Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

Analyse von webbasierten eGovernment-Anwendungen hinsichtlich der Optimierung von Suchmechanismen mit Methoden der Automatischen Sprachverarbeitung

Diplomarbeit

Vorgelegt von: Patrick Mairif

Betreut durch: Prof. Dr. Gerhard Heyer

Leipzig, August 2004

Zusammenfassung

Als Vertreter einer webbasierten eGovernment-Anwendung wird die Website der Stadt Leipzig in Hinblick einer möglichen Optimierung der Suchmechanismen analysiert. Dazu wird zu Beginn die Ausgangssituation auf vorhandene Daten und existierende Probleme untersucht.

Es wird eine These bzgl. des unterschiedlichen Sprachgebrauchs auf Redaktions- und auf Nutzerseite formuliert. Der Analyse liegen die Dokumente der Website sowie Suchanfragen der Nutzer zu Grunde. Aus den Dokumenten werden Fachbegriffe mit Hilfe des ConceptComposers extrahiert und diese zusätzlich mit den Suchanfragen verglichen. Andersherum werden auch die Suchanfragen mit den Dokumenten der Website verglichen und spezifische Begriffe des Sprachgebrauchs der Nutzer ermittelt.

Während der Analyse wird eingehend auf das Thema Mehrwortbegriffe eingegangen, sowie verschiedene Methoden der Automatischen Sprachverarbeitung beleuchtet.

Im direkten Zusammenhang mit der Analyse sind Werkzeuge entstanden, die es ermöglichen die Analyse in andere Umgebungen zu portieren.

Es ist ein einfaches Framework für die Integration von Verfahren zur Berechnung von Synonymen entstanden, das auf die gewonnen Daten aufsetzt und es werden Wege für die Generierung eines Wörterbuches „Amt-Bürgersprache“ aufgewiesen.

Vorwort

Diese Arbeit entstand in Kooperation mit dem Amt für Wirtschaftsförderung der Stadt Leipzig und der Abteilung für Anwendungsspezifische Informationssysteme am Institut für Informatik der Universität Leipzig.

An dieser Stelle möchte ich Herrn Körner für die Möglichkeit der Zusammenarbeit und der Bereitstellung der Daten der Website der Stadt Leipzig danken.

Mein Dank gilt ebenfalls Herrn Prof. Fähnrich und seinen Mitarbeitern für die Unterstützung bei diesem Projekt.

Nicht zuletzt danke ich Herrn Prof. Heyer und den Mitarbeitern der Abteilung Automatische Sprachverarbeitung. Ohne sie wäre diese Arbeit nicht möglich gewesen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ASCII	American Standard Code for Information Interchange
ASP	Active Server Pages
ASV	Automatische Sprachverarbeitung
CC	ConceptComposer
CLF	Common Logfile Format
CSS	Cascading Style Sheets
DIN	Deutsche Industrie-Norm
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IIS	Internet Information Server
J2SE	Java 2 Platform, Standard Edition
JDBC	Java Database Connectivity
MB	Mega Byte
PDF	Portable Document Format
PHP	Hypertext Preprocessor
RTF	Rich Text Format
SDK	Software Development Kit
SQL	Structured Query Language
Tsd.	Tausend

URI ..... Universal Resource Identifier

URL ..... Uniform Resource Locators

XML ..... Extensible Markup Language

1. Einleitung

Diese Arbeit beschäftigt sich mit der Optimierung von Suchmechanismen in webbasierten eGovernment-Anwendungen.

Unter eGovernment-Anwendung versteht man alle Formen der elektronischen Verarbeitung von Regierungs- und Verwaltungsaufgaben mit Hilfe des Internets.

Als spezielle Form der eGovernment-Anwendungen wird hier das Bürgerportal betrachtet. Bürgerportale sind Websites von Städten für Bürger, mit dem Schwerpunkt der Pflege der Nähe zu den Bürgern. Damit sind im wesentlichen zwei Aufgaben verbunden: Die Information der Bürger und die Kommunikation mit ihnen.

Die Bürger einer Stadt mit Informationen zu versorgen, umfasst vielfältige Themenbereiche wie z. B. Politik, Verwaltung, Kultur und Wirtschaft. Das stellt eine umfangreiche Aufgabe dar und bringt einen großen Dokumentenbestand mit sich. Trotz Strukturierung und ausgetüftelter Navigationshilfen führt solch eine größere Dokumentenmenge schnell zu Unübersichtlichkeit und macht Suchfunktionen unabdingbar.

Als Grundlage für die Untersuchungen in dieser Arbeit dient die Website der Stadt Leipzig „leipzig.de“. Die existierende Suchfunktionalität der Website

und die damit verbundenen Probleme werden untersucht, mit Text-Mining-Verfahren, sowie Methoden und Technologien der Automatischen Sprachverarbeitung werden die Daten analysiert und Strategien sowie Lösungsansätze für eine bessere Suchfunktionalität entwickelt.

Probleme bei der Suche sind nicht immer technischer Natur und lassen sich nicht immer allein mit technischen Mitteln lösen. Gemeint sind z. B. Diskrepanzen im Informationsangebot und Informationsbedarf oder Unterschiede im Sprachgebrauch. Um diese Problematik zu betrachten, wird das Suchverhalten und der Sprachgebrauch der Besucher analysiert. Als Datenbasis dafür stehen Suchanfragen des Suchformulars der existierenden Website zur Verfügung. Die gewonnenen Informationen werden mit dem Angebot der Website verglichen und zeigen Stellen auf, an denen Redakteure Lücken schließen und Angebote anpassen können.

Die Automatische Sprachverarbeitung hat verschiedene Methoden und Werkzeuge zur Informatonsextraktion und Verarbeitung von großen Textmengen hervorgebracht. Es wird betrachtet, welche sich eignen, die existierenden Probleme zu lösen, Verfahren zu optimieren und Suchergebnisse zu verbessern.

Bürgerportale verfolgen die Ziele

- Information der Bürger und
- Kommunikation mit den Bürgern.

Der interessantere der beiden Aspekte betrifft die Bereitstellung von Informationen. Dies erfolgt in Form von Texten, welche durch Redakteure des Bürgerportals gepflegt werden. Auf Basis dieser Texte wird mit Indizierungs- und Suchmechanismen eine Suchfunktionalität für die Besucher der Website be-

reitgestellt. Diese Texte bilden auch die Grundlage für die Analyse mit Text-Mining-Verfahren im Rahmen dieser Arbeit.

Der Kommunikationsaspekt beschränkt sich meist auf Umfragen, Kontaktformulare, sowie die Bereitstellung von E-Mail-Adressen, Telefonnummern u. ä. Letzteres stellt natürlich ebenfalls eine Informationsfunktion dar, weswegen sich auch Suchfunktionen darauf beziehen und es für diese Arbeit interessant ist. Der eigentliche Kommunikationsaspekt entzieht sich aber einer Suchfunktion und ist nicht in Form von textuellen Inhalten auf dem Webserver vorhanden. Eine entsprechende Arbeit könnte E-Mails und Ergebnisse von Umfragen auswerten. Solche Daten stehen hier nicht zur Verfügung und der Aspekt wird nicht weiter untersucht.

Dieser Arbeit liegt u. a. folgende Annahme zu Grunde: Die Redakteure eines Bürgerportals sind typischerweise Mitarbeiter der Stadt. Sie besitzen aufgrund ihrer Tätigkeit einen gewissen Fachwortschatz, der sich in den Dokumenten des Bürgerportals wiederfindet und über den die Mehrzahl der Besucher, die Bürger, nicht verfügen. Nach den Wörtern dieses Fachwortschatzes wird demnach nicht gesucht, sie sind also nicht in den Suchanfragen vorhanden. Umgekehrt besitzen die Bürger einen umgangssprachlichen Wortschatz („Bürgersprache“), der in den Texten des Bürgerportals nicht berücksichtigt wird. Besucher werden also durch reine Stichwortsuchen nicht immer in der Lage sein, alle relevanten Dokumente zu finden.

Um bessere Suchergebnisse zu erzielen und dem Bürger die Informationen zur Verfügung zu stellen, die ihn interessieren, muss die Diskrepanz zwischen der Fachsprache der Redakteure und der Bürgersprache der Nutzer überwunden und die Suchmechanismen optimiert werden.

Dies zu untersuchen, soll Ziel dieser Arbeit sein.

2. Ausgangssituation und Anforderungen

2.1. Beschreibung der Ausgangssituation

Um eine Analyse der Dokumente der Website durchführen zu können, muss vorher die Art und Organisation der vorhandenen Daten untersucht werden. Aber auch in Hinblick auf eine spätere technische Umsetzung und Integration in die Infrastruktur der Website, soll hier die Ausgangssituation und das technische Umfeld beschrieben werden, bevor existierende Probleme und Lösungsansätze besprochen werden.

Die Website greift auf mehrere verteilte Webserver zurück. Dies sind vor allem der IIS von Microsoft und der Lotus Domino-Server. Zusätzlich kommen externe Server von Content-Anbietern zum Einsatz.

Folgende statische Dateitypen mit textuellen Inhalten finden Verwendung:

- HTML
- PDF
- verschiedene Office-Formate (Word, RTF, ...)

- XML

XML wird vorrangig intern für den Bezug externer Inhalte verwendet. Diese Inhalte werden vor der Verwendung auf der Website konvertiert (HTML) und aufbereitet.

Neben den statischen enthält die Website auch dynamische Inhalte aus verschiedenen Datenbanken. Die Datenbankumgebung ist verhältnismäßig inhomogen. So sind MS Access, dBase, MS SQL und Lotus Notes vertreten.

Die Notes-Datenbankinhalte werden über den Domino-Server wiedergegeben, alle anderen Datenbanken über ASP-Skripte. Neben ASP wird auch JavaScript und ActiveState Perl verwendet.

Die Website verfügt über einen einfachen Suchmechanismus, über den die statischen Dokumente indiziert sind.

2.2. Problemaspekte

In Gesprächen mit Mitarbeitern der Stadt Leipzig, speziell mit Redakteuren und Verantwortlichen für die Webpräsenz, wurden relevante existierende Probleme erfasst.

Die Problemanalyse fand hinsichtlich einer möglichen Problemlösung mit Hilfe von ASV-Technologien statt und war auf die Suchmechanismen konzentriert.

Bei der Betrachtung der bisherigen Suche auf „leipzig.de“ und der Erarbeitung einer Lösungsstrategie lassen sich die folgenden grundlegenden Problemaspekte erkennen. Nicht alle aufgeführten Problemaspekte sind Schwerpunkt dieser Arbeit. Der Vollständigkeit halber seien aber alle aufgeführt.

dezentral: Die Dienste der Website sind auf verschiedene physische wie virtuelle Server verteilt.

technisch: Die Website besteht aus statischen HTML-Dateien und Skripten, die dynamische Seiten erzeugen.

administrativ: Z. B. die Konfiguration des Indexdienstes.

funktional: Mangelnde Funktionalität schränkt die Möglichkeiten der Suche ein.

inhaltlich: Dokumente werden nicht gefunden oder falsche Dokumente werden gefunden.

2.2.1. dezentral

Die Website „leipzig.de“ ist nicht auf einem einzigen zentralen Server angesiedelt. Ihre Dienste sind auf verschiedene physische und virtuelle Server verteilt. Dabei handelt es sich um Rechner der Stadt, welche aus organisatorischen Gründen getrennt sind. Zusätzlich werden Dienste externer Content-Anbieter genutzt, die sich wiederum auf anderen Rechnern befinden.

Wird bei der Indizierung ausschließlich auf Dokumente per HTTP zugegriffen, spielt es keine Rolle, wo sich die Dokumente befinden. Bei dynamischen Inhalten ist eine Indizierung der publizierten HTML-Dateien aber mit Nachteilen verbunden (siehe Abschnitt 2.2.2 auf Seite 16). Bei solchen Inhalten ist ein direkter Zugriff von Vorteil.

Der Direktzugriff auf die Dateien verschiedener virtueller Server ist unproblematisch. Die Dateien liegen auf einem physischen Rechner. Suchwerkzeuge können direkt auf Dateisystem-Ebene auf die Dateien zugreifen.

Verschiedene physische Server müssen in soweit betrachtet werden, als dass es bei solchen nicht ohne weiteres möglich ist, auf Dateisystem-Ebene auf die zugrunde liegenden Daten zuzugreifen. Hier kommen Lösungen mit Netzwerk-Dateisystemen, verteilten Indizierungsdiensten oder Indizierungen per HTTP in Frage. Sollen nicht Dateien, sondern Datenbanken indiziert werden, kann auf diese evtl. auch direkt von entfernten Rechnern zugegriffen werden.

Externe Content-Anbieter werden in der Regel keinen direkten Dateisystem- oder Datenbank-Zugriff erlauben. Solche Inhalte werden normalerweise nur über den einen Kanal bereitgestellt, über den sie bezogen werden, um in die Website integriert zu werden. Bei diesem Kanal wird es sich in den meisten Fällen um einen Webserver handeln. Auf diesen Webserver kann ebenfalls ein Indexdienst zugreifen, um die Dokumente zu indizieren. Wird der Inhalt nicht dynamisch zum Zeitpunkt des Seitenaufbaus eingebunden, sondern in regelmäßigen Abständen auf einem beliebigen Weg bezogen und lokal zwischengespeichert, kann er wieder durch einfachen Dateisystemzugriff indiziert werden.

2.2.2. technisch

Die vorliegenden Inhalte lassen sich hinsichtlich der Art der Auslieferung durch den Webserver in dynamische und statische Inhalte unterteilen.

Statische Inhalte entsprechen den zugrundeliegenden Dateien. Sie werden vom Webserver nur zur Verfügung gestellt, aber nicht verändert oder generiert. Es macht also keinen Unterschied, ob die Dateien im Dateisystem indiziert werden oder die Inhalte, die vom Webserver ausgeliefert werden.

Dynamische Inhalte dagegen werden zum Zeitpunkt der Anfrage an den Webser-

ver generiert. Die Generierung geschieht durch Skripte oder Programme auf dem Server anhand von Datenbank-Inhalten und/oder Parametern, die in der URL codiert sind.

statisch

Die statischen Dateien liegen als HTML, PDF sowie in verschiedenen Office-Formaten vor. Diese Formate können mit geeigneten Werkzeugen recht einfach konvertiert werden.

dynamisch

Verschiedene Skripte generieren Inhalte als HTML und XML. Die generierten Formate selbst sind unproblematisch; die Skripte, die diese generieren, bedürfen jedoch einer näheren Betrachtung.

Die Skripte dienen verschiedenen Zwecken:

Formulargenerierung Skripte, die Formulare bereitstellen, liefern meist statische Inhalte. Sie können einfach über einen HTTP-Zugriff indiziert werden.

Datenbankunabhängige Formularverarbeitung Skripte, die Informationen aus Formularen entgegennehmen, speichern und/oder verarbeiten, liefern entweder keine Ausgabe oder aber Informationen, die sich direkt auf die Nutzereingabe beziehen, wie z. B. Fehlermeldungen. Diese Informationen sind für eine Suche eher uninteressant. Solche Skripte sollten auch nicht durch HTTP-Zugriffe indiziert werden, da dies eine unerwünschte Ausführung des Skripts mit sich bringen würde.

Repräsentation von Datenbank-Inhalten Datenbankinhalte sind für die Suche sehr interessant. Die Wiedergabe durch Skripte basiert meist auf formulargesteuerten Abfragen, was die automatische Indizierung sehr schwierig gestaltet. Eine vollständige Abfrage kann dabei nur schlecht garantiert werden. Wenn möglich, sollten solche Daten direkt aus der Datenbank indiziert werden. Beispiele sind das Medienhandbuch und der Apotheken-Notdienst.

Bei der aktuellen Suche werden teilweise Frames als Suchergebnis ausgegeben, die nicht dazu gedacht sind, einzeln angezeigt zu werden. Dieses Problem muss aber in diesem Kontext nicht weiter betrachtet werden, da die Website im Zusammenhang mit einer geplanten Umstellung keine Frames mehr verwenden wird.

