen Dokumente werden einer Liste übergeben, aus der wiederum Threads versuchen, darin enthaltene Hyperlinks (und andere Informationen) zu extrahieren. Der Dokumenttext, die zugehörigen Hyperlinks und Metadaten werden in eine weitere Liste eingetragen. Die zuständige Datenbankkomponente fügt das Dokument und dessen Metainformationen in Tabellen ein, ebenso werden die dazu gehörenden Hyperlinks nach

Validitätsprüfung in die Tabelle eingefügt, aus der dann wieder die Liste der zu übertragenden Hyperlinks gefüllt wird. Zu Beginn wird vom Benutzer eine Startseite vorgegeben. Der Vorgang endet, wenn keine Hyperlinks mehr zu übertragen sind. Der Kreislauf wird veranschaulicht in Abbildung 4-3: Architektur der Retrieval - Komponente.

Zu jedem Hyperlink existieren in der Datenbank folgende Informationen:

Bezeichnung	Beschreibung
id	Eine eindeutige Nummer
origurl	Der originale, vollständige Hyperlink, z.B. http://www.test.com/t.html
host	Die IP-Adresse des zuständigen Web-Servers
Hashcode	40-stellige Zahl, generiert aus Metadaten und Inhalt des Dokuments
lastupdate	Datum der letzten Veränderung des Originaldokuments
keywords	Information aus dem Meta-TAG „keywords“ des Orignaldokuments
type	MIME - Typ des Dokuments, der UNICO – Web-Roboter akzeptier bislang nur HTML-Dokumente, mögliche Erweiterungen sind PDF-, Postscript- oder auch reine Text-Dokumente
length	Größe des Orignaldokuments in Bytes
Title	Der Titel des Originaldokuments
language	Information über die Sprache, in der das Dokument verfasst ist
cdate	Datum des Eintrags des vollständigen Dokuments in die Datenbank
content	Das Originaldokument
fulltext	Der sichtbare Text des Dokuments ohne Stopwörter, Interpunktionszeichen, reduziert auf Kleinschreibung; dient als Basis für den Volltextindex

depth	Tiefe des Dokuments, d.h. Abstand vom Startdokument
doNotFollow	Information, ob ein Hyperlink bearbeitet werden soll
Status	Status der Bearbeitung: wartend, in Bearbeitung, fertig, fehlerhaft, Dublette
invalid	Ist eine Bearbeitung fehlgeschlagen, so wird die Fehlerursache hier festgehalten

Arbeitsablauf im Detail:

Aus der Datenbank werden folgende Daten in die URL-Liste übertragen**: id, origurl, host, depth**. Beispiel:

id = 210
origurl = http://www.test.com/test/index.html
host = 100.100.101.26
depth = 4

Der Status dieser ID wird in der Datenbank auf „in Bearbeitung gesetzt“, um zu vermeiden, dass derselbe Hyperlink mehrfach in die Liste eingefügt wird.

Ein zuständiger „arbeitsloser“ Thread erhält diese Daten aus der Liste. Er prüft zunächst, ob auf Seiten des anzusprechenden Web-Servers Einschränkungen bezüglich bestimmter Web-Roboter bestehen. Dazu versucht er die Datei http://www.test.com/robot.txt aus dem WWW zu übertragen. Falls diese vorhanden ist, werden die Einschränkungen ausgewertet und überprüft, ob das zu übertragende Dokument geholt werden darf. Da die Informationen den gesamten Web-Server betreffen, werden diese in einem Cache aufbewahrt, um nicht jedes Mal die Datei „robot.txt“ laden zu müssen. Diese Datei liegt, wenn sie überhaupt vom Administrator des Web-Servers angelegt ist, immer im Wurzelverzeichnis des Web-Servers. Gibt es keine Einschränkung bezüglich dieses Hyperlinks, so beginnt der Thread mit der Übertragung. Fehlermeldungen des Web-Servers, wie z.B. „Error 404 = Dokument nicht vorhanden“, eine Überschreitung der maximalen Übertragungsdauer oder Übertragung eines falschen Dokumenttyps (z.B. eine Bilddatei), werden festgehalten unter der Bezeichnung „invalid“. War die Übertragung fehlerfrei, so wird der Dokumentinhalt unter der Bezeichnung „content“ festgehalten. Ebenso wird die Größe ermittelt. Folgende Informationen werden in die Data-Liste eingetragen: **id, origurl, host, depth, content, length, invalid**. Beispiel:

id = 210
origurl = http://www.test.com/test/index.html
host = 100.100.101.26
depth = 4
content = „<HTML> .... </HTML>“
length = 3245 Bytes
invalid = „“

Beispiel für den Fehlerfall:

id = 210
origurl = http://www.test.com/test/index.html
host = 100.100.101.26
depth = 4
content = ““
length = 0 Bytes
invalid = „Retrieval Timeout“

Ein für die Auswertung von Dokumenten zuständiger Thread erhält jetzt die Informationen aus dem obigen Listeneintrag. Wenn kein Fehler zuvor aufgetreten ist (invalid = „“), so wird das HTML-Dokument von einem HTML-Parser zerlegt und es werden zunächst die Informationen aus dem Meta-TAG „robot“ ausgewertet. Darin kann der Autor des Dokuments z.B. angeben, ob das Dokument oder/und die enthaltenen Hyperlinks indexiert werden soll(en) oder nicht. Ist die Indexierung durch den Autor gestattet, so werden folgende Informationen aus dem Dokument ermittelt: Titel des Dokuments, über Meta-TAGs vom Autor angegebene Schlüsselwörter und alle enthaltenen Hyperlinks. Zusätzlich wird der lesbare Dokumenttext von Sonderzeichen und Interpunktionszeichen sowie Stopwörtern befreit und die verwendete Sprache „geraten“. Der reduzierte Dokumenttext wird mit „fulltext“ bezeichnet, die ermittelte Sprache mit „language“. Ein 40-stelliger Hashcode zur Erkennung zur Wiedererkennung des Dokuments wird berechnet (hashcode). Die gefundenen Hyperlinks werden vervollständigt, der Host bestimmt, gegen die Einschränkungen der Web-Präsenz und des Autors abgeprüft und in zwei Gruppen eingeteilt: „toFollow“ = Gruppe der Hyperlinks, die den Beschränkungen genügen und weiter verfolgt werden sollen und „toMemo“ = Gruppe der Hyperlinks, deren Daten nicht zur Web-Präsenz gehören bzw. auf Wunsch des Autors nicht indexiert werden sollen. Zu jedem Hyperlink aus „toFollow“ wird zusätzlich der Ankertext ermittelt und von Stopwörtern befreit. Doppelte Einträge in der Hyperlink-Liste werden eliminiert, zugehörige Ankertexte werden zusammengefasst. Ist die Indexierung des Dokuments durch den Autor nicht gestattet, so wird der Dokumentinhalt „content“ gelöscht, „length=0“ und „invalid=meta robot exclusion“ gesetzt. Die Daten werden nun in die ParsedData-Liste eingetragen. War die Bearbeitung bisher erfolgreich so werden z.B. folgende Daten eingefügt:

id = 210
normurl = http://www.test.com/test/index.html
host = 100.100.101.26
depth = 4
content = „<HTML> .... </HTML>“
length = 3245 Bytes
invalid = „“
title = “Testbetrieb“
hashcode = „000012613452270676905512007729603782568“
language = „german“
fulltext = „testseite ... testseite“
toFollow = { { http://www.test.com/a.html, 100.100.100.26, „seite a“ },
{ http://www.test.com/b.html, 100.100.100.26, „“ },
{ http://www.test.com/c.html , 100.100.100.26 „seite c“} }
toMemo = { { http://www.unwichtig.com/index.html, 111.111.111.0 } }

Im letzten Schritt werden die ermittelten Daten wieder in Datenbanktabellen eingetragen. Der (Bearbeitungs-)Status der Id wird im Fehlerfall auf „fehlerhaft“ gesetzt. Der Grund für den

Fehler wird gegebenenfalls eingetragen. Ist kein Fehler aufgetreten, wird zunächst der Hashcode des einzufügenden Dokuments mit allen schon vorhandenen verglichen. Ist er bereits vorhanden, ist der seltene Fall eingetreten, dass das gleiche Dokument unter verschiedenen Adressen innerhalb der Web-Präsenz vorhanden ist oder der Hyperlink zusammen mit dem angegebenen Host ist nicht eindeutig. In diesem Fall wird der Status der Id als „Dublette“ vermerkt. In zahlreichen Tests mit unterschiedlichen Web-Präsenzen von mehr als 10.000 bearbeiteten Web-Seiten bewegte sich die Anzahl der Dubletten im Bereich von 5% - 12% der bearbeiteten Hyperlinks. Häufigste Ursache war die unterschiedliche Schreibweise der Hyperlinks. Wenn ein Dokument fehlerfrei übertragen wurde und keine Dublette ist, dann werden zu seiner Id folgende Daten eingetragen: „content“ als BLOB[14], „fulltext“ als CLOB[15], „length“, „title“, „hashcode“ und „language“. Jetzt werden die Hyperlinks mit schon vorhandenen aus der Datenbank verglichen und nur die noch nicht vorhandenen werden dann eingetragen. Zuletzt werden die Ankertexte und die Rückwärtsreferenzen in die Datenbank eingefügt.

Um den Kreislauf überhaupt in Gang zu bekommen, muss zu Beginn natürlich ein Hyperlink als Ausgangspunkt dienen. Diese Startadresse gehört zu der Beschreibung einer Web-Präsenz (s. 4.3.2.3.4). Das Verfahren terminiert, wenn keine „neuen“ Hyperlinks mehr hinzugefügt werden, da bereits alle zur Web-Präsenz gehörenden eingetragen sind.

4.3.2.3.3.Algorithmus

Gegeben:

Bezeichner	Beschreibung
dbOut	Thread, der zu holende URLs in den Kreislauf (URL – Liste) einfügt
dbIn	Thread, der fertig bearbeitete Daten aus dem Kreislauf in die Datenbank einfügt
U_{max}	Menge an Daten, die sich maximal im Kreislauf befinden darf
t_1	Wartezeit in Sekunden für dbOut, dbIn
t_2	Wartezeit für Retrieval und Parse Threads, abhängig vom Web-Server
R	Menge der Retrieval Threads $R = \{r_1, r_2, r_3, ..., r_{max}\}$
P	Menge der Parse Threads $P = \{p_1, p_2, p_3, ..., r_{max}\}$
input	Flag, das anzeigt, ob Retrieval-Daten vorhanden sind
$U_{aktuell}$	Anzahl der Daten, die sich zu einem bestimmten Zeitpunkt im Kreislauf befinden
UL	URL – Liste, |UL| Größe der URL –Liste
DL	Data – Liste, |DL| Größe der Data –Liste
PL	ParsedData – Liste, Größe der ParsedData –Liste

[14] BLOB = **B**inary **L**arge **Ob**ject
[15] CLOB = **C**haracter **L**arge **Ob**ject, in vielen kommerziellen Datenbanksystem Basis einer Volltextindexierung

Datenbankteil:

Ausgabe dbOut:

1. Warte t_1 Sekunden
2. Falls input = false: Gehe zu 1.
3. Eintrag von maximal (U_{max} - $U_{aktuell}$) Hyperlinks in die URL-Liste
4. Sind keine weiteren Hyperlinks in der Datenbank, dann input = false
5. Gehe zu 1.

Eingabe dbIn:

1. Warte t_1 Sekunden
2. Falls |PL| = 0, gehe zu 1.
3. Lese alle Daten aus PL
4. Eintrag der Daten in die Datenbank
5. Eintrag der zur Verfolgung gekennzeichneten Hyperlinks in die Datenbank, falls nicht schon vorhanden
6. Falls neue Hyperlinks zur Verfolgung eingetragen wurden, setze input = true
7. Gehe zu 1.

Retrieval – Teil: alle r_i aus R

1. Warte t_2 Sekunden
2. Falls |UL| = 0, gehe zu 1.
3. Lese ersten Eintrag (Hyperlink) aus UL
4. „Materialisiere“ Hyperlink, d.h. übertrage Daten vom Web-Server
5. Falls Fehler (z.B. Timeout der Datenübertragung, ungültiger Hyperlink, etc), markiere Daten als fehlerhaft
6. Trage Daten in DL ein
7. Gehe zu 1.

Parsing – Teil: alle p_i aus P

1. Warte t_2 Sekunden
2. Falls |DL| = 0, gehe zu 1.
3. Lese ersten Eintrag: Hyperlink + Daten aus DL
4. Falls Daten als fehlerhaft markiert, gehe zu 7.
5. Parse Daten: Extraktion von enthaltenen Hyperlinks
6. Einteilung der weiter zu verfolgenden Hyperlinks anhand der „allow“ – und „deny“ – Einschränkungen (s.o.)
7. Trage Daten in PL ein
8. Gehe zu 1.

Alle Threads arbeiten parallel, lediglich der Zugriff auf bestimmte Datenobjekte ist eingeschränkt. So ist gleichzeitiges Lesen und Schreiben auf allen Listen nicht gestattet, ebenso ist der Datenbankzugriff auf die Tabelle mit den noch zu bearbeitenden Hyperlinks exklusiv. Damit der Kreislauf überhaupt in Gang kommt, muss zu Beginn eine Startadresse eingetragen

werden. Ab diesem Zeitpunkt läuft der Algorithmus solange bis keine Daten mehr zur Übertragung anstehen.

4.3.2.3.4.Struktur einer Web-Präsenz

Eine Web-Präsenz ist eine Teilmenge aller Dokumente des WWW, definiert durch Dokumente einer Institution wie z.B. einer Universität. Um genau diese Dokumente zu beschreiben sind folgende Einschränkungen notwendig:

1. Angabe eines Start- oder Einstiegsdokuments, auch mehrere Angaben sind möglich
2. Einschränkung auf bestimmte Internetdomänen oder/und IP-Adressen von Web-Servern

Beispiel:

Die Einstiegsseite
http://www.test.com

Erlaubte Domänen
http://100.100.100.*/
http://100.100.101.*/
http://100.100.102.*/
http://100.100.103.*/

Auszuschließende Verzeichnisse
http://www.test.com/cgi-bin
http://www.test.com/~
http://www.test.com/tmp/
http://100.100.101.25/

Zu einer gegebenen Web-Präsenz kann man jetzt die *Tiefe* eines Dokuments definieren:

Die Einstiegsseite hat die Tiefe 0.
Alle Dokumente, die durch Hyperlinks der Einstiegsseite erreichbar sind, haben die Tiefe 1. Jedes Dokument B außer dem Einstiegsdokument besitzt genau einen Vorgänger, d.h. ein Dokument A, das durch einen Hyperlink auf B verweist. Wenn A die Tiefe t hat, so hat B die Tiefe t+1. Die Eindeutigkeit des Vorgängers ist gegeben, da jedes Dokument nur einmal in die Datenbank eingefügt wird und zu diesem Zeitpunkt wird die Tiefe bestimmt.

Die Tiefe ist somit der kürzeste Abstand eines Dokuments vom Ursprung (dem Einstiegsdokument). Wenn man für jede Tiefenschicht die Anzahl der Dokumente bestimmt, erhält man in der Praxis einen Graphen wie in Abbildung 4-4.

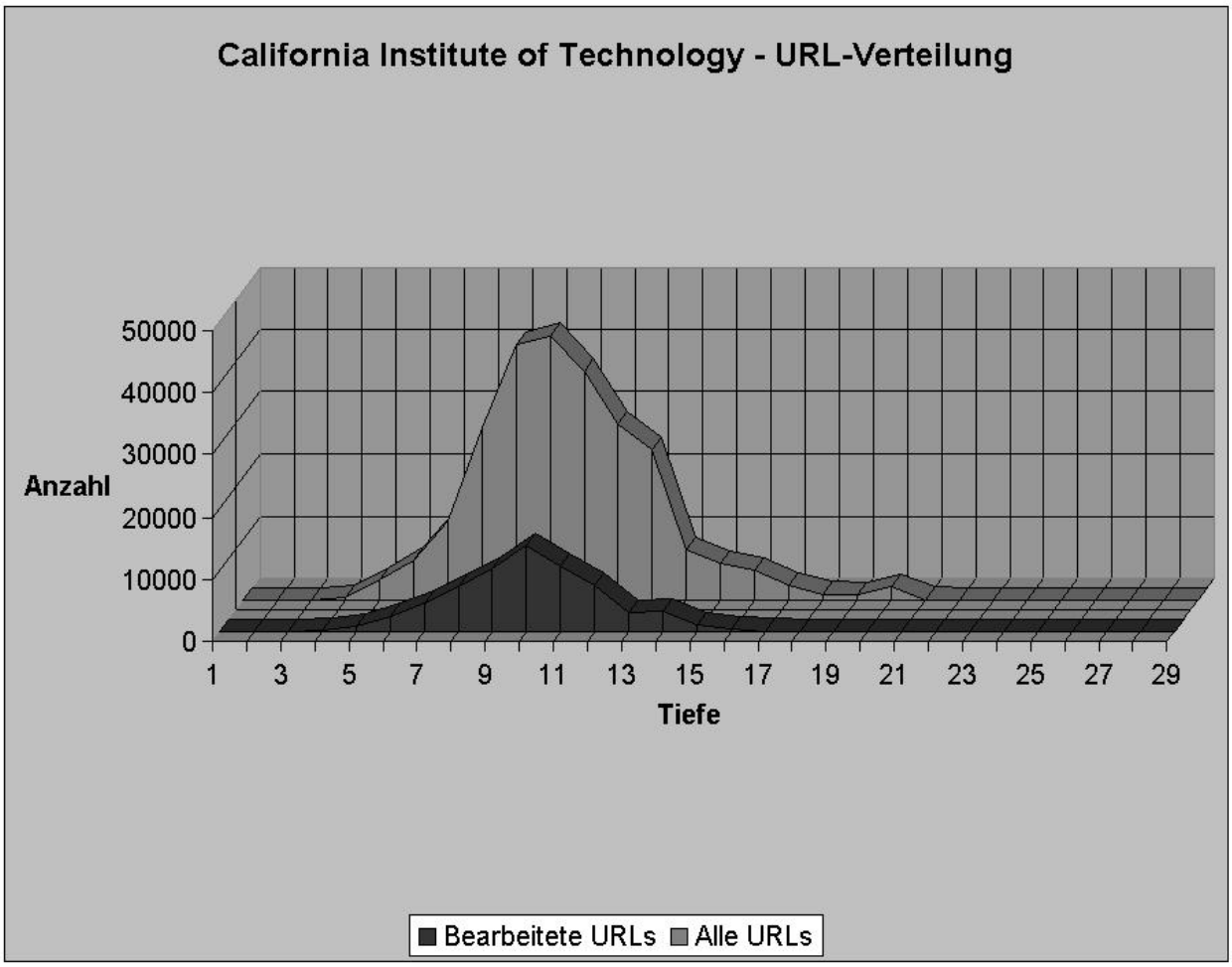

Abbildung 4-4: Verteilung von Hyperlinks für das California Institute of Technology

Es ergibt sich also eine Verteilung, die einer Gauß-Kurve ähnelt. Der höhere Graph im Hintergrund beschreibt die Anzahl aller Hyperlinks, der Graph im Vordergrund beschreibt nur die Anzahl der Hyperlinks, deren Dokumente auch tatsächlich in die Datenbank eingefügt wurden. Alle anderen Hyperlinks verweisen auf Dokumente außerhalb der Web-Präsenz, Dokumente falscher Typs oder „dangling“ Links[16]. Zum Vergleich ein weiterer Graph zur Web-Präsenz der Informatik / Mathematik der TU München in Abbildung 4-5.

Wegen der Symmetrie bezüglich des Maximums ist es möglich zu einem bestimmten Zeitpunkt die Menge der noch zu erwartenden Dokumente abzuschätzen. Der Zeitpunkt muss dabei so gewählt sein, dass das Maximum bereits überschritten wurde. Dieses Maximum ist einfach zu erkennen, da die Mengen von Dokumenten es deutlich beschreiben: s. Abbildung 4-6.

Wenn die Tiefe t erreicht ist, d.h. alle Dokumente dieser Tiefe bearbeitet sind, und die Anzahl der Dokumente mit der Tiefe t deutlich geringer ist als die Anzahl der Dokumente der Tiefe t-1, so ist die Tiefe t-1 als das Maximum anzunehmen.

[16] Ein „dangling“ Link ist ein Hyperlink, der zum Zeitpunkt des Retrievals auf ein nicht vorhandenes Dokument verweist. Dadurch wird beim Abruf des Dokuments ein Fehler auf der Seite des Web-Servers erzeugt (Code 404)

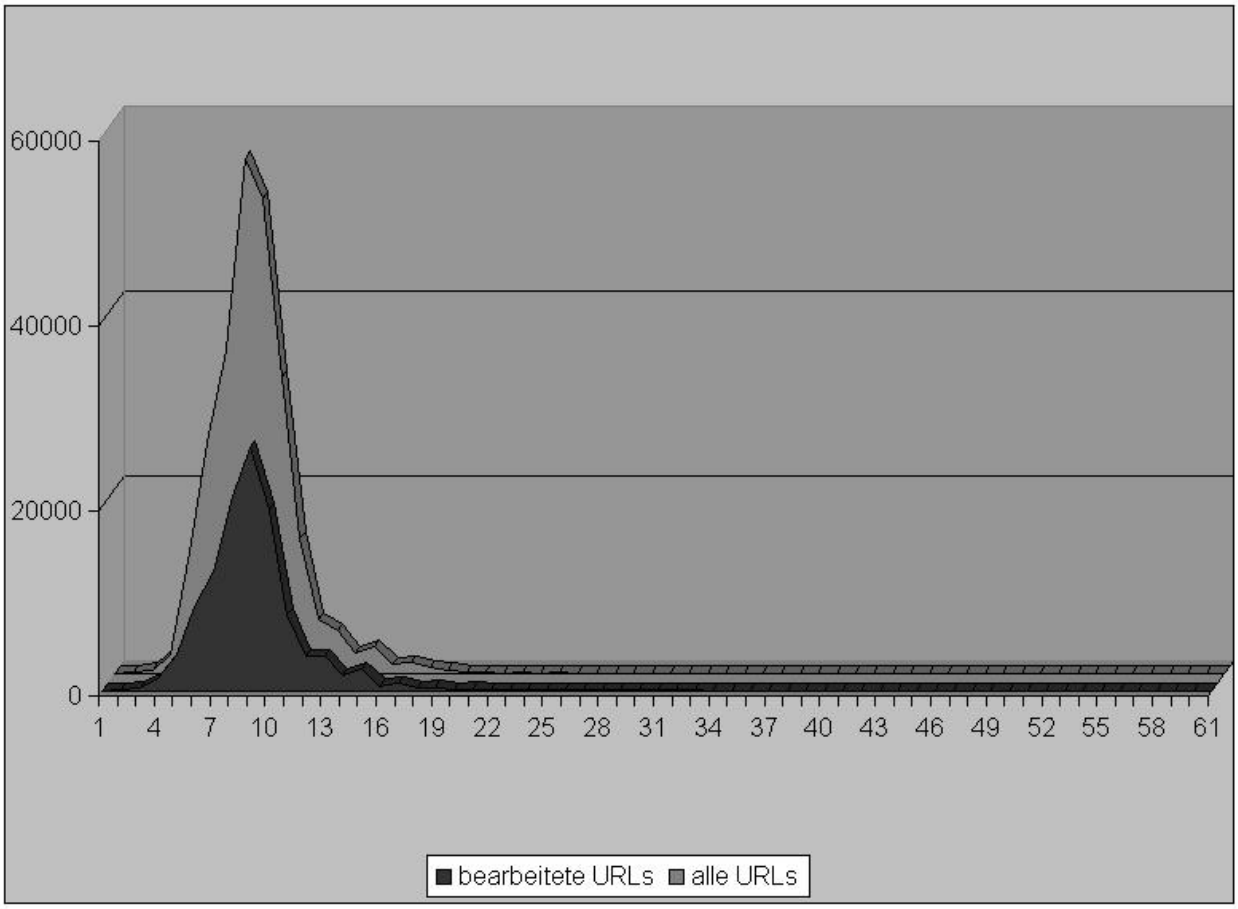

Abbildung 4-5: Verteilung von Hyperlinks für die TU München (Informatik und Mathematik)

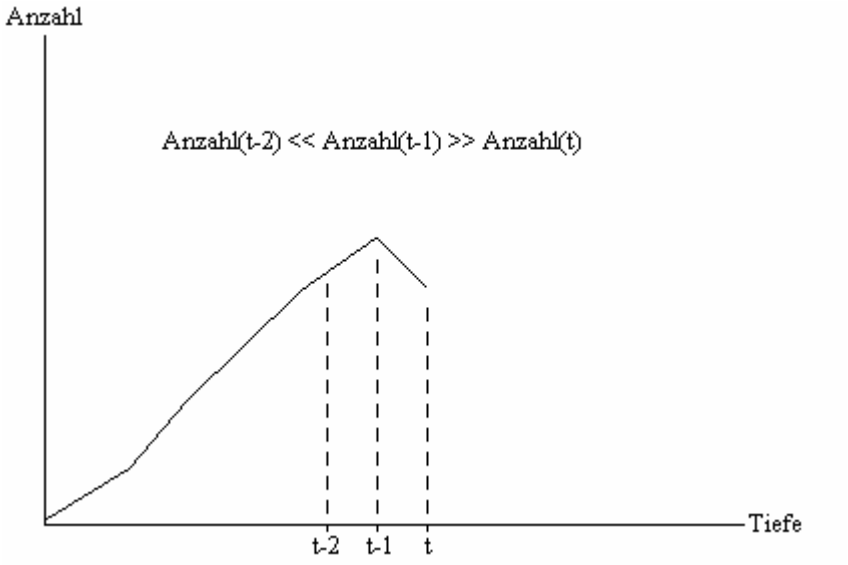

Abbildung 4-6: Bestimmung des Maximums einer Hyperlinkverteilung

In der Praxis hat sich bei allen Tests gezeigt, dass die Anzahl der Dokumente mit der Tiefe **stetig** ansteigt bis zum Erreichen des Maximums. Erst danach treten vereinzelt lokale Maxima auf. Das bedeutet: in 100% der Versuche ist das erste erreichte Maximum auch das absolute Maximum. Ebenfalls in allen Fällen waren die Unterschiede in der Anzahl der Dokumente vom Maximum zu den Tiefen davor und danach sehr deutlich, z.B.

Web-Präsenz der TU München, Informatik und Mathematik, bearbeitete Hyperlinks:

Tiefe	Anzahl
0	1
1	40
2	205
3	1251
4	3822
5	8984
6	12787
7	20993
8	**26248**
9	19538
10	8190

...

- Maximum bei Tiefe 8: 26248 Dokumente
- Unterschied in der Anzahl von Tiefe 8 zu Tiefe 7: 5255 entspricht ca. 25,0 % Anstieg
- Unterschied in der Anzahl von Tiefe 8 zu Tiefe 9: 6710 entspricht ca. 25,6 % Abfall
- Anzahl der Dokumente vor dem Maximum insgesamt: 48083
- Anzahl Dokumente nach dem Maximum insgesamt: 41379

Zum Zeitpunkt des Erreichens des Maximums hätte man die noch zu erwartende Dokumentmenge auf ca. 48000 schätzen können, tatsächlich wurden noch etwas über 41000 Dokumente übertragen. Versuche mit anderen Web-Präsenzen ergaben ähnlich gute Ergebnisse für die Schätzung.

4.3.2.3.5.UNICO – Web-Roboter

Ziel: Volltextindexierung von HTML-Daten der Web-Präsenz einer gewünschten Universität

Bevor überhaupt Daten über eine Suchschnittstelle recherchierbar sind muss zunächst die Möglichkeit geschaffen werden, auf deren Inhalt zuzugreifen. D.h. die Daten müssen aus dem WWW auf einen lokalen Rechner kopiert werden, bevorzugt in ein „volltextfähiges“ Datenbanksystem. Gemeint sind Datenbanksysteme, die einen Volltextindex aus vorhandenen Dokumenten selbständig generieren können. Ein Volltextindex bedeutet dabei eine Zuordnung von Worten (später entsprechen diese den Suchbegriffen) zu Dokumenten und wird auch „invertierte Liste“ genannt. In den meisten kommerziellen Datenbanksystemen sind entsprechende Werkzeuge zur Erzeugung eines Volltextindex integriert oder als Erweiterungen verfügbar. In manchen kleineren oder frei verfügbaren Datenbanksystemen fehlt diese

Möglichkeit jedoch, so dass man den Volltextindex „händisch“ erstellen muss. Grob betrachtet gliedert sich die Problemstellung folglich in zwei Hauptaufgaben:

Übertragung der „gewünschten“ Daten:
Die Schwierigkeit liegt in dem Begriff „gewünscht“: man möchte sich innerhalb einer Web-Präsenz bewegen und **nur** die dazu gehörigen Daten in die eigene Datenbank kopieren. Das bedeutet, dass man zuvor entsprechende Einschränkungen auf bestimmte IP-Domänen oder/und –Adressen machen muss. Dazu kommen ein „gutes Benehmen“ der Übertragungskomponente gegenüber den zuständigen Web-Servern und die Behandlung des Eindeutigkeitsproblems von URLs.

Erstellung des Volltextindex:
Ein entscheidender Schritt vom Browser zum Roboter ist die automatische Erkennung von Hyperlinks im HTML-Text. Diese müssen aus dem HTML-Dokument extrahiert werden und mit den Einschränkungen der Web-Präsenz abgeglichen werden. So kann entschieden werden, welche Hyperlinks weiterverfolgt werden und welche nicht. Bevor man das Dokument zur Volltextindexierung gibt, muss noch der „sichtbare“ Text von den HTML-Tags getrennt werden. Dabei kann man auch noch weitere Meta-Information zu dem Dokument erzeugen, wie z.B. Größe, Datum der letzten Änderung.

4.3.2.3.6.Datenübertragung

Diese Komponente erhält als Eingabe einen Hyperlink und überträgt die zugehörigen Daten aus dem WWW. Da dieser Vorgang in Abhängigkeit von der Internetverbindungsqualität, der Leistung des Web-Servers und der Größe der Datenmenge sehr schnell gehen kann, muss sichergestellt werden, dass die Kapazität des Web-Servers durch fortwährende Anfragen von HTML-Dokumenten nicht überfordert wird. Das würde dazu führen, dass Anfragen durch andere Anwender sehr langsam oder gar nicht mehr beantwortet würden. Dieser Vorgang ist bekannt unter der Bezeichnung „rapid fire“ oder allgemein „Denial of Service“ - Angriff und wird von „Hackern“ dazu missbraucht, Web-Server lahm zu legen. Eine Beschränkung der Anfragen des Web-Roboters und somit der auch Serverlast ist deshalb notwendig.