2.2.3. administrativ

Administrative Aspekte spielen durchaus eine relevante Rolle. Beispielsweise wurden in der bisherigen Konfiguration Dokumente gefunden, die nicht vom Webserver zur Verfügung gestellt werden und demzufolge auch nicht in Suchergebnissen auftauchen dürfen. Solche Fehler sind auf falsche Konfigurationen zurückzuführen und wären in einer zukünftigen Lösung zu vermeiden.

2.2.4. funktional

Die vorliegende Suche entbehrt vieler Funktionen, die die Suche erleichtern oder es überhaupt erst ermöglichen würden, die richtigen Dokumente zu finden. Folgende funktionale Mängel und Wünsche wurden von den Mitarbeitern festgestellt:

- Semantische Suche: Die Suche nach Dokumenten auf Basis ihrer Inhalte. Dabei sollen Dokumente gefunden werden, die inhaltlich der Suchanfrage entsprechen, aber nicht zwingenderweise genau die Wörter enthalten, die in der Suchanfrage verwendet wurden.
- Vernünftiges Ranking der Suchergebnisse, evtl. auch mit der Möglichkeit, dieses manuell beeinflussen zu können.
- Boolsche Verknüpfungen: Die Möglichkeit, mehrere Suchbegriffe mit boolschen Operatoren (und, oder, nicht) miteinander zu verknüpfen.
- Schlagwortsuche: Zusätzliche Schlagworte, die nicht im Text vorkommen.
- Meta-Angaben sollten bei der Suche berücksichtigt werden.
- Phonetische Suche: Erkennen verschiedener Schreibweisen (oe/ö, Meier/Meyer, ...).
- Wahl/Einschränkung der Sprache des Suchergebnisses.

Der Wunsch nach semantischer Suchfunktionalität, unterstreicht die Problematik der Sprachdiskrepanz zwischen Redakteuren und Nutzern.

2.2.5. inhaltlich

Es werden falsche Treffer, also Ergebnisse, die nicht zur Suchanfrage passen, und auch unvollständige Ergebnismengen ausgegeben.

Ein typischer Problemfall ist das Suchbeispiel „kita-nordstraße“. Es gibt nur ein Dokument, das diesen Suchbegriff enthält, trotzdem liefert die Suche mehr als 300 Treffer und zumindest bei den ersten Suchergebnissen ist die gewünschte Seite nicht dabei.

2.3. Vorhandene Behelfslösungen

Das Redaktionsteam hat bereits im Vorfeld versucht, einigen Problemen mit verschiedenen Workarounds entgegenzuwirken. Diese im Folgenden erläuterten Maßnahmen zielen zum einen darauf ab, mit manuellen Mitteln die Qualität der Treffermenge zu verbessern und zum anderen, Inhalte überhaupt auffindbar zu machen, die vom Suchwerkzeug technisch nicht erreichbar sind.

2.3.1. Qualität der Treffermenge

Es existiert ein manuell gepflegtes Stichwortverzeichnis, das zur Suche mit herangezogen wird.

Wird im Suchformular ein Begriff verwendet, der in diesem Verzeichnis enthalten ist, so wird zusätzlich zu den normalen Suchergebnissen eine „Wir empfehlen“-Liste ausgegeben. Diese Liste enthält dann jene Links, die von Redakteuren als gute Ergebnisse zu diesem Suchbegriff befunden wurden.

Stellt ein Redakteur fest, dass zu einem bestimmten Suchbegriff kein vernünftiges Suchergebnis geliefert wird, so hat er die Möglichkeit, in dem Verzeichnis einen Eintrag mit dem entsprechenden Suchbegriff, dem zugehörigen Link und einem beschreibenden Text anzulegen.

2.3.2. Erweiterung des Suchraumes

Der verwendete Indexdienst hat technisch nur die Möglichkeit, statische Dokumente zu lesen. Um dynamische Inhalte, die aus Datenbanken gespeist werden, auffindbar zu machen, werden zwei Tricks eingesetzt:

1. Auf jeder Seite des Suchergebnisses werden statische Verweise zu den vorhandenen dynamischen Verzeichnissen, wie z. B. dem Behördenwegweiser oder dem Apothekennotdienst, platziert. Abbildung 2.1 auf Seite 22 zeigt ein Beispiel dafür. Verfolgt man einen der Links, kommt man jeweils zu einem neuen speziellen Suchformular oder einer Index-Seite und kann in der dort eingeschränkten Dokumentenmenge suchen.

2. Aus den Datenbanken werden regelmäßig SQL-Dumps erzeugt. Diese Dumps stellen statische Dokumente dar, die vom Indexdienst gelesen werden können. Sie enthalten jeweils den kompletten Datenbankinhalt als Plaintext. Damit ist es möglich, alle enthaltenen Begriffe mit dem Suchformular zu finden.

   Nun wäre es nicht sinnvoll, wenn sich beim Klick auf das Suchergebnis der Datenbank-Dump präsentieren würde. Entsprechend wird beim Aufruf des Links auf die zur jeweiligen Datenbank gehörende Seite weitergeleitet. Bei diesem Verfahren ist es nicht möglich, einzelne Datensätze zu identifizieren, da der Indexdienst nichts über die Struktur der SQL-Datei weiß, die er indiziert. Die Weiterleitung erfolgt entsprechend auf die Startseite des jeweiligen Bereichs, welche ein weiteres Suchformular enthält.

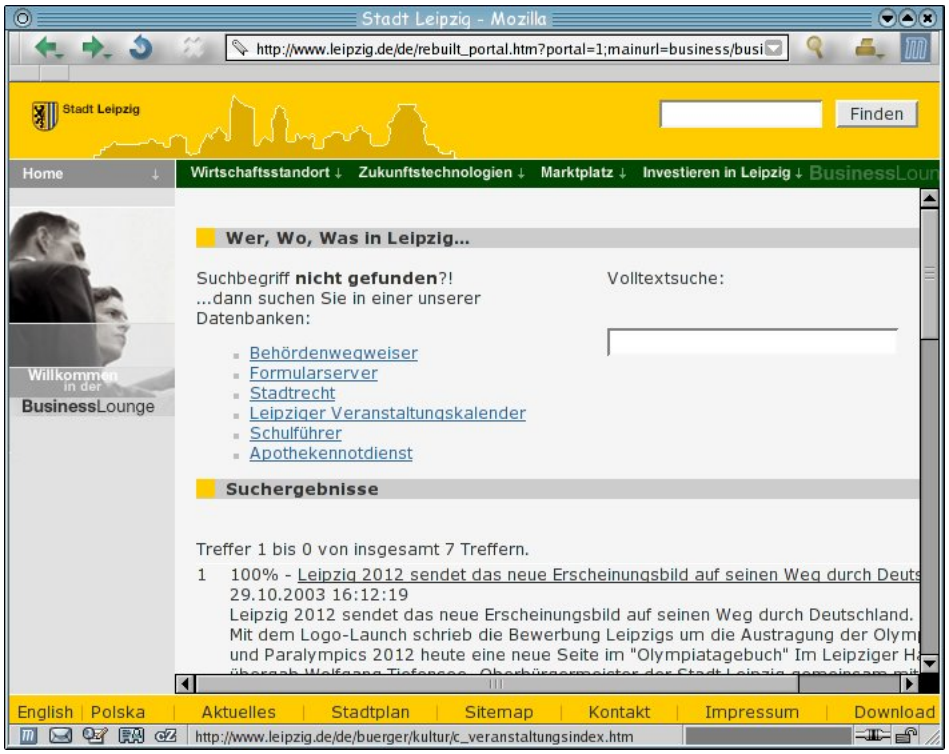

Abbildung 2.1.: *Statische Links im Suchergebnis*

3. Analyse

Anfangs wurde die Annahme formuliert, dass die Besucher eines Bürgerportals einen anderen Sprachgebrauch haben, als die Redakteure und dadurch bedingt musterbasierte Suchverfahren ins Leere laufen können. Um diese Annahme zu untersuchen, wird die Webpräsenz der Stadt Leipzig „leipzig.de“ betrachtet. Sie verfügt über einen genügend großen Dokumentenbestand sowie einen Suchmechanismus, dessen Daten verfügbar sind.

Als Ressource für die „Bürgersprache“ dienen die Suchanfragen und für den Sprachgebrauch im Amt die Dokumente der Website.

Die Wortmenge, die sich aus den Suchbegriffen ergibt, sei mit S bezeichnet, die Wortmenge der Wörter der Web-Dokumente mit W.

Unter den Suchbegriffen S sind die Wörter für die „Bürgersprache“ spezifisch, die nicht in W enthalten sind: $S_1 = S \setminus W$

Eine Teilmenge von W sind die Fachbegriffe $F \subset W$.

Für den Sprachgebrauch im Amt spezifisch sind nun die Wörter aus F, die nicht gesucht werden: $F_1 = F \setminus S$

Um zu untersuchen, ob die Annahme zutrifft und um dann die „Bürgersprache“ auf den Sprachgebrauch im Amt abzubilden, sind zwei Wortmengen interessant:

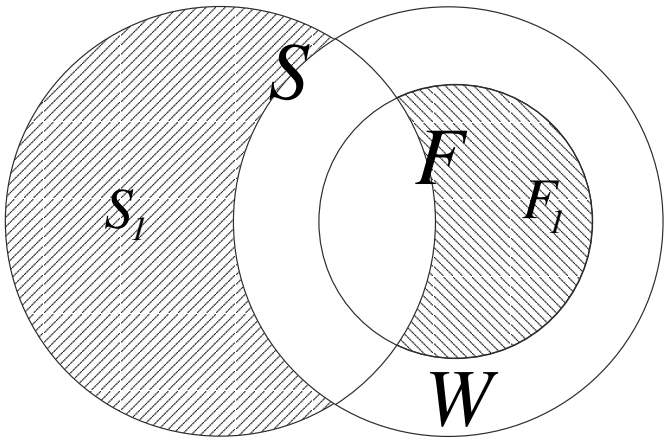

Abbildung 3.1.: *Wortmengen*

- Die Suchbegriffe, die nicht gefunden werden: S_1.
- Die Fachbegriffe, die nicht gesucht werden: F_1.

3.1. Datenbasis

3.1.1. Suchanfragen

Um statistische Aussagen zum Verhalten der Besucher der Website treffen zu können, werden die Suchanfragen von einem Zeitraum von sechs Monaten betrachtet.

Alle Zugriffe auf eine Website werden vom Webserver in einer Log-Datei protokolliert. Die Suchfunktionalität auf der untersuchten Web-Präsenz ist über HTTP-GET implementiert, entsprechend existiert für jede Suchanfrage ein Eintrag in den Log-Dateien des Web-Servers in Form der aufgerufenen URI.

```
"GET /de/search_results.asp?Finden=Motorrad+Messe HTTP/1.0"
```

Abbildung 3.2.: *Beispiel-Request mit Suche nach „Motorrad Messe“*

Es sind verschiedene Formate für Webserver-Log-Dateien verbreitet. Im untersuchten Beispiel liegen die Log-Dateien im Common Logfile Format (CLF) vor.

Log-Dateien im Common Logfile Format sind ASCII-Dateien. Jede Zeile stellt einen Datensatz mit folgendem Format dar [11]:

```
remotehost rfc931 authuser [date] "request" status bytes
```

Für diese Untersuchung relevant ist nur das Feld `request`. Es enthält den HTTP-Request, so wie er vom Client abgeschickt wurde. Abbildung 3.2 auf dieser Seite zeigt einen Beispiel-Request.

Ein solcher HTTP-GET-Request enthält u. a. die URI der ASP-Datei, die für die Suchanfrage verantwortlich ist, und den Such-String, der in der Suchanfrage verwendet wurde, als URL-encodierten [4] Parameter.

Filtert man diese Strings aus den Log-Dateien und decodiert man die URL-Parameter, erhält man alle Suchanfragen aus dem Zeitraum der Log-Dateien. In diesem Fall wurden insgesamt 361 Tsd. Suchanfragen extrahiert.

Definition 3.1 (Suchanfrage) *Als Suchanfrage soll jeder einzelne Eintrag in der Log-Datei des Webservers verstanden werden, der einen Aufruf der Such-Funktionalität dokumentiert. Die Suchanfrage umfasst eine Zeitangabe, zu der die Anfrage gestellt wurde, und den Such-String, so wie er vom Nutzer eingegeben wurde.*

Definition 3.2 (Such-String) *Mit Such-String sei die Eingabe des Nutzers*

bei einer Suchanfrage bezeichnet. Gleiche Such-Strings werden zusammengefasst. Zählt man jeweils die zugehörigen Suchanfragen, erhält man die Häufigkeit eines Such-Strings.

3.1.2. Dokumente

Zum Zeitpunkt der Untersuchung umfasste der Dokumentenbestand der Website heruntergeladen und unkomprimiert 800 MB. Nach dem Konvertieren der Dokumente in Plaintext umfassten sie unkomprimiert rund 65 MB. Die 50 Tsd. Dokumente enthielten 99 Tsd. Sätze (keine doppelt) und 101 Tsd. verschiedene Wörter.

3.2. Vorgehen

Eine schematische Übersicht der Vorgehensweise bei der Analyse der Daten zeigt Abbildung 3.3 auf Seite 27.

Wenn keine Textdokumente der Website vorliegen, können diese mit geeigneten Werkzeugen heruntergeladen werden.

Zur weiteren Analyse sind im wesentlichen die textuellen Inhalte interessant. Im nächsten Schritt werden entsprechend die vorhandenen Dokumente aus den verschiedenen vorliegenden Formaten nach Plaintext [1] konvertiert.

Parallel dazu werden aus den vorhanden Log-Dateien des Webservers die enthaltenen Suchanfragen extrahiert.

[1] Reine ASCII-Dateien ohne Formatanweisungen.

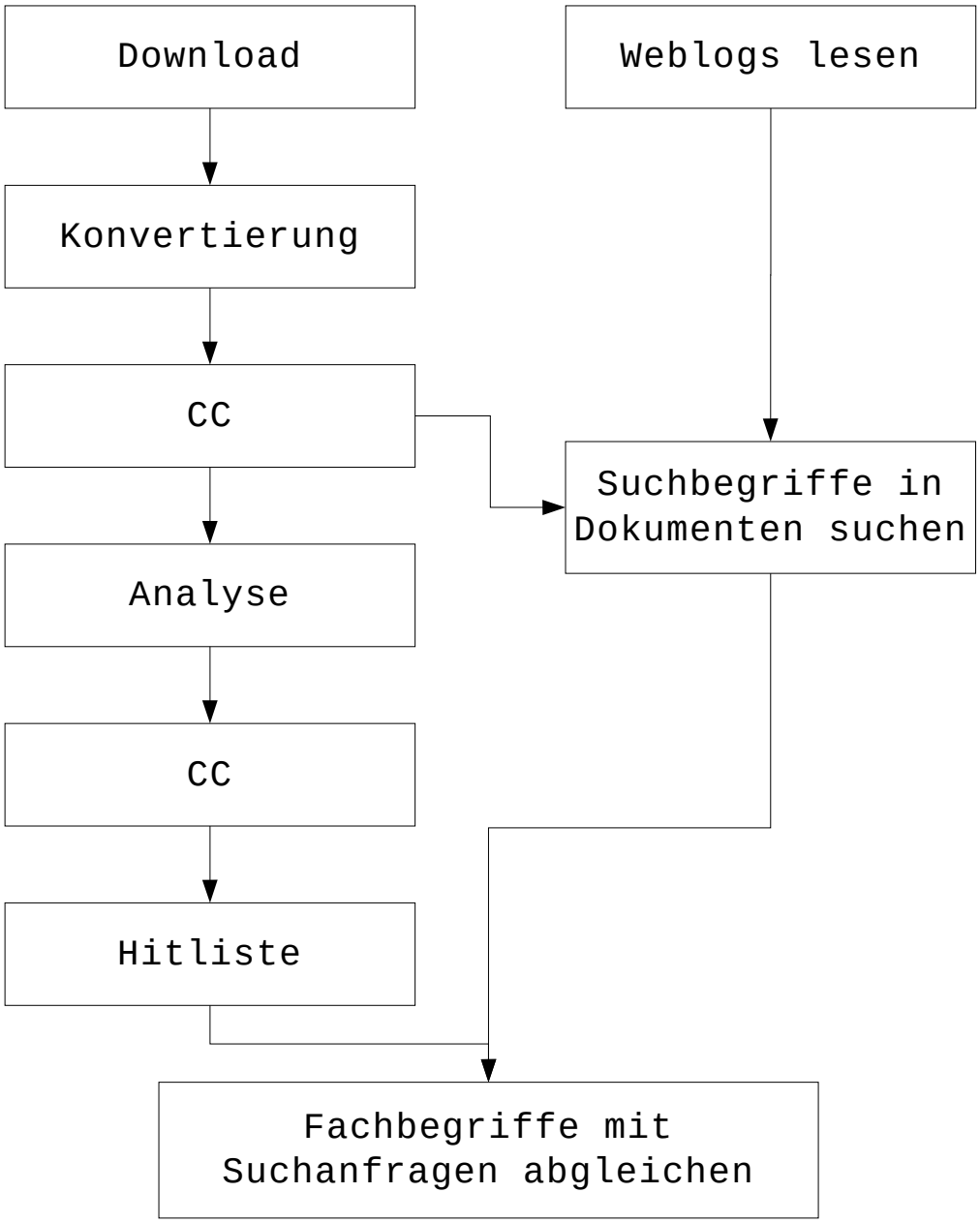

Abbildung 3.3.: *Analyse-Prozess*

Auf dem konvertierten Fließtext der Dokumente der Website wird mit dem ConceptComposer (CC) ein Text-Mining durchgeführt, d. h. es wird u. a. eine Datenbank mit einem Volltextindex erzeugt. Auf Basis dieser Datenbank werden die nichtgefundenen Suchbegriffe errechnet und die Mehrwortbegriff-Analyse wird durchgeführt.

Mit den errechneten Mehrwortbegriffen wird ein erneutes Text-Mining angestoßen, um die Hitliste und damit die Fachbegriffe des Korpus zu errechnen.

Im letzten Schritt werden die Fachbegriffe mit den nichtgefundenen Suchbegriffen abgeglichen.

3.3. Werkzeuge

Zum Zwecke der Analyse in dieser Arbeit wurden mehrere Werkzeuge entwickelt. Die Aufgabenbereiche dieser Werkzeuge gliedern sich wie folgt:

- Die Konvertierungen und Berechnungen werden weitestgehend von einem Java-Package übernommen.
- Oberflächen für die Interaktion mit Nutzern sind webbasiert auf PHP-Basis implementiert.
- Datenbank-Abfragen sind teilweise webbasiert und teilweise im Java-Package integriert.

Die Werkzeuge bieten folgende Funktionalitäten:

- Suchanfragen aus Log-Dateien extrahieren

- Suchanfragen verarbeiten (zusammenfassen, zählen, mit Website vergleichen, Mehrwortbegriffe markieren)
- Plaintext-Extraktion aus heruntergeladenen Dokumenten der Website
- Hitliste konvertieren und mit Suchanfragen vergleichen
- Mehrwortbegriffe finden (aufeinander folgende Nomen)
- Oberfläche für die Bearbeitung der Mehrwortbegriffe
- Oberfläche zum Markieren ungültiger Fachbegriffe
- Oberfläche zur Synonym-Bearbeitung

Zusätzlich existieren verschiedene Datenbank-Abfragen sowie ein Framework für die Berechnung von Synonymen, in das beliebige Verfahren integriert werden können.

Die Werkzeuge finden sich komplett auf der beiliegenden CD. Eine Installations- und Bedienungsanleitung befindet sich im Anhang A ab Seite 75.

3.3.1. Datenbank

Für die Daten, die während dieser Analyse anfallen, wird eine separate Datenbank angelegt, auf der alle Werkzeuge aufsetzen.

Diese Datenbank beinhaltet

- die extrahierten Fachbegriffe der Website (Tabelle `fachbezeichner`),
- die gefundenen aufeinander folgenden Nomen (Tabelle `mehrwortbegriffe`),
- die Suchanfragen aus den Webserver-Log-Dateien (Tabelle `such_log`),

- die (aus den Suchanfragen zusammengefassten) Such-Strings (Tabelle `such_strings`) und
- die berechneten Synonyme (Tabelle `synonyme`).

3.4. Analyse der Website

3.4.1. Konvertieren und Filtern der Texte

Der ConceptComposer benötigt Plaintext als Eingabe. Einige Formate wie z. B. HTML kann er selbst konvertieren. Andere Formate müssen nach dem Herunterladen entsprechend konvertiert werden.

Die Website im vorliegenden Beispiel enthält im Wesentlichen Dateien in den folgenden Formaten:

- Text: HTML, PDF, Plaintext, Microsoft Word, RTF
- CSS, JavaScript, ...
- verschiedene Grafikformate

Die Grafiken sind für die Textanalyse uninteressant und werden weitestgehend bereits beim Herunterladen gefiltert. Inhalte mit eher technischer Orientierung wie CSS und JavaScript enthalten in der Regel keinen natürlichsprachlichen Text. Sie enthalten vielmehr alphanumerische Strings mit syntaktischer Bedeutung für den jeweiligen Interpreter, wie z. B. Bezeichner, Steueranweisungen, Variablen. Bei einer späteren Indizierung können diese fälschlicherweise als natürlichsprachliche Wörter erkannt werden. Diese Dateien werden also ebenfalls ignoriert. CSS und JavaScript können aber auch direkt in HTML-Dateien

eingebettet sein. Dies bedarf bei der Konvertierung nach Plaintext besonderer Aufmerksamkeit.

Interessant sind die Dateien mit textuellen Inhalten. Tabelle 3.1 auf Seite 31 zeigt eine Übersicht über geeignete Werkzeuge zur Konvertierung der jeweiligen Formate.

Format	Werkzeug	Eingesetzte Version
HTML	lynx [34]	Lynx 2.8.5dev.7 (06 Jan 2002), libwww-FM 2.14
PDF	pdftotext/xpdf [15]	Version 2.01
Microsoft Word, RTF	wvWare [30]	0.7.3

Tabelle 3.1.: *Werkzeuge zur Plaintext-Konvertierung*

Zur einfacheren Handhabung werden die verschiedenen Werkzeuge in einem Shellscript (`convweb_file.sh`) gekapselt. Dieses Shellscript konvertiert einzelne Dateien.

Um die Konvertierung insgesamt für alle Dateien zu automatisieren existiert ein weiteres Shellscript: `convweb_all.sh`. Es liest alle Dateien eines Verzeichnisses rekursiv ein, übergibt sie an `convweb_file.sh` und sammelt die konvertierten Inhalte in einer Ausgabe-Datei.

Die Werkzeuge, die zur Konvertierung genutzt werden, verfolgen die Zielstellung möglichst viele Informationen zu transportieren. In diesem Fall liegt der Schwerpunkt jedoch auf möglichst „schönen“ Texten. Also auf Texten, die sich gut für das folgende Text-Mining eignen. Aus diesem Grund wird die Ausgabe zusätzlich gefiltert.

Die Filterung umfasst folgende Punkte:

- Doppelpunkte (:) sollen als Satztrenner interpretiert werden. Dazu werden sie jeweils durch eine Leerzeile ersetzt.
- Backslashes (\) werden entfernt.
- Texte zwischen eckigen Klammern ([]) werden komplett gelöscht.

Doppelpunkte erwiesen sich als geeignete Trenner für lange Sätze. Backslashes wurden beispielsweise in der „Breadcrumb“-Navigation als Trenner der einzelnen Punkt verwendet. Diese Zeichen sind im Fließtext nicht interessant und werden entsprechend gelöscht.

Lynx fügt für Bilder, die keinen Alternativ-Text besitzen, den Dateinamen in eckigen Klammern in den Text ein. Diese Dateinamen stellen meist keine Wörter im lexikalischen Sinne dar und würden das Rauschen bei der späteren Berechnung der Fachbegriffe erhöhen. Zusätzlich kann der Punkt, der oft in Dateinamen enthalten ist, vom Parser als Satzendezeichen missverstanden werden.

Bei der Liste der unterstützten Formate gibt es Einschränkungen. So bietet das PDF-Format die Möglichkeit die Verwendung beim Nutzer einzuschränken. Beispielsweise kann das Kopieren von Text aus dem Dokument unterbunden werden. In einem solchen Fall ist technisch auch keine Plaintext-Extraktion mehr möglich - andernfalls wäre dieser Schutz auch nicht wirkungsvoll durchzusetzen.

3.4.2. ConceptComposer

> „Text repräsentiert Wissen. Im Unterschied zu den strukturierten Daten in einer Datenbank stellen Texte unstrukturierte Daten

> dar. Mit Hilfe von Text-Mining Werkzeugen können aus digital vorliegenden Texten neue und relevante sachliche und inhaltliche Zusammenhänge extrahiert werden. Text Mining basiert auf statistischen und musterbasierten Verfahren und erlaubt innovative Anwendungen im Wissensmanagement.“ [20]

Als zentrales Analyse- und Text-Mining-Werkzeug dient der ConceptComposer.

Der ConceptComposer zerlegt Texte in Sätze, indiziert erkannte Begriffe und legt diese Informationen in einer relationalen Datenbank ab [33]. Zusätzlich werden u. a. Frequenzen von Wörtern, Nachbarschaftskollokationen sowie Satzkollokationen errechnet.

Auf dieser Datenbank können andere Werkzeuge aufsetzen, um Informationen zu extrahieren, darzustellen oder weiterzuverarbeiten.

Der ConceptComposer ist ein komplexes Werkzeug und Inhalt dieser Arbeit ist es nicht, dieses Werkzeug komplett zu beschreiben. Dennoch soll auf einige Details, die für das Verständnis der weiteren Arbeit wichtig sein können, eingegangen werden.

Im Kontext dieser Arbeit wurde die Version 6.2.2, Build 170 des ConceptComposers verwendet.

Daten

Die Daten werden komplett in einer MySQL-Datenbank verwaltet. Im folgenden soll ein Überblick gegeben werden, in welcher Form die für dieses Projekt relevanten Daten vorliegen.

Dokumente Um die einzelnen Sätze ihren Dokumenten zuordnen zu können, werden Verweise auf die Quellen in der Tabelle `quelle` abgelegt.

Sätze Die Texte der Dokumente werden in einzelne Sätze zerlegt. Diese werden in der Tabelle `saetze` gespeichert. Diese Tabelle enthält einen Verweis auf das Quelldokument.

Wörter Alle erkannten Wörter mit Anzahl und Gewichtung für die Hitliste (siehe Abschnitte 3.7 auf Seite 62) finden sich in der `wortliste` wieder. Eine Verknüpfung zwischen den einzelnen Wörtern und den Sätzen stellt die Relation in Tabelle `inv_liste` dar.

Kollokationen Satz- sowie Nachbarschaftskollokationen werden als Relationen zwischen Wörtern in den Tabellen `kollok_sig` und `kollok_nb` gespeichert.

Feinheiten der Konfiguration

Die Konfiguration des ConceptComposer erfolgt mittels Java-Property-Dateien. Einige Einstellungen haben sich bei diesem Projekt als wichtig erwiesen.

Anzahl Sätze Mit der Default-Einstellung werden für jedes Wort maximal 256 Sätze als Beispiele gespeichert. Möchte man alle Sätze und damit einen Volltextindex erzeugen, erreicht man das über die Option `maxBsp=-1`.

Doppelte Sätze Websites enthalten oft wiederkehrende Muster, wie Menüs oder Urheberrechtsangaben. Dies führt zu entsprechend hohen Häufigkeiten der enthaltenen Wörter. Die Option `checkDuplicates=-S` verhindert, dass Sätze doppelt aufgenommen werden.

Lange Sätze Oft liegen Texte auf Websites nicht in Form ganzer Sätze vor. Die Option `dotRequired=-e -l 2048` veranlasst den ConceptComposer, auch Sätze zu akzeptieren, die nicht auf einen Punkt enden und bis zu 2048 Zeichen beinhalten, was das momentane Maximum darstellt.

Mehrwortbegriffe Hat man eine Datei mit bekannten Mehrwortbegriffen (vgl. Abschnitt 3.6 auf Seite 46), kann man diese dem ConceptComposer über die Option `wlFile=/path/to/file` bekanntgeben.

3.5. Analyse der Suchanfragen

3.5.1. Zipfsche Gesetze

Der vorliegende Datenbestand umfasst 361 Tsd. Suchanfragen, wovon rund 80 Tsd. verschieden sind (Such-Strings). Für eine manuelle Analyse wäre das zu viel. Die Art und Menge läßt die Daten aber zu guten Kandidaten für die Anwendung der Zipfschen Gesetze [48] werden.

Sortiert man die Such-Strings absteigend nach ihrer Häufigkeit, so sagen die Zipfschen Gesetze eine hyperbolische Verteilung voraus. D. h. es gibt nur wenige Such-Strings, die häufig gesucht werden und damit für eine weitere Analyse interessant sind.

In Abbildung 3.4 auf Seite 36 ist die Häufigkeitsverteilung aller Such-Strings grafisch dargestellt. Dabei ist sehr gut zu erkennen, dass nur wenige Such-Strings (ganz links im Bild) häufig vorkommen.

Betrachtet man nur die ersten 500 Such-Strings (Abbildung 3.5 auf Seite 37), so erkennt man auch den hyperbolischen Charakter der Häufigkeitsverteilung,

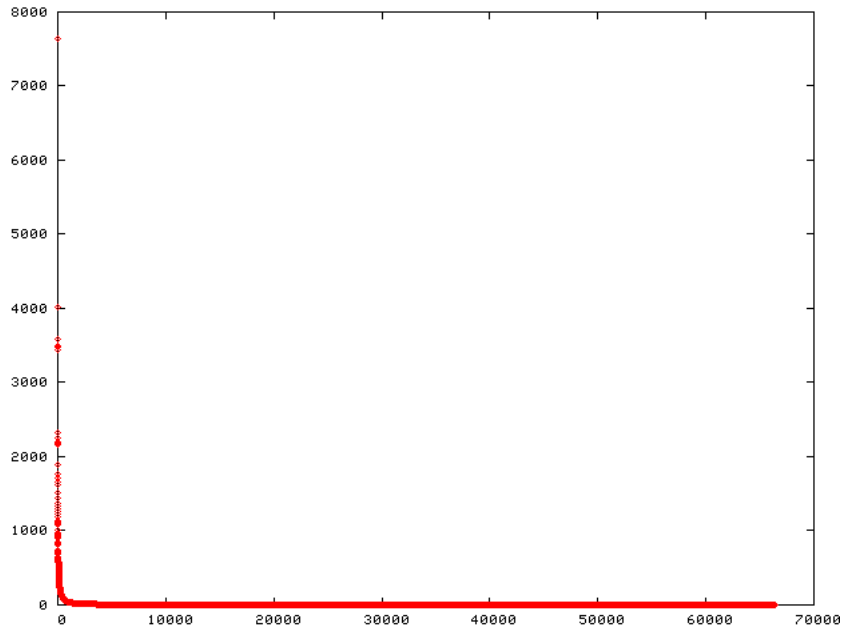

Abbildung 3.4.: *Häufigkeitsverteilung der Such-Strings*

was ein Hinweis darauf ist, dass das Gesetz hier zutrifft.

In Tabelle 3.2 auf Seite 38 findet sich eine Übersicht über die 20 häufigsten Such-Strings.

Der Rang r eines Such-Strings ergibt sich aus dessen Position n in der sortierten Liste, wobei man Such-Strings mit gleicher Häufigkeit f zusammenfasst und den gleichen Rang zuweist. Wenn also f_n die Häufigkeit und r_n der Rang des Such-Strings an Position n sind, dann gilt:

$$r_1 = 1$$

$$\forall n > 1 : r_n = \begin{cases} n & \text{falls } f_n < f_{n-1}, \\ r_{n-1} & \text{falls } f_n = f_{n-1} \end{cases}$$

Die Abbildungen 3.6 (S. 37) und 3.7 (S. 39) zeigen jeweils eine Abbildung des Ranges (x-Achse) auf das Produkt aus Rang und Häufigkeit (y-Achse).