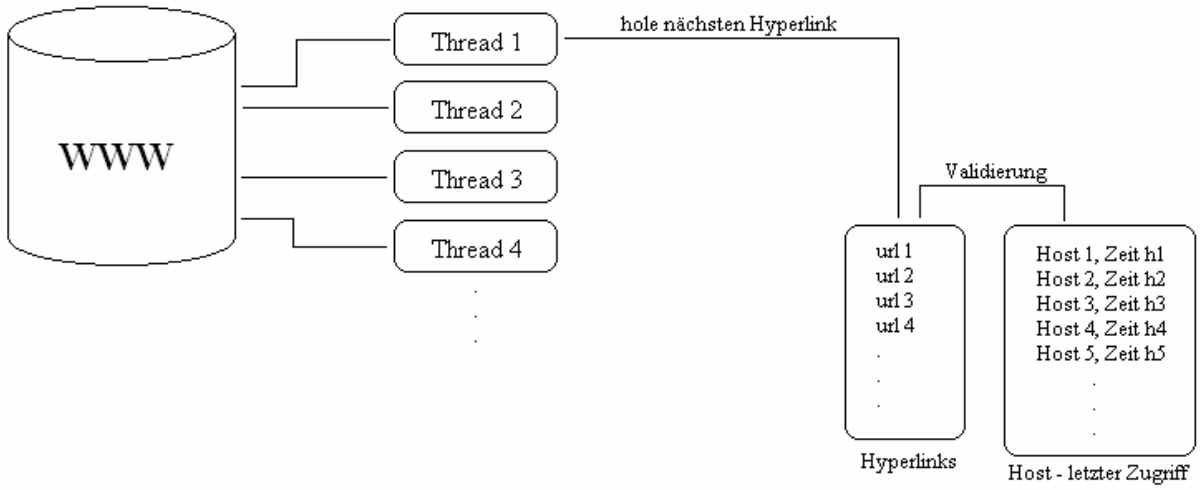

Abbildung 4-7: Zeitliche Verzögerung beim Zugriff auf Web-Server

Das gehört nicht nur zum „guten Benehmen“ der Web-Roboter, sondern ist notwendig, um Gegenmaßnahmen von Web-Server-Administratoren zu vermeiden, wie z.B. Sperren der IP-Adresse oder sogar der Domäne, von der aus der „Angriff“ erfolgt ist. Die Organisation dieser Einschränkung ist im UNICO - Web-Roboter wie in Abbildung 4-7 realisiert. Der UNICO – Web-Roboter realisiert die Übertragung von Daten aus dem WWW durch parallel arbeitende Teilkomponenten (Threads). Jede dieser Komponenten hat Zugriff auf eine Liste, in der die Hyperlinks der zu übertragenden Daten festgehalten sind. Der Zugriff erfolgt exklusiv, um Redundanzen zu vermeiden. Wenn ein Thread „arbeitslos“ ist, versucht er Zugriff auf die Hyperlinkliste zu erhalten. Er wird auf den Zugriff warten, sollte die Liste durch einen anderen Thread blockiert sein. Wenn er den Zugriff erhält, liest er den ersten Hyperlink der Liste aus und bestimmt den zugehörigen Host, so ist z.B. zu http://www.in.tum.de/index.html der Host: www.in.tum.de oder 131.159.24.107. Aus einer Hashtabelle wird dann die Zeit des letzten Zugriffs auf diesen Host ermittelt. Der Zugriff darauf erfolgt ebenfalls exklusiv. Hier ergeben sich drei Möglichkeiten:

Gegeben:
Toleranz x = Zeit in ms, die zwischen zwei Anfragen auf denselben Web-Server minimal liegen muss

Kein Eintrag für diesen Host. Aktion: Der Hyperlink wird übertragen und der Host zusammen mit dem Zeitpunkt **nach** der Übertragung wird in die Hashtabelle eingetragen.
Die Zeit für den letzten Zugriff liegt mindestens x ms vor dem aktuellen Zeitpunkt. Aktion: Der Hyperlink wird übertragen und der Eintrag für diesen Host wird **nach** der Übertragung aktualisiert.
Die Zeit für den letzten Zugriff liegt nicht mindestens x ms vor dem aktuellen Zeitpunkt. Aktion: Nächsten Hyperlink aus der Liste testen oder frühesten Zeitpunkt der Übertragung abwarten.

So ist sichergestellt, dass ein Host bzw. Web-Server im schlechtesten Fall alle x ms eine Anfrage durch den Web-Roboter erhält. Ein üblicher Wert für x ist 60000, das entspricht also einer Mindestwartezeit von einer Minute für jeden Web-Server. Das „Worst-Case“ - Szenario ist also in diesem Fall eine Web-Präsenz, die aus einem einzigen Web-Server besteht. Dann könnte man lediglich ein Dokument pro Minute übertragen. Der Idealfall wäre eine Web-Präsenz, deren Daten auf viele Server verteilt sind. Anzumerken ist auch noch, dass nicht für jeden Server der gleiche Toleranzwert sinnvoll ist. Es gibt sehr schnelle Web-Server, die z.B. auf mehrere Rechner verteilt sind, da sie ein enormes Datenaufkommen handhaben müssen: Beispiel: http://www.microsoft.com. Ebenso ergab sich bei Testläufen des gerade beschriebenen Verfahrens, dass für manche Server 10 Anfragen pro Minute einem „Denial of Service“ – Angriff entsprechen. Auch die Tageszeit in der Zeitzone, in der sich ein Web-Server befindet, spielt eine große Rolle für das Datenaufkommen. Für kommerzielle und wissenschaftliche Webseiten ist zwischen 9 Uhr und 19 Uhr die „Hauptverkehrszeit“, somit ist es für Web-Roboter günstiger in den Nachtstunden die Anfragen zu stellen, da die Web-Server dann über mehr freie Resourcen verfügen. Die jeweilige Zeitzone ist dabei natürlich zu beachten.

4.3.2.3.7.Beschränkungen durch den Web-Seiten-Betreiber

Ein typisches Problem der Betreiber privater Homepages besteht darin, dass nach der Indexierung ihrer Homepage für eine Internet-Suchmaschine auf eine Suchanfrage hin alle möglichen Teilstücke der Homepage (einzelne Frames oder Seiten ungleich der Einstiegsseite) als Treffer liefern, jedoch nicht die Hauptseite. Ein extremes Beispiel ist in Abbildung 4-8 zu

sehen: eine Einstiegsseite einer Homepage bestehend aus zwei Frames. Gewünscht wäre, dass bei einer Suchanfrage bei der indexierenden Suchmaschine die

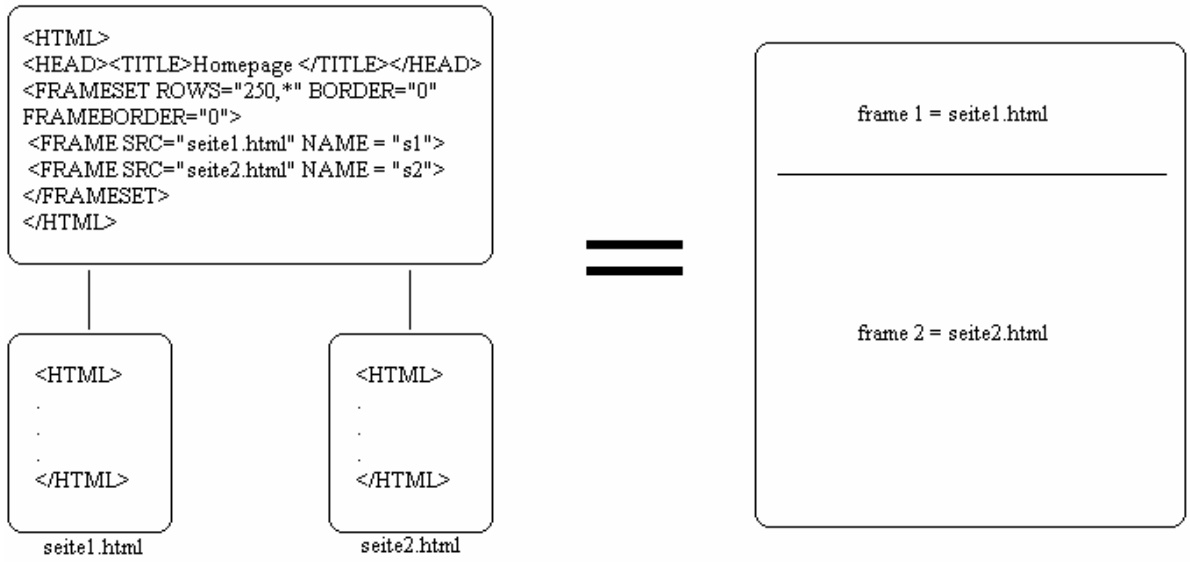

Abbildung 4-8: HTML-Dokument bestehend aus zwei Frames

Einstiegsseite als Treffer geliefert wird. Leider ist dort kein Text vorhanden, so dass dieser Fall nicht eintreten wird. Als Treffer werden im Allgemeinen die Teilstücke, also Frame1 oder Frame2 geliefert. Ein erster Schritt zur Abhilfe ist das HTML-Meta-Tag „keywords“. Der Inhalt dieses Tags wird von allen gängigen Suchmaschinen indexiert, obwohl es sich um keinen sichtbaren Taxt handelt.

Beispiel:

```
<META NAME=“KEYWORDS“ CONTENT=“datenbank suchmaschine [...]“ >
```

Als Ergebnis erhält man jetzt zu einer Suchanfrage „datenbank“ die Einstiegseite sowie alle Seiten, die diesen Begriff ebenfalls enthalten. Um zu erreichen, dass nur die Einstiegsseite als Treffer erscheint, muss man dem Web-Roboter der Suchmaschine mitteilen, welche Seiten indexiert werden sollen und welche nicht. Das geschieht über das Meta-Tag „ROBOTS“.

Beispiel:

```
<META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
```

„NOINDEX“ bedeutet hierbei, dass die Seite, die dieses Tag enthält nicht indexiert werden soll. „NOFOLLOW“ zeigt an, dass keiner der enthaltenen Hyperlinks dieser Seite ausgewertet und somit verfolgt werden soll. Um den gewünschten Effekt zu erzielen, sollte also der Betreiber der Beispiel-Homepage die entsprechenden „KEYWORDS“ per Meta-Tag einfügen und dem Web-Roboter der Suchmaschine mittels dem „ROBOTS“ - Meta-Tag mit Inhalt „NOINDEX“ mitteilen, dass lediglich die Einstiegsseite zu indexieren ist.
Das bedeutet auf der anderen Seite, dass sich Web-Roboter an diese Wünsche der Web-Seiten-Autoren halten sollten. Eine weitere Möglichkeit zur Kommunikation von Web-Administratoren

und Suchmaschinenbetreibern bietet eine Datei „ROBOTS.TXT“. Die dort enthaltenen Einschränkungen beziehen sich auf den gesamten Web-Server. Häufiges Einsatzgebiet sind dabei Verzeichnisse mit dynamisch generierten HTML-Dokumenten.

Beispiel:

Verbietet die Indexierung der Verzeichnisse /cgi-bin, /tmp und /~joe für alle Web-Roboter:

```
User-agent: *
Disallow: /cgi-bin/
Disallow: /tmp/
Disallow: /~joe/
```

Hier ist einem Web-Roboter der gesamte Server zur Indexierung verboten:

```
User-agent: BadBot
Disallow: /
```

Eine ausführliche Beschreibung der Robot-Exclusions findet man unter [http://www.robotstxt.org/]. Die Datei liegt dabei immer im Wurzelverzeichnis des Web-Servers, also z.B. http://www.in.tum.de/robots.txt. Tests mit dem UNICO - Web-Roboter haben ergeben, dass weniger als 5% der getesteten Web-Server dieses Mittel einsetzen. Beide Möglichkeiten werden vom UNICO - Web-Roboter unterstützt, d.h. ausgewertet und die gebotenen Regeln befolgt.

4.3.2.3.8.Eindeutigkeit von Hyperlinks

Folgendes Problem ergibt sich für alle Konstrukteure eines Web-Roboters: Eine Kopie der Web-Seite http://www.in.tum.de wird aus dem WWW in die eigene Datenbank gelegt. Im Laufe der Bearbeitung der enthaltenen Hyperlinks, weiterer Dokumente und wiederum deren Hyperlinks (transitive Hülle des Web-Graphen) erhält man den Hyperlink http://www.in.tum.de/index.html. Dieses Dokument wurde bereits geholt, jedoch unter einer anderen Adresse. Weitere Synonyme für dasselbe Dokument sind z.B. http://www.cs.tum.de/, http://www.informatik.tu-muenchen.de. Das bedeutet, dass es mehrere Namen für ein und dasselbe Dokument geben kann. Ein erster Schritt zur Lösung ist die Auflösung der namentlichen IP-Adresse: alle gerade genannten Adressen würden reduziert auf zwei: http://131.159.24.107/ und http://131.159.24.107/index.html. Dazu kommt die Möglichkeit, dass der Web-Server die Anfragen auf mehrere Rechner und damit auch auf mehrere IP-Adressen verteilt:

Beispiel:

Hinter www.microsoft.com verbergen sich folgende Rechner zum Zeitpunkt der Anfrage: 207.46.197.100, 207.46.230.219, 207.46.197.102, 207.46.197.113, 207.46.230.220, 207.46.230.218

Das bedeutet, dass die Einschränkung auf numerische IP-Adressen in der Praxis gute Chancen bietet, redundante Daten in der Datenbank zu vermeiden, aber keine wirklich ausreichende Sicherheit bieten. Man möchte dem Anwender der Suchmaschine nicht fünf Mal dieselbe Seite als Treffer liefern. Im UNICO - Web-Roboter gibt es einen dritten Test: Es wird aus jedem übertragenen HTML-Dokument ein 40-stelliger Hashcode generiert, zusammengesetzt aus Dokumentgröße, Anzahl der Hyperlinks, Zeichen an festgelegten Positionen. Vor dem Einfügen

eines neuen Dokuments in die Datenbank wird überprüft, ob der gleiche Hashcode bereits vorhanden ist. So kann im schlechtesten Fall ein Dokument zwar mehrfach in die Datenbank eingetragen werden, aber nur, wenn sich während der Laufzeit des Web-Roboters dessen Inhalt und somit auch sein Hashcode verändert hat. Die Wahrscheinlichkeit, dass zwei Dokumente den gleichen Hashcode besitzen, obwohl sie völlig verschiedene Dokumente sind, ist in Betracht der 40 Stellen des Hashcodes sehr gering. In der Praxis des UNICO – Web-Roboters hat sich diese Methode als zuverlässig erwiesen.

Ein weiteres Problem ist die Zyklenbildung durch relative Adressierung von HTML-Dokumenten. Dieses Phänomen wird auch als „Schwarzes Loch“ bezeichnet.
Beispiel:

Das Dokument „http://www.test.com/index.html“:

```
<HTML>
<A HREF=“./index.html“> Diese Seite </A>
<HTML>
```

Beim Zusammensetzen der enthaltenen Hyperlinks erhält man folgenden Hyperlink:

Iteration 1: http://www.test.com/./index.html

Wird dieses (dasselbe unter anderem Hyperlink) Dokument übertragen, erhielte man einen Verweis auf dasselbe Dokument wieder mit anderem längerem Hyperlink. Diesen Vorgang kann man beliebig fortsetzen, wenn man sich nur auf die Eindeutigkeit des Hyperlinks verlässt:

Iteration 2: http://www.test.com/./.index.html
Iteration 3: http://www.test.com/./.index.html
...
Zweifellos liefern diese Hyperlinks dasselbe Dokument immer wieder, aber mittels des 40-stelligen Hashcodes erkennt man diese Adresse als „Dublette“. Ohne den Hashcode würde der Web-Roboter dieselbe Seite wieder und wieder übertragen und die Datenbank eintragen, da die Adresse (auch die numerische) jedes Mal verschieden ist. Es ist jedoch sinnvoll, diese unnütze Arbeit grundsätzlich auszuschließen und eine derartige Zyklenbildung bei der Generierung der Hyperlinks zu erkennen. Der UNICO – Web-Roboter erkennt diese Zyklen und eliminiert sie bei der Bildung der Hyperlinks. Das Gleiche gilt auch für Zyklen mit übergeordneten Verzeichnissen (http://www.test.com/test/../test/../index.html).

4.3.2.3.9.Formate

Sinnvoll für einen Aufbau eines Volltextindex sind natürlich nur Dokumente, die Text enthalten. Daher ist notwendig, dass sich der Web-Roboter auf Text-Dokumente beschränkt. Diese Aufgabe ist nicht immer leicht zu lösen, da mancher Hyperlink z.B. eine Bilddatei liefert oder gepackte Daten. Auch Datei - Endungen sind diesbezüglich nicht immer verlässlich. Das bedeutet, dass der Web-Roboter nur bestimmte MIME[17]-Typen akzeptieren soll. Ideal wäre es, wenn vor der Datenübertragung bekannt ist, ob es sich um einen gewünschten Typ handelt. Diese Art Voraussage ist schwierig, manche Server bieten Hilfestellung und man kann den Typ vor der Übertragung abfragen. Im schlimmsten Fall überträgt man Megabytes von Daten, nur um

[17] Multipart Internet Mail Extensions

anschließend festzustellen, dass es sich z.B. um eine gezippte Datei handelt. In der Praxis des UNICO – Web-Roboters beläuft sich das Transfervolumen von nicht benötigten Daten auf weniger als 1%, d.h. die Retrieval-Komponente erwartet ein HTML-Dokument und es wird vom betroffenen Web-Server eine Bild-Datei (häufigster Fall) gesendet.

4.3.2.3.10.Verarbeitung der Dokumente

Sobald die Daten zu einem Hyperlink übertragen worden sind, werden sie zur weiteren Bearbeitung in einer Liste abgelegt. Eine weitere Menge von Threads ist für die Aufbereitung der Dokumente zuständig. Dieser Vorgang soll an einem Beispiel illustriert werden:

Ansicht des Test-Dokuments in einem Web-Browser:

Mail an Webmaster

Test-Dokument: Hyperlink http://www.testbg.com/test/test.html
Der HTML-Quelltext zu dem gezeigten HTML-Dokument:

```
<HTML>
<META NAME=“KEYWORDS“
CONTENT=“Test Tester Testen Testbetrieb Testseite“>
<META NAME=“ROBOTS“ CONTENT=“NOFOLLOW“>
<HEAD>
<TITLE>Testseite</TITLE>
</HEAD>
<BODY BGCOLOR="#FBFBFF">
<A HREF="http://www.testbg.com">
<IMG ALT="[Test Logo]" BORDER=0 SRC="/img/test_logo.gif">
</A><FONT SIZE=2 FACE="Helvetica">
<BR><I><B>
Testseiten für Testzwecke   -  
Institut für Testservice
<FONT COLOR="#00CC00">
<B>T</B></FONT>est
<FONT COLOR="#CC0000"><B>B</B>
</FONT>etriebs
<FONT COLOR="#0000CC">
<B>G</B></FONT>ruppe
</B></I>
<HR NOSHADE SIZE=1>
</FONT>
<BR CLEAR="all">
<P>
<CENTER>
<FONT FACE="Helvetica" SIZE="+4">
<B>Testbetriebsgruppe</B>
</FONT>
<BR><BR>
<FONT FACE="Helvetica" SIZE="+1">
<B>
Informationen zum Testbetrieb im Bereich des Lokalen Netzwerks (LAN) der
Testbetriebsgruppe
</B>
</FONT>
</CENTER>
<BR><BR><BR><BR>
<FONT FACE="Helvetica" SIZE="+1">
<UL>
<LI><B>
<A HREF="kennung/">Teilnahme am Testbetrieb</A>
</B><BR><BR>
<I>Benutzerkennungen, Zugang zum LAN mit dem eigenen Rechner,
Benutzungsrichtlinien, ...</I><BR></LI><BR><BR>
<LI><B>
<A HREF="lan/">
Hinweise zur Nutzung des LAN des Tests
</A></B><BR><BR>
<I>Öffnungszeiten, E-Mail, Plattenplatz, Drucken, ...</I>
<BR></LI>
<BR><BR>
<LI><B>
<A HREF="kontakt/">Ansprechpartner und Formulare</A>
```

```
</B><BR><BR>
<I>WWW, E-Mail, Kennungen, Mailverzeichnis, Hardware-Wartung, IP-
Addreßvergabe, ...</I>
<BR></LI>
</UL>
</FONT>
<FONT SIZE=2 FACE="Helvetica">
<HR SIZE=1 NOSHADE>
<DIV ALIGN=RIGHT>
<A HREF="mailto:webmaster@www.testbg.com?subject=Test-Email">
Mail an Webmaster
</A><BR>
</DIV>
</FONT>
</BODY>
</HTML>
```

Zunächst muss der textliche Inhalt aus dem Dokument extrahiert werden. Dazu ist ein HTML – Parser nötig, der den HTML-Quellcode in eine Baumstruktur umsetzt:

```
<HTML>
 <META>
   KEYWORDS CONTENT="Test Tester Testen Testbetrieb Testseite"
 </META>
 <META>
   ROBOTS CONTENT="NOFOLLOW"
 </META>
 <HEAD>
   <TITLE>
        Testseite
   </TITLE>
 </HEAD>
 <BODY ABGCOLOR="#FBFBFF">
   <A AHREF CDATA http://www.testbg.com>

      <IMG   AALT CDATA [Test Logo]
        ABORDER CDATA 0
        ASRC CDATA /img/test_logo.gif>
      </IMG>
   </A>
   <FONT ASIZE CDATA 2
      AFACE CDATA Helvetica>
   <BR>
   <I>
      <B>
            Testseiten für Testzwecke - Fakultät für Testservice
.
.
.
 </BODY>
</HTML>
```

Hervorgehoben ist der Text, der für einen Anwender mit einem HTML-Browser sichtbar ist. Es ist sehr viel strukturelle Information enthalten, wie z.B. Formatierungsangaben. Wichtig für die Volltextindexierung sind jedoch der sichtbare Text und die Meta-Informationen über das Dokument:

Titel: `Testseite`
Sichtbarer Text:

```
Testseiten für Testzwecke – Institut für Testservice Test BetriebsGruppe
Testbetriebsgruppe Informationen zum Testbetrieb im Bereich des lokalen
Netzwerks (LAN) der Testbetriebsgruppe Teilnahme am Testbetrieb
Benutzerkennungen, Zugang zum LAN mit eigenem Rechner, Benutzungsrichlinien,
... Hinweise zur Nutzung des LAN des Tests Öffnungszeiten, E-Mail,
Plattenplatz, Drucken, ... Ansprechpartner und Formulare WWW, Email,
Kennungen, Mailverzeichnis, Hardware-Wartung, IP-Adreßvergabe, ... Mail an
den Webmaster
```

Meta-Keywords: `Tester Testen Testbetrieb Testseite`

Diese drei Teilstücke bilden die Grundlage für den Text, der später in den Volltextindex eingehen soll. Zuvor werden aber noch überflüssige Zeichen wie z.B. Klammern oder Interpunktionszeichen entfernt. Ebenso werden deutsche Umlaute ersetzt, z.B. ä = ae, und für den Text wird einheitlich Kleinschreibung festgelegt. Ergebnis:

```
testseite testseiten fuer testzwecke institut fuer testservice test
betriebsgruppe testbetriebsgruppe informationen zum testbetrieb im bereich
des lokalen Netzwerks lan der testbetriebsgruppe teilnahme am testbetrieb
benutzerkennungen zugang zum lan mit eigenem rechner benutzungsrichlinien
hinweise zur nutzung des lan des tests oeffnungszeiten e-mail plattenplatz
drucken ansprechpartner und formulare www email kennungen mailverzeichnis
hardware-wartung ip-adressvergabe mail an den webmaster tester testen
testbetrieb testseite
```

Der letzte Schritt der Verarbeitung beinhaltet die Reduktion des Textes durch das Entfernen von Stopwörtern.
Ein *Stopwort* ist ein Wort, das keine inhaltliche Bedeutung besitzt oder keinerlei inhaltliche Bedeutung zum Text hinzufügt. Letzteres meint Begriffe, die in einem vorgegebenen Kontext keinen Gewinn an Information enthalten. So kann z.B. der Kontext die Web-Präsenz einer Universität sein, dann bedeutet das Vorkommen des Begriffs „Universität“ für die zugehörigen Dokumente keinen Gewinn an Information. Wenn der Kontext z.B. die Web-Präsenz der Firma Siemens ist, so bedeutet das Vorkommen des Begriffs „Universität“ für die entsprechende Seite eine eventuell wichtige Zusatzinformation. Diese Form der Stopwörter ist jedoch zum aktuellen Zeitpunkt der Bearbeitung nicht zu bestimmen, da der Gesamtkontext noch nicht bekannt ist. Beispiele für Wörter ohne inhaltliche Bedeutung sind „und“, „oder“, „als“, „wenn“, „ich“, „du“, „er“, „sie“, „es“, „ein“, „eine“, „einer“, „eines“ usw. Zur Bestimmung der Stopwörter werden im UNICO – Web-Roboter Listen verwendet, für jede Sprache eine eigene Liste (als Hashtabelle). Jeder Begriff des Dokuments (nach der Vorverarbeitung) wird mit den Stopwortlisten abgeglichen und das Wort gegebenenfalls entfernt. Zusätzlich wird für jede Liste ein Zähler erhöht, wenn daraus ein Stopwort im Text gefunden wurde. Als Ergebnis erhält man eine relativ gute Vorhersage, in welcher Sprache das Dokument verfasst ist. Der UNICO – Web-Roboter verwendet aktuell eine deutsche Stopwortliste mit ca. 3.000 Einträgen und eine englische mit etwa 2.500 Einträgen.

Erkennen der Stopwörter:

```
testseite testseiten fuer testzwecke institut fuer testservice test
betriebsgruppe testbetriebsgruppe informationen zum testbetrieb im bereich
des lokalen Netzwerks lan der testbetriebsgruppe teilnahme am testbetrieb
benutzerkennungen zugang zum lan mit eigenem rechner benutzungsrichlinien
hinweise zur nutzung des lan des tests oeffnungszeiten e-mail plattenplatz
drucken ansprechpartner und formulare www email kennungen mailverzeichnis
hardware-wartung ip-adressvergabe mail an den webmaster tester testen
testbetrieb testseite
```

Ergebnis nach dem Entfernen der Stopwörter:

```
testseite testseiten testzwecke institut testservice test betriebsgruppe
testbetriebsgruppe informationen testbetrieb bereich lokalen Netzwerks lan
testbetriebsgruppe teilnahme testbetrieb benutzerkennungen zugang lan eigenem
rechner benutzungsrichlinien hinweise nutzung lan tests oeffnungszeiten e-
mail plattenplatz drucken ansprechpartner formulare www email kennungen
mailverzeichnis hardware-wartung ip-adressvergabe mail webmaster tester
testen testbetrieb testseite
```

Entfernte Stopwörter: 15, davon 100% aus der deutschen Stopwortliste

Die Reihenfolge der Wörter bleibt unverändert, ebenso werden mehrfach vorkommende Begriffe nicht auf ein Vorkommen reduziert. Das ist damit zu begründen, dass im Volltextindex Positionsinformationen festgehalten werden, um z.B. eine Nähe-Suche zu realisieren, d.h. Anfragen, die nur dann erfolgreich sind, wenn ein Dokument gefunden wird, das zwei Begriffe enthält, die in einem bestimmten Abstand voneinander stehen müssen. Die Positionsinformationen müssen aus diesem Grund erhalten bleiben. Durch das Entfernen der Stopwörter ist diese Information bereits verfälscht, jedoch hat das nur in Ausnahmefällen Auswirkungen auf die Ergebnisse bei entsprechenden Anfragen.

Zusätzlich zu dem Textstück werden folgende Zusatzinformationen aus dem Dokument gewonnen:

- Dokumentgröße
- Sprache
- Hashcode (40-stellig)
- Datum der letzten Änderung
- Titel
- Liste von Hyperlinks

4.3.2.3.11.Extraktion von speziellen Informationen

Ausgehend von der ursprünglichen Form der UNICO – Datenbank stammt die Forderung, zu einem bestimmten Suchbegriff nicht nur inhaltliche Information finden zu können, sondern Möglichkeiten zur Kommunikation, zur Kontaktaufnahme, mit dem Autor der Information. Bei Dokumenten im WWW ist diese Information nicht immer leicht zu finden, nur selten findet man folgenden Eintrag in einem HTML-Dokument:

```
<META name=“Author“ content=“Karl Vielschreiber“>
```

Zusätzlich wäre auch eine Anschrift, Telefonnummer oder zumindest eine E-Mail-Adresse wünschenswert. Im UNICO – Web-Roboter wird eine einfache Methode eingesetzt, um heraus zu finden, ob ein Dokument Adressinformationen beinhaltet. Dazu werden deutsche und englische Begriffe, die auf derartige Informationen schließen lassen, im Text gesucht und deren Vorkommen gezählt.

Deutsche Begriffe:

```
anschrift, adresse, strasse, postleitzahl, plz, tel, telephon, telefon, email, mail, ort, raum, titel, vorname, nachname, name, familienname, fax, telefax, ...
```

Mehrfaches Vorkommen des gleichen Begriffs wird nicht mitgezählt, da sonst z.B. Telefonlisten von Mitarbeitern zu hohe Werte erreichen. Es hat sich gezeigt, dass bei einem Auftreten von mindestens vier dieser Begriffe das Dokument bei einer Erfolgsquote von 80% relevante Adressinformationen enthält. Komplizierte Berechnungsmethoden mit einer Gewichtung der Begriffe haben bei Tests keine wesentlichen Steigerungen der Quote erzielen können. Zwei Arten von Fehlern sind festzustellen:

- Es werden viele Adressinformationen im Dokument gefunden, jedoch enthält das Dokument keine einzige Adresse, Telefonnummer oder E-Mail-Adresse. Dieser Fall tritt auf z.B. bei Beschreibungen von Internetzugängen, da Begriffe wie „email“, „tel“, „telefon“, „name“, „ort“ häufig im Text vorkommen.

- Es werden Adressinformationen im Dokument gefunden, jedoch betreffen sie nicht den Autor des Dokumentinhalts.

Stichproben aus großen Dokumentmengen haben ergeben, dass Dokumente, die weniger als vier der oben genannten Begriffe enthalten, nur in etwa 6% der Fälle doch wichtige Kontaktinformationen beinhalteten.

4.3.2.3.12.Extraktion von Hyperlinks

Der entscheidende Punkt für die Selbständigkeit eines Web-Roboters ist seine Fähigkeit, aus Hypertext-Dokumenten Referenzen auf andere Dokumente zu erkennen, auszuwerten und weiterzuverfolgen. Ein Hyperlink im obigen Beispieldokument:

```
<A HREF="kennung/">Teilnahme am Testbetrieb</A>
```

Es handelt sich um das „Anchor“-Tag, dessen Attribut „HREF“ das Zieldokument beschreibt, das geladen wird, wenn man z.B. innerhalb eines Web-Browers auf den zum Anker gehörenden Text „Teilnahme am Testbetrieb“ klickt. Eine ausführliche Beschreibung aller Tags von HTML findet man unter http://www.w3.org/MarkUp/. Hyperlinks im WWW sind unidirektional, d.h. für ein Dokument im WWW kann man nicht feststellen, welche anderen Dokumente dieses referenzieren, außer man kennt alle Dokumente und kann somit alle Verbindungen rekonstruieren. Die Referenzen eines Dokuments sind interessant als Parameter für Bewertungsverfahren von WWW-Dokumenten wie z.B. Google’s PageRank-Algorithmus (s. 3.5.1). Man ist dabei natürlich auf die Dokumente beschränkt, die bereits ausgewertet worden

sind. Zusätzliche Information bietet der Ankertext, da der Autor einer Web-Seite dem Leser in der Regel damit inhaltliche Information über das referenzierte Dokument vermitteln möchte. Bei umfangreichen Tests mit dem UNICO – Web-Roboter hat sich gezeigt, dass in etwa 50% der Fälle wirklich relevante Information zu einem Dokument durch den Ankertext gewonnen wird. Suchmaschinen wie Google nutzen den Ankertext auch, um die Suche auf Dokumente zu erweitern, deren Volltext noch gar nicht indexiert wurde. Wenn also eine Seite mit dem Beispiel-Hyperlink ausgewertet würde, dann erhielte man zu dem Suchbegriff „Testbetrieb“ auch die Seite als Treffer, die durch HREF=“kennung/“ referenziert ist. Das ist auch dann der Fall, wenn der Inhalt der Seite noch nicht in den Volltextindex der Datenbank aufgenommen worden ist.

Der UNICO – Web-Roboter wertet folgende Information zu Hyperlinks aus:

- Ankertext
- Rückwärtsreferenz
- Vervollständigung des Hyperlinks

Ein Hyperlink auf ein HTML-Dokument hat z.B. folgendes Format:

http://www.test.com/test/index.html

Vollständige Informationen zu Hyperlinks findet man unter http://www.ietf.org/rfc/rfc2396.txt. Innerhalb von HTML-Seiten muss die Angabe einer Referenz auf ein anderes Dokument nicht absolut erfolgen, sondern kann auch relativ zur Position des referenzierenden Dokuments angegeben werden. Beispiel:

Ausgangsdokument: http://www.test.com/test/index.html

Hyperlink im Ausgangsdokument	Vervollständigter Hyperlink
kennung/	http://www.test.com/test/**kennung/**
../index.html	http://www.test.com/**index.html**
http:/test2.html	http://www.test.com/test/**test2.html**
http://www.testbetrieb.com/index.html	**http://www.testbetrieb.com/index.html**
/index.html	http://www.test.com/**index.html**

Zu beachten ist die Zusammensetzung des vollständigen Hyperlinks bei Auftreten von „../“ und „./“. Diese verzeichnisrelativen Angaben müssen eliminiert werden, da es sonst zu einer Zyklenbildung kommen kann. Beispiel:

http://www.test.com/test/index.html + ../index.html	Ausgangsseite + Hyperlink
http://www.test.com/test/ + ../index.html	Aktuelles Verzeichnis
http://www.test.com/test/../index.html	Zusammenfügen
http://www.test.com/index.html	test/../ wird eliminiert

Sobald alle Hyperlinks vervollständigt sind, wird geprüft, ob deren weitere Bearbeitung gewünscht, d.h. ob sie sich innerhalb der zu bearbeitenden Web-Präsenz befinden [s. Struktur einer Web-Präsenz, 4.3.2.3.4]. Die Beschränkungen einer Web-Präsenz als Beispiel:

Erlaubte Domänen

http://100.100.100.*/
http://100.100.101.*/
http://100.100.102.*/
http://100.100.103.*/

Auszuschließende Verzeichnisse

http://www.test.com/cgi-bin
http://www.test.com/~
http://www.test.com/tmp/
http://100.100.101.25/

Für jeden Hyperlink wird zunächst die namentliche IP-Adresse festgestellt und mit den erlaubten Domänen verglichen. Wenn er nicht enthalten ist, wird dieser Hyperlink als **nicht zu verfolgen** gekennzeichnet. Ist er jedoch enthalten, wird überprüft, ob sich der Hyperlink innerhalb der auszuschließenden Verzeichnisse befindet. Ist das der Fall, so wird er ebenfalls als **nicht zu verfolgen** gekennzeichnet. Falls nicht, wird dieser Hyperlink in die Liste der zu bearbeitenden Dokumente eingefügt. Beispiel:

http://www.test2.com/index.html	vollständiger Link
http://100.100.101.25/index.html	numerische Host IP-Adresse bestimmt
http://100.100.101.*/	liegt innerhalb der erlaubten Domänen
http://100.100.101.25/	ist ein auszuschließendes Verzeichnis

Ergebnis: http://www.test2.com/index.html wird nicht weiter verfolgt.

Jedoch wird jeder Hyperlink, der nicht weiter bearbeitet wird, in einer Datenbanktabelle festgehalten. So kann man nach der vollständigen Übertragung einer Web-Präsenz in eine Datenbank feststellen, welche Web-Server häufig referenziert worden sind und bei Bedarf kann man diese zur Übertragung freigeben. Dazu müssen die Einschränkungen der Web-Präsenz entsprechend angepasst werden, z.B. im obigen Beispiel http://100.100.101.25/ aus den auszuschließenden Verzeichnissen entfernen.

In der Datenbank werden die Metadaten zu den Hyperlinks in einer Tabelle festgehalten.
Beispiel:

ID (Schlüssel)	Link	Referenz - Link	Ankertext
0	http://www.a.de/	http://www.b.de/	Seite A
1	http://www.c.de/index.html	http://www.b.de/	Seite C
2	http://www.d.de/test.html	http://www.a.de/	Test
3	http://www.e.de/f/e.html	http://www.c.de/index.htm	
4	http://www.test.com/	http://www.a.de/test.html	Testbetrieb
...	...	...	...

Die Darstellung in der Tabelle ist aus Gründen der besseren Lesbarkeit idealisiert, d.h. die Hyperlinks sind in der Originaltabelle durch Id-Nummern repräsentiert. Wichtig ist, dass Selbstreferenzen nicht eingetragen werden, ebenso werden Mehrfachreferenzen einer Seite auf die gleiche Seite ignoriert. Wenn also das Dokument A ein Dokument B 25 Mal referenziert, wird **eine** Referenz in die Tabelle eingetragen, alle 25 Ankertexte werden zusammengefasst und darin doppelt vorkommende Begriffe eliminiert. Auch Stopwörter werden grundsätzlich aus jedem Ankertext entfernt. Es werden auch Referenzen eingetragen, die keinen Ankertext besitzen, weil z.B. ein Bild als Anker dient oder weil es sich um ein Stopwort handelt. Wichtig ist dann allein die Information, welches Dokument wie häufig und von welchen anderen Dokumenten referenziert wird. Beispiel für einen Hyperlink mit inhaltslosem Ankertext:

```
Informationen zur Teilnahme am Testbetrieb
finden Sie <A HREF="kennung/">hier</A>
```

4.3.2.3.13.Export von Daten

Um die Erzeugung von Daten für das UNICO – Projekt von Suche und dauerhafter Haltung zu trennen, wurde ein Konzept zum Datentransfer entwickelt. Die ermittelten Daten werden in ein festgelegtes Format exportiert und können per Skript in unterschiedliche Informationssysteme eingefügt werden. Somit ist Gewinnung der Daten und die Aufbewahrung der Daten für die Recherche z.B. nicht abhängig von einem bestimmten Datenbanksystem. Der UNICO – Web-Roboter bietet Schnittstellen zu MS Access (für kleinere Web-Präsenzen[18]) und MS SQL-Server, das Siemens Fachinformationszentrum verwendet jedoch Oracle[19] als Relationales Datenbanksystem und FULCRUM als eigenständige Volltextkomponente. Zur Beschreibung der Daten dient XML[20]. Die folgende Dokumenttypdefinition (DTD[21]) eingebettet in ein Beispieldokument soll einen kurzen Überblick geben:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- internal DTD -->
<!DOCTYPE university

 [
  <!ELEMENT university (description, amount, htmldata+)>
  <!ELEMENT description (#PCDATA)>
  <!ELEMENT amount      (#PCDATA)>
  <!ELEMENT htmldata (id, url, title, ranking, date, depth, length,
    keywords, anchortext, data)>
  <!ELEMENT id          (#PCDATA)>
  <!ELEMENT url         (#PCDATA)>
  <!ELEMENT title       (#PCDATA)>
  <!ELEMENT ranking     (#PCDATA)>
  <!ELEMENT date        (#PCDATA)>
  <!ELEMENT depth       (#PCDATA)>
  <!ELEMENT length      (#PCDATA)>
  <!ELEMENT keywords    (#PCDATA)>
  <!ELEMENT anchortext  (#PCDATA)>
```

[18] Es gibt bei Microsoft Access eine Begrenzung der Dateigrösse einer Datenbank auf max. 2 GB
[19] http://www.oracle.com
[20] **E**xtended **M**arkup **L**anguage, eine ausführliche Beschreibung findet sich unter _____
[21] Document Type Definition

```
  <!ELEMENT data        (#PCDATA)>
 ]
>
<university>
 <description>
  Informatik an der TU Muenchen
 </description>
 <amount>
  80
 </amount>
 <htmldata>
  <id>
   0
  </id>
  <url>
   http://www.in.tum.de/
  </url>
  <title>
   TUM Informatik
  </title>
  <ranking>
   25.626
  </ranking>
  <date>
   0
  </date>
  <depth>
   0
  </depth>
  <length>
   1830
  </length>
  <keywords>

  </keywords>
  <anchortext>
   'informatik'[1] 'informatik xii'[1] 'fakultaet  informatik'[6]
  </anchortext>
  <data>
   tum informatik fakultaet informatik technischen universitaet muenchen,
   [...] ankuendigungen lehrveranstaltungen personen raeumen web-seiten
  </data>
 </htmldata>

 <htmldata>
...
</htmldata>
...
</university>
```

4.3.2.3.14.Ergebnisse im Überblick

Die „Pro-Dokument“ - Leistung der Retrieval – Komponente ist von verschiedenen Faktoren abhängig:

- Verbindungsqualität zum Internet
- Leistungsfähigkeit der betroffenen Web-Server

- Durchschnittliche Größe der Dokumente
- Datenbankperformance
- Qualität der Hardware, z.B. CPU-Leistung, Festplatten-Leistung, ...

Hardware im Testeinsatz:

Datenbankserver:
Dual Pentium 3 500 MHz, 512 MB Arbeitspeicher, HDD RAID-0 mit 2 * 100 GB, Microsoft SQL-Server 2000

Web-Roboter-Server:
Dual Pentium 3 1 GHz, 1 GB Arbeitsspeicher, Bandbreite der Internetverbindung: 130 kByte/s downstream, 35 kByte upstream

Bei Tests wurden folgende Web-Präsenzen von Universitäten verwendet:

TU München
LMU München
Georgia Institute of Technology
California Institute of Technology
Massachusetts Institute of Technology
Princeton University
Texas Institute of Technology
Carnegie Mellon
Stanford University
University of California Berkeley

Ergebnisse aller Tests:

- Gesamte Datenmenge: > **15 Mio. Dokumente**.
- Die durchschnittliche Dokumentgröße beträgt etwa 9,8 kByte.
- Zu etwa 9% der Dokumentmenge wurden während des Programmlaufs Dubletten (d.h. verschiedene Hyperlinks für das gleiche Dokument) gefunden.
- Weniger als 4 % der Dokumente enthält META – Informationen („keywords“).
- Jedes Dokument enthielt ca. 25 Referenzen auf andere Dokumente innerhalb seiner Web-Präsenz.
- Jedes Dokument wurde im Schnitt 10 Mal von einem anderen Dokument seiner Web-Präsenz referenziert.
- Die Anzahl der Hyperlinks, die nicht innerhalb der Web-Präsenz liegende Dokumente referenzieren, ist etwa 1,8 mal größer als die Menge der tatsächlich bearbeiteten Hyperlinks. So erhält man z.B. für die Web-Präsenz des Instituts für Informatik und Mathematik bei 115.710 erfolgreich bearbeiteten Dokumenten eine Menge von 132.357 Hyperlinks, die nach „Außen“ verweisen. Das entspricht einem Faktor von „nur“ 1,14. Den Grund für den unterdurchschnittlichen Wert könnte man darin vermuten, dass durch vielfache Tests auf dieser Web-Präsenz die immer wieder veränderten Einschränkungen auf bestimmte Domänen und Web-Server die Menge an „überflüssigen“ Daten reduziert hat. Das betrifft sowohl die in der Datenbank enthaltenen Daten als auch die Menge der zu bearbeitenden Hyperlinks.
- Die Verteilung der Dokumente in Abhängigkeit von ihrer Tiefe ähnelt in allen Tests einer Gauß’schen Glockenkurve.

- Auf der angegebenen Hardware wurden im Durchschnitt etwa 6.500 Dokumente pro Stunde bearbeitet. Als „Flaschenhals“ der Bearbeitung war die Komponente auszumachen, die für die Datenbankeinträge verantwortlich ist.
- Etwa 36% aller Hyperlinks, die übertragen werden sollten, waren fehlerhaft. Häufigste Ursachen hierfür waren:
  40% Bilddateien: gif, jpg, png, usw.
  30% nicht unterstützte Dokumentformate: PDF, PS, TXT, usw.
  11% Dokument nicht vorhanden
  9% Zeitüberschreitung bei der Übertragung
  10% alle restlichen Fehler, hauptsächlich nicht unterstützte Formate: Video, Music, ausführbare oder gepackte Dateien, usw.

4.3.2.4.Evaluation

Wenn alle Dokumente einer Web-Präsenz in die Datenbank übertragen worden sind und der Volltextindex dazu erstellt worden ist, so ist das die Basis für eine Recherche auf diesen Daten. Gibt der Anwender einen oder mehrere Suchbegriffe ein, so liefert ihm der Volltextindex der Datenbank alle Dokumente, die den/die Begriff(e) enthalten. Da jedoch die meisten allgemeinen Suchmaschinen im Internet riesige Mengen von Dokumenten indexiert haben, ist die Wahrscheinlichkeit groß zu einer Suchanfrage sehr viele Treffer zu erhalten. Die Anwender versuchen in der Regel die ersten 10 bis 20 Treffer des Ergebnisses, um an die gewünschte Information zu gelangen. Dabei ist durchaus möglich, dass die gesuchte Seite erst an Stelle 5.000 von z.B. 1 Mio. Treffern liegt. Da hilft selbst die zusätzlich zu jedem Hyperlink in der Trefferliste angezeigte Zusatzinformation nicht. Zusätzliche Informationen können z.B. Ausschnitte des Inhalts sein, die einen der Suchbegriffe enthalten oder die Kategorien der gefundenen Dokumente, wie in der folgenden Abbildung zu sehen ist:

International **Index** to the Performing Arts, Full Text, **...** - [Diese Seite übersetzen]
Subscription Options. IIPA Full Text is available as an annual subscription
with monthly updates on the Web. Subscriptions may begin at any time. **...**
iipaft.chadwyck.com/ - 2k - 9 Aug. 2002 - Im Archiv - Ähnliche Seiten

•••

Im Beispielfall kann der Anwender nur hoffen, das gesuchte Dokument oder die gewünschte Information auf einer der ersten Positionen zu finden, will er nicht 53.300 Dokumente durcharbeiten und selbst dann besteht die Möglichkeit, dass das Gesuchte gar nicht vorhanden ist. Alternativ kann man auch versuchen, die Suche mit weiteren Suchbegriffen einzuschränken. D.h. nur Dokumente, auf denen alle angegebenen Begriffe zusammen vorkommen, sind dann Ergebnis der Anfrage (zur Beschreibung der Suchmaschinen s. Kaptitel Kapitel 2). Im obigen Beispiel reduziert das Hinzunehmen des Begriffes „molina“ die Treffermenge auf 201 Dokumente. Um dem Anwender Hilfestellung bei der Suche in der Trefferliste zu geben, bieten alle populären Suchmaschinen ein Bewertungsverfahren an, das es ermöglichen soll, den „optimalen“ Treffer an erster Stelle zu platzieren und auch das gesamte Ergebnis nach diesem Maßstab zu sortieren. Dabei wird jeder Seite ein Wert zugewiesen, der die Güte dieses Dokuments in Bezug auf die Anfrage wiedergeben soll, dieser Wert wird als „Ranking“ bezeichnet. Gelegentlich wird der Wert auch dem Anwender gezeigt, wobei das „beste“ Dokument z.B. die Bewertung „100%“ erhält. Es gibt sehr viele mehr oder weniger komplizierte Verfahren, eine einfache Variante wäre z.B. die Gewichtung der Position der Suchbegriffe: Befindet sich der Begriff etwa im Titel der Seite, so ist das höher zu bewerten als wenn er nur im einfachen Text vorkommt. Ebenso kann man die Häufigkeit des Auftretens der gesuchten Begriffe mit in den Gesamtwert des Dokuments eingehen lassen. Grundsätzlich kann man zwei Bewertungsarten unterscheiden: Die Bewertung kann abhängig oder unabhängig von Suchbegriffen sein. Bei unabhängigen Methoden ist z.B. Google’s Pagerank – Algorithmus (s. 3.5.1) zu nennen, der Dokumente anhand ihrer Position im Netz gewichtet. Vereinfacht formuliert erhalten Seiten einen Bonus, wenn sie oft von anderen „gewichtigen“ Seiten referenziert werden. Wenn man private Homepages in allgemeinen kategorisierten Verzeichnissen wie z.B. Yahoo’s „Web Site Directory“ einträgt, so erhalten diese einen Aufstieg in Google’s Bewertung und damit auch in der Trefferliste. In der Regel wird das allgemeine „Gewicht“ eines Dokuments in die Bewertung, die von den Suchbegriffen abhängt, mit eingehen. Alle mehr oder weniger guten Bewertungsmethoden stehen jedoch vor dem Problem, dass Begriffe mehrere Bedeutungen besitzen können, so ist z.B. Java sowohl eine Insel als auch eine populäre Programmiersprache. Interessanterweise ergibt bei vielen Suchmaschinen im Internet eine Anfrage mit den Begriffen „Java“ und „island“ als besten Treffer ein Dokument über Java als Programmiersprache. Dieses Problem der „Intention des Anwenders“ lässt sich aus Sicht der Suchmaschinenbetreiber nicht zufrieden stellend lösen. In der Regel gilt für die Suche, dass populäre Themengebiete höheres Gewicht besitzen als Randgebiete des allgemeinen Interesses. So ist auch zu erklären, dass der „optimale“ Treffer für die Insel Java von einem anderen, aber eben populären, Thema handelt.

Die Bewertung der Dokumente mit dem UNICO – Web-Roboter verwendet eine von Suchbegriffen unabhängige Bewertungsmethode, die gegenüber den Verfahren bei allgemeinen Suchmaschinen im Internet den Vorteil besitzt, dass ein Themengebiet vorgegeben ist. So sind die Mitarbeiter der Firma Siemens bei ihrer Recherche in der UNICO – Datenbank hauptsächlich an Themen aus dem Bereich der Ingenieurswissenschaften interessiert. Bezogen auf das Java-Insel-Beispiel sollte hier die Bewertung klar die Programmiersprache bevorzugen. Diese Einschränkung könnte man als Suchanfrage auffassen, die als Treffer alle Dokumente geordnet

nach ihrer Relevanz bezüglich aller ingenieurswissenschaftlichen Themen ergeben soll. Die Menge der dazu nötigen Suchbegriffe wäre allerdings enorm. Eine Methode, um ein themenabhängiges Ranking zu erstellen soll im Folgenden vorgestellt werden. Der Ablauf ist größtenteils automatisiert, jedoch ist an einer Stelle administrativer Eingriff nötig, um dem System das Themengebiet zu vermitteln. Das Verfahren zur Bewertung beginnt erst nach Abschluss des Retrieval-Vorgangs, da für einige Teile die gesamten Informationen der Web-Präsenz vorliegen müssen.

Arbeitsschritte:

- Generierung der Wortliste
- Berechnung des „Positionsgewichts“ der Dokumente
- Automatische Gruppierung der Dokumente
- Bewertung der automatischen Gruppierung durch einen Experten
- Berechnung der endgültigen Bewertung für alle Dokumente

Wichtige Maßgaben bei der Entwicklung des Systems waren vor allem Effektivität und Effizienz, d.h. Aufwand und Ertrag sollen in einem günstigen Verhältnis stehen. Das beste Bewertungsverfahren verliert an Bedeutung, wenn die Berechnung für 1 Mio. Dokumente einen Monat oder länger dauert oder enorme Anforderungen an die Hardware stellt. Leider gibt es viele Methoden, die dieser Vorgabe nicht genügen, obwohl die Qualität des Ergebnisses sehr gut ist.

4.3.2.4.1.Generieren der Wortliste

Um den Inhalt von Dokumenten zu erfassen und zu verstehen, ob zwei Dokumente vom gleichen Thema handeln, muss man die Bedeutung der Wörter kennen oder zumindest das Auftreten von gleichen Begriffen auf unterschiedlichen Dokumenten erkennen. Auch möchte man alle vorhandenen Wörter gewichten, am Besten automatisch. Eine einfache Methode zur Gewichtung von Wörtern in großen Textmengen soll im Folgenden beschrieben werden. Zunächst ist es notwendig für jedes Dokument den reduzierten Text (s. 4.3.2.3.10) zu betrachten und die enthaltenen Wörter in die zu erzeugende Liste aufzunehmen. Dabei fließt auch die Anzahl des Auftretens eines Begriffs mit in die Liste ein. Ist das Wort bereits in der Liste, wird lediglich die Anzahl aufaddiert. Das Ergebnis sieht z.B. aus wie in folgender Tabelle dargestellt:

Term	Vorkommen
datenbanken	15
datenbank	21
volltext	4
sql	8
indexierung	3
...	...

Ein gravierendes Problem ist bereits in der Tabelle angedeutet: verschiedene Formen des gleichen Begriffs bedeuten auch mehrere Einträge, wie hier für die Wörter „datenbanken“ und

„datenbank“. Ein Wort in der Mehrzahl beinhaltet keinen Informationsgewinn gegenüber einem Wort im Singular, daher ist ein Mehrfacheintrag hier nicht erwünscht. Um dies zu erreichen, muss man alle Begriffe auf ihre Stammform[22] reduzieren, dazu gibt es unterschiedliche Verfahren, die in 3.3 beschrieben sind. Für den UNICO – Web-Roboter kommt ein regelbasiertes Verfahren zum Einsatz, ein so genannter Porter-Stemmer (s. 3.3.2). Im obigen Fall würde damit das Ergebnis aussehen wie in nachstehender Tabelle. Auch der Originalbegriff bleibt erhalten, da man bei späteren Schritten sonst eventuell Schwierigkeiten hätte, das Originalwort in den Dokumenten wieder zu finden. Auch kann man in der Tabelle

Term	Stammform	Vorkommen
datenbanken	datenbank	36
datenbank	datenbank	36
volltext	volltext	4
sql	sql	8
indexierung	index	3
...	...	...

erkennen, dass die Reduktion auf eine Stammform auch das Entfernen von Endungen beinhaltet: „indexierung“ wird zu „index“. Im nächsten Schritt wird die Anzahl der Vorkommen zur Gesamtanzahl von Dokumenten in Relation gesetzt. Bei z.B. 30 Dokumenten würde man folgendes Ergebnis erhalten:

Term	Stammform	Relative Häufigkeit
datenbanken	datenbank	1,2
datenbank	datenbank	1,2
volltext	volltext	0,13
sql	sql	0,27
indexierung	index	0,1
...	...	...

Relative Häufigkeit eines Terms t:

$$\frac{\text{Anzahl des Auftretens von t im gesamten Textkorpus}}{\text{Anzahl aller Dokumente im Textkorpus}}$$

Also enthält jedes Dokument im Durchschnitt 1,2 Mal eine Form des Begriffs „Datenbank“ oder Formen des Wortes „Index“ sind nur in jedem zehnten Dokument zu finden. Folgende Ergebnisse ergab die Generierung einer solchen Wortliste für die Web-Präsenz des Instituts für Informatik und Mathematik mit 115.710 erfolgreich bearbeiteten Dokumenten:

- 843.207 unterschiedliche Begriffe.
- 677.831 unterschiedliche Stammformen, d.h. eine Reduktion um 19,6 %.

[22] Grundform eines Wortes, z.B. Nomen im Nominativ, Singular, Verben im Präsenz, Infinitiv

- Begriffe, die genau einmal auftreten: 270.397, das entspricht 32% der Gesamtmenge und ist damit die größte Menge von Begriffen, die die gleiche relative Häufigkeit besitzen.

- Dauer der Generierung der Wortliste einschließlich Stammformen und relativer Häufigkeit: ca. 3,5 Stunden.

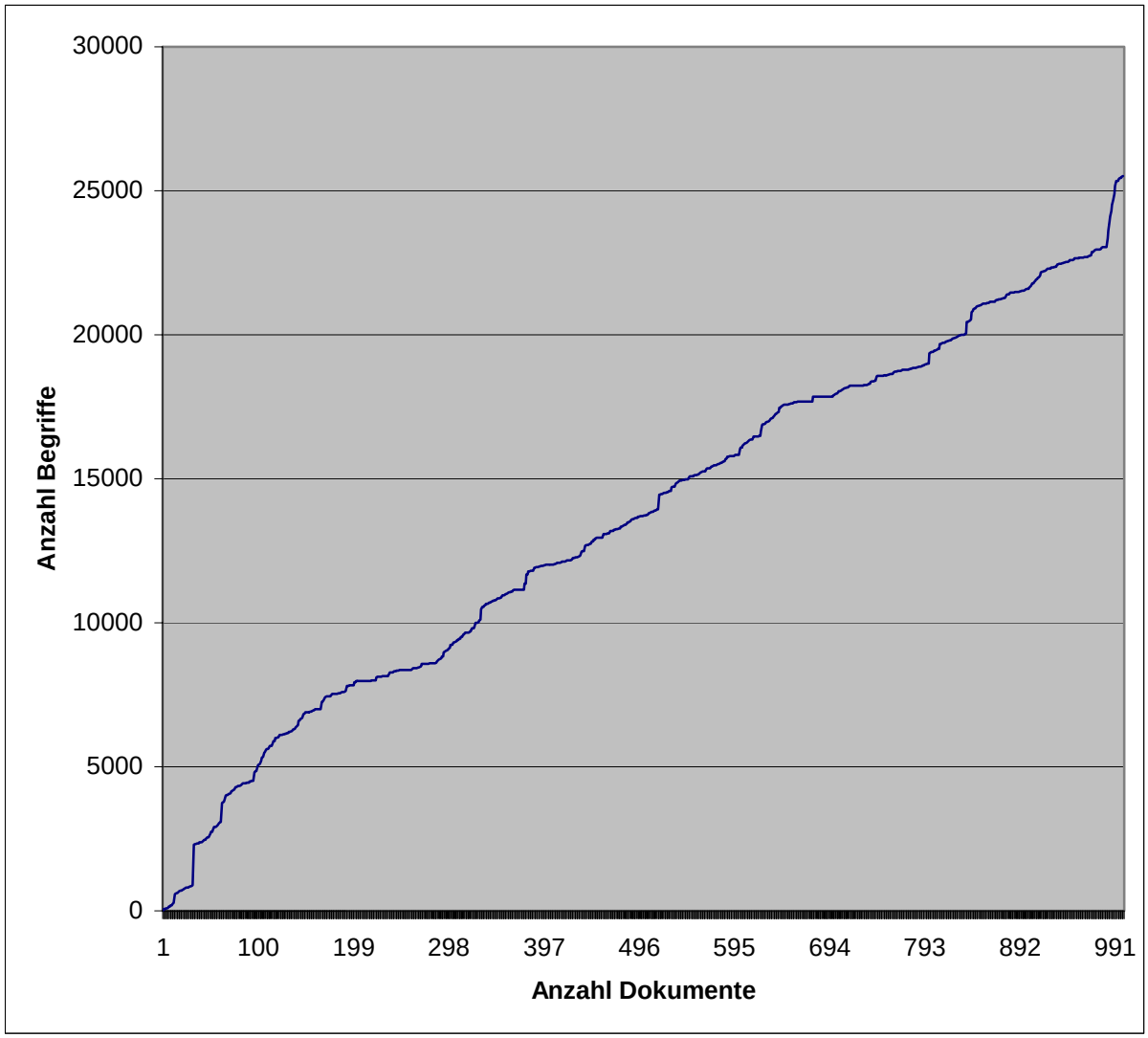

Abbildung 4-9: Anstieg der Anzahl der Begriffe mit zunehmender Dokumentanzahl

Die Dokumente werden einzeln nacheinander nach Begriffen durchsucht, dabei wächst die Menge der gefundenen Begriffe zu Beginn stark und lässt mit zunehmender Dokumentanzahl langsam nach. Dieses Verhalten ist im nachfolgenden Diagramm veranschaulicht: Im Mittel nimmt die Anzahl der Begriffe stetig zu, jedoch werden mit steigender Anzahl von Dokumenten immer weniger neue Wörter hinzukommen. Tests haben gezeigt, dass jedoch auch noch nach Verarbeitung von mehr als 1 Mio. Dokumenten eine stetige Zunahme der Gesamtmenge von Begriffen zu verzeichnen ist. Im Diagramm Abbildung 4-9 ist der Anstieg der Wortmenge bis etwa 1000 Dokumente zu sehen. Bei bestimmten Punkten kommen Dokumente zur Bearbeitung, die ein neues Themengebiet anschneiden und somit für eine kurzfristige starke Steigung des Graphen durch viele neue Begriffe verantwortlich sind, so z.B. im Bereich von Dokument 980 – 990. Dokumente aus diesem Bereich könnten als Repräsentanten einer Klassifikation verwendet werden, da sie sich inhaltlich deutlich von den bisherigen abheben.

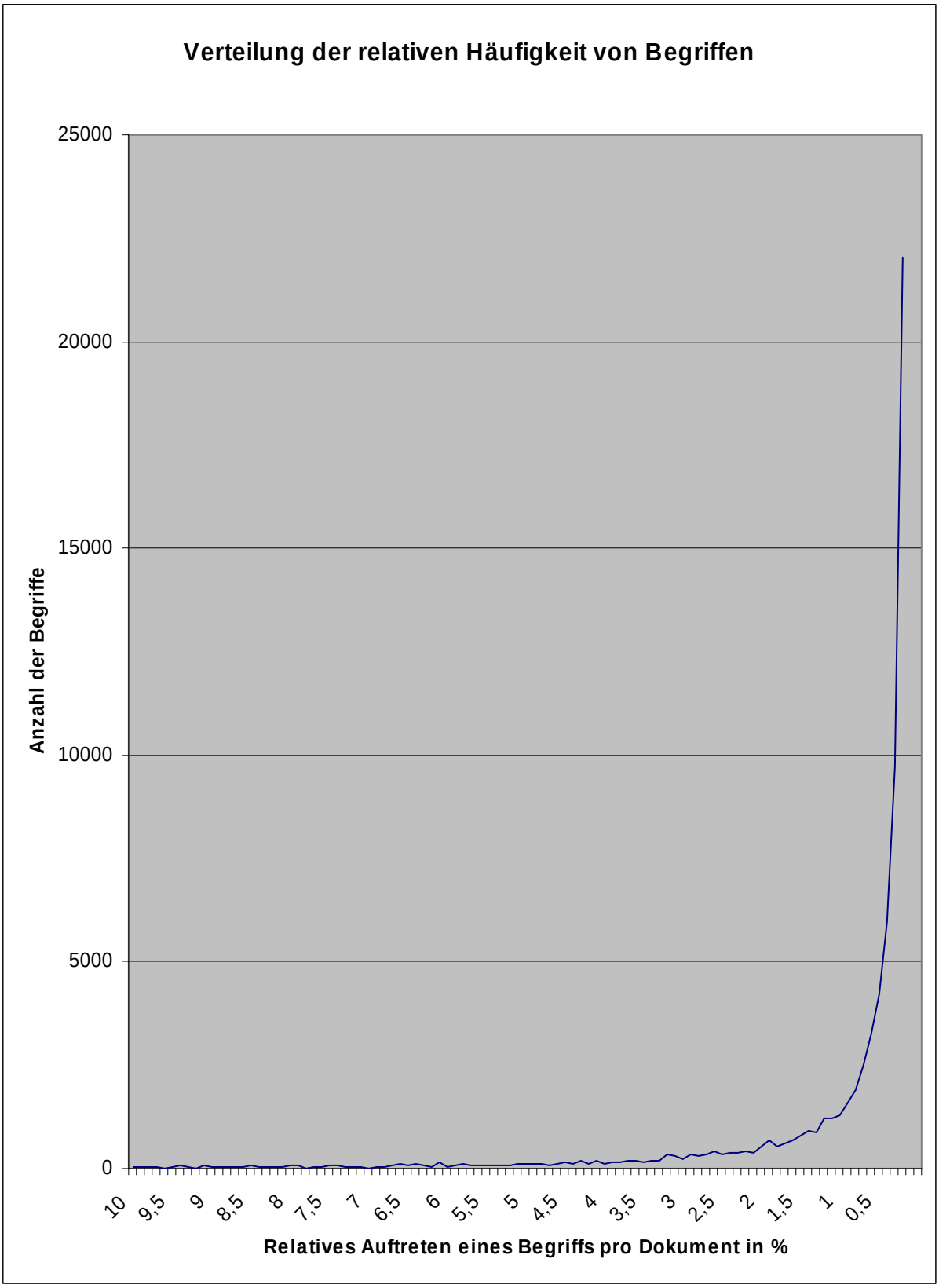

Abbildung 4-10: Verteilung der Begriffe auf ihr prozentuales Vorkommen

In Abbildung 4-10 sind die relativen Häufigkeiten von Begriffen pro Dokument graphisch veranschaulicht. D.h. es gibt z.B. 518 unterschiedliche Begriffe, die auf 1,9% – 2,0% der Dokumente enthalten sind oder, anders formuliert, auf etwa jedem 25ten Dokument auftreten. Die Anzahl der Begriffe nimmt mit fallendem Prozentwert immer weiter zu bis das Maximum erreicht ist bei 1 / 115710 = 0.000008642 = 0.0008642 %. Diese Verteilung war bei allen Tests sehr ähnlich, das Maximum bildete jeweils die Gruppe von Begriffen, die in der gesamten Dokumentmenge genau einmal aufgetreten ist.

Häufigste Begriffe:

Term	Stammform	Relative Häufigkeit
start	start	0,406079
starten	start	0,406079
started	start	0,406079
starting	start	0,406079
starts	start	0,406079
starte	start	0,406079
starter	start	0,406079
startes	start	0,406079
startens	start	0,406079
inform	inform	0,4060783
information	inform	0,4060783
informed	inform	0,4060783
informations	inform	0,4060783
informal	inform	0,4060783
informally	inform	0,4060783
informational	inform	0,4060783
informing	inform	0,4060783
informative	inform	0,4060783
informs	inform	0,4060783
informer	inform	0,4060783
informant	inform	0,4060783
informatively	inform	0,4060783
informated	inform	0,4060783
reference	refer	0,3955105
references	refer	0,3955105
refer	refer	0,3955105
referred	refer	0,3955105
referent	refer	0,3955105
referring	refer	0,3955105
refers	refer	0,3955105
referate	refer	0,3955105
refered	refer	0,3955105
refering	refer	0,3955105

referende	refer	0,3955105
referents	refer	0,3955105
referer	refer	0,3955105
refere	refer	0,3955105
referend	refer	0,3955105
...	...	...

Die Frage nach der Bedeutung der relativen Häufigkeit von Begriffen wird in den nächsten Teilschritten der Bewertungskomponente erklärt. Als weiteres Ergebnis bleibt festzuhalten, dass Begriffe, die im gesamten Textkorpus genau einmal vorkommen, keinen Informationsgewinn hinsichtlich themenverwandter Dokumente bringen, das Gleiche gilt für Dokumente mit einer relativen Häufigkeit, die größer als 1 ist. Das bedeutet dann, dass dieser Begriff im Mittel pro Dokument mindestens einmal auftritt, also auf jedem Dokument vorhanden ist und damit kein Dokument gegenüber einem besonders auszeichnet. Eine mögliche Fehlerquelle hierbei ist durch folgendes Beispiel illustriert: Ein Textkorpus bestehend aus 10 Dokumenten, Dokument d1 enthält 10 Mal den Begriff „datenbank“, die restlichen 9 Dokumente enthalten diesen Begriff kein Mal. Es ergibt sich eine relative Häufigkeit für den Begriff „datenbank“ von 1,0, d.h. im Mittel enthält jedes Dokument diesen Begriff, er wäre somit wenig relevant, um Dokumente zu unterscheiden, obwohl er es in diesem extremen Beispiel tut. Eine Möglichkeit der Korrektur dieses Fehlers wäre es, pro Dokument jeden Begriff nur einmal zu werten, d.h. für das Beispiel: das Wort „datenbank“ erhält die relative Häufigkeit 0,1, was hier wesentlich sinnvoller wäre.

4.3.2.4.2.Inhaltsunabhängige Bewertung von Dokumenten

Viele Dokumente, vor allem wissenschaftliche Aufsätze und Veröffentlichungen, enthalten Bibliographien oder ein Liste von Referenzen. Das sind im allgemeinen Verweise auf themenverwandte Dokumente. Wenn man also z.B. wissenschaftliche Veröffentlichungen aus dem Bereich der Informatik nimmt, so erhält man einen Graph von Dokumenten, die durch Referenzen verbunden sind. Die Struktur des Graphen, unabhängig vom Inhalt, kann dabei Aufschluss geben über Ähnlichkeit von Dokumenten und die Verteilung der Information.
Überblick:

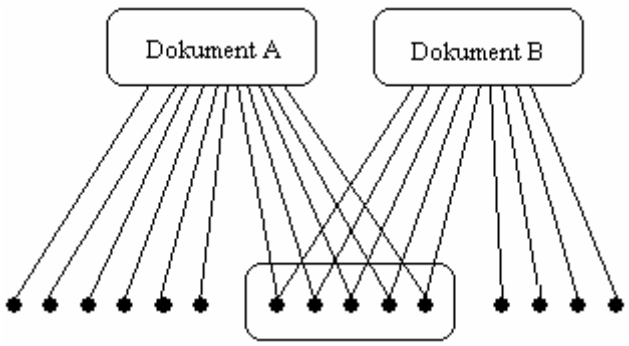

Abbildung 4-11: Ähnlichkeit von zwei Dokumenten durch gemeinsame Referenzen

Bereits in den 70er Jahren wurde versucht, ein derartiges Maß zu entwickeln, das wissenschaftliche Journale hinsichtlich Qualität und Einfluss bewerten soll. Gemessen wurde,

wie oft Papiere in einem Journal von Wissenschaftlern zitiert wurden. Der so genannte „Impact Factor“ eines Journals in einem bestimmten Jahr j errechnete sich aus der durchschnittlichen Anzahl von Verweisen eines Papiers auf Papiere aus dem gleichen Journal der Vorjahre j-1 und j-2. Der gewonnene Wert sagt jedoch nichts über die Qualität einer einzelnen Veröffentlichung aus.
1963 wurde ein Maß zu Bestimmung der „Ähnlichkeit“ von Dokumenten entwickelt: die „bibliographische Kopplung“ von zwei Dokumenten ist bestimmt durch die Zahl der gemeinsam referenzierten Dokumente, also der Schnittmenge ihrer Bibliographien. Um einheitliche Werte zu erhalten ist es dabei günstig, den Wert durch die Größe der Bibliographien zu normalisieren: s. Abbildung 4-11.
Ein anderer Ansatz basierend auf Referenzen wurde 1973 vorgestellt: Die Qualität der Ähnlichkeit von zwei Dokumenten wird durch die Anzahl der Dokumente gemessen, die auf beide Dokumente verweisen: s. Abbildung 4-12

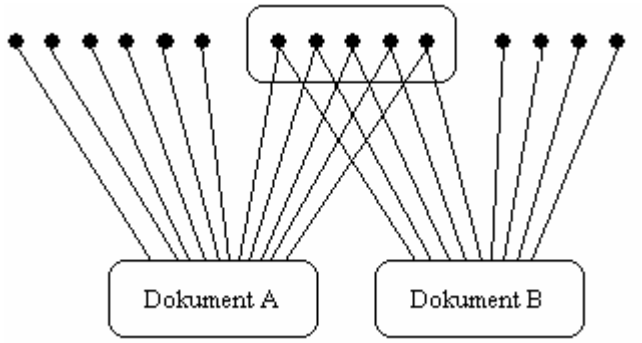

Abbildung 4-12: Verweise auf unterschiedliche Dokumente implizieren deren Ähnlichkeit

Der Vergleich von bibliographischen Referenzen und Hyperlinks in Dokumenten des WWW ist nicht ohne weiteres möglich, folgende Schwierigkeiten sind zu beobachten:

- Viele Hyperlinks dienen nur zur Navigation, so z.B. bei der Umsetzung von Postscriptdokumenten in HTML –Dokumente: „nächste Seite“, „vorhergehende Seite“, „zum Anfang des Dokuments“, usw.
- Viele Dokumente des WWW dienen als Portal und nicht zur Vermittlung von Inhalten, so z.B. Web-Verzeichnisse bei Providern wie Yahoo oder Web.de
- Nicht jeder Hyperlink dient zur Bestätigung des Inhalts
- Web-Seiten von Firmen verweisen in der Regel nicht auf Seiten ihrer Konkurrenz

Definition: Authorities sind Dokumente, die bekannt sind für sichere, umfassende, nützliche und wichtige Informationen zu einem spezifischen Themengebiet. *Hubs* sind Dokumente, die Hyperlinks auf viele relevante inhaltsvolle Dokumente zu einem bestimmten Themengebiet enthalten.

Ein einfaches Maß zur Bestimmung von Authorities wäre das simple Zählen der Verweise auf eine Seite. Jedoch hätte dann jeder Hyperlink das gleiche Gewicht und wird die Frage aufgeworfen, ob Referenzen von Seiten, die bereits als Authority bekannt sind, höheren Einfluss haben sollten. Wenn man die Authorities zu einem bestimmten Themengebiet ermittelt hat, dann lassen sich auch die dazu gehörigen Hubs ermitteln, indem man die Seiten mit den meisten Referenzen auf diese Authorities ermittelt. Ein bekanntes Verfahren zur Bestimmung von Hubs und Authorities ist bekannt unter dem Namen „HITS“. Es ermittelt die Authorities anhand des

Inhalts der Dokumente und bestimmt anschließend die Hubs über die Hyperlinkstruktur. Eine ausführliche Beschreibung ist unter 3.5.2 zu finden.

Für den UNICO – Web-Roboter wird ein bekanntes Verfahren zur Bestimmung von Authorities eingesetzt: *PageRank*. Das Verfahren wurde 1998 entwickelt (s. 3.5.1) und wird von Google verwendet. Es beschränkt sich auf die Ermittlung von Authorities und ist auf sehr große Dokumentmengen (das gesamte WWW im Idealfall) ausgelegt. Eine ausführliche Beschreibung des Algorithmus befindet sich unter 3.5.1. Mit der bei 4.3.2.3.14 angegebenen Hardware lassen sich zurzeit 1.000 Dokumente in wenigen Sekunden berechnen, für 100.000 Dokumente werden etwa 2,5 Stunden benötigt. Da die Struktur einer Web-Präsenz im Vergleich zum gesamten WWW nur sehr wenige Web-Server betrachtet, werden die überwiegend als Navigationshilfen eingeschätzten Hyperlinks innerhalb eines Web-Servers entsprechend markiert und bei der Berechnung des PageRank über den E-Wert (s. 3.5.1) gegenüber anderen Hyperlinks in der Bewertung benachteiligt.

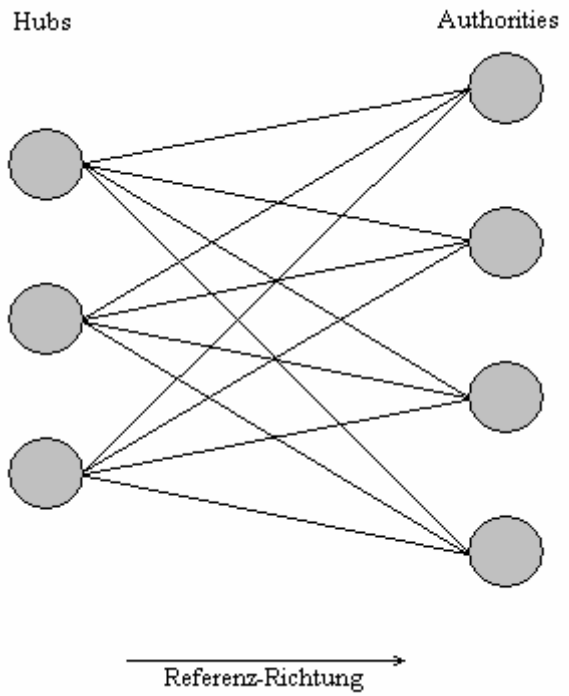

Abbildung 4-13: Authorities

Beispiel: Web-Präsenz der LMU München Mathematik, 1000 Dokumente

Hyperlink	PageRank	Referenzen
http://www.mathematik.uni-muenchen.de/	100	286
http://www.mathematik.uni-muenchen.de/aktuell.html	58,82	200
http://www.mathematik.uni-muenchen.de/kontakt/index.html	58,72	200
http://www.mathematik.uni-muenchen.de/arbeitsgruppen.html	58,53	199
http://www.mathematik.uni-muenchen.de/einrichtungen.html	58,42	198

Hyperlink	PageRank	Referenzen
http://www.mathematik.uni-muenchen.de/personen/index.html	58,41	199
http://www.mathematik.uni-muenchen.de/studium.html	58,31	198
http://www.mathematik.uni-muenchen.de/extern/fach.html	58,27	201
http://www.mathematik.uni-muenchen.de/suche/suche.html	57,07	197
http://www.mathematik.uni-muenchen.de/welcome_en.html	36,83	95
http://www.mathematik.uni-muenchen.de/personen/index_en.html	32,67	91
http://www.mathematik.uni-muenchen.de/kontakt/index_en.html	32,34	90
http://www.mathematik.uni-muenchen.de/aktuell_en.html	32,33	91
http://www.mathematik.uni-muenchen.de/studium_en.html	32,25	90
...	...	...

Die Spalte Referenzen bezeichnet die Anzahl von Referenzen auf das Dokument. In der Regel bedeutet eine hohe Zahl an Referenzen auch einen hohen PageRank-Wert, jedoch fließt auch die Güte des referenzierenden Dokuments mit in die Bewertung ein. Den besten Wert erhält die Einstiegsseite der LMU Mathematik, da von etwas mehr als jeder vierten Seite auf sie verwiesen wird. Innerhalb größerer Web-Präsenzen entstehen jedoch auch Authorities zu spezielleren Themengebieten. Diese treten dann hervor, wenn man die Menge der Seiten durch Suchbegriffe einschränkt. Durch verändern der Startparameter (s. 3.5.1) ist es auch möglich, die Bewertung in eine bestimmte Richtung zu steuern, indem ausgesuchte Seiten (die z.B. besonders wichtige Themen betreffen) bevorzugt werden. Beim UNICO – Web-Roboter wird diese Einschränkung auf für Siemensmitarbeiter relevanter Themengebiete erst zu einem späteren Zeitpunkt erreicht, da dann inhaltliche Gesichtspunkte von Dokumenten bereits bekannt sind und deren Bewertung leichter fällt. Die Beschreibung eines ersten Schritts zu einer inhaltlichen Sortierung von Dokumenten einer Web-Präsenz erfolgt im nächsten Abschnitt.

4.3.2.4.3.Inhaltsabhängige Bewertung von Dokumenten - Vorbereitungen

Ziel des gesamten im UNICO – Web-Roboter eingesetzten Bewertungssystems soll es ein, thematische Präferenzen für jede gewünschte Web-Präsenz herauszuarbeiten und danach die einzelnen enthaltenen Dokumente zu bewerten. Der Wert des PageRank-Algorithmus soll dabei einer der Einflussfaktoren auf die abschließende Bewertung darstellen. Ein weiterer wesentlich komplexerer Schritt ist jetzt die Aufspaltung eines Textkorpus in mehrere Themenbereiche und der anschließende Abgleich mit den gewünschten Themen. Speziell für die UNICO – Datenbank waren folgende Hauptinteressensbereiche von universitären Web-Präsenzen vorgegeben:

Mathematik
Informatik
Physik
Chemie
Wirtschaftswissenschaften
Elektrotechnik
Informationstechnik

Durch gründliches Vorarbeiten bezüglich der Einschränkungen beim Retrieval (s. 4.3.2.3) kann schon eine gute Vorauswahl an Dokumenten getroffen werden. Jedoch werden immer noch viele nutzlose Daten enthalten sein, wie z.B.

Private Homepages
Studentische Seiten
Dokumente von Institutionen innerhalb der Universität, z.B. Mensa, Rechenzentrum
Technische Informationen zum Studium
Softwarearchive
Softwaremanuale
Dokumente von Fachbibliotheken
Adresslisten von Studenten und Mitarbeitern
Informationen zu Freizeitaktivitäten, z.B. Hochschulsport

Vor allem Softwaremanuale stellen ein sehr großen Anteil an nutzlosen Informationen, so finden sich innerhalb der Web-Präsenz der Informatik und Mathematik der TU München etwa 25 HTML-Manuale für die Programmiersprache Java in allen Versionen und Varianten. Jedes der Manuale besteht aus mehreren tausend Dokumenten. Die Erkennung und Eliminierung dieser Informationen aus den Daten einer Web-Präsenz ist zuverlässig nur durch Ausschluss des Verzeichnisses beim Retrieval effizient möglich. Eine Erkennung durch inhaltliche Analyse im Allgemeinen ist schwierig, da es eine Vielzahl von Software-Anleitungen gibt, deren inhaltliche Merkmale selten homogen sind. Speziell zur Erkennung von Java-Dokumentation ist ein einfaches Konzept im UNICO – Web-Roboter integriert: Enthält ein Dokument eine Zeile im HTML-Quelltext wie die nachstehende, so wird das Dokument nicht indexiert.