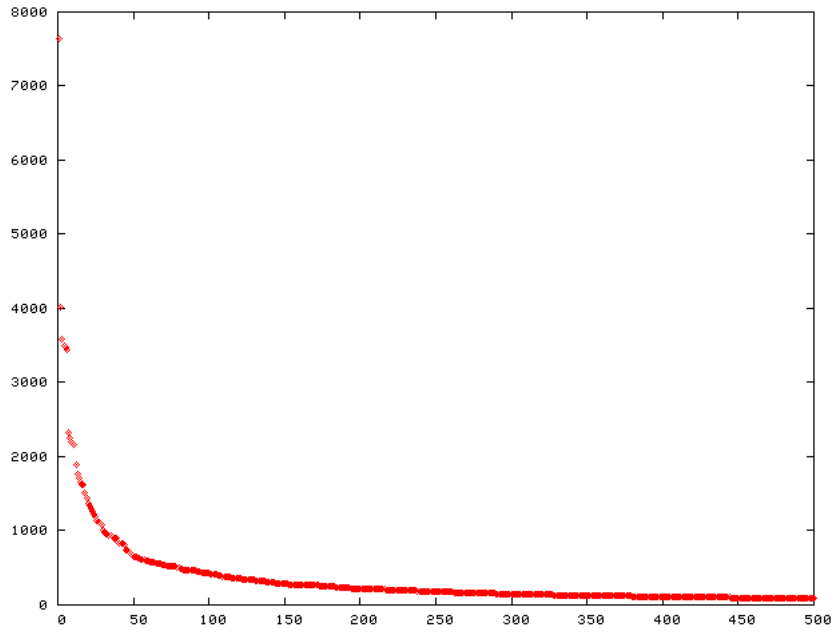

Abbildung 3.5.: *Häufigkeitsverteilung der 500 häufigsten Such-Strings*

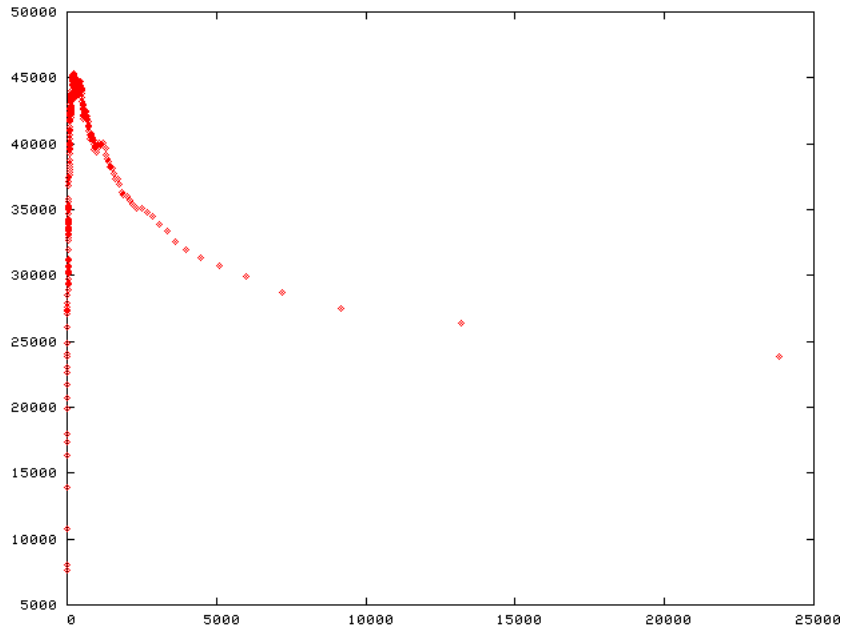

Abbildung 3.6.: *Such-Strings: $r \cdot f$*

Such-Strings	Häufigkeit f	Rang r	$r \cdot f$
stadtplan	7638	1	7638
kino	4012	2	8024
flughafen	3593	3	10779
hotels	3492	4	13968
zoo	3473	5	17365
hotel	3446	6	20676
schulen	2333	7	16331
messe	2244	8	17952
gewandhaus	2206	9	19854
arena	2173	10	21730
stellenangebote	2171	11	23881
sport	1884	12	22608
oper	1773	13	23049
universität	1718	14	24052
stadtbibliothek	1656	15	24840
bibliothek	1629	16	26064
hauptbahnhof	1623	17	27591
weihnachtsmarkt	1507	18	27126
bahnhof	1441	19	27379
wohnungen	1367	20	27340

Tabelle 3.2.: *Die 20 häufigsten Such-Strings*

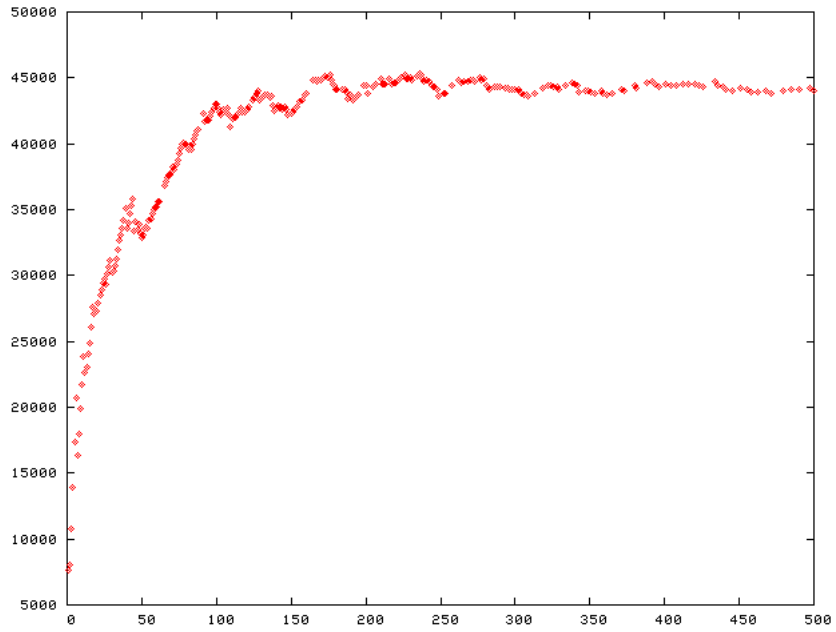

Abbildung 3.7.: *Such-Strings: $r \cdot f$, die ersten 500*

Die Zipfschen Gesetze sagen für das Produkt aus Rang und Häufigkeit einen konstanten Wert voraus. Diese Vorhersage wird nicht hundertprozentig erfüllt, die Tendenz ist aber zu erkennen.

Man kann festhalten, dass die Anzahl der häufigen Such-Strings gering ist. Tabelle 3.3 auf Seite 40 zeigt die Anzahl a der Such-Strings bei der jeweiligen Mindesthäufigkeit h. D. h. die Such-Strings, die jeweils mindestens h mal vorkommen, werden gezählt und ergeben die Anzahl a. Am Beispiel heißt das, dass es nur 79 verschiedene Such-Strings gibt, die 500 mal oder öfter gesucht wurden.

Mindesthäufigkeit h	Anzahl Suchanfragen a
20	1814
50	812
100	440
500	79

Tabelle 3.3.: *Anzahl der Suchanfragen in Abhängigkeit zur Mindesthäufigkeit*

3.5.2. Nichtgefundene Suchbegriffe

Um die Begriffe der „Bürgersprache“ (siehe Abschnitt 3 auf Seite 23) zu ermitteln, wird die Menge der Begriffe, nach denen die Bürger suchen, auf jene eingeschränkt, die bei einer Suche auf der Website keine Treffer liefern.

Das führt zu zwei Fragestellungen:

- Wann kann man davon ausgehen, dass ein Such-String einen Begriff darstellt?
- Und wie lässt sich feststellen, wann eine Suchanfrage nicht erfolgreich war, also keine Treffer geliefert hat?

Ein Such-String kann prinzipiell aus einem oder mehreren Wörtern bestehen. Besteht er nur aus einem Wort, kann man annehmen, dass es sich dabei um einen Begriff handelt. Wenn er aus mehreren Wörtern besteht, muss man unterscheiden, ob es sich um einen Mehrwortbegriff (z. B. „Alte Messe“) oder um eine reine Aneinanderreihung einzelner Suchbegriffe (z. B. „Leipzig Messe“) handelt, die vom Nutzer in dieser Form intuitiv eingegeben wurde, um das Suchergebnis mittels einer boolschen Verknüpfung einzuschränken.

Bei der Aneinanderreihung einzelner Wörter gibt es keine übliche Reihenfolge. Es können beliebige Wörter in beliebiger Weise kombiniert werden. Die Zusam-

mensetzung des Such-Strings wird also bei jedem Nutzer anders aussehen. Im Gegensatz dazu sind bei Mehrwortbegriffen Position und Einzelwörter nicht variabel. Entsprechend wird die Anzahl der Suchanfragen bei Mehrwortbegriffen höher sein als bei individuell verknüpften Begriffen.

Gängige Begriffe werden mit entsprechender Häufigkeit auch alleine in Suchanfragen verwendet werden. Die Anfragen mit verknüpften Begriffen bedürfen also keiner eingehenden Betrachtung.

Zusammenfassend kann man die Begriffe, die gesucht werden, wie folgt als Suchbegriffe definieren:

Definition 3.3 (Suchbegriff) *Als Suchbegriffe sollen jene Such-Strings verstanden werden, die keine Aneinanderreihung mehrerer Begriffe enthalten, sondern jeweils als ganzes einen Begriff darstellen. Das sind zum einen die Such-Strings, die nur aus einem Wort bestehen, und zum anderen die Such-Strings, die Mehrwortbegriffe enthalten.*

Die hier gefundenen Kriterien zur Identifizierung von Mehrwortbegriffen in Suchanfragen finden auch unter 3.6.1 auf Seite 47 Beachtung.

Die Frage, wann eine Suchanfrage erfolgreich war, wann sie also Treffer geliefert hat, soll hier auf die Fragestellung reduziert werden, ob der Suchbegriff in den Dokumenten der Website enthalten ist.

Die vom ConceptComposer erzeugte Datenbank enthält den vollständig indizierten Text der Website in Form einer Wortliste mit zugehörigen Beispielsätzen. Um die Frage zu beantworten, ob ein Begriff auf der Website enthalten ist, reicht es aus, den Datenbankinhalt zu betrachten.

Für die Suche in der Datenbank gibt es folgende Möglichkeiten:

1. Suche in der Wortliste
2. Suche in den Beispielsätzen (Volltextsuche)

Suche in der Wortliste

Für die Suche von Begriffen in Sätzen sei eine Teil-String-Relation $k \otimes l$ für Begriffe k und Sätze l definiert, die genau dann wahr ist, wenn der Begriff k an beliebiger Stelle in dem Satz l vollständig enthalten ist. Die Begriffe und Sätze werden dabei als Zeichenketten mit vorgeschriebener Reihenfolge gesehen. Technisch wird diese Relation in der MySQL-Datenbank durch den SQL-Befehl `like` [35] umgesetzt.

Die Menge der Beispielsätze sei mit B, die Menge der Einträge der Wortliste mit W und die Menge der Such-Strings mit S bezeichnet.

Die Wortliste enthält alle Einzelwörter und die zum Zeitpunkt der Indizierung bekannten Mehrwortbegriffe. Für die Einträge der Wortliste gilt:

$$W = \{w / \forall w \exists b \in B(w \otimes b)\}$$

Eine Suche in der Wortliste ergibt die Treffermenge S_W mit allen Such-Strings, die in der Wortliste enthalten sind:

$$S_W = \{s_w / \forall s_w \in S \exists w \in W(s_w = w)\}$$

Dabei werden jeweils die kompletten Strings der einzelnen Begriffe aus W mit denen aus S verglichen.

Die Menge der nichtgefundenen Such-Strings ist entsprechend:

$$N_W = S \setminus S_W$$

Such-String	Häufigkeit
Arena Leipzig	417
webcam	414
Haus Auensee	323
kinoprogramm	301
stellenausschreibung	283
Stellenausschreibung	280
arena leipzig	276
Kinoprogramm	273
Botanischer Garten	253
eisdom	249

Tabelle 3.4.: *Suchbegriffe, die nicht gefunden werden (Wortliste)*

Tabelle 3.4 auf Seite 43 zeigt die häufigsten Suchbegriffe, die bei einer Suche in der Wortliste nicht gefunden werden.

Suche in den Beispielsätzen

Der jeweilige Such-String gilt als gefunden, wenn es einen Beispielsatz gibt, in dem er enthalten ist:

$$S_B = \{s_b / \forall s_b \in S \exists b \in B(s_b \otimes b)\}$$

Die Menge der nichtgefundenen Such-Strings ergibt sich aus:

$$N_B = S \setminus S_B$$

Die Suche in den vorhandenen Beispielsätzen entspricht einer Volltextsuche. Enthält ein Such-String mehr als ein Wort, kann dieser nur gefunden werden,

Such-String	Häufigkeit
webcam	414
eisdom	249
flohmarkt	208
Eisdom	201
zulassungsstelle	183
Flohmarkt	162
Zulassungsstelle	135
Paunsdorfcenter	121
paunsdorfcenter	117
werk 2	116

Tabelle 3.5.: *Suchbegriffe, die nicht gefunden werden (Beispielsätze)*

wenn die Wörter in gleicher Reihenfolge direkt hintereinander im Text auftreten. Mehrwortbegriffe werden dabei alle erfasst, Kombinationen einzelner Suchbegriffe nicht.

Tabelle 3.5 auf Seite 44 enthält die häufigsten Suchbegriffe, die bei einer Suche in den Beispielsätzen nicht gefunden werden. Es fällt auf, dass Mehrwortbegriffe wie „Arena Leipzig“, „Haus Auensee“ und „Botanischer Garten“ nicht mehr in der Liste enthalten sind, jetzt also gefunden werden. Wörter wie „Kinoprogramm“ werden jetzt ebenfalls gefunden. Das Wort wird bei der Suche in der Wortliste nicht gefunden, da es in dieser Form nicht in der Wortliste existiert. Es wird aufgrund des Teil-String-Vergleichs in „www.kinoprogramm-leipzig.de“ gefunden. Wörter wie „Stellenausschreibung“ kommen im Text nur im Plural oder als „Stellenausschreibung-Nr“ vor, weswegen sie auch erst bei der Volltextsuche gefunden werden.

Vergleich

Um ein eindeutiges Entscheidungsmerkmal zu bekommen, wann ein Suchbegriff gefunden wird, muss man sich für eine Suchmethode entscheiden. Die Suche in der Wortliste ist verglichen mit der Suche in den Sätzen sehr schnell. Es werden aber nur die vom ConceptComposer erkannten Wörter gefunden. Die Suche in den Sätzen entspricht einer Volltextsuche und ist damit für die Fragestellung, welche Suchbegriffe auf der Website gefunden werden, das realistischere Kriterium.

Gibt es aber vielleicht Such-Strings, die bei einer Suche in der Wortliste gefunden werden und nicht bei einer Suche in den Sätzen? Intuitiv ist die Frage leicht beantwortet. Formal ergibt sich die Frage, ob es eine Nichtleere Menge $A = S_W \setminus S_B$ gibt.

Die Such-Strings würden in der Wortliste gefunden werden, aber nicht in den Sätzen:

$$A = \{a/\forall a \in S_W \ \nexists b \in B(a \otimes b) \wedge \forall a \in S_W \exists w \in W(a = w)\}$$

Zusammen mit der Bedingung, dass alle Wörter der Wortliste in den Sätzen gefunden werden $W = \{w/\forall w \exists b \in B(w \otimes b)\}$, ergibt sich:

$$A = \{a/\forall a \in S_W \ \nexists b \in B(a \otimes b) \wedge \forall a \in W \exists b \in B(w \otimes b)\}$$

Diese beiden Bedingungen schließen sich gegenseitig aus und die Menge A muss leer sein.

Die Such-Strings S_W, die bei einer Suche in der Wortliste gefunden werden, werden ebenfalls alle bei einer Suche in den Beispielsätzen gefunden. Es gilt:

$$S_W \subseteq S_B$$

und

$$S_W \setminus S_B = \emptyset$$

Um die Frage, wann ein Suchbegriff gefunden wird, zu beantworten, reicht es also aus, die Suche in den Beispielsätzen heranzuziehen.