```
<!-- Generated by javadoc on ... -->
```

Dieser Eintrag ist in der Regel bei automatisch generierter Java-Dokumentation enthalten. Die Datenqualität in der Datenbank wird durch das Auslassen solcher Dokumente erheblich verbessert. So ist z.B. die Wortliste aller Dokumente einschließlich der Java – Dokumentationen für die Web-Präsenz der TU München: Informatik und Mathematik besonders gravierend betroffen:

Wort	Wortstamm	Relative Häufigkeit
class	class	4,570694
classes	class	4,570694
classe	class	4,570694
classed	class	4,570694
classen	class	4,570694
classer	class	4,570694
java	java	3,25757
javaes	java	3,25757
methoden	method	2,469323
methods	method	2,469323
methodik	method	2,469323
methodische	method	2,469323
method	method	2,469323
methode	method	2,469323
methodischen	method	2,469323

methodischer	method	2,469323
methodisch	method	2,469323
methodisches	method	2,469323
methodical	method	2,469323
methodiker	method	2,469323
methodically	method	2,469323
methodics	method	2,469323
methodic	method	2,469323
methodiken	method	2,469323
methodes	method	2,469323
object	object	2,044426
objects	object	2,044426
objective	object	2,044426
...	...	...

Es folgen noch 291 weitere Wörter und Wortstämme, die alle ihre hohe relative Häufigkeit aus den zahllosen Seiten von Java-Dokumentationen beziehen. Es ist für spätere Berechnungen wichtig, dass die Wortliste so klein wie möglich gehalten wird. Auch die Berechnung des PageRank-Wertes der Dokumente profitiert durch die Eliminierung der Java-Dokumentationen, sowohl im Ergebnis als auch in der Performance durch die geringe Anzahl der Dokumente.

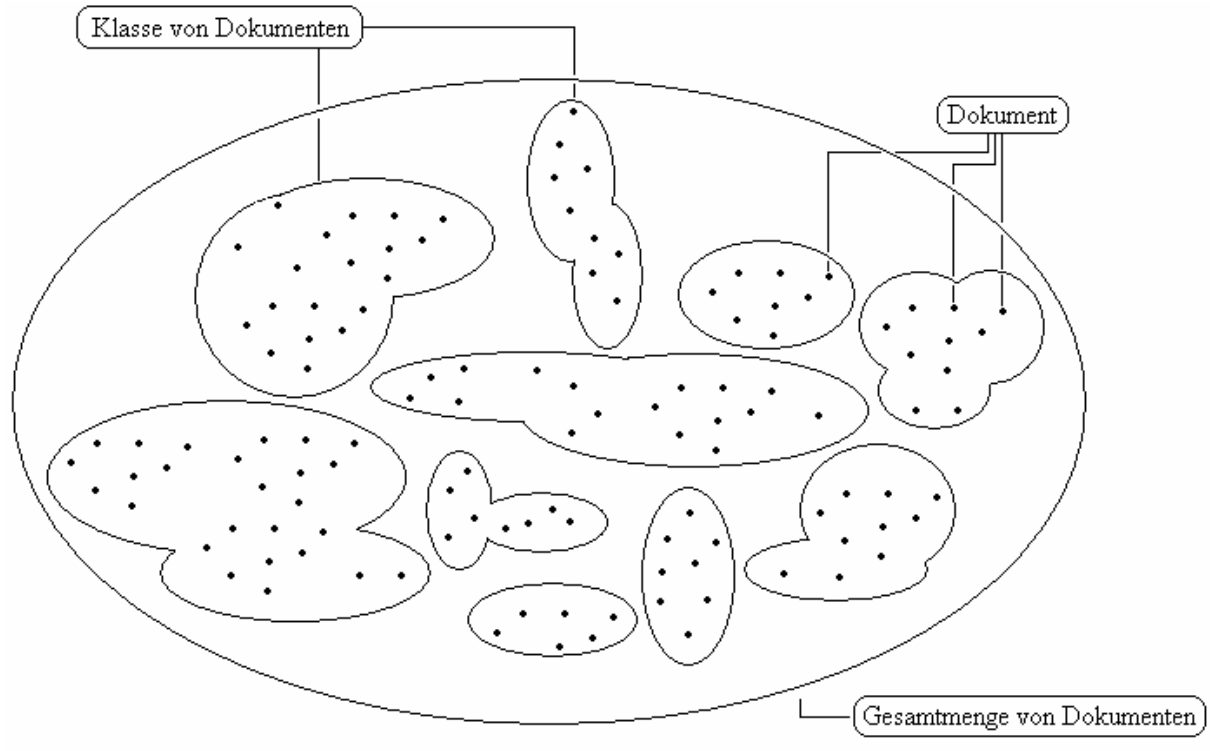

Abbildung 4-14: Einteilung in Dokumentklassen

Definition:

Ein *Cluster* oder *Klasse* von Dokumenten beschreibt eine Menge von Dokumenten, die inhaltlich zusammenhängen. Ein Verfahren zur automatischen Erzeugung von Dokumentklassen wird *Clustering* oder *Klassifikation* genannt.

Um Dokumente nach Themen zu klassifizieren (Clustering), existieren viele Verfahren. Einige sind im Detail in Abschnitt 3.4 genauer beschrieben. Bei der Entwicklung des UNICO – Web-Roboters sind einige davon getestet oder neu entwickelt worden, um eine Klassifizierung in einem günstigen Zeitrahmen zu erhalten. In Abbildung 4-14 wird eine Einteilung in Dokumentklassen gezeigt. Dabei wird inhaltliche Ähnlichkeit von Dokumenten zur Veranschaulichung durch räumliche Nähe ausgedrückt.

Folgende Klassifizierungsvarianten wurden getestet:

K-Means (klassisch)
K-Means (reverse)
WebSOM (n-gramm-basiert)
WebSOM (wortbasiert)
WordCon

Nach vielen Tests (s. 3.4.7) wurde schließlich K-Means (reverse) für den Einsatz im UNICO – Web-Roboter ausgewählt. Er zeichnet sich durch lineare zeitliche Komplexität (d.h. nur Abhängigkeit von der Anzahl der Dokumente) aus und bietet trotz guter Gesamtperformanz gute Klassifizierung der Dokumentmenge. Da durch eine automatische Klassifizierung einer großen Dokumentmenge immer Information verloren geht, z.B. durch das „Aufsaugen“ von kleineren Themengebieten in großen Clustern, ist die Anzahl der zu berechnenden Klassen sehr groß gehalten: In den meisten Tests wurden 200-250 Cluster von Dokumenten erstellt. So ist die Wahrscheinlichkeit größer, dass einzelne Themenbereiche nicht „verschluckt“ werden. Der verwendete Algorithmus ist im Detail in 3.4.3 beschrieben, soll jedoch hier kurz skizziert werden:

Zunächst wird die Wortliste eingeschränkt: Es werden nur Begriffe betrachtet, die mindestens *n* mal im gesamten Textkorpus auftreten. Günstige Werte für n lassen sich bei Betrachtung der Wortliste bestimmen. Dabei sollte in jedem Fall $n \geq 2$ gelten, damit Begriffe ausgeschlossen werden, die im gesamten Textkorpus nur einziges Mal vorkommen. Für die Web-Präsenz der TU München wurde ein Wert n=10 festgelegt. Eine weitere Einschränkung ist die Begrenzung noch oben: z.B. maximal 100 000 Wörter der Liste. Abbildung 4-15 soll die Einschränkung illustrieren.

Nachdem die Wortliste eingeschränkt ist, erhalten die Begriffe jetzt eine Bewertung, d.h. ein Gewicht für ihre „Wichtigkeit“: Dazu sollen Terme, die häufig auftreten, einen geringen Wert erhalten, seltenere Wörter einen hohen Wert. Das wird erreicht durch das IDF – Maß:

Für einen Term t ist

$$IDF(t) = \log_e\left(\frac{1}{tf(t)}\right)$$

tf(t): relative Häufigkeit (tf = term frequency) des Terms t

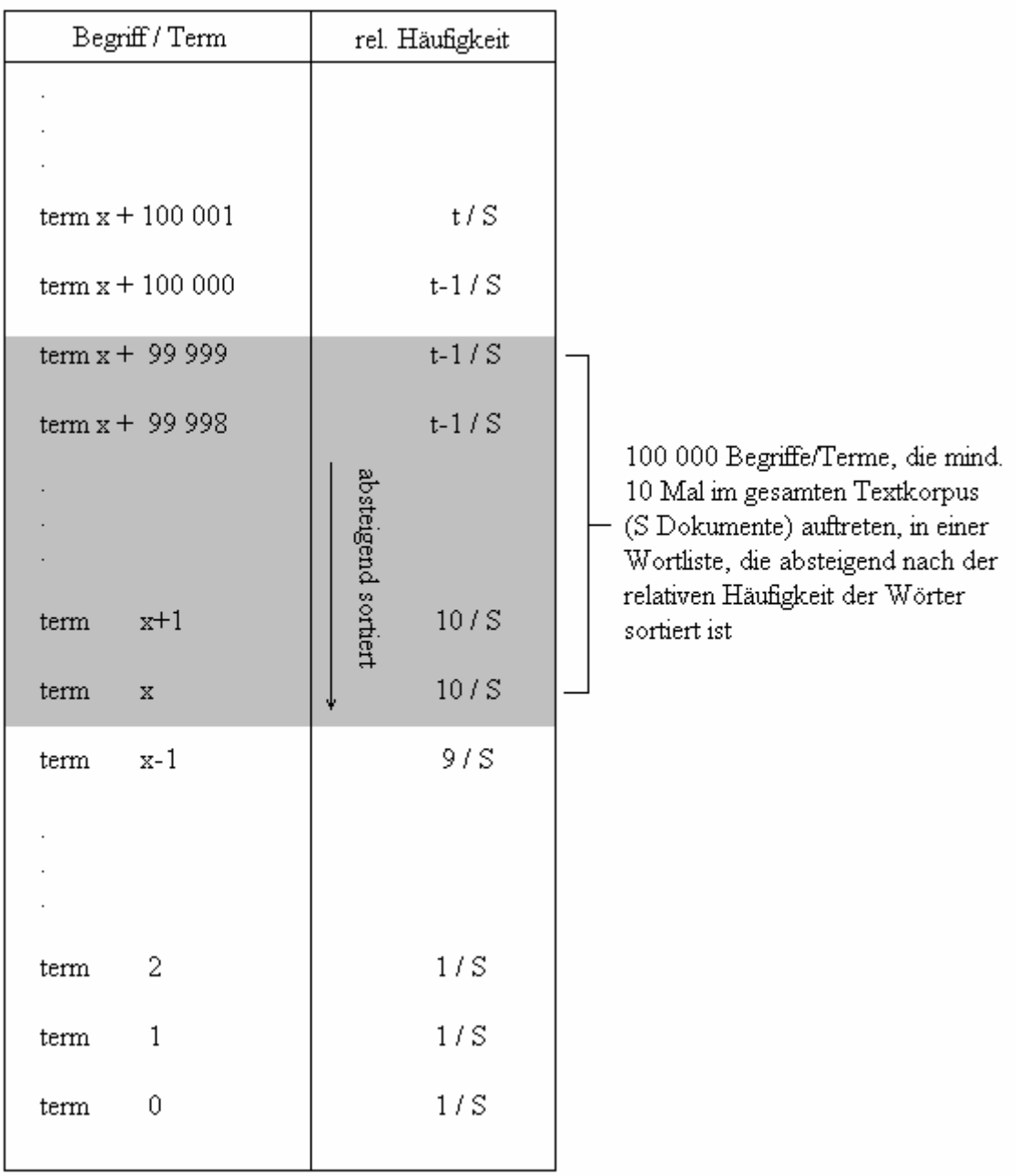

Abbildung 4-15: Einschränkung der Wortliste

Das entscheidende an der Berechnung des IDF-Werts ist die Verwendung des Kehrwerts der relativen Häufigkeit eines Begriffs, der Logarithmus soll lediglich die Werte nicht zu stark ansteigen lassen: sonst wäre bei 1 Mio. Dokumente der IDF-Wert für ein Wort, das zwei Mal auftritt: 500.000. Verwendet man die angegebene Formel so ergibt sich für den IDF-Wert:

$$idf(t) = \log e\left(\frac{1}{tf(t)}\right) = \log e\left(\frac{1000000}{2}\right) = \log e(500000) = 13{,}12\,.$$

Für jedes Dokument kann nun ein Vektor gebildet werden aus den Begriffen der reduzierten Wortliste und den dazugehörigen IDF-Werten. Dabei wird gezählt, wie oft jeder Begriff der Liste in dem Dokument auftritt und dieser Wert mit dem IDF-Wert des Begriffs multipliziert:

Term	Auftreten im Dokument	IDF(term)	**Gewicht**
Analytisch	1	1,75	**1,75**
Wissenschaft	3	1,65	**4,95**
Gleichung	0	1,54	**0**
linear	0	1,33	**0**
Mathematik	2	1,21	**2,42**
Funktion	0	1,06	**0**
Hörsaal	4	0,99	**3,96**
...	...	...	**...**

Ein Wortvektor zu einer auf 100.000 Einträge reduzierten Wortliste besteht aus 100 000 Termen mit den zugehörigen Gewichten. Die überwiegende Zahl der Einträge wird als Gewicht 0 sein, da im Allgemeinen nur ein kleiner Bruchteil der Wortmenge in einzelnen Dokumenten auftritt.

4.3.2.4.4.Inhaltsabhängige Bewertung von Dokumenten - Klassifikation

Nach den Vorarbeiten beginnt jetzt die Klassifikation der Dokumente. Zu Beginn befinden sich alle Dokumente in einer Klasse. Diese Klasse wird nach K-Means-Verfahren in zwei Unterklassen geteilt. Man nimmt dann die jeweils größte der neu entstandenen Klassen und teilt diese noch mal. Der Teilungsschritt wird sooft wiederholt bis die gewünschte Anzahl an Klassen entstanden ist (s.Abbildung 4-16). Dabei werden zwei Dokumente aus dem zu teilenden Cluster c_1 ausgewählt und dienen als Startrepräsentanten r_1 und r_2 für die neu zu bildenden Cluster c_2 und c_3. Jedes Dokument aus c_1 wird jetzt mit den beiden neuen Repräsentanten verglichen und demjenigen zugeordnet, dem es nach der p-Korrelation (s. 3.4.1.2.4) „näher“ ist.
Sind alle Dokumente zugeordnet, so wird überprüft, ob die Teilung sinnvoll ist. dafür gibt es unterschiedliche Kriterien:

Die Größe der neu entstandenen Klassen: Im Extremfall besteht einer der Cluster nur aus einem einzigen Dokument, dem Repräsentanten. Im günstigsten Fall erfolgt die Teilung genau in der Mitte, d.h. gleich viele Dokumente in c_2 und c_3. Die Mindestgröße sollte abhängig sein von der gewünschten Anzahl von Clustern, die entstehen soll: bei 50 zu bilden Klassen sollte bei einer Teilung die Minimalgröße 2% der Ausgangsklasse c_1 sein, also 1/50.
Die Qualität der neu entstandenen Klassen: Masse hierfür sind in 3.4.6.1 beschrieben.

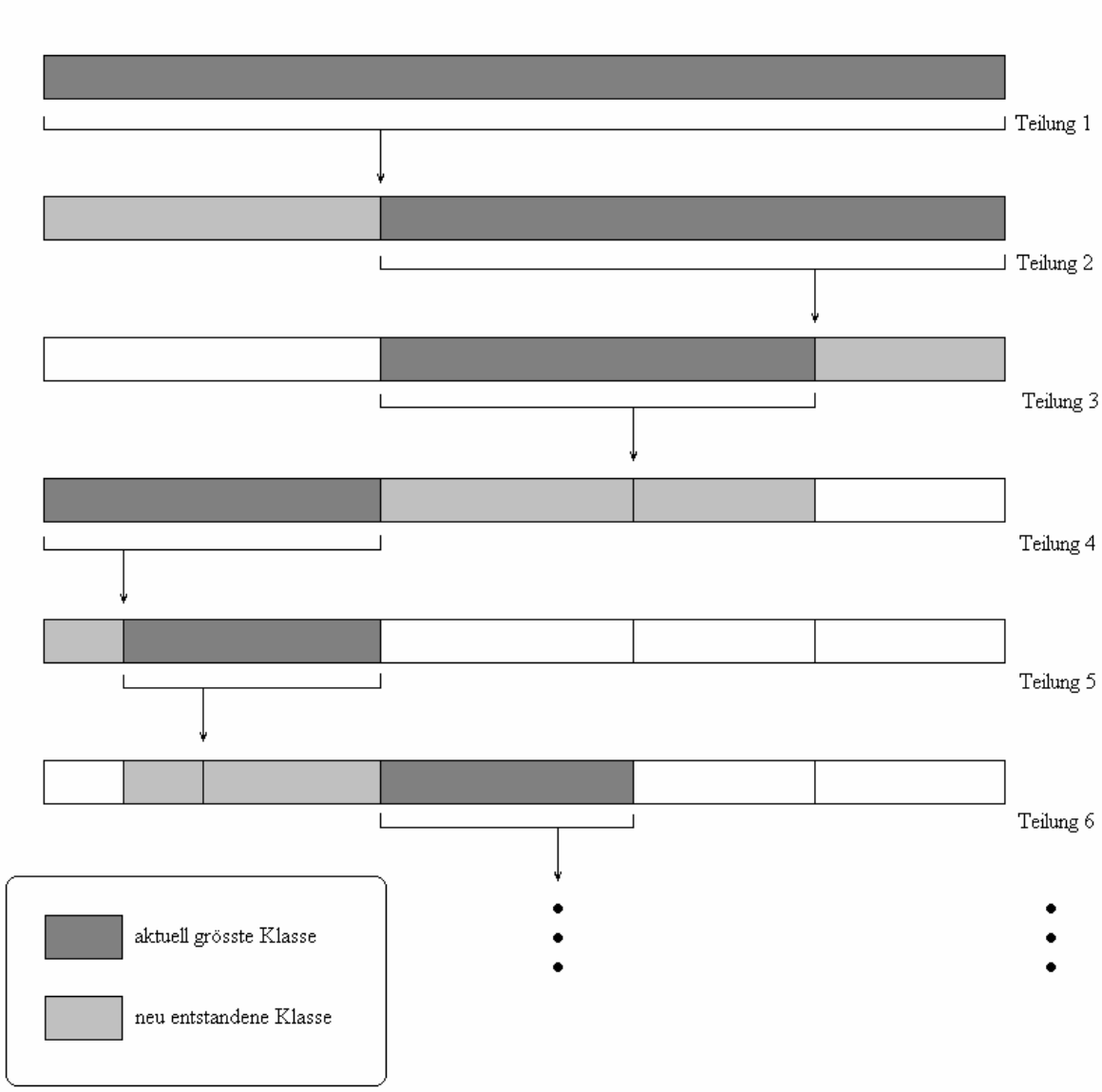

Abbildung 4-16: Teilung in K-Means(reverse)

Der UNICO – Web-Roboter verwendet die Größenbedingung bei der Teilung. Wenn die Bedingung nicht erfüllt ist, werden die Startvoraussetzungen verändert und der Vorgang wiederholt. Ist nach einer vorgegebenen Anzahl von Versuchen die Bedingung immer noch nicht erfüllt, so wird die bis dahin beste Lösung genommen und der Teilungsprozess kann fortgesetzt werden. Im Wiederholungsfall gibt es die Auswahl der neuen Repräsentanten r_{1neu} und r_{2neu} wiederum mehrere Möglichkeiten:

- Zufällige Auswahl aus c_1
- Auswahl von zwei Dokumenten mit stark besetzten Vektoren, d.h. möglichst wenige Nullwerte
- $r_{1neu} = r_1$ und r_{2neu} wird zufällig aus c_1 gewählt
- $r_{1neu} = r_1$ und r_{2neu} = Dokument mit größtem Abstand zu r_1
- r_{1neu} = Dokument mit bestem PageRank-Wert in c_1 und r_{2neu} = Dokument mit schlechtestem oder durchschnittlichen PageRank-Wert in c_1
- Zufällige Reihenfolge der zuzuordnenden Dokumente

- weitere Kombination aus den bisherigen Varianten

Bis auf den Punkt mit den PageRank-Werten sind alle Varianten gleich gut oder schlecht. Es gibt immer wieder Teilungssituationen, bei denen alle Wiederholungsversuche verbraucht werden, ohne dass eine ausreichende Lösung entsteht. Grundsätzlich fällt auf, dass Dokumente mit gut besetztem Wortvektor, d.h. in der Regel große Dokumente, bessere Repräsentanten darstellen als kleinere. Deshalb wird im UNICO – Web-Roboter einer der Repräsentanten im Wiederholungsfall beibehalten und ein Neuer wird unter den großen Dokumenten gesucht, die einen großen Abstand zu dem Beibehaltenen hatten.

Bildung der Repräsentanten:

Wird ein Dokument einem Klassenrepräsentanten zugeordnet, so wird sofort ein Schwerpunkt über alle zugehörigen Dokumente gebildet. Das entspricht einem durchschnittlichen Dokument, dieses wird also wiederum durch einen Wortvektor dargestellt:

Term	Dokument 1: Anzahl	Dokument 2: Anzahl	**Durchschnittliches Dokument**
Analytisch	1	0	**0,5**
Wissenschaft	3	2	**2,5**
Gleichung	0	4	**2**
linear	0	0	**0**
Mathematik	2	1	**1,5**
Funktion	0	1	**0,5**
Hörsaal	4	0	**2**
...	...	...	**...**

Eine andere Möglichkeit ist die Schwerpunktbildung erst nach der Zuordnung aller Dokumente. In der Praxis ist die Qualität der Zuordnung besser, wenn der Klassenrepräsentant kontinuierlich auf seine endgültige Position zuwandert. Dokumente, die dem anderen Repräsentanten zugeordnet werden, lassen diesen durch den Abstand dieser Dokumente zum ersten Repräsentanten eine Position mit deutlicher Abgrenzung beziehen. So können auch zwei relativ ähnliche Dokumente als Startrepräsentanten noch ein gutes Ergebnis für die Klassifikation erzielen.

Man kann jedes Dokument und jeden Repräsentanten auch als Punkt in einem *n*-dimensionalen Raum ansehen, *n* ist dabei die Größe des Wortvektors. Nachstehendes Beispiel reduziert die Darstellung auf zwei Dimensionen und soll die Entwicklung des Repräsentanten und deren Vorzüge sichtbar machen:

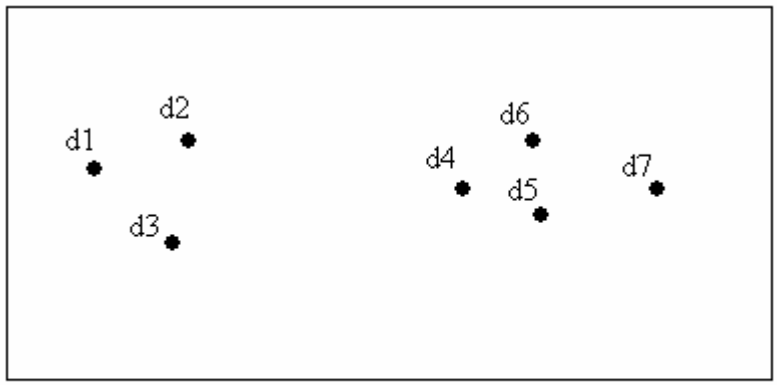

Abbildung 4-17: Dokumente als Vektoren im 2-dim. Raum - 1

Es sollen die sieben Dokumente (d1, d2, ... d7) durch das Clusteringverfahren K-Means in zwei Gruppen gespalten werden. Die graphische Darstellung in Abbildung 4-17 stellt die Verteilung der Dokumente im 2-dimensionalen Raum dar. Ein optimales Ergebnis wäre die Zuordnung der Dokumente d1 – d3 zu Klasse 1 und die Dokumente d4 - d7 zu Klasse 2. Der erste Schritt in diesem Verfahren ist die Bestimmung zweier Dokumente als Startrepräsentanten für die zu bestimmenden Cluster. Für diese Auswahl gibt es wenig Hilfsmittel, in der Regel werden die Dokumente zufällig gewählt. Günstig für das Ergebnis wären etwa Startrepräsentanten d1 und d7, da die Zuordnung der restlichen Dokumente dann einfach wäre. Um die Probleme des Verfahrens aufzuzeigen, werden aber die Dokumente d5 und d6 als Startpunkte gewählt. Jetzt werden nacheinander alle Dokumente gemäß einer Abstandsfunktion den Klassenrepräsentanten zugeordnet, dieser Abstand ist hier durch den euklidischen Abstand im 2-dimensionalen Raum angedeutet. Bleibt der Repräsentant bis zum Ende der Zuordnungen unverändert, so erhielte man als Ergebnis Abbildung 4-18.

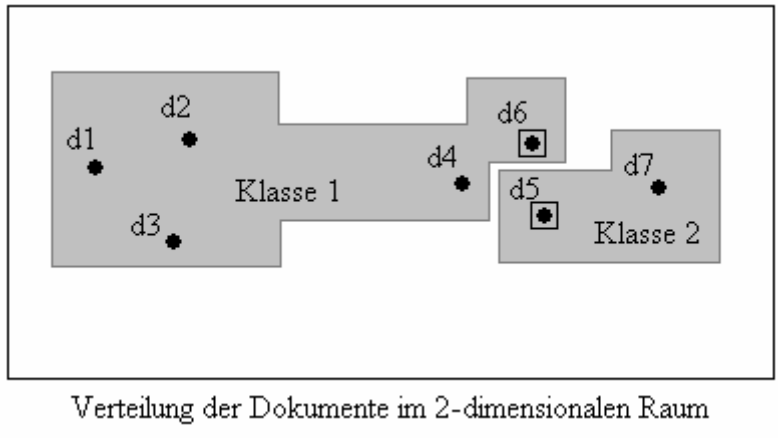

Abbildung 4-18: Dokumente als Vektoren im 2-dim. Raum - 2

Aufgrund ihrer räumlichen Nähe werden die Dokumente d5 und d7 zu Klasse 2 zugeordnet, die restlichen Dokumente zu Klasse 1. Gegenüber dem gewünschten Ergebnis sind zwei Dokumente „falsch“ zugeordnet bedingt durch eine ungünstige Wahl des Startrepräsentanten. Das Ergebnis wäre schlechter ausgefallen, hätte man ein Dokument vom äußeren Rand der Abbildung, also

etwa d1 oder d7 als Startpunkt bestimmt. Eine Verbesserung des Verfahrens ist die sofortige Neuberechnung des betroffenen Repräsentanten nach der Zuordnung eines Dokuments. Dieses durchschnittliche Dokument bewegt sich mit jedem neu zur Klasse hinzukommenden Dokument in den räumlichen „Mittelpunkt“ des Clusters.

Im Gegensatz zu einem statischen Repräsentanten beeinflusst jedoch die Reihenfolge der Dokumente die Zuordnung und damit auch das Ergebnis. Repräsentiert wird Klasse 1 zu Beginn von Dokument d6, Klasse 2 von Dokument d5. Wird zum Beispiel das Dokument d1 zuerst zugeordnet (zu Klasse 1) verschiebt sich der Schwerpunkt in Richtung dieses Dokuments:

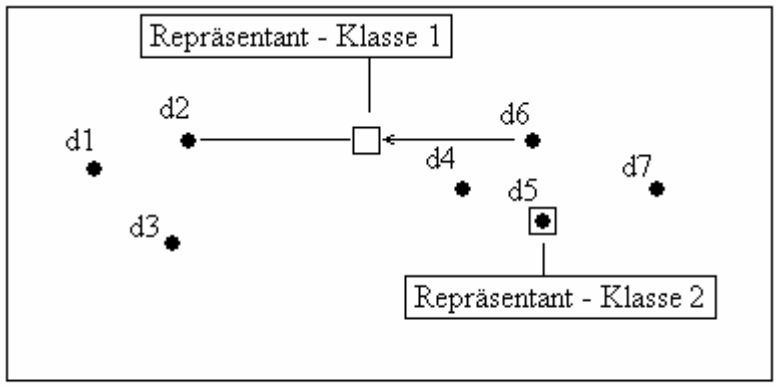

Abbildung 4-19: Dokumente als Vektoren im 2-dim. Raum - 3

Wichtig bei diesem Verfahren ist, dass die Dokumente, die zu Beginn die Repräsentanten stellen, nicht automatisch den entsprechenden Klassen zugeordnet werden. In Abbildung 4-19 wäre das Dokument nach der Verschiebung des Repräsentanten der Klasse 1 dem der Klasse 2 bereits näher. Entscheidend für das Ergebnis sind die ersten Dokumente, die zugeordnet werden. Sie bestimmen den „Kurs“ der Klassifikation. In der Praxis kann man das ausnützen, in dem man die Repräsentanten im Falle einer Wiederholung der Teilung wieder verwendet und nur die Reihenfolge der Dokumente variiert. Im Mittel führt das zu genauso guten Ergebnissen wie eine Neuwahl der Startpunkte. Wenn man das Verfahren nach Abbildung 4-19 fortführt erhält man folgendes Ergebnis:

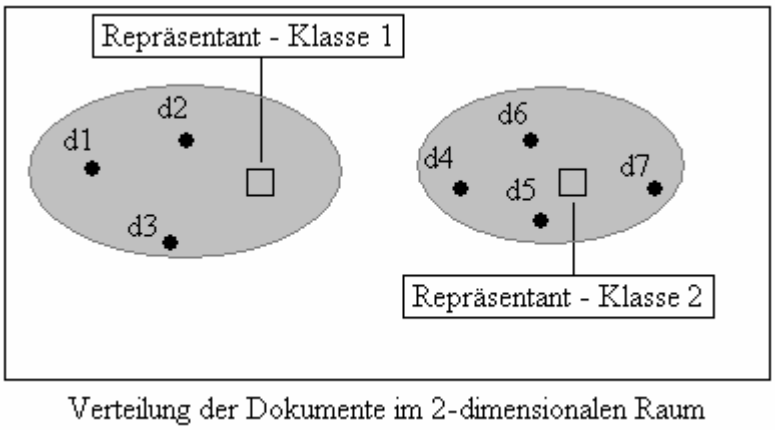

Abbildung 4-20: Dokumente als Vektoren im 2-dim. Raum - 4

Abschließend kann man sagen, dass die Berechnung des Schwerpunkts nach jedem zugeordneten Dokument im Schnitt bessere Ergebnisse liefert, da die Wahl der Startrepräsentanten weniger großen Einfluss auf das Resultat ausübt. Durch eine weitere Verfeinerung des K-Means-Verfahrens verliert dieser Vorteil jedoch wieder etwas an Gewicht. Gemeint ist das „Refining“ der Klassen:

Refining:

„Refining“ bedeutet soviel wie Nachbesserung des Resultats. Die ermittelten Schwerpunkte der Klassen sollten sich bereits in die Zentren von „Dokumenthaufen“ bewegt haben. Jedoch kann es einzelne Ausreißer geben, die dem Startrepräsentanten oder einem seiner Nachfolger zugeordnet wurden, aber eigentlich dem endgültigen Schwerpunkt einer anderen Klasse jetzt näher sind. Als Beispiel dient Abbildung 4-18: bestimmt man jetzt die Repräsentanten der dort vorhandenen Klassen, so erhält man Abbildung 4-21.

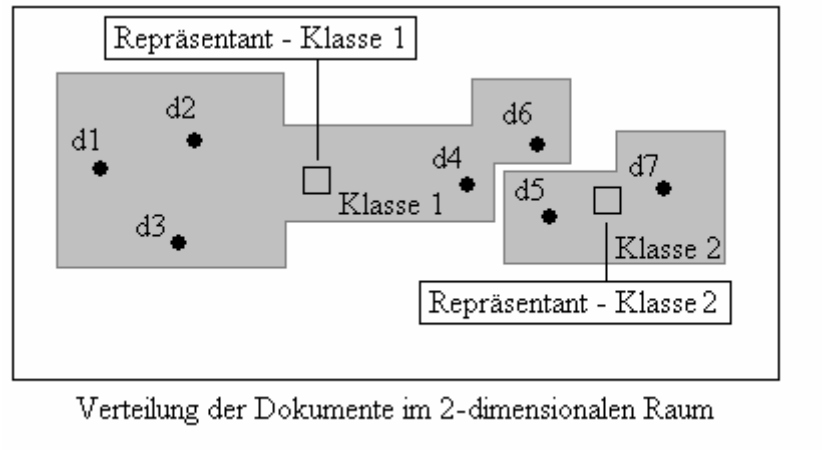

Abbildung 4-21: Dokumente als Vektoren im 2-dim. Raum - 5

Deutlich zu erkennen ist z.B., dass Dokument d6 dem Repräsentanten der Klasse 2 näher ist als dem der Klasse 1, welchem es zugeordnet worden ist. Beim Nachbessern werden nochmals alle Dokumente den Repräsentanten zugeordnet und der Schwerpunkt neu berechnet. Für das Beispiel bedeutet das: Dokument d1, d2 und d3 werden Klasse 1 zugeordnet, Dokument d4, d5, d6 und d7 zu Klasse 2: siehe Abbildung 4-22.
Der Refining-Schritt kann mehrmals wiederholt werden, bis die Repräsentanten eine stabile Position erreicht haben. Im Beispiel würde sich bei weiteren Refining-Schritten an der Zuordnung der Dokumente zu den Klassen nichts mehr ändern, lediglich die Schwerpunkte würde näher an die jeweiligen Mittelpunkte der beiden Klassen heranrücken. Auch hier ist es günstig, nach jeder Dokumentzuordnung den Repräsentanten neu zu berechnen. Damit konvergiert das Verfahren schneller. Zu Beginn jedes Refining-Schritts wird dabei das Gewicht des Repräsentanten auf 1 gesetzt, d.h. er entspricht einem Dokument. Wird ihm das erste Dokument zugeordnet, wandert der neue Schwerpunkt auf den Mittelpunkt einer Geraden zwischen altem Schwerpunkt und der Dokumentposition: siehe Abbildung 4-19. Sein Gewicht beträgt jetzt 2 Dokumente, d.h. mit zunehmender Anzahl an zugeordneten Dokumenten verändert ein neu hinzukommendes Dokument die Position des Repräsentanten immer weniger.

Durch das Rücksetzen des Gewichts ergibt sich ebenfalls eine Beschleunigung des Konvergenzverhaltens.

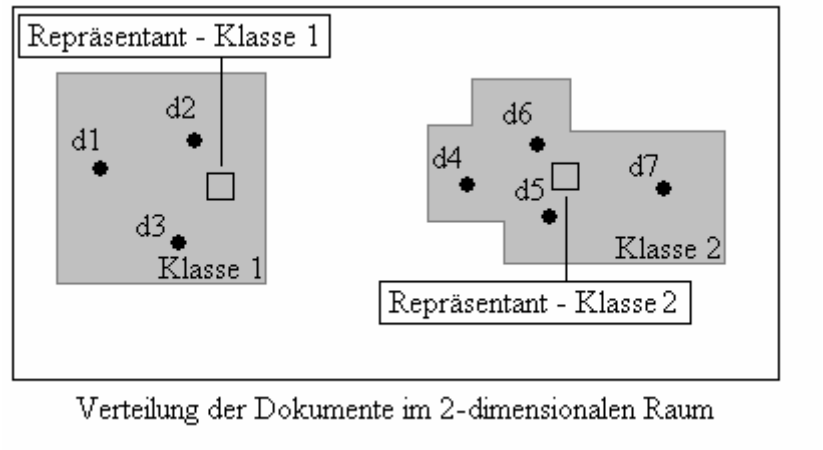

Abbildung 4-22: Dokumente als Vektoren im 2-dim. Raum - 6

Die Ergebnisse kann man jetzt betrachten, indem man die Klassenrepräsentanten als Dokumente ansieht und deren wichtigste Wörter durch Gewichtung mit den IDF-Werten der Begriffe bestimmt: die Anzahl der Vorkommen jedes Wortes des Repräsentanten wird mit seinem IDF-Wert multipliziert und die entstandene Liste sortiert nach der höchsten Gewichtung. Nachstehend ein Beispiel einer Dokumentklasse für die Web-Präsenz der TU München: Mathematik und Informatik:

Begriff (Stammform)	Anzahl Vorkommen	IDF-Wert	Gewichtung
schleifeninvariant	6,8878	6,5028	44.79
Bytecod	7,3715	5,9920	44.17
Escap	6,6961	5,6555	37.87
Call	10,6928	3,3387	35.7
Direkt	6,2758	4,8934	30.71
Orleansstrass	6,5670	4,3825	28.78
Zusammenfassung	4,1028	6,5028	26.68
Eu	4,3029	5,2988	22.8
Transit	4,3978	4,8934	21.52
Bearbeitungsfrist	3,3933	6,2151	21.09
Address	10,8301	1,5780	17.09
Null	6,9066	2,3919	16.52

wwr-praktikum	1,7444	7,6014	13.26
Webapp	2,0194	5,9920	12.1
Ergang	1,9260	6,2151	11.97
html-version	2,1892	5,2988	11.6
Come	1,9627	5,6555	11.1
Berufsbild	1,7586	6,2151	10.93
Fog	1,5639	6,5028	10.17
Chines	1,9095	5,1165	9.77
...	...	...	...

Angegeben sind die gewichtigsten 20 Terme. Dieses Gewicht wird hauptsächlich durch häufiges Auftreten der Begriffe gebildet, die Multiplikation mit dem zugehörigen IDF-Wert setzt die Häufigkeit mit dem allgemeinen Gewicht des Begriffs im gesamten Textkorpus in Relation.
In diesem Cluster sind hauptsächlich Dokumente enthalten, die zu einem bestimmten Informatik-Lehrstuhl der TU München gehören. Inhaltlich werden zumeist Programmieraufgaben für Studenten behandelt, wie aus folgenden enthaltenen unveränderten Begriffen zu ersehen ist:

Schleifeninvariante(n)
Bytecode
escape
call
direkt(en)
Transition(en)
addresse
null
html-version
webapplication(s)

Teile des Lehrstuhls befinden sich in der „Orleansstrasse“, „Bearbeitungsfrist“ und „Zusammenfassung“, beziehen sich auf die Angaben von Übungsaufgaben. Die restlichen Begriffe können von einem menschlichen Betrachter nur als Rauschen aufgefasst werden, der Algorithmus sieht jedoch ein konstantes Auftreten von Begriffen wie „chinesisch“ oder „ergangen“ auf vielen der zur Klasse gehörenden Dokumente als gewichtig an. Ebenfalls gut zu erkennen ist, dass alle 20 Begriffe im Durchschnitt mindestens 1,5 Mal pro Dokument dieser Klasse vorkommen, die häufigsten knapp 11 Mal pro Dokument. Das ist der Kerngedanke des Verfahrens: inhaltlich verwandte Texte enthalten eine feste Menge gemeinsamer Begriffe. Je besser die Wortliste auf inhaltlich bedeutsame Terme reduziert ist, desto besser das Ergebnis dieses Verfahrens.

4.3.2.4.5. Verbindung von inhaltlichen und strukturellen Gewichten der Dokumente

Zwei Teilziele sind bis hier erreicht worden:

1. Bewertung der Dokumente anhand ihrer Position im Graphen der Web-Präsenz (PageRank)
2. Einteilung der Dokumente in inhaltlich verwandte Klassen

Gewünschtes Resultat des gesamten Bewertungsverfahrens ist die Einteilung der Dokumente in für Siemens relevante und nicht relevante Dokumente. Der PageRank-Wert eines Dokuments alleine reicht dafür nicht aus, ebenso wenig genügt dafür die Klassifikation aller Dokumente. Bei 1 Mio. Dokumente ergibt eine Berechnung von 250 Clustern im Schnitt 4.000 Dokumente pro Klasse. Viele Dokumente werden in Klassen eingeteilt, die man als thematisch relevant beurteilen kann (z.B. durch Betrachtung der 20 gewichtigsten Terme), jedoch wird durch Unzulänglichkeiten der Wortliste (Rauschen) und durch die Ungenauigkeit des Verfahrens (s. 3.4.6) ein gewisser Prozentsatz der Dokumente falsch eingeordnet. Zur Lösung des Problems wird im UNICO – Web-Roboter ein mehrstufiges Verfahren eingesetzt:

Im Folgenden gilt: der **Experte** ist eine oder mehrere Personen, die fähig ist/sind, Themengebiete festzulegen und eine Menge von Dokumenten nach diesen Themengebieten zu bewerten.

- Festlegung von Themengebieten durch den Experten
- Beurteilung der Klassen durch den Experten
- Erlernen der Expertenbeurteilung durch ein neuronales Netz
- Bewertung aller Dokumente durch das trainierte neuronale Netz

Ergebnis soll eine Bewertung der automatisch klassifizierten Dokumente (z.B. durch ein K-Means-Verfahren) bezüglich vom „Experten“ vorgegebener Themen sein. D.h. jedes Dokument eines automatisch erstellten Clusters wird einem bestimmten Thema zugeordnet.

Zunächst werden alle verfügbaren Informationen festgelegt, die für jedes Dokument bestimmt werden können. Bei einer Einteilung des gesamten Textkorpus in 250 Klassen ist pro Dokument nicht nur die jeweilige Klasse bestimmt, sondern auch der Abstand zu allen anderen Klassen. Ist der Abstand eines Dokuments zu verschiedenen unterschiedlichen Klassen gering, so geht diese Information durch das Clustering verloren. Jedoch ist es häufig der Fall, dass der Inhalt eines Dokuments mehrere Themengebiete schneidet. Auch kann z.B. große Distanz eines Dokuments zu einem bestimmten Thema für sich genommen Aussagekraft besitzen. Abschließend kommt der PageRank-Wert des Dokuments dazu. Das bedeutet 251 unterschiedliche Daten sind in diesem Beispiel verfügbar. Die Schwierigkeit besteht jetzt darin, diese zwei unterschiedlichen Arten von Informationen, den PageRank-Wert und den Abstand zu allen Klassenrepräsentanten, zu gewichten. Ohne den PageRank könnte man eine Variante der Bayes’schen Klassifikation verwenden, da alle Eingangswerte gleichen Typs sind. In diesem Fall würde ein Experte jedem Cluster ein festes Gewicht zuteilen, indem er z.B. die 20 wichtigsten Begriffe der Klasse zur Bestimmung des Inhalts heranzieht. Dieses Gewicht drückt die Nähe einer Klasse von Dokumenten zu einem vom Experten vorgegebenen Thema aus: z.B.

Zuvor festgelegte Themengebiete:
Informatik, Biologie, Chemie, Mathematik, Physik, Elektrotechnik

Klasse von Dokumenten, erstellt durch ein Clustering - Verfahren	Themenzuordnung durch Experten	Gewichtsangabe durch Experten
Klasse 0	Informatik	0,8
Klasse 1	Biologie	0,3
	Chemie	0,6
Klasse 2	Mathematik	0,9
Klasse 3	Informatik	0,1
	Mathematik	0,5
Klasse 4	Physik	0,7
Klasse 5	Informatik	0,2
	Elektrotechnik	0,5
	Physik	0,5
...	...	...

Zuordnung der Klassen/Cluster zu den Themengebieten

Erläuterung:

Zur Beurteilung der durch das Clustering-Verfahren erstellten „Klasse 0“ betrachtet der Experte die wichtigsten Begriffe der Klasse, die aus dem Repräsentanten (ein Wortvektor) zu ersehen sind (Auswahl der z.B. 20 Wörter mit dem höchsten Wert). Zusätzlich wird ihm eine Auswahl an Dokumenten aus dieser Klasse angezeigt. Anhand dieser Informationen entscheidet der Experte, dass die „Klasse 0“ zu 80% Themen und Dokumente aus der Informatik behandelt. Die restlichen 20% sind keinem der vom Experten vorgegebenen Themen zuzuordnen.

Für jedes Dokument wird der Grad seiner Zugehörigkeit zu einem Themengebiet des Experten bestimmt durch Aufsummieren aller Produkte des Abstands zu einer Klasse mit der jeweiligen Gewichtsangabe. Dabei ist zu beachten, dass der Dokumentabstand zu einem Klassenrepräsentanten ausgedrückt wird durch unterschiedliche Verfahren. So ist bei der im UNICO - Web-Roboter verwendeten p-Korrelation (s. 3.4.1.2.4) ein Dokument „näher“ zu einer Klasse, je größer der Ergebniswert ist. In jedem Fall ist es notwendig, die Abstandswerte der Dokumente zu normieren: so wird beim UNICO – Web-Roboter das Ergebnis der p-Korrelation auf einen Wert zwischen 0 und 100 abgebildet. Das ist vor allem bei dieser Abstandsfunktion wichtig, da hier sehr große Werte im negativen Bereich auftreten können. Wird eine

Abstandsfunktion mit kleinen Werten als bestem Ergebnis verwendet, wie z.B. der euklidische Abstand, so ist der Kehrwert des Ergebnisses zu verwenden. Für das nachstehende Beispieldokument d ergibt sich:

Klasse von Dokumenten, erstellt durch ein Clustering - Verfahren	Nähe des **Beispieldokuments d** (p-korreliert) zum Klassenrepräsentanten	Themenzuordnung durch Experten	Gewichtsangabe durch Experten
Klasse 0	100	Informatik	0,8
Klasse 1	5	Biologie	0,3
	3	Chemie	0,6
Klasse 2	34	Mathematik	0,9
Klasse 3	25	Informatik	0,1
	19	Mathematik	0,5
Klasse 4	4	Physik	0,7
Klasse 5	13	Informatik	0,2
	11	Elektrotechnik	0,5
	7	Physik	0,5
...	...	...	...

Das Bespieldokument d stammt aus der „Klasse 0“, was durch seinen hohen Wert (100) aus der p-Korrelation erkennbar ist. Die Korrelationswerte zu den restlichen Klassen sind ebenfalls bekannt.

Damit ergibt sich für das Beispieldokument d bezüglich der angegeben Klassen 0-5 die nachstehende Themenzuordnung:

Informatik:	100 * 0,8 + 25 * 0,1 + 13 * 0,2	= **85,1**
Biologie:	5 * 0,3	= **1,5**
Chemie:	3 * 0,6	= **1,8**
Mathematik:	34 * 0,9 + 19 * 0,5	= **40,1**
Physik:	4 * 0,7 + 7 * 0,5	= **6,3**
Elektrotechnik:	11 * 0,5	= **5,5**

... (weitere Themengebiete)

Somit ist anzunehmen, dass es sich bei Dokument d um einen Text aus der Informatik handelt, der hohe Wert für Mathematik ist vielleicht durch die Verwandtschaft der beiden Gebiete begründet.

Das Bewertungsverfahren des UNICO – Web-Roboters versucht jedoch, auch den PageRank-Wert mit in das Ergebnis einfließen zu lassen. Ein möglicher Ansatz wäre z.B. die Multiplikation der Werte der Themenzuordnung mit dem PageRank-Wert, also z.B. sei der PageRank-Wert von d: pr(d) = 0,5. Dann würden sämtliche Werte der Zuordnung halbiert. Eventuell ist damit der Einfluss des PageRanks zu groß und eine Skalierung des PageRanks auf die Hälfte brächte ein gutes Ergebnis oder es ist günstig, für jede Klasse oder Themengebiet den PageRank eigens zu skalieren. Viele Tests mit dem UNICO – Web-Roboter haben im Allgemeinen kein befriedigendes Ergebnis geliefert. Allenfalls in Einzelfällen konnten ausreichende Lösungen durch Versuche bestimmt werden, jedoch ist die Skalierung des PageRanks bezüglich eines Themengebietes bei jeder Web-Präsenz unterschiedlich. Der Aufwand für jede Web-Präsenz steigt durch diese Versuche enorm an.

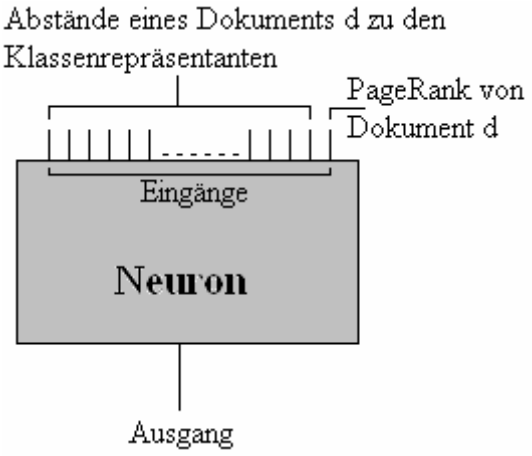

Abbildung 4-23: Aufbau eines Neurons

Daher wird ein anderes Verfahren eingesetzt, um diese speziellen Muster für jede Web-Präsenz zu erkennen: ein Verfahren zur Mustererkennung in Form eines neuronalen Netzes. Im ersten Schritt wird ein Trainingsset an Dokumenten zufällig aus dem Textkorpus ausgewählt und durch einen Experten in Themengebiete eingeteilt und bewertet. Die Anzahl der unterschiedlichen Themengebiete ist ausschlaggebend für den Aufbau des Netzes. Für **jedes Thema** wird ein Neuron angelegt, das **speziell für dieses Themengebiet** die Verhältnisse von PageRank und den Abständen zu allen Klassenrepräsentanten erlernen soll. Ein solches Neuron ist in Abbildung 4-23 zu sehen, Details über Neuronen und neuronale Netze sind in [39] ausführlich beschrieben. Jedes Neuron ist für ein durch den Experten bestimmtes Thema zuständig und liefert Wert zwischen 0 und 1 als Ergebnis, wobei gilt: Je höher der Ergebniswert, umso besser paßt ein Dokument zu diesem Thema.

Im UNICO – Web-Roboter wird in der ersten Einsatzphase mit nur einem „Themengebiet“ gearbeitet, das sämtliche Ingenieurswissenschaften einschließt, d.h. das Netz besteht nur aus einem Neuron, das erlernen soll, welche Dokumentklassen bei welchem PageRank–Wert relevant für dieses Themengebiet sind.

Das gelernte „Wissen“ eines Neurons steckt dabei in den Gewichtungen der Eingänge, die im Laufe der Trainingsphase so verändert werden sollten, um das Erlernte zu reproduzieren und auch auf andere Eingaben hin anzuwenden. Auf diese Weise soll die Verbindung von PageRank und Abstandswerten entstehen. Ein neuronales Netz für drei Themengebiete ist in Abbildung 4-24 zu sehen. Um dieses Wissen zu erlernen, werden Trainingsdaten benötigt:

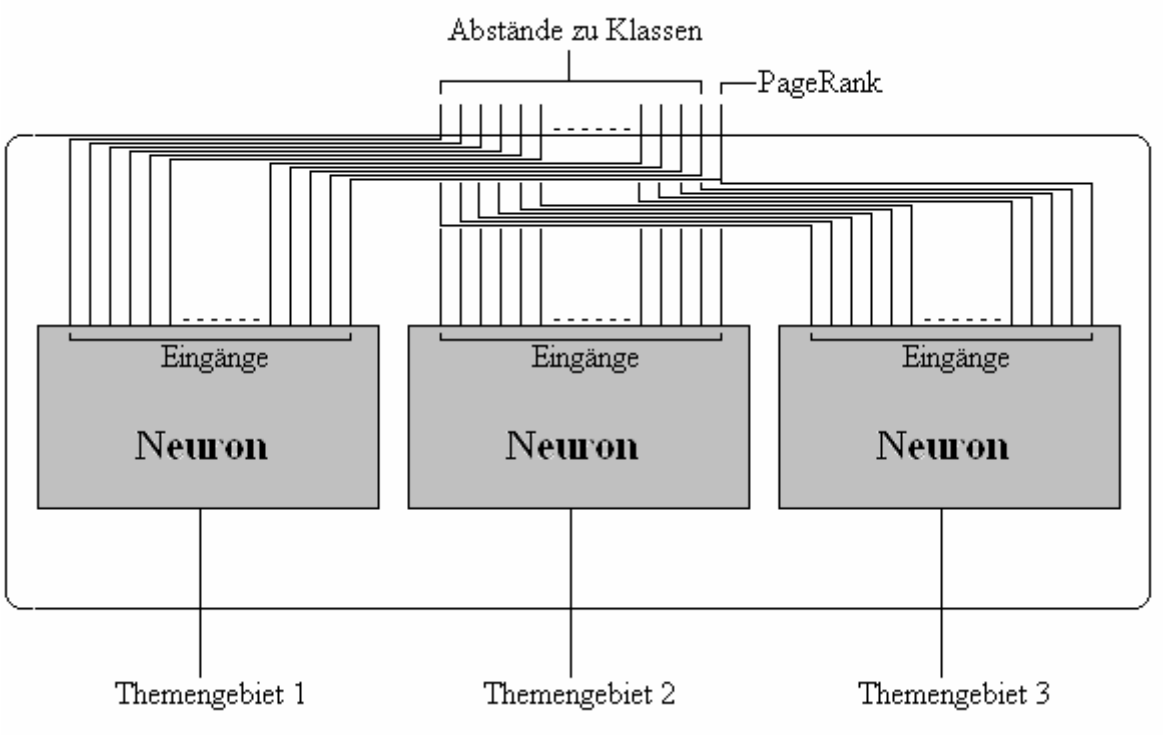

Abbildung 4-24: Neuronales Netz für drei Themengebiete

Erstellen der Trainingsdaten:

Aus jeder Klasse von Dokumenten wird abhängig von der Klassengröße eine Anzahl von Dokumenten zufällig ausgewählt und dem Experten (für die zu bestimmenden Themengebiete) zur Bewertung vorgelegt. Dieser bewertet jedes Dokument mit Werten zwischen 0 und 1. Zusätzlich kann er den Klassenrepräsentanten, der einem durchschnittlichen Dokument entspricht, betrachten, um die gesamte Klasse einschätzen zu können. Je mehr Dokumente in die Bewertung eingehen, umso genauer wird die vom betroffenen Neuron gelernte Information sein.

Ein bearbeitetes Trainingset für das Themengebiet „Informatik“ könnte z.B. so aussehen:

Klasse 0, Dokument 2: Relevanz für das Thema Informatik nach Expertenurteil: 90%
Klasse 0, Dokument 6: Thema Informatik 60%
Klasse 1, Dokument 13: Thema Informatik 80%
Klasse 1, Dokument 22: Thema Informatik 50%
Klasse 2, Dokument 54: Thema Informatik 0%
Klasse 2, Dokument 77: Thema Informatik 10%
…

Für jedes Themengebiet ist ein bestimmtes Neuron zuständig, das unabhängig von den anderen Neuronen den Zusammenhang von Dokumenten, Klassen und PageRank zu einem bestimmten Themengebiet erlernen soll.
Für das zuständige Neuron wird das Trainingsset erweitert durch die Abstände der Trainingsdokumente zu allen Klassen, die durch das Clustering ermittelt worden sind. So entstehen bei 250 Klassen 250 Abstandswerte für jedes Dokument und einem PageRank-Wert. *Jedes Trainingsdokument besteht somit aus 251 Eingangswerten und einem Sollwert*: der Expertenbeurteilung für dieses Dokument. So ergibt sich für aus obiges Beispiel:

	Eingangswerte für das Neuron „Informatik“					Zu lernender Ausgangswert
Dokument	Abstand zu Klasse 0	Abstand zu Klasse 1	…	Abstand zu Klasse 250	PageRank	**Sollwert: Expertenbewertung für das Themengebiet „Informatik“**
2	100	51	…	9	45,66	**0,9**
6	37	29	…	17	11,30	**0,6**
13	78	12	…	2	42,06	**0,8**
22	12	10	…	11	34,59	**0,5**
54	15	5	…	34	7,81	**0,0**
77	22	49	…	0	55,89	**0,1**
…	…	…	…	…	…	**…**

Im folgenden Schritt wird das überwachte Lernen des Netzes vollzogen: alle Trainingsdaten mit ihren Sollwerten werden an das Netz gegeben und ein Lernalgorithmus verändert die anfangs zufälligen Eingangsgewichtungen bei jedem Dokument. Grundlage für die Veränderung ist die Abweichung des Ergebniswertes vom Sollwert. Die Eingabe des Trainingssets in das Netz wird solange fortgesetzt, bis der Fehler der Ausgabe einen genügend kleinen Wert erreicht hat. Damit kann das Netz die Expertenbewertung der Trainingsdaten reproduzieren. Wenn die Trainingsmenge ausreichend groß gewählt wurde und die Bewertungen durch den Experten sinnvoll gesetzt worden ist, so genügt das erlernte „Wissen“ des Netzes, um auch alle anderen Dokumente der Web-Präsenz zu beurteilen.

Reproduktion

Für alle Dokumente der Web-Präsenz werden in diesem Schritt ihr jeweiliger Abstand zu allen Klassenrepräsentanten berechnet und zusammen mit ihrem PageRank-Wert als Eingabe in das neuronale Netz gegeben. Die erhaltenen Ergebnisse werden für jedes Dokument und Themengebiet in der Datenbank abgelegt und dienen als allgemeine Bewertung für die Recherche.

Kapitel 5 Ergebnisse

5.1.Testbetrieb des UNICO-WebRoboters

Web-Präsenz:	TU München: Informatik und Mathematik
Bestimmung der Einschränkungen:	Dauer ca. 2 Stunden
Anzahl Dokumente:	115.710 (Retrieval in ca. 15 Stunden)
Anzahl Klassen:	250 (Berechnung in ca. 3,5 Stunden)
Größte Klasse:	993 Dokumente
PageRank:	Berechnung in etwa 2,5 Stunden
Trainingsset:	etwa 900 Dokumente bei 3-5 Dokumente pro Klasse
Themengebiete:	1 (relevant/ nicht relevant)
Expertenbewertung des Trainingssets:	Dauer etwa 3 Stunden
Lernphase:	500 Iterationen, Dauer etwa 1 Stunde

Bewertung aller Dokumente der Web-Präsenz: Dauer etwa 5 Stunden

Ergebnis: Jedes Dokument hat einen Relevanzwert zu dem Themengebiet durch das neuronale Netz zwischen 0 und 1 erhalten. Wenn man einen Schwellwert von 0,4 wählt, ergibt sich folgende Aufstellung:

	Relevant	Nicht Relevant	Verteilung der Dokumente
Relevanzwert > 0,4	74%	26%	81%
Relevanzwert <= 0,4	32%	68%	19%

Wenn man also die Dokumente, die als nicht relevant eingestuft sind (Schwellwert <= 0,4), aus der Datenbank löscht, so bleiben 81% der Daten erhalten, wovon 74% wirklich relevant sind (durch Stichproben ermittelt). Man hat 32% von 19% aller Daten fälschlich gelöscht: etwa 7.000 Dokumente (6% der Gesamtmenge).

Bei einem Gesamtaufwand von ungefähr 32 Stunden entfallen auf den selbständigen Teil des UNICO – Web-Roboters 27 Stunden, etwa 85%. Der zeitliche Aufwand des Experten beträgt etwa 5 Stunden. Größere Web-Präsenzen erfordern höheren zeitlichen Aufwand für beide Seiten, jedoch bleibt der prozentuale Anteil etwa gleich. Das Ergebnis entspricht einer Einteilung der Daten nach Themengebieten, die Grundlage einer spezialisierten Suchmaschine.

Stichproben bei anderen Web-Präsenzen ergaben qualitativ vergleichbare Resultate.

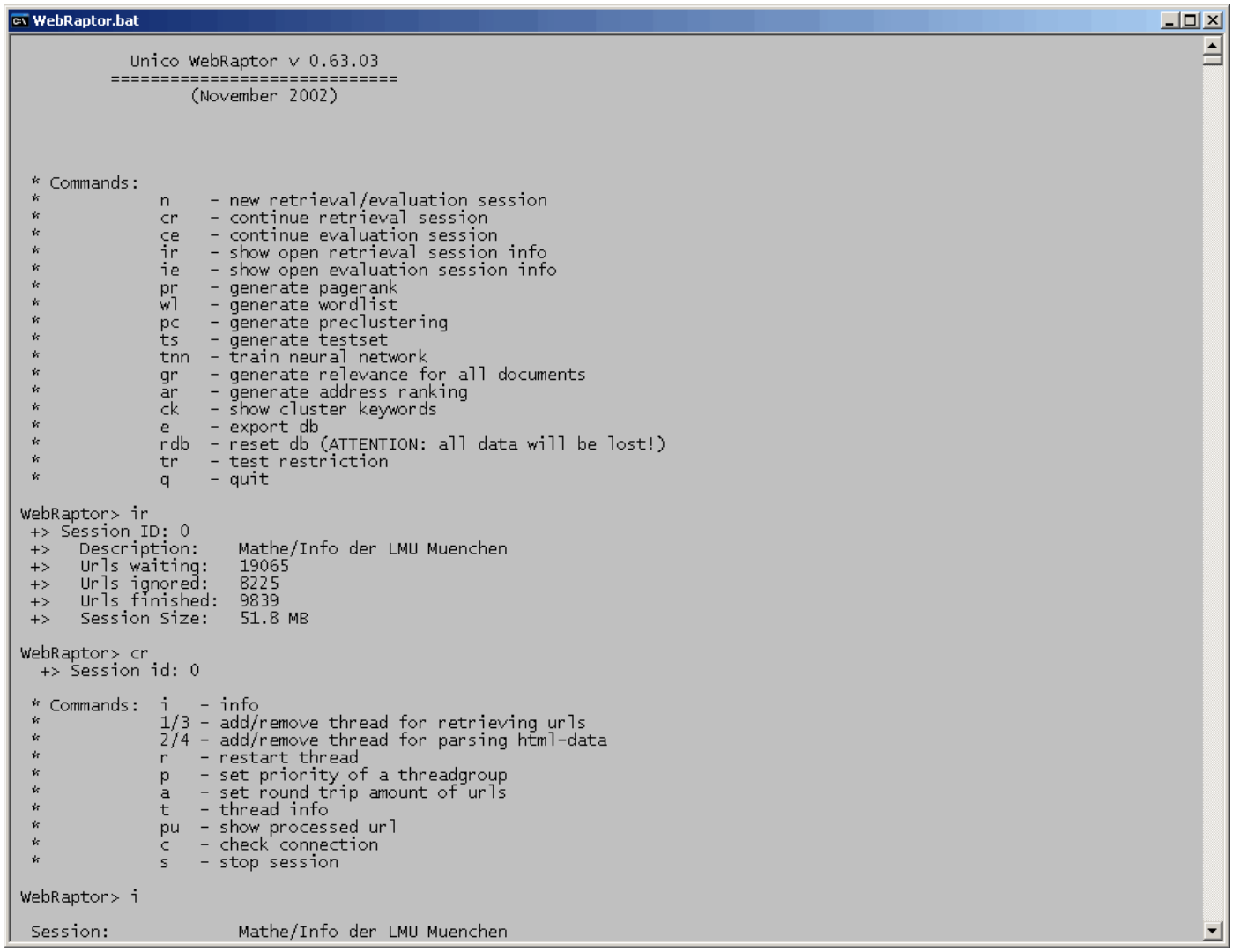

Abbildung 5-1: Textbasierte Oberfläche des UNICO-WebRoboters

5.2.Bewertung

Große Schwierigkeiten bereitet die Bewertung der Tests. Bei einer Million Dokumenten hätte eine aussagekräftige Stichprobe (2%) eine Größe von 20.000 Dokumenten. Der Aufwand zur Überprüfung der Zuordnung zu den Themengebieten wäre enorm hoch. Kleinere Stichproben dagegen beinhalten eine größere Fehlerquote. Letztendlich muss der Benutzer an der Recherche-Oberfläche des UNICO-Portals die Entscheidung über die Qualität des Ergebnisses treffen. Aber selbst hier tritt das Problem auf, dass der durchschnittliche Anwender maximal die ersten 20 Treffer des Ergebnisses betrachtet unabhängig von der Gesamtzahl der Treffer. Somit würden z.B. bei 100 Treffern 80% unbewertet bleiben. Zudem bietet das vorgestellte Konzept des UNICO-WebRoboters noch keine Integration eines oder mehrerer Suchbegriffe in die Bewertung der Dokumente, sondern lediglich eine Einteilung des Textkorpus in Themengebiete. Fehlerquellen bei der Zuordnung sind beim automatischen Clustering, im PageRank-Verfahren, in der Expertenzuordnung und im Erlernen von Wissen durch das neuronale Netz zu erwarten. Je größer die Gesamtmenge an Daten ist, desto schlechter wird auch das Ergebnis im Verhältnis sein, da man die Anzahl der automatisch generierten Cluster nicht beliebig erhöhen kann ohne dem Experten erheblichen Mehraufwand in der Zuordnung zuzumuten. Grundsätzlich kann man feststellen, dass die Qualität des Ergebnisses steigt bei zunehmendem Experteneinfluss. Je

geringer die Anzahl der vom Experten festgelegten Themengebiete und der betrachteten Dokumente, umso schlechter ist das Resultat. In den Tests der einzelnen Verfahren kann man feststellen, dass in einigen Fällen auch die Zuordnung eines Dokuments zu mehreren Themengebieten die Qualität des Ergebnisses erheblich verbessert. Insgesamt gesehen konnte im Rahmen dieser Arbeit jedoch keine Verbesserung im Allgemeinen festgestellt werden. Auch ist anzumerken, dass selbst bei maximalem Aufwand bei Themenzuordnung durch einen Experten das Resultat nicht optimal sein kann. Das liegt an der Qualität der verwendeten Clustering-Verfahren, die aufgrund von beschränkten Informationen nur ein mittelmäßiges Ergebnis liefern können. Durch menschliche „Zusatzarbeit“ kann man das Gesamtergebnis lediglich etwas verbessern.

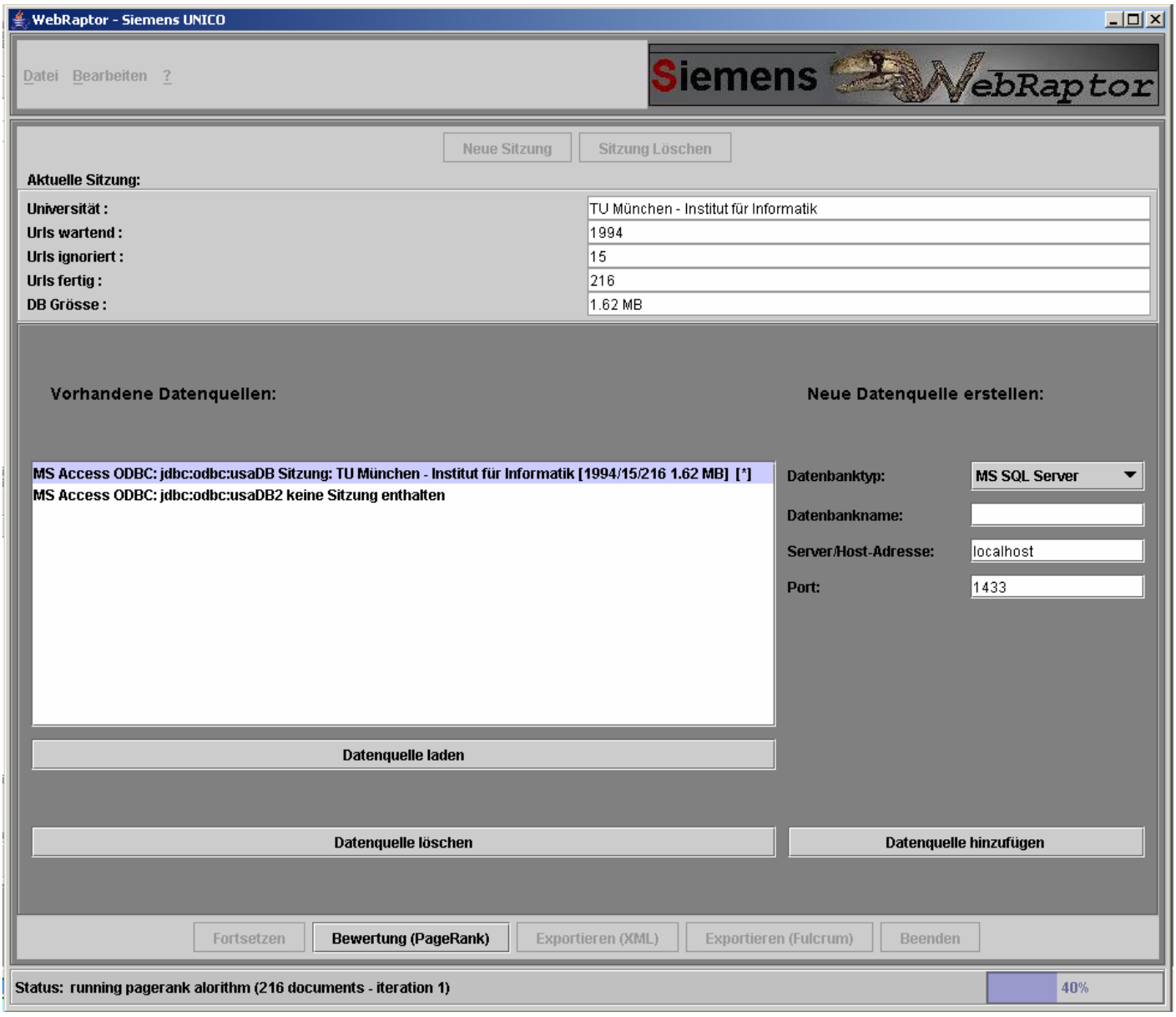

Abbildung 5-2: Graphische Oberfläche des UNICO-WebRoboters – Sitzungsauswahl

5.3.Perspektiven

Zum jetzigen Zeitpunkt befindet sich noch eine text-basierte Oberfläche des UNICO-WEbRoboters im Einsatz (s. Abbildung 5-1. Im Rahmen einer Diplomarbeit wird für die zukünftige Verwendung ein graphischer Client (s. Abbildung 5-2 und Abbildung 5-3) entwickelt und getestet, um die Benutzung auch für Benutzer ohne Hintergrundwissen über das System sicher zu stellen. Geplant sind zudem die Anbindung an das von Siemens verwendete Datenbanksystem Oracle, sowie die Entwicklung eines eigenen Storage-Systems ähnlich der Architektur von Google (s. 2.3).

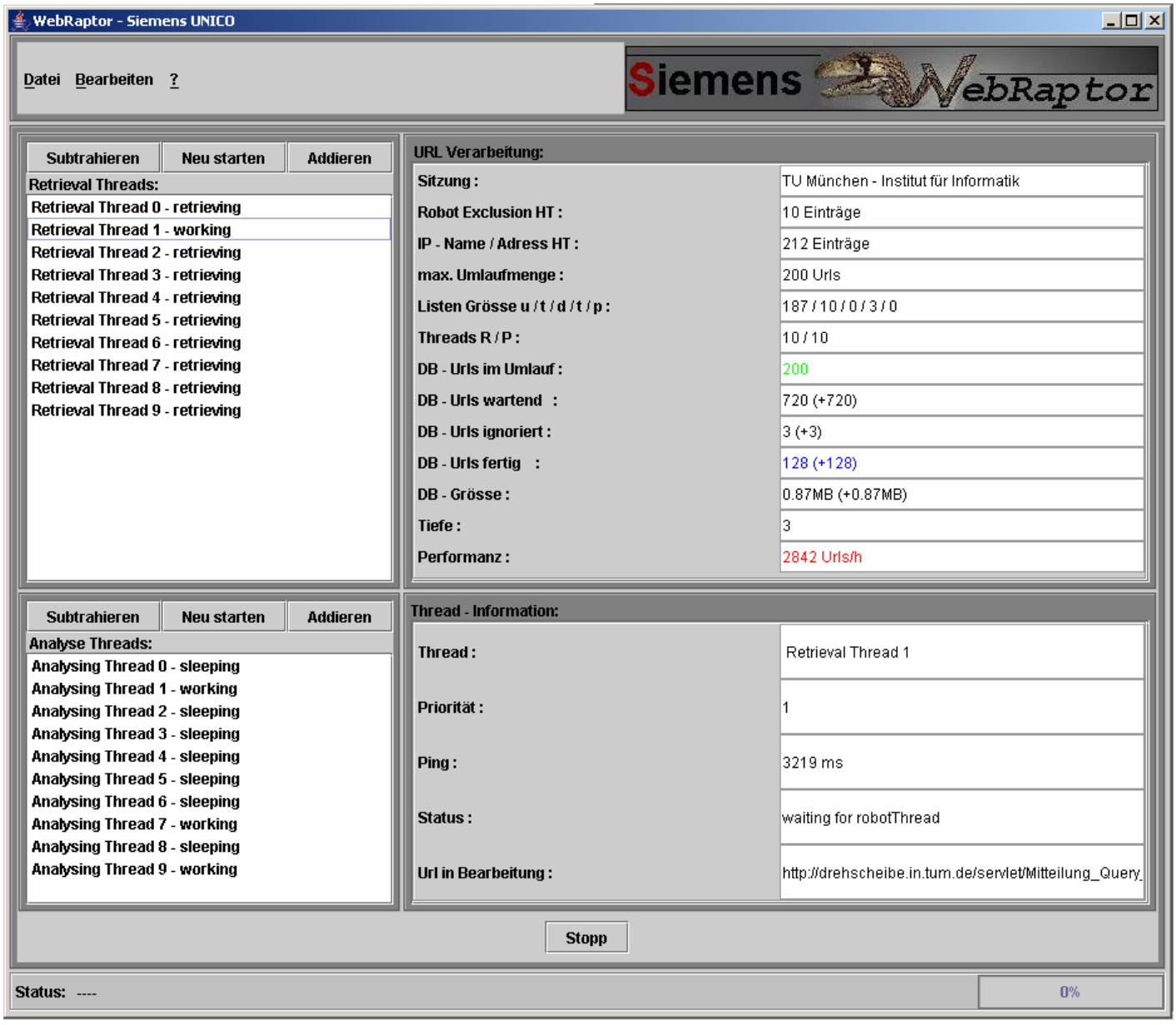

Abbildung 5-3: Graphische Oberfläche des UNICO-WebRoboters - Aktive Sitzung

Dazu ist auch die Entwicklung eines Ranking-Verfahrens für Anfragen geplant. Entscheidend für die Zukunft dieser spezialisierten Suchmaschine ist jedoch das Feedback der Anwender im

Siemens- Intranet. Schwierig gestaltet sich jedoch auch hier die Bewertung dieser Resonanz, sowie die damit verbundene Umsetzung von Änderungen von Parametern in der Bewertung oder gar der Austausch von einzelnen Bewertungskomponenten wie z.B. eines Clusteringverfahrens. Da jedoch die Daten ständig (etwa alle 3-6 Monate) aktualisiert werden sollten, kann man Detailverbesserungen des Verfahrens sukzessive mit neu zu integrierenden Daten einfließen lassen.

Abschließend muss man feststellen, dass keines der heutzutage verwendeten Textbewertungsverfahren wirklich gute Ergebnisse liefert. Dem Anfragenden einer Suchmaschine fällt jedoch die geringe Qualität des Ergebnisses nur schwer auf, da er das optimale Ergebnis nicht kennt und das Ergebnis durch die Einschränkung auf bestimmte Suchbegriffe in aller Regel brauchbare Ergebnisse auch ohne Bewertung resp. Ranking enthalten wird.

Kapitel 6 Referenzen

[1] http://www.searchenginewatch.com/reports/sizetest.html

[2] H. Chen. Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*, 46(3): 194-216, April 1995

[3] Christian Strohmaier. UNICO – Eine Datenbank zur Hochschulkooperation mit der Industrie, *Diplomarbeit an der TU München*, November 1997

[4] Shokri Z. Selim and M.A. Ismail. K-means-type algorithms: Ageneralized convergence theorem and characterization of local optimality. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(1):81-87, January 1984

[5] Charu C. Aggarwal, Stephen C. Gates, and Philip S. Yu. On the merits of building categorization systems by supervised clustering. *In Proc. of the Fifth ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pages 352-356, 1999

[6] D. Boley, M. Gini, R. Gross, E.H. Han, K. Hastings, G. Karypis, V. Kumar, B. Mobasher, and J. Moore. Document categorization and query generation for web document categorization. *Decision Support Systems (accepted for publication)*, 1999

[7] D. Boley. Principal direction divisive partitioning. *Data Mining and Knowledge Discovery*, 2(4), 1998

[8] M. Koster. Robots in the Web: threat or treat, *http://www.robotstxt.org/wc/threat-or-treat.html*, 1997

[9] M. Koster. The Web Robots FAQ, *http://www.robotstxt.org/wc/faq.html*

[10] M. Koster. Guidelines for Robot Writers, *http://www.robotstxt.org/wc/guidelines.html*, 1993

[11] Chen, H., Chung, Y. Ramsey, M., and Yang, C. A Smart Itsy-Bitsy Spider for the Web. *Journal of the American Society for Information Science, Special Issue on AI Techniques for Emerging Information Systems Applications*, 49(7) (1998), 604- 618.

[12] M. Koster. *Evaluation of the Standard for Robot Exclusion*, *http://www.robotstxt.org/wc/eval.html*, 1996

[13] M. Koster. A Standard for Robot Exclusion, *http://www.robotstxt.org/wc/norobots.html*

[14] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. *Proceedings of the Seventh World-Wide Web Conference*, 1998.

[15] M. Steinbach, G. Karypis, V. Kumar. A comparison of document clustering techniques. *KDD Workshop on Text Mining*, 2000

[16] D. Zhang and Y. Dong. An efficient algorithm to rank web resources. *11th World-wide Web Conference*, 2000

[17] S. Brin and L. Page. The anatomy of a large-scale hypertextual (Web) search engine. *Proc. 7th International World Wide Web Conference (WWW7)/Computer Networks*, 30(1-7):107--117, 1998

[18] http://www.cis.uni-muenchen.de/people/strohmaier/

[19] Wrapper-Tool: Araneus (http://www.difa.unibas.it/Araneus)

[20] Wrapper-Tool: Ariadne (http://www.isi.edu/ariadne)

[21] Wrapper-Tool: Garlic (http://www.almaden.ibm.com/cs/garlic/adagency.html)

[22] Wrapper-Tool: Internet Softbots (http://www.cs.washington.edu/research/irdb.intro.html)

[23] Wrapper-Tool: JEDI (http://www.darmstadt.gmd.de/oasys/projects/jedi)

[24] Wrapper-Tool: LAPIS (http://www-2.cs.cmu.edu/rcm/lapis)

[25] Wrapper-Tool: Lixto (http://www.dbai.tuwien.ac.at/proj/lixto)

[26] Wrapper-Tool: NoDoSE (http://www.cs.nwu.edu/adelberg/nodose/nodose.html)

[27] Wrapper-Tool: SoftMealy (http://kaukoai.iis.sinica.edu.tw/mealy/mealyindex.html)

[28] Wrapper-Tool: TSIMMIS (http://www-db.stanford.edu/tsimmis/tsimmis.html)

[29] Wrapper-Tool: University of Maryland Wrapper Generation Project (http://umiacs.umd.edu/labs/CLIP/DARPA/ww97.html)

[30] Wrapper-Tool: W4F (http://db.cis.upenn.edu/Research/w4f.html)

[31] Wrapper-Tool: WIEN (http://www.cs.ucd.ie/staff/nick/home/research/wrappers)

[32] Wrapper-Tool: WebL (http://www.research.digital.com/SRC/WebL)

[33] Wrapper-Tool: XWRAP (http://www.cse.ogi.edu/DISC/XWRAP)

[34] Wrapper-Tool: XWRAP Elite (http://www.cc.gatech.edu/projects/disl/XWRAPElite)

[35] Kohonen, Teuvo. Self-Organizing and Associative Memory. *3rd Edition, Springer,* 1989

[36] Kohonen, Teuvo. Self-Organizing Maps. *Springer,* 1989

[37] Dersch, D.R. Eigenschaften neuronaler Vektorquantisierer und ihre Anwendung in der Sprachverarbeitung. *Reihe Physik, Band 54, Verlag Harry Deutsch*, 1995

[38] Villmann, Thomas. Topologieerhalung in selbstorganisierenden neuronalen Merkmalskarten. Dissertation, *Universität Leipzig*, 1995

[39] Rojas, Raul. Theorie der neuronalen Netze. *Springer*, 1993

[40] Gerald Kowalski, Information Retrieval Systems – Theory and Inplementation, *Kluwer Academic Publishers*, 1997.

[41] C.J. van Rijsbergen, Information Retrieval, *Buttersworth, London, 2nd Edition*, 1989

[42] Bjorner Larsen and Chinatsu Aone, Fast and Effective Text Mining Using Linear-time Document Clustering, *KDD-99, San Diego, California*, 1999

[43] Mu-Chun Su, Hsiao-Te Chang, Fast Self-Organizing Feature Map Algorithm, *Department of Electrical Engineering, Tamkam University*, Taiwan

[44] Text REtrieval Conference (TREC): http://trec.nist.gov

[45] Ken Lang, Newsweeder: Learning to filter netnews, *International Conference on Machine Learning, S. 331-339*, 1995

[46] W.B. Frakes, R. Baeza-Yates: Information Retrieval – Data Structures & Algorithms, *Prentice Hall PTR*, 1992

Kapitel 7 Anhang

7.1. Liste der Abbildungen

7.2. Konfiguration des UNICO-WebRoboters

7.2.1.Allgemein