3.6. Mehrwortbegriffe

Alexander Kaiser definiert Mehrwortbegriffe kurz als:

> „sprachliches Konstrukt mit ganz bestimmten syntaktischen und semantischen Eigenschaften“ [28]

Aus Sicht eines Übersetzers, könnte man Mehrwortbegriffe als

> „alles, was als Einheit übersetzt werden muß“ [46]

definieren.

Hier sollen Mehrwortbegriffe wie folgt definiert werden:

Definition 3.4 (Mehrwortbegriff) *Ein Mehrwortbegriff besteht aus mehreren Einzelwörtern, die als semantische Einheit eine bestimmte Bedeutung haben. Die Einzelwörter können unabhängig vom Mehrwortbegriff evtl. auch separat in einer anderen Bedeutung gebraucht werden.*

Mehrwortbegriffe können Namen von Personen, Institutionen oder Orten, idiomatische Wendungen wie z. B. „ins Gras beißen“ oder Zusammensetzungen wie „kinetische Energie“ sein (vgl. [46]).

Damit der ConceptComposer (siehe Abschnitt 3.4.2 auf Seite 32) Mehrwortbegriffe nicht als einzelne Wörter, sondern als zusammenhängende Einheiten erkennt, muss man ihm diese vorher in Form einer Liste bekannt machen.

Es sind viele Ansätze denkbar, Mehrwortbegriffe zu finden. Im Rahmen dieser Arbeit werden folgende untersucht bzw. sind zum Einsatz gekommen:

- Nutzung von Suchanfragen
- Mustererkennung im vorhandenen Text
- Aufeinander folgende Nomen
- Pendelalgorithmus, NameRec-Package
- Kollokationen

3.6.1. Mehrwortbegriffe in Suchanfragen

Es ist durchaus gängige Praxis bei Suchmechanismen, die Eingabe mehrerer einzelner durch Freizeichen getrennte Suchbegriffe intern mit einer logischen Und-Verknüpfung zu versehen. Die Eingabe wird also vom System so interpretiert, dass die gesuchten Dokumente alle Wörter der Anfrage beinhalten müssen.

Unabhängig von der tatsächlich vorhandenen Funktionalität wird dieses Verhalten vom Benutzer oft intuitiv vorausgesetzt und die Suchanfragen werden entsprechend formuliert.

Wie auch bereits im Abschnitt 3.5.2 (S. 40) formuliert, gibt es zwei Möglichkeiten, eine Suchanfrage mit mehr als einem Wort zu deuten:

1. Es handelt sich um eine lose Aneinanderreihung einzelner Suchbegriffe, mit dem Zweck, die Treffermenge einzuschränken.

2. Die einzelnen Wörter ergeben zusammen einen Mehrwortbegriff.

Bei der ersten Möglichkeit kann man davon ausgehen, dass die Suchanfrage selten in gleicher Weise mehrfach gestellt wird, da die Anzahl, die Kombination und die Position der einzelnen Wörter vom Nutzer frei wählbar sind. Die dadurch entstehenden Kombinationsmöglichkeiten sind entsprechend zahlreich. Bei Mehrwortbegriffen ist die Wortstruktur vorgegeben. Ein zweiter Nutzer wird den Begriff in gleicher Weise verwenden, die Häufigkeit dieser Such-Strings wird also wesentlich höher sein.

Filtert man anhand der Anzahl ihres Auftretens die Mehrwort-Suchanfragen, so bekommt man qualitativ gute Mehrwortbegriffe.

Eventuelle Tippfehler werden ebenfalls durch die Anzahl des Auftretens gefiltert. Es sei denn, es handelt sich um einen typischen Tippfehler, der von vielen Menschen gern gemacht wird. In einem solchen Fall besteht aber auch die Möglichkeit, dass ein Redakteur den gleichen Fehler macht. Solche fehlerhaften Mehrwortbegriffe sollen ebenfalls als Mehrwortbegriffe und nicht als einzelne Wörter erkannt werden.

Von den insgesamt 80 Tsd. verschiedenen Suchanfragen enthalten 29 Tsd. mehrere Wörter. Wählt man einen geeigneten Schwellwert für die Häufigkeit der Such-Strings, so bekommt man eine qualitativ gute Menge von Mehrwortbegriffen. Bei den vorhandenen Daten erweist sich ein Schwellwert von vier als geeignet. Dabei erhält man rund zwei Tsd. Mehrwortbegriffe mit guter Qualität.

Tabelle 3.6 auf Seite 50 zeigt die 20 häufigsten Suchanfragen, die potentielle Mehrwortbegriffe enthalten.

Viele Nutzer sind es von verschiedenen Suchmechanismen gewohnt, dass die Groß-/Kleinschreibung ignoriert wird. Entsprechend werden Eingaben oft einfach klein geschrieben. In den gezeigten Such-Strings in Tabelle 3.6 auf Seite 50 sind bereits recht gute Mehrwortwortbegriffe zu sehen. Bei einigen ist die Groß-/Kleinschreibung auch korrekt, teilweise wurden die Wörter aber eben einfach klein geschrieben. Auffallend ist, dass zumindest bei den häufigen Wörtern für jeden klein geschriebenen Begriff die korrekt geschriebene Variante ebenfalls in der Liste enthalten ist. Ein einfacher Algorithmus könnte die Liste bereinigen, indem er bei jedem komplett klein geschriebenen Begriff überprüft, ob es eine groß geschriebene Variante in der Liste gibt. Wenn dem so ist, kann der Begriff aus der Liste der Mehrwortbegriff-Kandidaten gelöscht werden.

3.6.2. Mustererkennung im vorhandenen Text

Mehrwortbegriffe entsprechen im Deutschen oft bestimmten Mustern. Diese Eigenschaft kann man sich zu Nutze machen und mit Mustererkennungsverfahren einen Teil der verwendeten Mehrwortbegriffe aus einem vorhandenen Textbestand extrahieren.

Als Textbestand werden hier die Dokumente der Website genutzt.

Ämter

Ein Ansatz zielt auf die Namen von Ämtern ab. Diese heißen meist „Amt für ...“ oder „...amt ...“, z. B. „Amt für Wirtschaftsförderung“ oder „Standesamt Leipzig“. Diese lassen sich leicht extrahieren.

Mehrwortbegriff	Häufigkeit der Suchanfragen
Arena Leipzig	417
Haus Auensee	323
arena leipzig	276
Botanischer Garten	253
haus auensee	225
botanischer garten	214
Deutsche Bücherei	184
rolling stones	149
Leipzig Arena	145
Auerbachs Keller	128
honky tonk	125
Honky Tonk	118
Neues Rathaus	117
werk 2	116
Renaissance Hotel	115
alte messe	111
Hotel Mercure	108
Werk 2	107
gelbe seiten	104
Neue Messe	103

Tabelle 3.6.: *Mehrwortbegriffe aus Suchanfragen (Auszug)*

Die in Tabelle 3.7 auf Seite 52 zu sehenden Amtsbezeichnungen konnten mit dem Muster „Amt für ...“ aus den Beispielsätzen der ConceptComposer-Datenbank extrahiert werden. Dabei wurde der reguläre Ausdruck so formuliert, dass nach „Amt für“ optional beliebig viele klein geschriebene Wörter kommen können und dann ein groß geschriebenes kommen muss.

Wie man sieht, ist dieses Verfahren nicht sehr ergiebig und auch nicht perfekt. „Amt für die Region“ passt zwar in das Muster, ist aber sicher keine gesuchte Amtsbezeichnung. Mit manueller Kontrolle und mit größeren Textbeständen kann dieses Verfahren aber vielleicht auf andere Beispiele übertragen werden.

3.6.3. Aufeinander folgende Nomen

In der deutschen Sprache sind viele Mehrwortbegriffe aufeinander folgende Nomen. Nomen werden im Deutschen immer groß geschrieben. Neben den Nomen sollen auch Namen von Personen oder Einrichtungen als Mehrwortbegriffe erkannt werden. Diese werden auch in fast allen Fällen groß geschrieben. Man kann also alle aufeinander folgenden groß geschriebenen Wörter extrahieren.

Neben den Namen und Nomen werden Wörter am Satzanfang und einige Ausnahmen (z. B. Anreden wie „Du“ oder „Sie“) mitten im Satz groß geschrieben. Hinterlegt man eine Liste mit bekannten Wörtern als Ausnahmen, bekommt man schon recht gute Ergebnisse. Nach einem Probelauf finden sich die Wörter in Tabelle 3.8 auf Seite 53 für die Liste der Ausnahmen.

Aufzählungen von Nomen werden mit Kommata getrennt, sind also ebenfalls leicht zu filtern.

Die zur Verfügung stehende Textbasis ist leider nicht nur Fließtext. In den HTML-Dokumenten sind auch Menüstrukturen und Ähnliches eingebettet.

Amt für Ausbildungsförderung
Amt für Bauordnung
Amt für Stadterneuerung
Amt für Stadtsanierung
Amt für Statistik
Amt für Strahlenschutz
Amt für Umweltschutz
Amt für Verkehrsplanung
Amt für Wirtschaftförderung
Amt für Wirtschaftsentwicklung
Amt für Wirtschaftsfoerderung
Amt für Wirtschaftsförderung
Amt für Wirtschaftsfördeung
Amt für Wohnungswesen
Amt für Wohnungwesen
Amt für amtliche Veröffentlichungen
Amt für die Region
Amt für ländliche Neuordnung

Tabelle 3.7.: *Mit Mustererkennung gefundene Ämter*

Typische Satzanfänge:	Der, Die, Das, Des, Den, Dem, Als, Diese, Dieses, Dieser, Diesen, Diesem, Dies, Ein, Eine, Eines, Einer, Einem, Im, In, Ins, Vom, Von, Vor, Um, Was, Wie, Wer, Wo, Wann, Welche, Welches, Welcher, Welchem, Bei, Beim, An, Am, Zum, Für, Zur, Zu, Mit, Ab, Alle, Viele, Viel, Vieles, Nach, Jede, Jeder, Jeden, Jedes, Auch, Auf
Anrede von Personen:	Herr, Herrn, Frau, Ihr, Ihre, Ihrem, Ihren, Ihres, Ihnen, Sie, Du, Dein, Deinem, Deine, Deinen
Zeitangaben:	Uhr

Tabelle 3.8.: *Ausnahmen für aufeinander folgende Nomen*

Menüs bestehen oft aus mehreren Nomen, die einzeln verlinkt und durch grafische Mittel als einzelne Menüpunkte dargestellt sind. Nach der Konvertierung in Plaintext lassen sich diese jedoch nicht mehr als solche identifizieren. Sie stellen sich als aufeinander folgende Nomen dar und werden damit als Mehrwortbegriffe erkannt.

Eine weitere Eigenschaft von solchen Strukturen ist, dass sie sich oft in ähnlicher Weise wiederholen und aus mehr als nur drei Wörtern bestehen. D. h. man kann sie im Nachhinein leicht herausfiltern.

Alle weiteren Seiteneffekte, die auftreten, sollten sich über die Häufigkeit des Auftretens filtern lassen. Korrekte Mehrwortbegriffe werden häufiger auftreten als zufällige Aneinanderreihungen.

Ein kleines Programm, das nach aufeinander folgenden Nomen sucht, muss nach Mustern suchen, die folgende Bedingungen erfüllen:

1. beginnen mit Großbuchstaben

2. entsprechen nicht den festgelegten Ausnahmen

3. sind aufeinander folgend und nur mit Freizeichen getrennt

Als Textquelle können direkt die konvertierten Dokumente der Website oder alternativ die bereits vorbearbeiteten Sätze in der ConceptComposer-Datenbank dienen. Die Sätze in der Datenbank haben den Vorteil (bei entsprechender Konfiguration des ConceptComposers) nicht doppelt aufzutreten und damit die Häufigkeiten der einzelnen Wörter nicht zu verzerren. Im Folgenden wird die Datenbank als Textquelle verwendet.

Im ersten Schritt wurden 27.774 Kandidaten für Mehrwortbegriffe gefunden.

In einem zweiten Schritt werden alle Kandidaten in einem teils manuellen, teils automatischen Verfahren als korrekte bzw. nicht korrekte Mehrwortbegriffe markiert. Nur die als korrekt markierten Begriffe sollen am Ende verwendet werden.

Im Text werden natürlich nicht nur Grundformen, sondern auch flektierte Formen verwendet. Diese sollen hier ebenfalls erkannt werden.

Abbildung 3.8 auf Seite 55 zeigt zur Übersicht eine schematische Darstellung des Prozesses.

Manuelle Kontrolle

Zur manuellen Kontrolle steht ein webbasiertes Werkzeug zur Verfügung. In Abbildung 3.9 auf Seite 56 ist dieses am Beispiel zu sehen. Die ungeprüften Mehrwortbegriff-Kandidaten werden nach ihrer Häufigkeit absteigend sortiert anzeigt. D. h. der Kandidat, der am häufigsten im Text gefunden wurde, wird zuerst angezeigt.

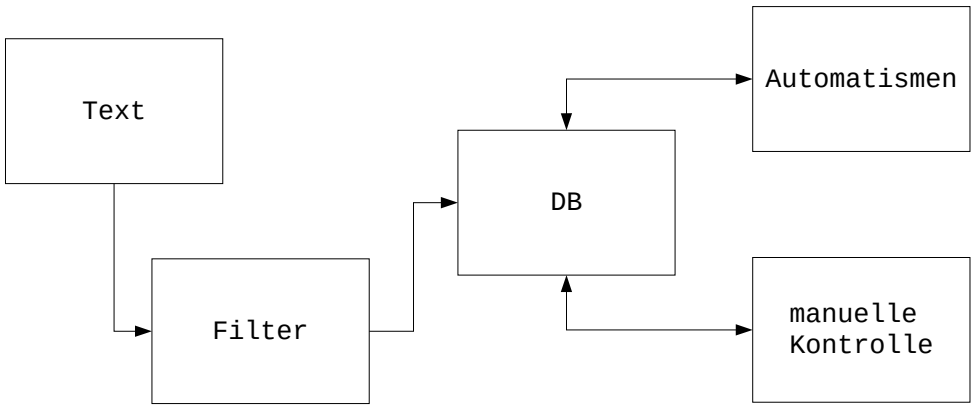

Abbildung 3.8.: *Aufeinander folgende Nomen - Prozess*

Für jeden Kandidaten kann der Bearbeiter wählen, ob es sich um einen bzw. keinen korrekten Mehrwortbegriff handelt. Die dritte Option, die auch per Default gewählt ist, belässt diesen Kandidaten als ungeprüft. Er wird dann nach dem Speichern zusammen mit den nächsten ungeprüften Datensätzen wieder angezeigt.

Mit Fortschreiten des Bearbeitungsvorgangs sinken sortierungsbedingt die Häufigkeiten der angezeigten Mehrwortbegriff-Kandidaten. Theoretisch kann man die Bearbeitung fortsetzen, bis alle Daten gesichtet wurden. Praktisch wird man feststellen, dass der Anteil der sinnvollen Kandidaten mit der Häufigkeit nachlässt und ab einer gewissen Häufigkeit so gut wie keine sinnvollen Kandidaten mehr auftreten. Bei den hier vorliegenden Daten war die Bearbeitung von Kandidaten mit einer Häufigkeit von größer als vier sinnvoll.

Bei der manuellen Kontrolle der Daten erkennt man schnell sich wiederholende Muster, die genutzt werden können, um einen Teil der Kandidaten automatisch

Begriff	anz	ok?	Begriff	anz	ok?
Innenstadtrestaurant Preis	6	ok not ok ?	Abend Möglichkeit	6	ok not ok ?
Südumfahrung Leipzig	6	ok not ok ?	Leipziger Kabaretts	6	ok not ok ?
Lutherstadt Wittenberg	6	ok not ok ?	Wörlitzer Park	6	ok not ok ?
Zoologischen Garten	6	ok not ok ?	Alter Johannisfriedhof	6	ok not ok ?
Napoleons Truppen	6	ok not ok ?	Leipziger Tieflandsbucht	6	ok not ok ?
Krostitzer Weg	6	ok not ok ?	Gosenschenke Ohne Bedenken	6	ok not ok ?
Mount Everest	6	ok not ok ?	Leipziger Bierbörse	6	ok not ok ?
Innenhof Sommertheater	6	ok not ok ?	Empio Punito	6	ok not ok ?
Leipziger Servicebetriebe	6	ok not ok ?	Leipzig Burgstraße	6	ok not ok ?
Christi Himmelfahrt	6	ok not ok ?	Bayerischen Bahnhof	6	ok not ok ?
Städtische Klinikum	6	ok not ok ?	Gemeinde Wiederitzsch	6	ok not ok ?
Adobe Acrobat Reader	6	ok not ok ?	Buches Gerichtsweg	6	ok not ok ?
Südraum Leipzigs	6	ok not ok ?	Hugo Licht	6	ok not ok ?
Kunststoffzentrum Leipzig	6	ok not ok ?	Deutschen Platz	6	ok not ok ?
Leistungen Dritter	6	ok not ok ?	Serena Williams	6	ok not ok ?
Tyczka Minol	6	ok not ok ?	Holger Tschense	6	ok not ok ?
Leipziger Freiheit	6	ok not ok ?	Bereich Medien	6	ok not ok ?
Johannes Rau	6	ok not ok ?	Bewerbungskomitee Leipzig	6	ok not ok ?
Internationalen Olympischen Komitees	6	ok not ok ?	Rayan Abdullah	6	ok not ok ?
Buchkunst Leipzig	6	ok not ok ?	Leipziger Straßen	6	ok not ok ?
					submit

Abbildung 3.9.: *Manuelle Bearbeitung der Mehrwortbegriff-Kandidaten*

markieren zu lassen.

Negative Kriterien

Als nicht korrekt markiert werden alle Kandidaten, die eines der folgende Kriterien erfüllen:

- mehr als drei Teil-Wörter
- erstes Teilwort: „Bezeichnung“, „Navigation“, „Name“, „Namen“, „Mail“, „Logo“ oder „Startseite“
- letztes Teilwort: „Schulen“, „Details“, „Produkte“

Bei den Kandidaten mit vielen Teil-Wörtern (mehr als drei) ist die Wahrscheinlichkeit sehr hoch, dass es sich nicht um sinnvolle Mehrwortbegriffe handelt. Das sind meist Menüpunkte und ähnliches. Die genannten Teilwörter am Anfang und Ende sind einzig auf Beobachtungen bei der manuellen Kontrolle zurückzuführen.

Positive Kriterien

Unter den Suchanfragen finden sich viele Straßennamen. Man kann also alle Kandidaten mit zwei Teil-Wörtern und dem zweiten Teil-Wort „Straße“ als korrekt markieren.

Desweiteren können die GmbHs und AGs, also die Kandidaten, die auf „AG“ oder „GmbH“ enden, als korrekt markiert werden.

Mehrwortbegriffe aus langen Folgen von Nomen

Die Mustererkennung ist so konzipiert, dass sie immer alle aufeinander folgenden Nomen erkennt. In manchen Fällen kommt es vor, dass nur eine Teilmenge der Nomen einen oder mehrere Mehrwortbegriffe enthält, während der erkannte String aber kein gültiger Mehrwortbegriff ist. Wenn der Textkorpus groß genug ist, stellt das kein Problem dar. Die einzelnen Mehrwortbegriffe kommen dann an anderen Stellen im Text auch alleine vor und werden dann an diesen Stellen erkannt. In kleineren Textkorpora mit insgesamt weniger Fundstellen kann es aber Sinn machen, auch alle Teilkombinationen erkennen zu lassen.

Am Beispiel „Alte Messe Leipzig“ würden entsprechend die Teilkombinationen „Alte Messe“, „Messe Leipzig“ und die Gesamtkombination „Alte Messe Leip-

zig“ erkannt werden. Man kann sich leicht vorstellen, dass dieses Verhalten den Rauschanteil unter den korrekten Mehrwortbegriffen sehr erhöhen kann.

Insgesamt können von den 27.774 Kandidaten 1.205 korrekte Mehrwortbegriffe extrahiert werden.

3.6.4. Pendelalgorithmus

Der Pendelalgorithmus extrahiert semantische Relationen aus natürlichsprachlichem Text [5, 6].

> „Den Kern des Verfahrens bildet die Iteration von Suchschritt und Verifikationsschritt, in denen in gesuchter Relation stehende Wörter gefunden und überprüft werden. Auf diese Weise ist es möglich, mit wenigen bekannten Wörtern eine große Anzahl in derselben Relation stehende Wörter zu gewinnen. So können mit wenig Aufwand große Listen von Wörtern erstellt werden, die in einem semantischen Zusammenhang stehen.“ [6]

Das Verfahren eignet sich besonders gut, um Namen zu finden. Dabei werden dem System Vor- und Zunamen zusammen mit einer Reihe von Regeln als „Wissen“ mitgegeben. Wenn das System im Text auf einen bekannten Vor- oder Zunamen in Verbindung mit einem unbekannten Gegenstück stößt, wird der neue Name nach einem Verifikationsschritt „gelernt“.

Im Versuch zeigt sich, dass der vorhandene Textbestand von fast 100 Tsd. Sätzen zu klein ist, um mit dem Pendelalgorithmus brauchbare Ergebnisse zu erzielen.

NameRec-Package

Für die Verarbeitung sehr kleiner Texte wurde eine Variation des Pendelalgorithmus, das NameRec-Package [5], geschaffen. Es nutzt für den Verifikationsschritt einen zweiten, größeren Textkorpus.

Der Algorithmus konnte aus dem hier vorliegenden Text 925 Personennamen extrahieren und erwies sich damit durchaus als brauchbar.

Nutzung vorhandener Personen- und Firmennamen

Mit dem Algorithmus wurden bereits im Vorfeld auf anderen Textbeständen gültige Mehrwortbegriffe, insbesondere Namen, gesammelt, die sich zur weiteren Nutzung anbieten. Der Umstand, dass die Daten nicht von der hier untersuchten Website stammen, hat keinen störenden Einfluss. Der Concept-Composer nutzt die mitgegebenen Mehrwortbegriffe, um gefundene Begriffe zu überprüfen. Die Begriffe selbst werden nur zur Wortmenge hinzugefügt, wenn sie im untersuchten Text tatsächlich enthalten sind. Die Art der bekannten Mehrwortbegriffe verändert also nicht den Inhalt des indizierten Textkorpus.

Firmennamen Mit Hilfe des Pendelalgorithmus wurden rund vier Tsd. Namen überwiegend deutscher Firmen gewonnen [7].

Personennamen Im Zusammenhang mit den „Wörtern des Tages“ [38] entstand eine lange Liste mit Personen-Namen, welche mehr als 386 Tsd. Einträge umfasst.

3.6.5. Kollokationen

Kollokationen beschreiben das gemeinsame Auftreten von Wörtern [19, 42].

Typische Kollokationsarten sind Nachbarschaftskollokationen und Satzkollokationen. Nachbarschaftskollokationen beschreiben die Häufigkeit des gemeinsamen Auftretens von zwei Wörtern in direkter Nachbarschaft. Von Satzkollokationen spricht man, wenn zwei Wörter gemeinsam im selben Satz auftauchen.

Nun könnte man annehmen, dass starke Nachbarschaftskollokationen (Kollokationen mit hoher Signifikanz) von Nomen gute Kandidaten für Mehrwortbegriffe sind.

Nachbarschaftskollokationen treten allerdings nicht nur auf, wenn die Wörter mit Freizeichen getrennt sind, sondern auch, wenn z. B. Klammern zwischen ihnen stehen. Solche Wortkombinationen stellen natürlich keine Mehrwortbegriffe dar.

Kollokationen haben genau dann eine hohe Signifikanz, wenn sie genügend oft gemeinsam im Text vorkommen. Nomen, die oft gemeinsam vorkommen, wurden ebenfalls im Abschnitt 3.6.3 (Seite 51) durch Mustererkennung ermittelt. Bei dem musterbasierten Verfahren ist sichergestellt, dass keine störenden Zeichen zwischen den Wörtern stehen. Es ist also nicht zu erwarten, mit dieser Methode neue oder mehr aufeinander folgende Nomen zu finden. Stattdessen findet man vielleicht Mehrwortbegriffe, die nicht nur aus Nomen bestehen.

Die Berechnung der Kollokationen erfolgt durch den ConceptComposer und die Ergebnisse finden sich in der resultierenden Datenbank.

Ob eine Wortkombination ein sinnvoller Mehrwortbegriff ist, kann nur manuell entschieden werden. Dazu werden die Kollokationen nach der Signifikanz absteigend sortiert, so dass die interessanten Kandidaten ganz oben stehen.

Im Versuch zeigt sich, dass unter den Kollokationen, bestehend aus Nomen, keine neuen Mehrwortbegriffe sind. Unter den obersten 500 Nachbarschaftskollokationen sind 179 bereits bekannte Wortkombinationen (durch Mustererkennung gefunden) und davon 65 Mehrwortbegriffe. Die Tatsache, dass Wörter bei Nachbarschaftskollokationen nicht zwingenderweise durch Freizeichen getrennt sein müssen, wirkt sich hier als störender Faktor aus, so dass die Ergebnisse nicht besser sind als die Mustererkennung im Abschnitt 3.6.3.

Die Aufhebung der Einschränkung auf Nomen, also die Betrachtung aller Wortkombinationen verbessert das Ergebnis nicht. Zwei Worte sind für Mehrwortbegriffe, die nicht nur aus Nomen bestehen, im Deutschen anscheinend zu wenig.

Mehrfachkombination von Kollokationen

Beim Durchsehen der Kollokationen fällt das Beispiel „Amt für Wirtschaftsförderung“ auf. D. h. „Amt für“ und „für Wirtschaftsförderung“ treten beide separat auf. Um dieses Beispiel und ähnliche Kandidaten als Ganzes zu erfassen, ist ein Index für das gemeinsame Auftreten von drei Wörtern nötig. Ein solcher Index könnte helfen, Mehrwortbegriffe, die aus drei beliebigen Wörtern und nicht nur aus Nomen bestehen, zu finden.

Nachbarschaftskollokationen beschreiben das gemeinsame Auftreten von zwei Wörtern. Verknüpft man zwei solche Kollokationen, erhält man einen Index für das gemeinsame Auftreten von drei Wörtern.

Wenn $k(a, b)$ den Wert der Nachbarschaftskollokation zwischen den Wörtern a und b beschreibt und $k(b, c)$ den Wert der Nachbarschaftskollokation zwischen den Wörtern b und c, dann soll $k(a, b, c)$ einen Index für das gemeinsame

Auftreten der drei Wörter a, b und c liefern. Dieser Index soll als Summe der beiden Einzelwerte errechnet werden:

$$k(a, b, c) = k(a, b) + k(b, c)$$

Der Index $k(a, b, c)$ hat einen hohen Wert, wenn

1. $k(a, b)$ und $k(b, c)$ jeweils hohe Werte haben oder
2. einer der beiden Einzelwerte einen hohen und der andere einen niedrigen Wert hat.

Es reicht also nicht aus, den Gesamtindex losgelöst von den Einzelindizes zu betrachten.

Aus Datenbanksicht werden zwei Kollokationsrelationen anhand des rechten Wortes der ersten Kollokation mit dem linken Wort der zweiten Kollokation verknüpft. Die Sortierung kann nach dem Gesamtindex $k(a, b, c)$ erfolgen, wobei jeweils ein Minimum für die Einzelindizes festgelegt werden sollte.

Die Ausbeute im Ergebnis war auch hier nicht sehr groß. Außer dem Ausgangsbeispiel „Amt für Wirtschaftsförderung“, was die Idee für das Verfahren geliefert hat, wurden keine weiteren Mehrwortbegriffe gefunden.

3.7. Fachbegriffe

Die Redakteure eines Bürgerportals verwenden aufgrund ihrer Arbeit in einem abgesteckten Bereich eine gewisse Fachsprache. Diese Fachsprache findet sich auch in den Dokumenten dieser Redakteure, den Dokumenten des Bürgerportals (der Website), wieder.

Die Deutsche Industrie-Norm (DIN) definiert den Begriff „Fachsprache“ in DIN 2342 wie folgt:

> „Bereich der Sprache, der auf eindeutige und widerspruchsfreie Kommunikation in einem Fachgebiet gerichtet ist und dessen Funktionieren durch eine festgelegte Terminologie entscheidend unterstützt wird.“ [41]

Das schließt natürlich nicht aus, dass ein Teil dieser Fachsprache evtl. den Besuchern der Website (den Bürgern) geläufig ist. Wenn sie also nach einem bestimmten Thema suchen, verwenden sie u. U. bereits den korrekten Begriff, obwohl sie diesen Begriff im allgemeinen Sprachgebrauch selten verwenden würden.

Es sind also zwei Dinge gesucht:

1. Die Fachbegriffe des Bürgerportals und
2. diejenigen Fachbegriffe, die den Besuchern nicht geläufig sind.

3.7.1. Terminologie-Extraktion

Es gibt mehrere Ansätze, Fachtermini zu errechnen. Zum einen sind das musterbasierte und zum anderen statistische Verfahren [24]. Letztere kommen in dieser Arbeit zum Einsatz.

Vergleicht man die relative Häufigkeit von Begriffen einer Domäne mit der Häufigkeit dieser Begriffe in einem allgemeinsprachlichen Textkorpus, so fällt auf, dass die Fachbegriffe dieser Domäne in den zugehörigen Dokumenten im Verhältnis häufiger auftreten als im allgemeinen Wortschatz [24].

Als Referenz für den allgemeinen Sprachgebrauch steht das Projekt „Deutscher Wortschatz“ [12, 43] zur Verfügung.

Mit dem ConceptComposer (vgl. Abschnitt 3.4.2 auf S. 32) steht ein Werkzeug zur Verfügung, das diesen statistischen Vergleich von Textkorpora durchführen kann. Dazu wird eine so genannte „Hitliste“ errechnet, die diejenigen Begriffe enthält, die im untersuchten Dokumentenbestand im Verhältnis häufiger vorkommen als im verwendeten Referenzkorpus.

Voraussetzung für die Erstellung der Hitliste ist eine existierende Concept-Composer-Datenbank. Existiert sie noch nicht, kann sie durch den Concept-Composer aus den Dokumenten erstellt werden. Die Hitliste selbst findet sich nach der Berechnung ebenfalls in dieser Datenbank in der `wortliste`, in den Feldern `qual_pos` und `anzahl` wieder. Die Anzahl ist die absolute Häufigkeit des Auftretens dieses Begriffs im Textbestand. `qual_pos` enthält einen errechneten Wert als Gewichtung für diesen Begriff.

Es können natürlich nur solche Begriffe bewertet werden, die zuvor als Begriffe erkannt wurden. Wenn der ConceptComposer auf einen für ihn unbekannten Mehrwortbegriff stößt, werden die einzelnen Wörter als separate Begriffe erkannt. Man muss dem ConceptComposer also vor der Indizierung des Textes möglichst viele Mehrwortbegriffe mitgeben. Siehe dazu Abschnitt 3.6 auf Seite 46.

In Tabelle 3.9 auf Seite 65 sind die obersten zehn Einträge der Hitliste für „leipzig.de“ zu sehen.

Begriff	Gewichtung	Anzahl
Carmen Jacob	25781	454
Leipzig	14465	165
Kopie	14286	364
Portal	8660	363
Statistik	5723	169
Mittelschule	5284	435
Bezeichnung	5283	212
Grundschule	5225	171
Navigation	4278	385
Stadt Leipzig	4001	454

Tabelle 3.9.: *Hitliste*

3.7.2. Selten gesuchte Fachbegriffe

Wie lassen sich nun die Fachbegriffe ermitteln, die den Bürgern nicht geläufig sind? Als Ressource für den Wortschatz der Bürger stehen die Suchanfragen zur Verfügung. Es lässt sich also überprüfen, welche Fachbegriffe nicht oder selten gesucht werden.

Technisch sind das jene Fachbegriffe, die nicht als Such-String enthalten sind bzw. deren Häufigkeit niedriger als ein geeignet gewählter Schwellwert ist.