```
# Error Log
C:\Work\Projects\WebRaptor\log\dba.log
# GetURL Log
C:\Work\Projects\WebRaptor\log\u.log
# ParseURL Log
C:\Work\Projects\WebRaptor\log\pd.log
# Datenbank
jdbc:microsoft:sqlserver://hobbes:1433;DatabaseName=CMellon
# DB-Typ (0=MS Access, 1=MS SQL-Server)
1
# DB Server
rmi://calvin:1098/DB_Server
# Retrieve Threads
10
# Parse Threads
10
# Round Trip Amount
50
# Maximum Redirects
10
# RThread Priority
5
# PThread Priority
5
# dbThread Priority
8
# KeyWords
C:\Work\Projects\WebRaptor\conf\Wordlists\key_tech_de.txt
# StopWords
C:\Work\Projects\WebRaptor\conf\Wordlists\stop_de.txt
```

7.2.2.Retrieval – Sitzungen

7.2.2.1.Beschreibung

Über die Software des UNICO – WebRoboters kann man die gewünschten Daten einer Retrieval-Sitzung von Hand eintragen oder über eine Datei laden. Es existieren Konfigurationsdateien von mehr als 3.000 Universitäten weltweit. Diese enthalten minimal eine Startadresse und eine Einschränkung auf mindestens eine Domäne. Alleine für die USA wurden Konfigurationsdateien für 1.548 Universitäten und andere Lehrinstitute erstellt.

7.2.2.2.Beispiel

```
# Name der Institution:
Informatik an der TU Muenchen
# Start-URL:
http://www.in.tum.de/
# Restrictions:
http://131.159.*.*/
# Ausschluss:
http://131.159.72.20
http://131.159.0.51/rechenbetrieb
http://131.159.0.51/images
http://131.159.0.51/cgi-bin
http://131.159.40.10/cgi-bin
http://131.159.58.16/cgi-bin
http://131.159.16.147/cgi-bin
http://131.159.46.188/cgi-bin
http://131.159.36.40/cgi-bin
http://131.159.60.46/cgi-bin
http://131.159.22.41/cgi-bin
http://131.159.20.1/cgi-bin
http://131.159.18.46/cgi-bin
http://131.159.32.66/cgi-bin
http://131.159.24.47/cgi-bin
http://131.159.38.128/cgi-bin
http://131.159.12.111/cgi-bin
http://131.159.44.2/cgi-bin
http://141.84.218.4/cgi-bin
# Bemerkung:
Test
```

7.3.Daten-Export: XML

Jede exportierte Datensammlung enthält üblicherweise eine Institution, im Beispiel hier die TU München. Eine interne „Document Type Definition“ zur Beschreibung der Daten ist enthalten.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!-- internal DTD -->
<!DOCTYPE university

[
<!ELEMENT university        (description, amount, htmldata+)>
<!ELEMENT description       (#PCDATA)>
<!ELEMENT amount            (#PCDATA)>
<!ELEMENT htmldata          (id, url, title, pagerank, date, depth, length, keywords, data)>
<!ELEMENT id                (#PCDATA)>
<!ELEMENT url               (#PCDATA)>
<!ELEMENT title             (#PCDATA)>
<!ELEMENT pagerank          (#PCDATA)>
<!ELEMENT date              (#PCDATA)>
<!ELEMENT depth             (#PCDATA)>
<!ELEMENT length            (#PCDATA)>
<!ELEMENT keywords          (#PCDATA)>
<!ELEMENT data              (#PCDATA)>
]
>
<university>
<description>
Informatik an der TU Muenchen
</description>
<amount>
28
</amount>
<htmldata>
<id>
0
</id>
<url>
http://www.in.tum.de/
</url>
<title>
TUM Informatik
</title>
<pagerank>
18.559
</pagerank>
<date>
0
</date>
<depth>
0
</depth>
<length>
1980
</length>
<keywords>

</keywords>
<data>
```