Tabelle 3.10 auf Seite 66 zeigt die obersten zehn Einträge der Hitliste, die max. fünf mal gesucht worden sind. Wie man sieht, sind die gängigsten Begriffe wie „Leipzig“, „Statistik“ oder „Grundschule“ nicht mehr enthalten.

Begriff	Gewichtung	Anzahl	Anzahl Suchanfragen
Carmen Jacob	25781	454	1
Kopie	14286	364	5
Portal	8660	363	1
Bezeichnung	5283	212	0
Navigation	4278	385	2
Dateianhang	3710	451	0
Typ	3663	101	0
Not Categorized	2801	454	0
Förderschule	2602	422	0
Bereich	2532	16	0

Tabelle 3.10.: *Hitliste - nicht oder selten gesuchte Begriffe*

3.8. Begriffsabgleich

Es wurden Suchbegriffe errechnet, die nicht auf der Website vorkommen (Begriffe der „Bürgersprache“) und es wurden Fachbegriffe errechnet, die nicht in den Suchanfragen verwendet werden. Jetzt stellt sich die Frage, wie man diese beiden Begriffsmengen sinnvoll miteinander kombinieren kann.

Die Idee war, dass diese beiden Wortmengen evtl. die gleichen Thematiken beschreiben. Wenn dem so ist, sollte es möglich sein, die beiden Wortmengen aufeinander abzubilden und auf diese Weise Synonyme zu errechnen.

Diese Synonymberechnung wird nicht vollautomatisch möglich sein. Auf der anderen Seite ist der Aufwand, die Wortmengen manuell miteinander zu vergleichen, sicher zu groß. Es ist aber möglich, mit geeigneten Verfahren Kandidaten zu berechnen, die evtl. Synonyme darstellen. Diese Kandidaten können dann manuell überprüft werden.

Dazu wurde ein Framework geschaffen, dass die grundlegende Funktionalität für die Abbildung der beiden Wortmengen und die manuelle Überprüfung bietet. In dieses Framework können dann verschiedene Verfahren integriert werden.

3.8.1. Framework für Synonymberechnung

Der Aufruf der Synonymgenerierung erfolgt über die Kommandozeilenoption `-generate_synonyms` bzw. der Methode `action_generate_synonyms()` der Klasse `Analyse`.

Um ein Verfahren zu implementieren, wird die abstrakte Klasse `SynonymGen` abgeleitet. Es müssen mindestens zwei Methoden implementiert werden:

- `getName()` gibt einen String mit dem Namen des Verfahrens zurück. Dieser Name wird zusammen mit den errechneten Synonymen in der Datenbank gespeichert, um die Ergebnisse einzelner Verfahren miteinander vergleichen oder alle Ergebnisse bestimmter Verfahren pauschal als gut oder schlecht bewerten zu können.
- `generateSynonyms()` nimmt einen Fachbegriff (als Instanz vom Typ `Fachbegriff`) entgegen und generiert alle Synonyme für diesen.

Den Begriff selbst als String liefert die Methode `getBegriff()` der Klasse `Fachbegriff`. Synonyme werden dieser über die Methode `addSynonym()` übergeben. Als Parameter erwartet diese Methode das Synonym als String sowie den Namen des Verfahrens (`this.getName()`).

Die Suchbegriffe, die bei der Synonym-Berechnung genutzt werden dürfen und sollen, werden nicht extra übergeben. Sie stehen über die Datenbank zur Verfügung.

Eine sehr einfache Beispiel-Implementierung ist in der Klasse `DummySynonymGen` vorhanden.

Um ein neues Verfahren zu integrieren, wird die Methode `action_generate_synonyms()` der Klasse `Analyse` entsprechend angepasst.

Die Ergebnisse der Generierung finden sich in der Tabelle `synonyme` der Projekt-Datenbank. Neben dem Synonym, dem Verfahren und einer Referenz auf den Fachbegriff enthält die Tabelle ein Feld `good`. Hat dieses Feld den Wert `null`, ist es noch ungeprüft. Ansonsten gibt der boolsche Wert an, ob ein Bearbeiter das Synonym für gut (true) oder schlecht (false) befunden hat.

Die Bearbeitung der Synonyme erfolgt über das Web-Frontend `synonyme.php`. Abbildung 3.10 auf dieser Seite zeigt einen Screenshot mit Beispieldaten, die von `DummySynonymGen` generiert wurden.


Synonyme

Fachbegriff	Synonym	ok?	Quelle
Kommunalwirtschaft	Test-Synonym	ja nein ?	dummy
Plagwitz	Test-Synonym	ja nein ?	dummy
Johannes-Kepler-Schule	Test-Synonym	ja nein ?	dummy
Liebertwolkwitz	Test-Synonym	ja nein ?	dummy
Zuwendungsempfänger	Test-Synonym	ja nein ?	dummy

Abbildung 3.10.: *Manuelle Bearbeitung der Synonyme*

3.8.2. Ideen für Verfahren

Als Verfahren für die Berechnung möglicher Synonyme gibt es viele Möglichkeiten. Man könnte verschiedene Wortähnlichkeitsalgorithmen implementieren.

Verfahren, die auf weitere Ressourcen (Textbestände, Wörterbücher) zurückgreifen, können zum Einsatz kommen. Evtl. bieten sich Verfahren aus dem Bereich des Information Retrieval an.

Unter den Fachbegriffen finden sich viele Namen von Personen, Straßen oder Einrichtungen. Ein Verfahren, das häufig im Bereich Namensähnlichkeit eingesetzt wird, ist „Soundex“ [45]. Dabei wird für einen Namen anhand dessen Aussprache ein Code generiert. Zwei Namen mit gleicher Aussprache bekommen dabei den gleichen Soundex-Code. Die beiden Namen „Mike“ und „Maik“ werden beispielsweise unterschiedlich geschrieben, aber identisch ausgesprochen. Mit dem Soundex-Algorithmus bekommen beide den gleichen Code „M200“ und werden damit algorithmisch vergleichbar.

Um nun die Fälle abzufangen, in denen ein Nutzer einen gesuchten Namen einfach anders schreibt, könnte man das Soundex-Verfahren als Synonym-Generator implementieren.

4. Ausblick

4.1. Kritische Würdigung

Die am Anfang aufgestellte These konnte nicht bestätigt werden - sie wurde aber auch nicht widerlegt. Es konnten Begriffe der „Bürgersprache“ gefunden und es konnten Fachbegriffe extrahiert werden. Diese gewonnenen Daten können die Betreiber und Redakteure des Bürgerportals nutzen, um ihr Angebot dem Interesse und Verhalten der Nutzer anzupassen.

Die am häufigsten gesuchten Begriffe spiegeln das Interesse der Besucher der Website wider. Aus dieser Liste sind folgende Informationen ableitbar:

Das Interesse der Besucher. Redakteure können erkennen, welche Themen besonders gepflegt werden sollten, welche Themen fehlen und welche weniger Aufmerksamkeit bedürfen. Die Ressourcen lassen sich damit besser einteilen.

Das Vokabular. Die häufigsten Suchbegriffe repräsentieren auch einen Teil des Sprachgebrauchs der Besucher. Wenn Inhalte existieren, die zu den Suchbegriffen passen, aber die konkreten Begriffe nicht verwenden, können Redakteure diese, zumindest für die häufigsten Suchbegriffe, als zusätzliche Schlagwörter ergänzen, um Inhalte auffindbar zu machen.

Die Analyse, die durchgeführt wurde, ist durchaus auf ähnliche Projekte portierbar. Voraussetzung dafür ist natürlich der mögliche Zugang zu den nötigen Daten (Website und Log-Dateien des Webservers) und eine vorhandene Such-Funktionalität auf der Website. Einige Punkte, wie z. B. die Suche der aufeinander folgenden Nomen, sind für die deutsche Sprache konzipiert. Mit einigen Anpassungen sollte die Lösung aber auch auf andere Sprachen angepasst werden können.

Der Aufwand für eine Portierung hängt sehr davon ab, wie verwandt ein Projekt ist. Handelt es sich um das Bürgerportal einer anderen deutschen Stadt, so kann die Lösung sicher 1:1 übernommen werden und der Aufwand beschränkt sich auf kleinere Anpassungen am Programm (z. B. das Muster zur Filterung der Suchanfragen) und die Durchführung der Analyse.

Liegt dem Projekt eine andere Sprache zugrunde, müssen einige Methoden angepasst oder weggelassen werden. Variiert der thematische Schwerpunkt des Projektes, muss überlegt werden, inwieweit die Methoden Sinn machen. Prinzipiell sollte sich die Idee aber auf alle Bereiche übertragen lassen, in denen sich Experten als Redakteure und Laien mit evtl. anderem Sprachgebrauch als suchende Nutzer gegenüberstehen.

4.2. Optimierungsansätze

Es gibt sicher einige Stellen, an denen noch Raum für Verbesserungen und weitere Arbeiten ist.

Da wäre z. B. die Implementierung verschiedener Verfahren für die Berechnung von Kandidaten für Synonyme (vgl. Abschnitt 3.8 auf Seite 66).

Alle berechneten oder extrahierten Begriffe wurden in der flektierten Form verarbeitet. Der Einsatz von Grundformreduktion könnte an manchen Stellen Qualitätsverbesserungen mit sich bringen.

Die entstandenen Analysewerkzeuge sind bereits in Hinblick auf eine spätere Erweiterung und einen Einsatz „außer Haus“ entwickelt wurden. Dennoch war das primäre Ziel Werkzeuge zu schaffen, die für die jeweiligen Analysen Ergebnisse liefern. Mögliche Verbesserungen liegen entsprechend in einer eleganteren Integration der einzelnen Werkzeuge oder einer nutzerfreundlicheren Schnittstelle.

4.3. Möglichkeiten der Umsetzung

Es gibt zwei Bereiche, an denen ein Einsatz sinnvoll erscheint:

- Clientseitig
- Serverseitig

4.3.1. Clientseitiger Einsatz

„Clientseitig“ soll hier nicht so verstanden werden, dass die Programmlogik auf Clientseite ausgeführt wird, sondern vielmehr, dass die Funktionalität benutzerorientiert ist. D. h. der Nutzer ist für die Initiierung der Funktion verantwortlich (Suchanfrage) und das Ergebnis wird im Browser des Nutzers dargestellt. Die Programmlogik kann durchaus auf dem Server verbleiben.

Bevor ein Nutzer eine Suchanfrage stellt, kennt er in der Regel die Dokumentenmenge nicht. Er weiß also nicht, wie umfangreich die Treffermenge zu einer

bestimmten Suchanfrage sein wird. Andersherum betrachtet weiß er also nicht, wie präzise eine Suchanfrage formuliert sein muss, um eine adäquate Treffermenge zu erzielen. Der Nutzer wird nach einem ersten Versuch entweder gleich fündig werden oder die Suchanfrage anpassen, um ein besseres Ergebnis zu erzielen. Diese Anpassung wird in der Regel eine Spezialisierung der Anfrage sein, bei der weitere Suchbegriffe ergänzt werden um die Treffermenge einzuschränken. Bei dieser Erweiterung der Anfrage soll der Rechner behilflich sein.

Diese Unterstützung des Nutzers geschieht mittels Query Expansion [2]. Dabei wird in einem ersten Durchlauf die Suchanfrage des Nutzers ausgewertet und mittels verschiedener Verfahren dem Nutzer zusätzliche Suchbegriffe angeboten, die die Treffermenge weiter einschränken können.

Zur Query Expansion können verschiedene Daten/Methoden eingesetzt werden. Wie z. B.:

- verwandte Begriffe aus der Kollokationsumgebung
- Disambiguierung (Auflösung von Mehrdeutigkeiten) [8, 9, 21]

4.3.2. Serverseitiger Einsatz

Auf Serverseite können mit verschiedenen Methoden zusätzliche Schlagwörter zu den jeweiligen Dokumenten generiert werden. Diese Schlagwörter können in einer geeigneten Form als Meta-Daten gespeichert werden, so dass sie vom verwendeten Index-Dienst mit indiziert werden können. Zum Format der Meta-Daten existieren verschiedene Standards bzw. Standardisierungversuche, z. B. die Dublin Core Metadata Initiative [13].

Zusätzliche Schlagwörter können mit folgenden Verfahren generiert werden:

- Hitliste
- Synonym-Wörterbuch

Die vom ConceptComposer (siehe 3.4.2 auf Seite 32) generierte Hitliste enthält die Wörter, die im aktuellen Dokumentenbestand im Verhältnis öfter vorkommen als im Vergleichskorpus. Diese Wörter zeichnen die Dokumente besonders aus. Der Concept Extractor kapselt diese Funktionalität für einzelne Dokumente und generiert Meta-Angaben für diese [14].

Diese Wörter sind keine zusätzlichen Wörter, da sie im Dokument bereits enthalten sind. Sie können aber bei der Indizierung besondere Beachtung finden, wenn sie separat als Schlagwörter codiert sind.

Wenn es gelingt, mit einem geeigneten Verfahren Synonyme für die Fachbegriffe zu errechnen (vgl. Abschnitt 3.8 auf Seite 66), eignen sich diese sehr gut für die Generierung zusätzlicher Schlagwörter.

A. Installations- und Bedienungsanleitung Analyse-Werkzeuge

A.1. Installation

Die Werkzeuge wurden auf einem Linux-System entwickelt und getestet. Aufgrund plattformunabhängiger Entwicklung der Kernkomponenten in Java und PHP sowie plattformübergreifend verfügbarer Datenbank, sollte sich das Werkzeug nach evtl. kleinen Anpassungen (Startscrips etc.) auch auf anderen Systemen verwenden lassen.

A.1.1. Datenbank

Alle Werkzeuge setzen auf eine gemeinsame Datenbank. Dabei handelt es sich um eine PostgreSQL-Datenbank, entsprechend wird eine lauffähige PostgreSQL-Installation vorausgesetzt. Während der Entwicklung wurde die Version 7.3.2 von PostgreSQL verwendet. Es sollten alle Versionen $\geq$7.3 und evtl. auch ältere funktionieren.

Nach Erstellen einer neuen Datenbank, können die nötigen Tabellen mit dem SQL-Script `db.sql` im Verzeichnis `db` erzeugt werden.

Als Beispiel wird hier ein Nutzer „lpz“ sowie eine Datenbank „lpz“ mit allen nötigen Tabellen erzeugt. Nähere Informationen findet man in der PostgreSQL Dokumentation [36].

```
createuser -U postgres -A -d -e -P lpz
createdb -U lpz --encoding=LATIN1 -e lpz
psql -U lpz lpz -f db/db.sql
```

A.1.2. Analyse-Werkzeug

Das Java-basierte Analyse-Werkzeug wurde mit dem J2SE SDK v1.4.2 von Sun entwickelt. Benötigt wird eine kompatible Laufzeitumgebung in der Version 1.4 oder höher.

Die Jar-Dateien im Verzeichnis `jars` auf der CD können an eine beliebige Stelle kopiert werden (oder auch auf der CD bleiben) und müssen dann jeweils dem Classpath hinzugefügt werden.

Wenn die Jar-Dateien im Verzeichnis „jars“ im Homedir liegen und die Bash [3] als Shell verwendet wird, kann das folgendermaßen geschehen:

```
CLASSPATH=$CLASSPATH:~/jars/analyse.jar
CLASSPATH=$CLASSPATH:~/jars/log4j-1.2.8.jar
CLASSPATH=$CLASSPATH:~/jars/mysql-connector-java-3.0.9-stable-bin.jar
CLASSPATH=$CLASSPATH:~/jars/postgresql-7.3.2.jar
export CLASSPATH
```

`analyse.jar` enthält die eigentliche Anwendung. `log4j-1.2.8.jar` enthält die log4j-Bibliothek, einen verwendeten Logging Service [32]. Die anderen beiden Jars enthalten die JDBC-Treiber [27] für die verwendeten Datenbanken.

Für die Verwendung wird ein Arbeitsverzeichnis benötigt. In diesem Verzeichnis wird u. a. eine Log-Datei erzeugt, es sind also Schreibrechte für den aktuellen Nutzer erforderlich. Das Default-Verzeichnis ist `~/analyse`, dies kann aber angepasst werden.