tum informatik fakultaet informatik technischen universitaet muenchen home studium forschung fakultaet dienste information alumni login neuer benutzer suche hilfe studium schulportal studienberatung internationale studierende informatik bioinformatik wirtschaftsinformatik vorlesungen skripte forschung schwerpunkte projekte publikationen stellenangebote fakultaet dekanat lehrstuehle professoren mitarbeiterinnen kolloquium dienste rechenbetrieb bibliotheken servicebueros information adresse anfahrt leo jobs alumni adressbuch jahrgangslisten impressum feedback willkommen fakultaet informatik web-portal ersetzt 15 maerz 2002 bisherigen webauftritt fakultaet informatik bisherige angebot informationsdrehscheibe auftretenden anfangs-problemen bitten verstaendnis feedback anregungen webmaster tum de drehscheibe tum de studium schulportal informationen studienfach informatik schuelerinnen schueler lehrerinnen lehrer studienanfaengerinnen bewerbungsfrist wintersemester 2002 2003 15 mai 2002 15 juli 2002 hinweise bewerbung eignungsfeststellungsverfahren fakultaet fakultaet zieht ende sommersemesters 2002 innenstadt campus garching drehscheibe wap-zugang informationen funktionalitaeten drehscheibe beta-test doku wap aktuelle termine tu-film italienisch anfaenger 28 05 2002 19 30 microsoft techtalk thema mobile internet toolkit 04 06 2002 18 tu-film fabelhafte welt amelie 04 06 2002 19 30 neue mitteilungen tu-film italienisch anfaenger 28 05 2002 19 30 systeme systemnahe programmierung terminaenderung hochschulgruendertag juni 2002 muenchen verschiebung sprechstunde studienberatung microsoft techtalk thema mobile internet toolkit 04 06 2002 18 university-competition pc-based simulation workshop bewerbungsaufruf center digital technology management fachschaftszeitung impulsiv nummer 72 drehscheibe item-preview hauptseminar mpi plaetze frei ankuendigungen suche ankuendigungen lehrveranstaltungen personen raeumen web-seiten student muenchen schaut studiosity de
</data>
</htmldata>

<htmldata>
<id>
1
</id>
<url>
http://www.in.tum.de/doku/wap
</url>
<title>
TUM Informatik
</title>
<pagerank>
9.535
</pagerank>
<date>
0
</date>
<depth>
1
</depth>
<length>
3432
</length>
<keywords>

</keywords>
<data>
tum informatik fakultaet informatik technischen universitaet muenchen home studium forschung fakultaet dienste information alumni login neuer benutzer suche hilfe studium schulportal studienberatung internationale studierende informatik bioinformatik wirtschaftsinformatik vorlesungen skripte forschung schwerpunkte projekte publikationen stellenangebote fakultaet dekanat lehrstuehle professoren mitarbeiterinnen kolloquium dienste rechenbetrieb bibliotheken servicebueros information adresse anfahrt leo jobs alumni adressbuch jahrgangslisten impressum feedback wapportal benutzerdokumentation wapportal handelt dienste drehscheibe wap-faehige handys momentan folgende dienste angeboten friends items calendar friends moeglich liste drehscheibe eingetragener freunde einzusehen anzurufen items ueberblick termine mitteilungen bookmarks konversationen interessengebiet communities verschaffen calendar

listet privaten regelmaessigen termine stundenplaneintraege heutigen tag aktuellen uhrzeit voraussetzung wap-faehigen handy drehscheibe account besitzen registriert anlegen neuer benutzer alternativ wap-emulator verwenden empfehlen deckit bedienen ansprechendes layout gibt version linux windows installation programms umstaendlich web-basierten emulator verwenden url geben http drehscheibe tum de index wml handy wap-seiten gelangen anmelden abmelden generell dienst anmelden benutzerkennung passwort identisch normalen drehscheibe-accounts neuen account web-schnittstelle drehscheibe erstellen neuer benutzer schritt schritt schritt angemeldet natuerlich abmelden waehlen menue options punkt logout moeglichkeit login-information anzuschauen session options waehlen herausfinden kennung angemeldet dienste sehen menueauswahl diensten friends items calendar waehlen dienst options gehen follow link anwaehlen befinden seiten wap-dienstes startseite kommen options gehen home anwaehlen dienst friends erfolgreich angemeldet sehen liste eingetragenen freunde personen persoenlichen einstellungen drehscheibe eingtragen liste selbstverstaendlich leer moeglichkeit freunde einzutragen besteht web-interface melden gehen benutzereinstellungen interessenseinstellungen editieren klicken freundesliste sehen liste eingetragenen freunde hilfe suchfunktion benutzerkennung freundes eingeben freundesliste hinzufuegen waehlen eintrag options follow link sehen telefonnummer mobilfunknummer ausgewaehlten person telefonnummer selektieren waehlt handy automatisch nummer dienst items items termine mitteilungen bookmarks anzeigen nachrichten communities aufgelistet persoenlichen einstellungen eingetragen communities entfernen hinzufuegen web-interface anmelden login gehen benutzereinstellungen interessenseinstellungen editieren link interessen neue community-liste zusammenstellen items erstellungsdatum erstellungszeit angezeigt wissen typ termin handelt waehlen kommen detailansicht auskunft typ naehere beschreibung gibt erfahren communities zugeordnet dienst calendar waehlen dienst calendar privaten regelmaessigen termine stundenplantermine heutigen tag aktuellen uhrzeit einsehen bevorstehenden termine persoenlich eingetragenen communities zugeordnet angezeigt eintraege automatisch anfangszeit sortiert zeiten selbstverstaendlich ueberschneiden besonderheit termine angezeigt naechsten tag enden steht fall endzeit enddatum korrekturen ergaenzungen dokumentation willkommen bitte e-mail drehscheibe tum de kochm tum de richten
</data>
</htmldata>

<htmldata>

</university>

7.4.Auszüge aus dem Quellcode

7.4.1.1.Überblick

Der UNICO – WebRoboter ist komplett in Java[23] implementiert. Das Programm ist zusammengesetzt aus 71 Klassen, die auf 9 Java-Packages verteilt sind. Die aktuelle Version besteht aus 46 219 Zeilen Code. Dazu kommen 2 Java-Packages mit zusammen 13 Klassen für Testverfahren, hauptsächlich Clustering und neuronale Netze betreffend. Diese Klassen bestehen aus 16 301 Zeilen Code. Die Entwicklungszeit des Gesamtsystems betrug effektiv etwa 1,5 Jahre.

Folgende Programmpakete sind vorhanden:

com.unico.neuronet: Framework für den Aufbau neuronaler Netze, sowohl zur Mustererkennung als auch für selbstorganisierende Karten

com.unico.raptor: Main-Klassen für die text-basierte und die graphische Oberfläche

com.unico.raptor.common: Allgemeine Hilfsmittel wie z.B. Log-Mechanismen, Datenstrukturen

com.unico.raptor.dbaccess: Datenbankschnittstelle zu MS ACCESS und MS SQL Server

com.unico.raptor.evaluation: Bewertungsroutinen

com.unico.raptor.gui: Bestandteile der graphischen Oberfläche

com.unico.raptor.html: HTML-Parser

com.unico.raptor.internal: Framework für den Retrieval- und Bewertungsprozess

com.unico.raptor.text: Textverarbeitungshilfsmittel wie z.B. Datenstrukturen für die Stopwortliste oder Inversed Document Frequency (IDF)

Im Folgenden sind einige Auszüge aus dem Quellcode dargestellt, vor allem die Implementierung der verwendeten Bewertungsverfahren.

7.4.1.2.PageRank

```
public void generatePageRank(StatusHandler sh)
 {
  inStatus  = "processing pagerank algorithm";
  outStatus = "processing pagerank algorithm";

  //
  // Berechnen des Google-PageRanks
  //
```

[23] http://www.javasoft.com

```
     final PageRankCache cache = new PageRankCache();
     double maxPR = 0.0;      // Maximaler PageRank-Wert für die aktuellen
                              // Daten;
     double E1  = 0.15;       // Konstante, Wahrscheinlichkeit "zufällig" auf
                              // eine Seite zu gelangen
     double E2  = 0.85;       // Konstante, Wahrscheinlichkeit per Link auf
                              // eine Seite zu gelangen
     double R_p = 100.0;      // R(p): Rank einer Seite p, Startwert
     double R_p_old = 100.0;  // R(p)_old: vorhergehender Wert von R(p)
     double R_q = 100.0;      // R(q): Rank einer Seite p, Startwert (p->q)
     int    N   = 0;          // N(q): Anzahl der Links auf der Seite q
     double V   = 1.0;        // Veränderung bei jeder Iteration
     double V_old = 1.0;      // Veränderung der vorhergehenden Iteration
     double change = 1.0;     // Differenz zwischen V und V_old, sollte gegen
                              // 0 gehen
     int    S = 0;            // Die Summe aller Seiten
     double C = 1.0;          // Normalisierungs-Wert
     double SumR = 100.0;     // Summe aller R(p)
     int maxId = -1;          // Grösste Id der zu bewertenden Seiten
     final int CHUNK = 10000; // Maximal 10000 Ids werden auf einmal aus der
                         // DB geholt
final int MAX_CACHE_SIZE = 134217728; // Cache-Grösse in Byte: 128 MB

final String query0   = "DELETE FROM Evaluation";
final String query1   = "SELECT count(*) FROM data";
final String query2_1 = "INSERT INTO evaluation ( id, pageRank, cluster ) SELECT
   data.id, ";
final String query2_2 = ", 0 FROM data";
final String query3   = "SELECT max(id) FROM data";
final String query4_1 = "SELECT id, pagerank FROM evaluation WHERE id >= ";
final String query4_2 = " AND id < ";
final String query4_3 = " ORDER BY id";
final String query5_1 = "SELECT id, pr, count(out) AS Expr1 " +
                        "FROM (SELECT DISTINCT b.srcUrlId AS id, e.pagerank AS pr,
   b2.urlId AS out " +
                        "FROM evaluation AS e, backlink AS b, backlink AS b2 WHERE
   e.id = b.srcUrlId " +
                                "AND NOT b.urlId=b.srcUrlId AND b2.srcUrlId =
   b.srcUrlId AND b.urlId = ";
final String query5_2 = " ) DERIVEDTBL GROUP BY id, pr";
final String query6_1 = "UPDATE evaluation SET pagerank=";
final String query6_2 = " WHERE id=";
final String query7 = "UPDATE evaluation SET pagerank = pagerank * 100 /";
final String query8 = "DELETE FROM cluster WHERE id=0";
final String query9_1 = "INSERT into cluster (id, size) VALUES(0,";
final String query9_2 = ")";

// initialisieren:
// Alle Ids von data lesen und in Evalutaion
// eintragen mit Default-Rank = 100.0
sh.setStatus("running pagerank alorithm (cache size: " + MAX_CACHE_SIZE + " bytes)");

try
 {
   dba.beginTransaction();
   taOpen = true;
  dba.runUpdate(query0);
  Result = dba.runQuery(query1);
  if(Result.length() != 0)
    S = ((Integer)Result.elementAt(0,0)).intValue();
  else S = 0;
```

```
  if(S == 0) return;

  E1 /= S; // E1 = 0.15 / S
  E2 /= S; // E2 = 0.85 / S
  StringBuffer query2 = new StringBuffer();
  query2.append(query2_1).append((100.0 / S)).append(query2_2);
  dba.runUpdate(query2.toString());
  Result = dba.runQuery(query3);
  if(Result.length() != 0)
    maxId = ((Integer)Result.elementAt(0,0)).intValue();
   dba.commitTransaction();
   taOpen = false;
 }
catch(Exception e)
 {
  e.printStackTrace();
  if(taOpen)
   {
    try { dba.abortTransaction(); taOpen = false; }
    catch(Exception e1) { e1.printStackTrace(); }
   }
 }

sh.setStatus("running pagerank alorithm (" + S + " documents)");

// Iteration
boolean cacheOpen = true;
int q, iter=0;
double conv = 1.0/(double)S;
int cacheHits = 0, cacheMiss = 0, nItems = 0;
int runs = maxId / CHUNK;
if(runs * CHUNK < maxId) runs++;

while(change > conv && iter <= 50)
 {
  maxPR = 0.0;
  iter++;
  nItems = 0;
  cacheHits = 0;
  cacheMiss = 0;

  sh.setStatus("running pagerank alorithm (" + S + " documents - iteration " + iter +
   ")");
  sh.setProgress(1);

  for(int k=0; k<runs; k++)
   {
    Result = null;
    final int min = k * CHUNK;
    final int max = min + CHUNK;
    StringBuffer query4 = new StringBuffer();

   query4.append(query4_1).append(min).append(query4_2).append(max).append(query4_3);
    try
     {
       dba.beginTransaction();
       taOpen = true;
      Result = dba.runQuery(query4.toString());
       dba.abortTransaction();
       taOpen = false;
     }
```

```
      catch(Exception e)
       {
         e.printStackTrace();
        if(taOpen)
         {
          try { dba.abortTransaction(); taOpen = false; }
          catch(Exception e1) { e1.printStackTrace(); }
         }
       }

      final int len1 = Result.length();
      for(int i=0; i<len1; i++) // Alle ids
       {
        nItems++;
         final int p = ((Integer)Result.elementAt(i,0)).intValue();  // p
        if(cache.containsSrc(p))
         {
          R_p_old = cache.getSrcPageRank(p);
          cacheHits++;
         }
        else
         {
           R_p_old = ((Double)Result.elementAt(i,1)).doubleValue(); // p_alt
          cacheMiss++;
         }
         R_p     = E1; // R(p)

        if(cache.contains(p))
            {
          cacheHits++;
             Vector src    = cache.getSrc(p);
             Vector srcPr  = cache.getSrcPr(p);
             Vector srcOut = cache.getSrcOutlinks(p);

          final int len = src.size();
             for(int j=0; j<len; j++)
              {
                q   = ((Integer)src.elementAt(j)).intValue();    // q
            R_q = ((Rank)srcPr.elementAt(j)).doubleValue();  // R(q)
                N   = ((Integer)srcOut.elementAt(j)).intValue(); // N
            if(N == 0) N = S;

            C = 100.0 / SumR;
            R_p += C * E2 * R_q / N;
              }
            }
           else
            {
          cacheMiss++;
          StringBuffer query5 = new StringBuffer();
          query5.append(query5_1).append(p).append(query5_2);
          try
           {
             dba.beginTransaction();
             taOpen = true;
            Result2 = dba.runQuery(query5.toString()); // q, R(q), N(q) mit q -> p
             dba.abortTransaction();
             taOpen = false;
           }
          catch(Exception e)
           {
             e.printStackTrace();
```

```
            System.out.println("q5=" + query5);
            System.out.flush();
           if(taOpen)
            {
             try { dba.abortTransaction(); taOpen = false; }
             catch(Exception e1) { e1.printStackTrace(); }
            }
         }

           if(cache.size() + (Result2.length() * 24) > MAX_CACHE_SIZE)
               cacheOpen = false;
           else
               cacheOpen = true;

         final int len2 = Result2.length();
          for(int j=0; j<len2; j++) // alle Seiten q mit q -> p
           {
              q = ((Integer)Result2.elementAt(j,0)).intValue(); // q
           if(cache.containsSrc(q))
            {
             R_q = cache.getSrcPageRank(q);                        // R(q)
             cacheHits++;
            }
           else
            {
             R_q = ((Double)Result2.elementAt(j,1)).doubleValue(); // R(q)
             cacheMiss++;
            }
              N       = ((Integer)Result2.elementAt(j,2)).intValue(); // N
           if(N == 0) N = S;

               if(cacheOpen)
                 cache.setSrc(p, q, N, R_q);

           C = 100.0 / SumR;
           R_p += C * E2 * R_q / N;
          }
          }

       if(R_p > maxPR) maxPR = R_p;
       SumR += R_p - R_p_old;

          if(cache.containsSrc(p) ||
          cache.size() + 12 <= MAX_CACHE_SIZE)
        {
           cache.setSrcPr(p, R_p);
         cacheHits++;
        }
       else
        {
         cacheMiss++;
         final StringBuffer query6 = new StringBuffer();
         query6.append(query6_1).append(R_p).append(query6_2).append(p);
         try
          {
           dba.beginTransaction();
           taOpen = true;
          dba.runUpdate(query6.toString());
           dba.commitTransaction();
           taOpen = false;
          }
         catch(Exception e)
```

```
          {
           e.printStackTrace();
           System.out.println("q6=" + query6);
           System.out.flush();
          if(taOpen)
           {
            try { dba.abortTransaction(); taOpen = false; }
            catch(Exception e1) { e1.printStackTrace(); }
           }
         }
       }

       sh.setProgress(1 + Math.round(79.0f * (float)nItems / (float)S));

       if(R_p < R_p_old)
         V += R_p_old - R_p;
       else
         V += R_p - R_p_old;

      } // Ende for(int i=0; i<Result.length(); i++) // Alle ids
    }   // Ende for(int k=0; k<runs; k++) ...

   V /= S;

   if(V_old < V)
     change = V - V_old;
   else
     change = V_old - V;

   V_old = V;
   V = 0.0;
  } // Ende while(...

 // Alle Items aus dem Cache in die DB zurückschreiben
 final HashMap tmp = cache.getPageRankHM();
 final Set keySet  = tmp.keySet();
 final int setSize = keySet.size();
 final Iterator keyIterator = keySet.iterator();

 try
  {
   dba.beginTransaction();
   taOpen = true;
  }
 catch(Exception e) {}

 sh.setProgress(1 + Math.round(79.0f * (float)nItems / (float)S));
 sh.setStatus("running pagerank alorithm (" + S + " documents - writing back cache");
 for(int i=0; i<setSize; i++)
  {

   final Integer key  = (Integer)keyIterator.next();
   final int pg       = key.intValue();
   final double R_pg  = ((Rank)tmp.get(key)).doubleValue();

    final StringBuffer query6 = new StringBuffer();
   query6.append(query6_1).append(R_pg).append(query6_2).append(pg);
   try
    {
     dba.runUpdate(query6.toString());
    }
   catch(Exception e)
```

```
    {
     e.printStackTrace();
     if(taOpen)
      {
       try { dba.abortTransaction(); taOpen = false; }
       catch(Exception e1) { e1.printStackTrace(); }
      }
     break;
    }
   sh.setProgress(80 + Math.round(10.0f * (float)i / (float)setSize));
  }

 try
  {
   if(taOpen)
    {
     dba.commitTransaction();
     taOpen = true;
    }
  }
 catch(Exception e)
  {
   if(taOpen)
    {
     try { dba.abortTransaction(); taOpen = false; }
     catch(Exception e1) { e1.printStackTrace(); }
    }
  }

 sh.setStatus("running pagerank alorithm (" + S + " documents - normalizing rank
    values");
 sh.setProgress(90);
 int counter = 0;
 try
  {
   dba.beginTransaction();
   counter = dba.runUpdate(query7 + maxPR);
   dba.commitTransaction();
  }
 catch(Exception e)
  {
   e.printStackTrace();
   System.out.println("q7=" + query7);
   System.out.flush();
   if(taOpen)
    {
     try { dba.abortTransaction(); taOpen = false; }
     catch(Exception e1) { e1.printStackTrace(); }
    }
  }

 sh.setProgress(95);
 final String query9 = query9_1 + S + query9_2;
 try
  {
   dba.beginTransaction();
   dba.runUpdate(query8);
   dba.runUpdate(query9);
   dba.commitTransaction();
  }
 catch(Exception e)
  {
```

```
    e.printStackTrace();
    System.out.println("q8=" + query8);
    System.out.flush();
    if(taOpen)
     {
      try { dba.abortTransaction(); taOpen = false; }
      catch(Exception e1) { e1.printStackTrace(); }
     }
   }

  sh.setProgress(90);
  outStatus = "sleeping";
  inStatus  = "sleeping";
  //
  // Ende Google PageRank Berechnung
  //
 }
```

7.4.1.3.K-Means

A. Initialisierung

```
    public void generatePreClustering(int nCluster,
                                      int minFrequency,
                                      int nWords)
     {
      // Bilde n Cluster als Vorbereitung für die Kategorisierung

      // Anzahl der Dokumente bestimmen
      final String queryC0 = "SELECT count(*) FROM data";
  // Cluster rücksetzen
  final String queryC1 = "DELETE FROM cluster WHERE id > 0";
  final String queryC2 = "UPDATE cluster SET size = ";
  final String queryC3 = "UPDATE evaluation SET cluster = 0";

  try
   {
    dba.beginTransaction();
    final DBResult r1 = dba.runQuery(queryC0);
    if(r1 == null || r1.length() == 0)
      return;
    final int size   = ((Integer)r1.elementAt(0,0)).intValue();
    dba.runUpdate(queryC1);
    dba.runUpdate(queryC2 + size);
    dba.runUpdate(queryC3);
    dba.commitTransaction();
   }
  catch(Exception e)
   {
    e.printStackTrace();
    System.out.println("queryC0=" + queryC0);
    System.out.println("queryC1=" + queryC1);
    System.out.println("queryC2=" + queryC2);
    System.out.println("queryC3=" + queryC3);
    System.out.flush();
    if(taOpen)
     {
      try { dba.abortTransaction(); taOpen = false; }
```

```
      catch(Exception e1) { e1.printStackTrace(); }
     }
    return;
   }
  buildWordCache(minFrequency, nWords);

  final HashMap cache = new HashMap(); // Hier soll das Ergebnis rein
  final int maxCacheSize = 100000;     // Max. Zwischenspeicher in Byte

  for(int i=0; i<nCluster-1; i++)
   {
    if(!bisection(word2id, id2idf, cache, maxCacheSize, 2)) break;
   }
  for(int i=0; i<5; i++)
   {
    if(!refineCluster(cache, maxCacheSize, word2id, id2idf, 2)) break;
   }
 }
```

B. Bisektion

```
     private boolean bisection(HashMap word2id,    // Hashtabelle: Wort -> ID
                               HashMap id2idf,     // IDF-Werte zu den Wort-IDs
                               HashMap cache,      // Rückgabe-Object
                               int maxCacheSize,   // Max. Zwischenspeicher in
                                                   // Byte
                               int method)         // Vergleichs-Methode:
                                                   // 1 => Skalarprodukt
                                                   // 2 => p-Korrelation
      {
       boolean retval = true;

       final String query0  = "SELECT max(id), max(size) FROM cluster";
       final String query1  = "SELECT id FROM cluster WHERE size = ";
       final String query2_0 = "SELECT MAX(d.length) FROM data d INNER JOIN evaluation
e ON d.id = e.id WHERE e.cluster=";
       final String query2_1 = "SELECT MAX(d.length) FROM data d INNER JOIN evaluation
e ON d.id = e.id WHERE e.cluster=";
       final String query2_2 = " AND NOT d.length=";
       final String query2_3 = "SELECT d.id FROM data d INNER JOIN evaluation e ON d.id
= e.id WHERE (d.length=";
       final String query2_4 = " OR d.length=";
       final String query2_5 = ") AND e.cluster=";

       final String query3   = "SELECT max(id) FROM evaluation WHERE cluster = ";
       final String query4_1 = "SELECT id FROM evaluation WHERE id >= ";
       final String query4_2 = " AND id < ";
       final String query4_3 = " AND cluster = ";
       final String query5_1   = "UPDATE evaluation SET cluster=";
       final String query5_2   = " WHERE id=";

       int maxClusterId      = -1;
       int selectedClusterId = -1;
       long intermediate;
       int S = 0;         // Anzahl der Docs
       int maxId = -1;    // Grösste id
       int CHUNK = 1000;  // Maximal 1000 Ids werden auf einmal aus der DB geholt
       int id1 = -1;
       int id2 = -1;
       int maxDataLen1 = 0, maxDataLen2 = 0;
```

```
Stopwords sw = ctrl.getStopwords();

try
 {
   dba.beginTransaction();
   taOpen = true;
  Result = dba.runQuery(query0);
  if(Result.length() != 0)
   {
    maxClusterId = ((Integer)Result.elementAt(0,0)).intValue();
    S = ((Integer)Result.elementAt(0,1)).intValue();
   }
  else S = 0;

  if(S == 0)
   {
    dba.abortTransaction();
    return false;
   }

  System.out.println("   +> max. cluster size = " + S);

  Result = dba.runQuery(query1 + S);
  if(Result.length() != 0)
    selectedClusterId = ((Integer)Result.elementAt(0,0)).intValue();
  else
   {
    dba.abortTransaction();
    return false;
   }

  System.out.println("   +> selected cluster = " + selectedClusterId);

  Result = dba.runQuery(query2_0 + selectedClusterId);
  if(Result.length() == 1)
    maxDataLen1 = ((Integer)Result.elementAt(0,0)).intValue();
  else
   {
    Result.print();
    dba.abortTransaction();
    return false;
   }

  Result = dba.runQuery(query2_1 + selectedClusterId + query2_2 + maxDataLen1);
  if(Result.length() == 1)
    maxDataLen2 = ((Integer)Result.elementAt(0,0)).intValue();
  else
   {
    Result.print();
    dba.abortTransaction();
    return false;
   }

  final String query2 = query2_3 + maxDataLen1 +
                        query2_4 + maxDataLen2 +
                        query2_5 + selectedClusterId;
  Result = dba.runQuery(query2);
  if(Result.length() >= 2)
   {
    id1 = ((Integer)Result.elementAt(0,0)).intValue();
    id2 = ((Integer)Result.elementAt(1,0)).intValue();
```

```
     }
    else
     {
      Result.print();
      dba.abortTransaction();
      return false;
     }

    Result = dba.runQuery(query3 + selectedClusterId);
    if(Result.length() != 0)
      maxId = ((Integer)Result.elementAt(0,0)).intValue();
     dba.commitTransaction();
     taOpen = false;
   }
  catch(Exception e)
   {
    System.out.println("q2=" + query2_1 + selectedClusterId + query2_2 + maxDataLen1);
    e.printStackTrace();
    if(taOpen)
     {
      try { dba.abortTransaction(); taOpen = false; }
      catch(Exception e1) { e1.printStackTrace(); }
     }
    return false;
   }

  if(id1 == -1 && id2 == -1) return false;

  final Vector usedIds = new Vector();
  final int newClusterId = ++maxClusterId;
  HashMap centroid1, centroid2;
  int c1Size = 0;
  int c2Size = 0;
  int bestSize, nextBestId;
  Vector newCluster;
  // neue clusternr. generieren, id2 updaten in evaluation.cluster,
  // id1 bleibt der alte cluster
  if(!createCluster(newClusterId)) return false;
  // usedIds.addElement(new Integer(id1));
  // usedIds.addElement(new Integer(id2));

  while(true)
   {
    newCluster = new Vector();
    // newCluster.addElement(new Integer(id2));

    bestSize = 0;
    nextBestId = -1;

    if(cache.containsKey(new Integer(id1)))
     {
      // Kopie(!!) des Wort-Vektors von id1 anlegen,
      // damit der Wort-Vektor von id1 nicht verändert wird
      centroid1 = util.copyHashMap((HashMap)cache.get(new Integer(id1)));
     }
    else if(cache.size() == maxCacheSize)
     {
      centroid1 = buildWordVector(getFulltext(id1, sw), word2id, id2idf);
     }
    else
     {
      centroid1 = buildWordVector(getFulltext(id1, sw), word2id, id2idf);
```

```
      cache.put(new Integer(id1), centroid1);
     }

    if(cache.containsKey(new Integer(id2)))
     {
      // Kopie(!!) des Wort-Vektors von id2 anlegen,
      // damit der Wort-Vektor von id2 nicht verändert wird
      centroid2 = util.copyHashMap((HashMap)cache.get(new Integer(id2)));
     }
    else if(cache.size() == maxCacheSize)
     {
      centroid2 = buildWordVector(getFulltext(id2, sw), word2id, id2idf);
     }
    else
     {
      centroid2 = buildWordVector(getFulltext(id2, sw), word2id, id2idf);
      cache.put(new Integer(id2), centroid2);
     }

    System.out.println("   +> doc id used for centroid " + selectedClusterId + ": " +
                       id1 + " (size: " + centroid1.size() + ")");
    System.out.println("   +> doc id used for centroid " + newClusterId + ": " +
                       id2 + " (size: " + centroid2.size() + ")");

    c1Size = 0;
    c2Size = 0;

    // Iteration über alle Volltexte des grössten Clusters
    int nItems = 0;
    int runs = maxId / CHUNK;
    if(runs * CHUNK < maxId) runs++;

    for(int k=0; k<runs; k++)
     {
      final int min = k * CHUNK;
      final int max = min + CHUNK;
      final String query4 = query4_1 + min +
                            query4_2 + max +
                            query4_3 + selectedClusterId;
      try
       {
         dba.beginTransaction();
         taOpen = true;
        Result = dba.runQuery(query4);
         dba.abortTransaction();
         taOpen = false;
       }
      catch(Exception e)
       {
        System.out.println("q4=" + query4);
         e.printStackTrace();
        if(taOpen)
         {
          try { dba.abortTransaction(); taOpen = false; }
          catch(Exception e1) { e1.printStackTrace(); }
         }
        return false;
       }

      final int len = Result.length();
      if(len == 0) continue;
```

```
      int offset = 0;
      if(len > 1)
        offset = (new Random(System.currentTimeMillis())).nextInt(len-1);
      int j;
      for(int i=0; i<len; i++) // Alle Volltexte
       {
        j = i+offset;
        if(j>=len)
          j -= len;
        final int id = ((Integer)Result.elementAt(j,0)).intValue();
        // if(id == id1 || id == id2) continue;
        nItems++;

        HashMap v;
        if(cache.containsKey(new Integer(id)))
          v = (HashMap)cache.get(new Integer(id));
        else if(cache.size() == maxCacheSize)
          v = buildWordVector(getFulltext(id, sw), word2id, id2idf);
        else
         {
          v = buildWordVector(getFulltext(id, sw), word2id, id2idf);
          cache.put(new Integer(id), v);
         }

        if(v.size() > bestSize && !usedIds.contains(new Integer(id)))
         {
          nextBestId = id;
          bestSize = v.size();
         }

        // Vergleich ...
        float compare1, compare2;
        if(method == 1)
         {
          compare1 = compare(centroid1, v, id2idf);
          compare2 = compare(centroid2, v, id2idf);
         }
        else
         {
          compare1 = compare2(centroid1, v, id2idf);
          compare2 = compare2(centroid2, v, id2idf);
         }

        if(compare1 >= compare2)
         {
          c1Size++;
          centroid1 = intersect(centroid1, v, c1Size);
         }
        else
         {
          c2Size++;
          newCluster.addElement(new Integer(id));
          centroid2 = intersect(centroid2, v, c2Size);
         }
        System.out.print("     +> item: " + nItems + "/" + S + " - cluster " +
selectedClusterId + ": " + c1Size +
                         ", cluster " + newClusterId + ": " + c2Size + "
\r");
        System.out.flush();
       } // Ende Chunk-Schleife
     } // Ende Run-Schleife
```

```
    if(c1Size > c2Size)
     {
      final int minSize = S * 5 / 100;
      if(c2Size >= minSize)
        break;
      // id1 = nextBestId;
      // usedIds.addElement(new Integer(id1));
      System.out.println("   +> repeating run ... size of cluster " + newClusterId +
                         " is too low [" + c2Size + " should be " + minSize + "]");
     }
    else
     {
      final int minSize = S * 5 / 100;
      if(c1Size >= minSize)
        break;
      // id2 = nextBestId;
      // usedIds.addElement(new Integer(id2));
      System.out.println("   +> repeating run ... size of cluster " + newClusterId +
                         " is too low [" + c1Size + " should be " + minSize + "]");
     }
   } // Ende while-Schleife

  // Cluster-Size und Centroid-Vector -> DB
  System.out.println("\n   +> updating cluster " + selectedClusterId + ", " +
newClusterId);

  // Nur alle Docnos des neuen Cluster updaten
  for(int i=0; i<c2Size; i++)
   {
    final Integer id = (Integer)newCluster.elementAt(i);
    final String query5 = query5_1 + newClusterId + query5_2 + id;
    try
     {
      dba.beginTransaction();
       taOpen = true;
      dba.runUpdate(query5);
       taOpen = false;
       dba.commitTransaction();
     }
    catch(Exception e)
     {
      retval = false;
      System.out.println("q5 = " + query5);
      e.printStackTrace();
      if(taOpen)
       {
        try { dba.abortTransaction(); taOpen = false; }
        catch(Exception e1) { e1.printStackTrace(); }
       }
     }
   }

  final String query6 = "DELETE FROM cluster WHERE id = " + selectedClusterId;
  final String query7 = "INSERT INTO cluster VALUES("+ selectedClusterId + ", ?, " +
c1Size + ")";
  final String query8 = "DELETE FROM cluster WHERE id = " + newClusterId;
  final String query9 = "INSERT INTO cluster VALUES("+ newClusterId + ", ?, " + c2Size
+ ")";

  try
   {
    dba.beginTransaction();
```

```
     taOpen = true;
    dba.runUpdate(query6);
    dba.setBLOB(query7, new BLOB(centroid1));
    dba.runUpdate(query8);
    dba.setBLOB(query9, new BLOB(centroid2));
     taOpen = false;
    dba.commitTransaction();
   }
  catch(Exception e)
   {
    retval = false;
    e.printStackTrace();
    if(taOpen)
     {
      try { dba.abortTransaction(); taOpen = false; }
      catch(Exception e1) { e1.printStackTrace(); }
     }
   }

  return retval;
 }
```

C. Refinement

```
    private boolean refineCluster(HashMap cache,
                                  int maxCacheSize,
                                  HashMap word2id,
                              HashMap id2idf,
                                  int method)
     {
      boolean retval = true;

      int S = 0;
      int maxId = -1;
      int cSize = 0;

      final String query0 = "SELECT id FROM cluster WHERE size>0";
      final String query1 = "SELECT wordVector FROM cluster WHERE id=";
      final String query2 = "SELECT count(*), max(id) FROM evaluation";

      final String query3_1 = "SELECT id FROM evaluation WHERE id >= ";
      final String query3_2 = " AND id < ";

      HashMap centroid;
      int[] centroidId = null;
      Object[] centroidVector = null;
      int[] centroidSize = null;

      Stopwords sw = ctrl.getStopwords();

      try
       {
        dba.beginTransaction();
          taOpen = true;
        final DBResult r = dba.runQuery(query0);
        if(r == null || r.length() == 0) return false;
        cSize = r.length();
        centroidId     = new int[cSize];
        centroidVector = new Object[cSize];
        centroidSize   = new int[cSize];
```

```
        for(int i=0; i<cSize; i++)
         {
          final int id   = ((Integer)r.elementAt(i,0)).intValue();
          final BLOB bl = dba.getBLOB(query1 + id);
          if(!bl.isEmpty())
            centroid = (HashMap)bl.toObject();
          else
            centroid = new HashMap();

          centroidId[i]     = id;
          centroidVector[i] = centroid;
          centroidSize[i]   = 0;
         }
        final DBResult r2 = dba.runQuery(query2);
        if(r2 == null || r2.length() == 0) return false;
        S = ((Integer)r2.elementAt(0,0)).intValue();
        maxId = ((Integer)r2.elementAt(0,1)).intValue();
          dba.abortTransaction();
          taOpen = false;
       }
      catch(Exception e)
       {
        retval = false;
        e.printStackTrace();
        if(taOpen)
         {
          try { dba.abortTransaction(); taOpen = false; }
          catch(Exception e1) { e1.printStackTrace(); }
           }
       }

      // Iteration über alle Volltexte
      int nItems = 0;
      int CHUNK = 1000;
      int runs = maxId / CHUNK;
      if(runs * CHUNK < maxId) runs++;

      for(int k=0; k<runs; k++)
       {
        final int min = k * CHUNK;
        final int max = min + CHUNK;
        final String query3 = query3_1 + min +
                              query3_2 + max;
        try
         {
            dba.beginTransaction();
            taOpen = true;
          Result = dba.runQuery(query3);
            dba.abortTransaction();
            taOpen = false;
         }
        catch(Exception e)
         {
            e.printStackTrace();
          if(taOpen)
           {
            try { dba.abortTransaction(); taOpen = false; }
            catch(Exception e1) { e1.printStackTrace(); }
             }
          return false;
         }
```

```
        final int len = Result.length();
        for(int i=0; i<len; i++) // Alle Volltexte
         {
          final int id = ((Integer)Result.elementAt(i,0)).intValue();
          nItems++;

          HashMap v1;
          if(cache.containsKey(new Integer(id)))
            v1 = (HashMap)cache.get(new Integer(id));
          else if(cache.size() == maxCacheSize)
            v1 = buildWordVector(getFulltext(id, sw), word2id, id2idf);
          else
           {
            v1 = buildWordVector(getFulltext(id, sw), word2id, id2idf);
            cache.put(new Integer(id), v1);
           }

          // Vergleich mit allen Centroiden ...
          float best = -10000.0f;
          float res;
          int bestCluster = -1;
          for(int j=0; j<cSize; j++) // Vergleich mit allen Cluster-Repräsentanten
           {
            final HashMap v2 = (HashMap)centroidVector[j];

            if(v2 == null) continue;

            if(method == 1)
              res = compare(v1, v2, id2idf);
            else
              res = compare2(v1, v2, id2idf);

            if(res > best || j==0)
             {
              best = res;
              bestCluster = j;
             }
           }
          final int clusterId = centroidId[bestCluster];
          System.out.print("     +> refining item: " + nItems + "/" + S + "
\r");
          System.out.flush();
          updateClusterId(id, clusterId);
          centroidVector[bestCluster] =
intersect((HashMap)centroidVector[bestCluster],
                                                  v1,
                                                  ++centroidSize[bestCluster]);
         } // Ende Chunk-Schleife
       } // Ende Run-Schleife

      for(int i=0; i<cSize; i++)
       {
        final int id = centroidId[i];
        final HashMap wordVector = (HashMap)centroidVector[i];
        final int size = centroidSize[i];

        // Cluster-Size und Centroid-Vector -> DB
        final String query4 = "DELETE FROM cluster WHERE id = " + id;
        final String query5 = "INSERT INTO cluster VALUES(" + id + ", ?, " + size +
")";
```

```
        try
         {
          dba.beginTransaction();
          taOpen = true;
          dba.runUpdate(query4);
          dba.setBLOB(query5, new BLOB(wordVector));
            taOpen = false;
            dba.commitTransaction();
         }
        catch(Exception e)
         {
          retval = false;
          e.printStackTrace();
          if(taOpen)
           {
            try { dba.abortTransaction(); taOpen = false; }
            catch(Exception e1) { e1.printStackTrace(); }
             }
        }
       }
      System.out.println();
      return retval;
     }

     public WeightedObject[] clusterKeywords(int clusterNr, int n)
     {
      if(n<1) return null;

      if(id2word == null)
        buildWordCache();
      WeightedObject[] retval = null;
      final String query = "SELECT wordVector FROM cluster WHERE id=" + clusterNr;
      HashMap wordVector = null;

      try
       {
        dba.beginTransaction();
          taOpen = true;
        final BLOB bl = dba.getBLOB(query);
        wordVector = (HashMap)bl.toObject();
          taOpen = false;
          dba.commitTransaction();
       }
      catch(Exception e)
       {
        e.printStackTrace();
        if(taOpen)
         {
          try { dba.abortTransaction(); taOpen = false; }
          catch(Exception e1) { e1.printStackTrace(); }
           }
       }

      if(wordVector != null)
       {
        final int size = wordVector.size();
        final WeightedObject[] list = new WeightedObject[size];
        final Set set  = wordVector.entrySet();
        final Iterator iter = set.iterator();
        for(int i=0; i<size; i++)
         {
          Map.Entry e = (Map.Entry)iter.next();
```

```
          final Integer wordId = (Integer)e.getKey();
          final String word = (String)id2word.get(wordId);
          final float df   = ((Float)e.getValue()).floatValue();
          final float idf  = ((Float)id2idf.get(wordId)).floatValue();
          // System.out.println(" Word: " + word + ", df=" + df + ", idf=" + idf + " -
> weight=" + (df*idf));
          list[i] = new WeightedObject(word, df * idf);
         }
        Arrays.sort(list);
        if(list.length < n) n = list.length;
        retval = new WeightedObject[n];
        for(int i=0; i<n; i++)
          retval[i] = list[i];
       }

      return retval;
     }
```

D. Korrelation / Skalarprodukt

```
     // Implementierung der Korrelation / Skalarprodukt
     private float compare(HashMap v1, HashMap v2, HashMap id2idf)
     {
      HashMap v3, v4; // v3 soll der kleinere der beiden Wortvektoren werden
      if(v1.size() > v2.size())
       {
        v3 = v2;
        v4 = v1;
       }
      else
       {
        v3 = v1;
        v4 = v2;
       }

      float retval = 0.0f;
      final int size      = v3.size();
      final Set entry     = v3.entrySet();
      final Iterator iter = entry.iterator();
      for(int i=0; i<size; i++)
       {
        final Map.Entry e = (Map.Entry)iter.next();
        final Integer ID  = (Integer)e.getKey();
        if(!v4.containsKey(ID) || !id2idf.containsKey(ID)) continue;
        final float c1  = ((Float)e.getValue()).floatValue();
        final float c2  = ((Float)v4.get(ID)).floatValue();
        final float idf = ((Float)id2idf.get(ID)).floatValue();
        retval += c1 * c2 * idf * idf;
       }
      return retval;
     }
```

E. p-Korrelation

```
     // Implementierung der p-Korrelation mit p = -1
     private float compare2(HashMap v1, HashMap v2, HashMap id2idf)
     {
      HashMap v3, v4; // v3 soll der kleinere der beiden Wortvektoren werden
```

```
      if(v1.size() > v2.size())
       {
        v3 = v2;
        v4 = v1;
       }
      else
       {
        v3 = v1;
        v4 = v2;
       }

      float retval = 0.0f;
      final int size      = v3.size();
      final Set entry     = v3.entrySet();
      final Iterator iter = entry.iterator();
      for(int i=0; i<size; i++)
       {
        final Map.Entry e = (Map.Entry)iter.next();
        final Integer ID  = (Integer)e.getKey();
        if(!id2idf.containsKey(ID)) continue;

        final float c1  = ((Float)e.getValue()).floatValue();
        final float idf = ((Float)id2idf.get(ID)).floatValue();
        float c2 = -1.0f;
        if(v4.containsKey(ID))
          c2  = ((Float)v4.get(ID)).floatValue();
        retval += c1 * c2 * idf * idf;
       }

      return retval;
     }
```

7.4.1.4.Einsatz von neuronalen Netzen

A. Training eines neuronales Netzes

```
    public void trainNeuralNetwork()
     {
      // Centroid - Vectoren laden ...
      final String query0 = "SELECT id FROM cluster WHERE size>0 ORDER BY id";
      final String query1 = "SELECT wordVector FROM cluster WHERE id=";

      HashMap centroid;
      int[] centroidId = null;
      Object[] centroidVector = null;
      int[] centroidSize = null;

      int cSize = 0;

      try
       {
        dba.beginTransaction();
          taOpen = true;
        final DBResult r = dba.runQuery(query0);
        if(r == null || r.length() == 0) return;
        cSize = r.length();
        centroidId     = new int[cSize];
        centroidVector = new Object[cSize];
        centroidSize   = new int[cSize];
```

```
        for(int i=0; i<cSize; i++)
         {
          final int id   = ((Integer)r.elementAt(i,0)).intValue();
          final BLOB bl = dba.getBLOB(query1 + id);
          if(!bl.isEmpty())
            centroid = (HashMap)bl.toObject();
          else
            centroid = new HashMap();

          centroidId[i]     = id;
          centroidVector[i] = centroid;
          centroidSize[i]   = 0;
         }
       }
      catch(Exception e)
       {
        e.printStackTrace();
        if(taOpen)
         {
          try { dba.abortTransaction(); taOpen = false; }
          catch(Exception e1) { e1.printStackTrace(); }
           }
        return;
       }
      // Centroid - Vectoren geladen

      // WordCache bilden
      if(id2word == null || id2word.size() == 0 ||
         id2idf  == null || id2idf.size()  == 0 ||
         word2id == null || word2id.size() == 0)
        buildWordCache();
      // WordCache gebildet

      // TestSet laden ...
      final String query2 = "SELECT t.id, t.relevance, e.pageRank FROM nntestset t,
evaluation e WHERE t.id=e.id";

      Stopwords sw = ctrl.getStopwords();
      int tSize = 0;
      Object[] wordVector   = null;
      double[] docPageRank  = null;
      double[] docRelevance = null;
      int[] docId           = null;

      try
       {
        dba.beginTransaction();
          taOpen = true;
        final DBResult r = dba.runQuery(query2);
        if(r == null || r.length() == 0) return;
        tSize = r.length();
        wordVector   = new Object[tSize];
        docPageRank  = new double[tSize];
        docRelevance = new double[tSize];
        docId        = new int[tSize];

        for(int i=0; i<tSize; i++)
         {
          final int id      = ((Integer)r.elementAt(i,0)).intValue();
          final double rv   = ((Float)r.elementAt(i,1)).doubleValue();
          final double pr   = ((Double)r.elementAt(i,2)).doubleValue();
```

```
          final HashMap wv  = buildWordVector(getFulltext(id, sw), word2id, id2idf);

          docId[i]        = id;
          wordVector[i]   = wv;
          docPageRank[i]  = pr;
          docRelevance[i] = rv;
         }
        }
       catch(Exception e)
        {
         e.printStackTrace();
         if(taOpen)
          {
           try { dba.abortTransaction(); taOpen = false; }
           catch(Exception e1) { e1.printStackTrace(); }
           }
         return;
        }
       // TestSet geladen

       // Aufbau NN
       System.out.println("\n   +> neural network testset training");

       System.out.print("\n\n    +> creating basic net ...          ");
       final Net nn = new Net();
       final int[] connection = new int[cSize+1];
       final double[] weight  = new double[cSize+1];
       int newNeuron      = -1;
       int outputNeuron   = -1;

       System.out.println(" ok Neural Network v." + nn.getVersion());

       for(int i=0; i<cSize+1; i++)
       {
        newNeuron = -1;
        System.out.print("    +> creating input neurons ...      ");
        try
         {
          newNeuron = nn.newInputNeuron();
         }
        catch(Exception e)
         {
          e.printStackTrace();
         }
        connection[i] = i;
        System.out.println(" ok (N" + newNeuron + ")");
       }

       System.out.print("    +> building random weights ...     ");
       for(int i=0; i<cSize; i++)
         weight[i] = (new Random(System.currentTimeMillis() + i *
100000000)).nextDouble();
       System.out.println(" ok");

       System.out.print("    +> creating an output neuron ...   ");
       try
        {
         outputNeuron = nn.newNeuron(1, weight, connection);
        }
       catch(Exception e)
        {
         e.printStackTrace();
```

```
        }
       System.out.println(" ok (N" + outputNeuron + ")");

       System.out.print("    +> setting activation function ... ");
       try
       {
        nn.setActivationFct(Net.ACTIVATION_STD_DIGITAL, // Typ
                            0.0,                // Minimum
                            1.0,                // Maximum
                            0.0,                // Ruhewert
                            1.0,                // Abnahme
                            1.0                 // Skalierung
                           );
       }

       catch(Exception e)
       {
        e.printStackTrace();
       }
       System.out.println(" ok");

       System.out.print("    +> setting output function ...     ");
       try
        {
         nn.setOutputFct(Net.OUTPUT_FERMI,  // Typ
                         0.0,               // Minimum
                         1.0,               // Maximum
                         0.5,               // Schwellwert
                         0.5                // Steigung
                        );
        }
       catch(Exception e)
        {
         e.printStackTrace();
        }
       System.out.println(" ok");

       System.out.print("    +> set learning rule ...           ");
       try
        {
         nn.setLearningRule(Net.LEARNING_RULE_DELTA , // Typ
                            0.05                      // Lernrate
                           );
        }
       catch(Exception e)
        {
         e.printStackTrace();
        }
       System.out.println(" ok");

       System.out.print("    +> inserting training samples ...  ");
       Sample[] s = new Sample[tSize];
       double best = 0.0, worst = 0.0, len = 0.0;
       for(int i=0; i<tSize; i++)
        {
         final double[] input = new double[cSize+1]; // Eingangswerte: Anzahl Cluster +
1 (pageRank)
         final double[] result = new double[1];      // 1 Ausgangswert (relevance)
         result[0] = docRelevance[i];
         final HashMap wv = (HashMap)wordVector[i];

         for(int j=0; j<cSize; j++) // Abstand zu allen Clustern messen ...
```

```
          {
           final HashMap cluster = (HashMap)centroidVector[j];
           final double c = compare2(cluster, wv, id2idf);
           if(j==0)
            {
             worst = c;
             best  = c;
            }
           else if(c > best)
             best = c;
           else if(c < worst)
             worst = c;
           input[j] = c;
          }
         len = best - worst;
         for(int j=0; j<cSize; j++) // Abstand zu allen Clustern vereinheitlichen ...
           input[j] = (input[j] - worst) / len;

         input[cSize] = docPageRank[i] / 100.0;
         s[i] = new Sample(input, result);
        }

       int c = nn.setSamples(s);
       System.out.println(" ok (" + c + " samples)");

       System.out.print("    +> set sample handling ...         ");
       nn.setSampleHandling(Net.SAMPLES_ALL);
       System.out.println(" ok");

       System.out.print("    +> net is learning samples ...     ");
       int steps = 10000;
       try
       {
        for(int i=0; i<steps; i++)
        {
           if((i/100.0) - (i/100) == 0.0)
             System.out.print("\r    +> net is learning samples ...      " + i);
         // System.out.println("\n Step " + i);
         nn.operate(true);
        }
       }
       catch(Exception e)
       {
        System.out.print("    +> net is learning samples ...      failed.");
        e.printStackTrace();
       }
       System.out.print("\r    +> net is learning samples ...     ");
       System.out.println(" ok (" + steps + " steps)");

       System.out.print("    +> writing net to db ...           ");
       try
        {
         String queryNN1 = "DELETE FROM neuralnetwork WHERE sid=0";
         String queryNN2 = "INSERT INTO neuralnetwork(sid, neuralNetwork) VALUES (0,
?)";
         dba.beginTransaction();
           taOpen = true;
         dba.runUpdate(queryNN1);
         dba.setBLOB(queryNN2, new BLOB(nn));
           dba.commitTransaction();
           taOpen = false;
        }
```

```
       catch(Exception e)
        {
         System.out.println(" failed");
         e.printStackTrace();
         if(taOpen)
          {
           try { dba.abortTransaction(); taOpen = false; }
           catch(Exception e1) { e1.printStackTrace(); }
            }
         return;
        }
       System.out.println(" ok");

       System.out.println("    +> net is reproducing samples ...  ");
       Sample[] s2 = new Sample[1];
       nn.setSampleHandling(Net.SAMPLES_ONE);
       double err = 0.0;
       for(int i=0; i<tSize; i++)
        {
         s2[0] = s[i];
         nn.setSamples(s2);
         try
          {
           nn.operate(false);
          }
         catch(Exception e)
          {
           e.printStackTrace();
          }
         err += nn.computeOverallError(0);

         final double output = util.round(nn.getOutput(0), 2);
         System.out.println("     +> DocId: " + docId[i] + " relevance = " +
                            util.round(s2[0].getTarget(0), 2) + " ... nn.output = " +
output);
        }
       System.out.println("    +> ok");

       System.out.print("    +> compute overall error ...       ");
       err /= (double)tSize;
       System.out.println(" ok (" + err + ")");

       System.out.println("\n");
       System.out.flush();
       // Ende Aufbau NN
      }
```

B. Reproduktion des erlernten Wissens

```
     public void generateRelevance()
      {
       //
       // NN von DB laden ...
       //
       System.out.print("    +> reading net from db ...         ");
       Net nn = null;
       try
        {
         String queryNN = "SELECT neuralNetwork FROM neuralnetwork WHERE sid=0";
```

```
          dba.beginTransaction();
            taOpen = true;
          final BLOB bl = dba.getBLOB(queryNN);
          nn = (Net)bl.toObject();
            dba.commitTransaction();
            taOpen = false;
         }
        catch(Exception e)
         {
          System.out.println(" failed");
          e.printStackTrace();
          if(taOpen)
           {
            try { dba.abortTransaction(); taOpen = false; }
            catch(Exception e1) { e1.printStackTrace(); }
             }
          return;
         }
        System.out.println(" ok");

        //
        // Centroid - Vectoren laden ...
        //
        System.out.print("    +> reading centroids from db ...   ");
        final String query0 = "SELECT id FROM cluster WHERE size>0 ORDER BY id";
        final String query1 = "SELECT wordVector FROM cluster WHERE id=";

        HashMap centroid;
        int[] centroidId = null;
        Object[] centroidVector = null;
        int[] centroidSize = null;

        int cSize = 0;

        try
         {
          dba.beginTransaction();
            taOpen = true;
          final DBResult r = dba.runQuery(query0);
          if(r == null || r.length() == 0) return;
          cSize = r.length();
          centroidId     = new int[cSize];
          centroidVector = new Object[cSize];
          centroidSize   = new int[cSize];

          for(int i=0; i<cSize; i++)
           {
            final int id   = ((Integer)r.elementAt(i,0)).intValue();
            final BLOB bl = dba.getBLOB(query1 + id);
            if(!bl.isEmpty())
              centroid = (HashMap)bl.toObject();
            else
              centroid = new HashMap();

            centroidId[i]     = id;
            centroidVector[i] = centroid;
            centroidSize[i]   = 0;
           }
         }
        catch(Exception e)
         {
          e.printStackTrace();
```

```
          if(taOpen)
           {
            try { dba.abortTransaction(); taOpen = false; }
            catch(Exception e1) { e1.printStackTrace(); }
             }
          return;
         }
        System.out.println(" ok");

        //
        // Wort - Cache bilden ...
        //
        System.out.print("    +> building word cache ...         ");
        if(id2word == null || id2word.size() == 0 ||
           id2idf  == null || id2idf.size()  == 0 ||
           word2id == null || word2id.size() == 0)
          buildWordCache();
        System.out.println(" ok");

        //
        // Iteration über alle Volltexte
        //
        System.out.print("    +> computing relevance ...                    \r");
        final long maxCacheSize = 10000000; // max. 10 Mio. Einträge
        final HashMap cache = new HashMap();
        final Stopwords sw = ctrl.getStopwords();
        int S = 0;
        int maxId = -1;
        final String query2 = "SELECT count(*), max(id) FROM evaluation";
        final String query3_1 = "SELECT id, pageRank FROM evaluation WHERE id >= ";
        final String query3_2 = " AND id < ";
        try
         {
          dba.beginTransaction();
            taOpen = true;
          final DBResult r2 = dba.runQuery(query2);
          if(r2 == null || r2.length() == 0) return;
          S = ((Integer)r2.elementAt(0,0)).intValue();
          maxId = ((Integer)r2.elementAt(0,1)).intValue();
            dba.abortTransaction();
            taOpen = false;
         }
        catch(Exception e)
         {
          e.printStackTrace();
          if(taOpen)
           {
            try { dba.abortTransaction(); taOpen = false; }
            catch(Exception e1) { e1.printStackTrace(); }
             }
          return;
         }
        int nItems = 0;
        int CHUNK = 1000;
        int runs = maxId / CHUNK;
        if(runs * CHUNK < maxId) runs++;

        for(int k=0; k<runs; k++)
         {
          final int min = k * CHUNK;
          final int max = min + CHUNK;
          final String query3 = query3_1 + min +
```

```
                              query3_2 + max;
        try
         {
            dba.beginTransaction();
            taOpen = true;
          Result = dba.runQuery(query3);
            dba.abortTransaction();
            taOpen = false;
         }
        catch(Exception e)
         {
            e.printStackTrace();
          if(taOpen)
           {
            try { dba.abortTransaction(); taOpen = false; }
            catch(Exception e1) { e1.printStackTrace(); }
             }
          return;
         }

        final int len = Result.length();
        for(int i=0; i<len; i++) // Alle Volltexte
         {
          final int id     = ((Integer)Result.elementAt(i,0)).intValue();
          final float pr   = ((Double)Result.elementAt(i,1)).floatValue();
          final HashMap wv = buildWordVector(getFulltext(id, sw), word2id, id2idf);
          nItems++;

          // Action ...
          final double[] input  = new double[cSize+1]; // Eingangswerte: Anzahl
Cluster + 1 (pageRank)
          final double[] result = new double[1];       // 1 Ausgangswert (relevance)
          double best = 0.0, worst = 0.0, vlen = 0.0;
          for(int j=0; j<cSize; j++) // Vergleich mit allen Cluster-Repräsentanten
           {
            final HashMap cluster = (HashMap)centroidVector[j];
            final double c = compare2(cluster, wv, id2idf);
            if(j==0)
             {
              worst = c;
              best  = c;
             }
            else if(c > best)
              best = c;
            else if(c < worst)
              worst = c;
            input[j] = c;
           }
          vlen = best - worst;
          // System.out.print("     +> doc id " + id + "{");
          for(int j=0; j<cSize; j++) // Abstand zu allen Clustern vereinheitlichen ...
           {
            input[j] = (input[j] - worst) / vlen;
            // System.out.print(util.round(input[j], 2) +  ", ");
           }

          input[cSize] = pr / 100.0;
          // System.out.print(util.round(input[cSize], 2) +  "} -> ");
          final Sample[] s = { new Sample(input, result) };
          nn.setSampleHandling(Net.SAMPLES_ONE);
          nn.setSamples(s);
          try
```

```
            {
             nn.operate(false);
            }
           catch(Exception e)
            {
             e.printStackTrace();
            }
           final double relevance = nn.getOutput(0); // Ergebnis!!
           // System.out.println(relevance);
           if(cache.size() >= maxCacheSize)
            {
             final String query4 = "UPDATE evaluation SET relevance=" + relevance + "
WHERE id=" + id;
             try
              {
                 dba.beginTransaction();
                 taOpen = true;
               dba.runUpdate(query4);
                 dba.commitTransaction();
                 taOpen = false;
              }
             catch(Exception e)
              {
                 e.printStackTrace();
               if(taOpen)
                {
                 try { dba.abortTransaction(); taOpen = false; }
                 catch(Exception e1) { e1.printStackTrace(); }
                  }
               return;
              }
            }
           else
            {
             cache.put(new Integer(id), new Float(relevance));
            }
           System.out.print("    +> computing relevance: " + nItems + "/" + S + "
documents                \r");
           System.out.flush();
         } // Ende Chunk-Schleife
       } // Ende Run-Schleife

      System.out.println("    +> computing relevance ...          ok (" + S + "
documents)");
      //
      // Cache zurückschreiben ....
      //
      System.out.print("    +> writing back cache ...          ");
      final int cacheSize = cache.size();
      final Set set = cache.entrySet();
      final Iterator iter = set.iterator();
      try
       {
        dba.beginTransaction();
        taOpen = true;
       }
      catch(Exception ex)
       {
        ex.printStackTrace();
        if(taOpen)
         {
          try { dba.abortTransaction(); taOpen = false; }
```

```
           catch(Exception e1) { e1.printStackTrace(); }
          }
         return;
        }
       for(int i=0; i<cacheSize; i++)
        {
         Map.Entry e = (Map.Entry)iter.next();
         final int id = ((Integer)e.getKey()).intValue();
         final float relevance = ((Float)e.getValue()).floatValue();
         final String query4 = "UPDATE evaluation SET relevance=" + relevance + " WHERE
id=" + id;
         try
          {
           dba.runUpdate(query4);
          }
         catch(Exception ex)
          {
           ex.printStackTrace();
           if(taOpen)
            {
             try { dba.abortTransaction(); taOpen = false; }
             catch(Exception e1) { e1.printStackTrace(); }
            }
           return;
          }
        }
       try
        {
         dba.commitTransaction();
         taOpen = false;
        }
       catch(Exception ex)
        {
         ex.printStackTrace();
         if(taOpen)
          {
           try { dba.abortTransaction(); taOpen = false; }
           catch(Exception e1) { e1.printStackTrace(); }
          }
         return;
        }
       System.out.println(" ok");

      }
```