Für die einfache Verwendung gibt es ein Shellscript `analyse.sh` auf der CD im Verzeichnis `bin`. Dieses Shellscript wechselt in das Arbeitsverzeichnis und ruft die Java-Klasse `Analyse` auf. Es verfügt über zwei Konfigurationsparameter, welche durch Bearbeiten der Datei angepasst werden können: Das Arbeitsverzeichnis und der Name der Property-Datei mit der Datenbank-Konfiguration.

Am besten wird das Script zusammen mit den anderen aus dem Verzeichnis `bin` von der CD in ein Verzeichnis kopiert, das im „PATH“ enthalten ist.

Die Konfiguration des Datenbankzugangs erfolgt in einer Properties-Datei. Dazu gibt es das Muster `jars/muster_db.properties`, das in das Arbeitsverzeichnis kopiert und angepasst werden sollte.

A.1.3. Webbasierte Werkzeuge

Der webbasierte Teil der Werkzeuge setzt einen Webserver und eine PHP-Installation mit PostgreSQL- und MySQL-Unterstützung voraus. Getestet wurde mit dem Apache-Webserver 2.0.48 und PHP 4.3.4. Funktionieren sollte jeder Webserver mit PHP4-Unterstützung.

Die PHP-Skripte aus dem Verzeichnis `web` auf der CD können in ein beliebiges

Verzeichnis kopiert werden, für das der Webserver konfiguriert ist. Einige der PHP-Scripts benötigen eine (beiliegende) Template-Engine [25]. Beim Kopieren muss darauf geachtet werden, dass auch die Unterverzeichnisse mit kopiert werden.

Die verwendete Datenbank und die Zugangskennung wird in der Datei `var.inc.php` konfiguriert.

A.2. Bedienung

Die webbasierten Werkzeuge werden über einen herkömmlichen Webbrowser verwendet. Die anderen Werkzeuge werden direkt über die Kommandozeile aufgerufen. Für das Java-Package existiert zusätzlich ein Shell-Script (`analyse.sh`), das den Aufruf vereinfacht.

A.2.1. Plaintext-Konvertierung

Für die Konvertierung gibt es zwei Shellscripts: `convweb_file.sh` und `convweb_all.sh` (vgl. Abschnitt 3.4.1 auf Seite 30). Diese beiden Scripts liegen auf der CD im Verzeichnis `bin`. Um sie auszuführen, muss nur das Verzeichnis dem Such-Pfad (`PATH`) hinzugefügt oder die Dateien in ein Verzeichnis kopiert werden, das im Such-Pfad enthalten ist.

`convweb_file.sh` konvertiert einzelne Dateien. Die zu konvertierende Datei wird mit dem Argument `-file` übergeben. Mit dem Argument `-out` wird eine Datei angegeben, welche die konvertierte Plaintext-Ausgabe aufnimmt.

`convweb_all.sh` wird in dem Verzeichnis gestartet, in dem sich die heruntergeladene Website befindet. Es konvertiert alle Dateien (auch in Unterverzeich-

nissen) via `convweb_file.sh`. Die Ausgabe findet sich danach in einer Datei mit dem Präfix `conv_`, dem Datum und der Uhrzeit und dem Suffix `.txt`.

A.2.2. Java-Package

Das Java-Package läßt sich direkt über `java de.mairif.suchoptimierung.Analyse` aufrufen. Dieser Befehl ist bereits recht lang und zusätzlich muss noch über eine Option das Db-Property-File übergeben werden. Wesentlich kürzer ist also der Aufruf über `analyse.sh`.

Ein Aufruf ohne Parameter zeigt alle möglichen Optionen. Wobei die Option `-dbp` bei der Verwendung von `analyse.sh` weggelassen werden kann.

Webserver-Log-Dateien einlesen

Mit der Option `-read_weblog` wird eine Log-Datei eines Web-Servers im Common Logfile Format von STDIN eingelesen, die enthaltenen Suchanfragen extrahiert und in der Datenbank gespeichert.

Das Shellscript `read_weblogs.sh` kapselt das für mehrere Dateien. Es liest nacheinander alle Dateien im aktuellen Verzeichnis, welche die Dateiendung „.log“ haben ein.

Datenbank aktualisieren

Die Option `-update_db` veranlasst das Programm

- Mehrwortbegriffe unter den Such-Strings zu markieren,
- Suchanfragen zu zählen und

- Such-Strings in der ConceptComposer-Datenbank zu suchen.

Bevor diese Aktion ausgeführt wird, muss die ConceptComposer-Datenbank also bereits erstellt worden sein.

Hitliste konvertieren

Wenn die Suchanfragen in die Datenbank eingelesen wurden und `-update_db` bereits ausgeführt wurde, kann mit `-convert_hitliste` die Hitliste aus der ConceptComposer-Datenbank in die Projekt-Datenbank konvertiert werden.

Dabei wird aus den Werten der Wortliste die Hitliste errechnet, diese wird in die Projekt-Datenbank kopiert und die einzelnen Begriffe werden mit den Suchbegriffen verglichen.

Mehrwort-Nomen suchen

Mehrwort-Nomen (vgl. Abschnitt 3.6.3 auf Seite 51) können aus Plaintext-Dokumenten oder aus den Sätzen der ConceptComposer-Datenbank extrahiert werden. Entsprechend wird entweder die Option `-suche_mehrwort_nomen` mit Angabe einer Datei oder die Option `-suche_mehrwort_nomen_indb` verwendet. Anstelle des Dateinamens kann auch ein Minus (`-`) angegeben werden, um den Text von STDIN zu lesen.

Beide Varianten haben die gemeinsame Option `-tw`. Wenn diese angegeben wird, werden auch alle Teilkombinationen erkannt (vgl. Abschnitt 3.6.3 auf Seite 57).

Synonyme zu Fachbegriffen generieren

Die Option `-generate_synonyms` ruft die Methode `action_generate_synonyms()` der Klasse Analyse auf. In dieser werden mit Hilfe des `DummySynonymGen` Dummy-Synonyme generiert.

Um hier brauchbare Ergebnisse zu erzielen, muss noch ein vernünftiger Synonym-Generator implementiert (vgl. Abschnitt 3.8 auf Seite 66) und die Methode `action_generate_synonyms()` angepasst werden.

Abfragen

Hier werden ebenfalls einige Abfragen angeboten. Die Abfragen unterstützen alle Argumente, die das Ausgabeformat beeinflussen. So bewirkt kein Argument oder die Angabe von `-plain` eine Plaintextausgabe, während `-tex` Ausgaben erzeugt, die für die Einbettung in TEX- bzw. LATEX-Dokumente vorbereitet sind.

Einige Abfragen verfügen zusätzlich über das Argument `-l`, das die Anzahl der Datensätze auf die gewünschte Anzahl einschränkt.

Folgende Abfragen werden unterstützt:

Personen-Namen Liest Personen-Namen aus der Tabelle `person` der MySQL-Datenbank. Diese Tabelle wird vom NameRec erzeugt (vgl. Abschnitt 3.6.4 auf Seite 59).

Ämter Sucht in den Sätzen nach Mustern für Ämter (vgl. Abschnitt 3.6.2 auf Seite 49).

Mehrwortbegriffe Alle gewonnenen Mehrwortbegriffe (alle gültigen) aus der Tabelle `mehrwortbegriffe` aus der Projekt-Datenbank werden ausgegeben.

Fachbegriffe, die selten gesucht werden Es werden die Fachbegriffe angezeigt, die nicht häufiger als fünf mal gesucht wurden und nicht als ungültig markiert sind (vgl. Abschnitt 3.7.2 auf Seite 65).

Suchbegriffe, die nicht gefunden werden Hier werden alle Suchbegriffe ausgegeben, die bei einer Volltextsuche (Suche in den Sätzen) nicht gefunden wurden (vgl. Abschnitt 3.5.2 auf Seite 40).

A.2.3. Web-Schnittstelle

Die URL der Web-Schnittstelle ist abhängig von der Konfiguration des verwendeten Webservers. Wurden die PHP-Skripte in das Standard-Verzeichnis des Web-Servers kopiert, reicht evtl. bereits der Aufruf von „http://localhost/“.

Es gibt eine Index-Datei mit Links für den Aufruf der einzelnen Werkzeuge.

Mehrwortbegriffe bearbeiten

Hiermit können die Mehrwortbegriffe, die als aufeinander folgende Nomen gefunden wurden (vgl. Abschnitt 3.6.3 auf Seite 51 und Abschnitt A.2.2 auf Seite 80), bearbeitet werden.

Man kann für jeden Kandidaten wählen, ob der Mehrwortbegriff korrekt ist, nicht korrekt ist oder ihn als ungeprüft belassen.

Fachbegriffe, die nicht gesucht werden

Hierbei handelt es sich um Abfrage und Werkzeug zugleich. Es werden die Fachbegriffe, die selten oder nicht gesucht werden angezeigt. Gleichzeitig hat man die Möglichkeit ungültige Mehrwortbegriffe als solche zu markieren, um sie zukünftig ausblenden zu können.

Synonyme bearbeiten

Errechnete Synonyme können hier manuell bearbeitet werden (vgl. Abschnitt 3.8 auf Seite 66 und Abschnitt A.2.2 auf Seite 81).

Zu jedem Fachbegriff werden die gefundenen Synonyme angezeigt. Für diese kann jeweils gewählt werden, ob es korrekt ist, nicht korrekt ist oder als ungeprüft belassen werden soll.

Abfragen

Es existieren verschiedene Abfragen, die sich anhand verschiedener Parameter steuern lassen.

Suchbegriffe, die nicht gefunden werden Zeigt jene Suchbegriffe, die in den Dokumenten der Website nicht gefunden wurden (vgl. Abschnitt 3.5.2 auf Seite 40). Zu Vergleichszwecken werden die Ergebnisse aus der Suche in der Wortliste und in den Beispielsätzen berücksichtigt.

Fachbegriffe, die selten gesucht werden Es werden die Fachbegriffe angezeigt, die nicht häufiger als ein gewisser Schwellwert gesucht wurden (vgl.

Abschnitt 3.7.2 auf Seite 65). Der Schwellwert kann vom Nutzer bestimmt werden.

relevante Mehrwortbegriffe, nach denen gesucht wurde Extrahiert Mehrwortbegriffe aus den Such-Strings (vgl. Abschnitt 3.6.1 auf Seite 47). Der Schwellwert legt einen Mindestwert für die Häufigkeit des Such-Strings fest.

Kollokationen Dies ist eine Abfrage der signifikanten Nachbarschaftskollokationen in Hinblick auf eventuelle Mehrwortbegriffe (vgl. Abschnitt 3.6.5 auf Seite 60). Dabei werden die Wörter der Kollokationen zusammengesetzt und in der Datenbank überprüft, ob diese Kombination bereits bekannt ist (aus den aufeinander folgenden Nomen) und ob es sich um einen Mehrwortbegriff handelt. Die Kandidaten, die noch unbekannt sind, also die Kandidaten, die für den Nutzer interessant sind, werden fett gedruckt. Es ist eine Liste von Ausnahmen hinterlegt, welche denen in Tabelle 3.8 (Seite 53) entspricht. Diese Ausnahmen lassen sich ausblenden.

Mehrfach-Kollokationen Diese Abfrage ist der vorherigen sehr ähnlich. Hier werden jedoch zwei Kollokationen miteinander verknüpft (vgl. Abschnitt 3.6.5 auf Seite 61). Um diese Abfrage ausführen zu können ist eine zusätzlich Tabelle in der MySQL-Datenbank nötig: `pm_kollok_nb3`. Diese Tabelle und der SQL-Befehl, um diese zu füllen, findet man in der Datei `db/pm_kollok_nb3.sql` auf der CD.

Literaturverzeichnis

[1] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier: *Modern Information Retrieval*. Addison Wesley Longman, 1999. – ISBN 0-201-39829-X

[2] Baeza-Yates, Ricardo ; Ribeiro-Neto, Berthier: *Modern Information Retrieval*. Kap. Query Operations. Siehe [1]. – ISBN 0-201-39829-X

[3] *BASH*. – URL `http://www.gnu.org/software/bash/bash.html`. – Zugriffsdatum: 2004-08-11

[4] Berners-Lee, T. ; Masinter, L. ; McCahill, M.: *RFC1738: Uniform Resource Locators (URL)*. Dezember 1994. – URL `http://rfc.net/rfc1738.html`. – Zugriffsdatum: 2004-08-01

[5] Biemann, Christian: *Extraktion von semantischen Relationen aus natürlichsprachlichem Text mit Hilfe von maschinellem Lernen*, Institut für Informatik der Universität Leipzig, Diplomarbeit, September 2002

[6] Biemann, Christian: Extraktion semantischer Relationen aus natürlichsprachlichem Text mit Hilfe von maschinellem Lernen. In: Seewald-Heeg, Uta (Hrsg.): *Sprachtechnologie für die multilinguale Kommunikati-*

on, Beiträge der GLDV-Frühjahrstagung 2003. Sankt Augustin : Gardez!-Verlag, 2003

[7] Biemann, Christian ; Quasthoff, Uwe ; Böhm, Karsten ; Wolff, Christian: Automatic discovery and Aggregation of Compound Names for the Use in Knowledge Representations. In: *Journal of Universal Computer Science (JUCS)* 9 (2003), Juni, Nr. 6, S. 530–541

[8] Bordag, Stefan: *Vererbungsalgorithmen von semantischen Eigenschaften auf Assoziationsgraphen und deren Nutzung zur Klassifikation von natürlichsprachlichen Daten*. Leipzig, Institut für Informatik der Universität Leipzig, Diplomarbeit, Juni 2002

[9] Bordag, Stefan: Sentence Co-occurrences as Small-World Graphs: A solution to Automatic Lexical Disambiguation. In: Gelbukh, A. (Hrsg.): *CICLing 2003, LNCS 2588*, Springer-Verlag Berlin Heidelberg, 2003, S. 329–332

[10] Böhm, Karsten ; Heyer, Gerhard ; Quasthoff, Uwe ; Wolff, Christian: Topic Map Generation Using Text Mining. In: *Journal of Universal Computer Science (JUCS)* 8 (2002), Nr. 6, S. 623–633. – ISSN 0948-69x

[11] *The Common Logfile Format*. – URL `http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format`. – Zugriffsdatum: 2004-08-09

[12] *Deutscher Wortschatz*. – URL `http://www.wortschatz.uni-leipzig.de`. – Zugriffsdatum: 2004-08-03

[13] Dublin Core Metadata Initiative: *Dublin Core Metadata Initiative (DCMI) Home Page.* – URL `http://dublincore.org`. – Zugriffsdatum: 2004-03-03

[14] Faulstich, Lukas C. ; Quasthoff, Uwe ; Schmidt, Fabian ; Wolff, Christian: Concept Extractor - Ein flexibler und domänenspezifischer Web Service zur Beschlagwortung von Texten. In: [16], S. 165–180. – ISBN 3-89669-759-5

[15] Glyph & Cog: *Xpdf: A PDF Viewer for X.* 1996. – URL `http://www.foolabs.com/xpdf/`. – Zugriffsdatum: 2004-02-26

[16] Hammwöhner, Rainer (Hrsg.) ; Wolff, Christian (Hrsg.) ; Womser-Hacker, Christa (Hrsg.) ; Universität Regensburg (Veranst.): *Information und Mobilität - Proc. 8. Internationales Symposium für Informationswissenschaft.* Bd. *40.* Konstanz, Oktober 2002. (Schriften zur Informationswissenschaft). – ISBN 3-89669-759-5

[17] Hausmann, Franz-Josef: Kollokationen im deutschen Wörterbuch: Ein Beitrag zur Theorie des lexikographischen Beispiels. In: Bergenholtz, Henning (Hrsg.) ; Mugdan, Joachim (Hrsg.): *Lexikographie und Grammatik: Akten des Essener Kolloquiums zur Grammatik im Worterbuch 1984 (Lexicographica 3).* Tübingen : Niemeyer, 1985, S. 118–129

[18] Heyer, Gerhard ; Läuter, Martin ; Quasthoff, Uwe ; Wittig, Thomas ; Wolff, Christian: Learning Relations using Collocations. In: Maedche, A. (Hrsg.) ; Staab, S. (Hrsg.) ; Nedellec, C. (Hrsg.) ; Hovy, E. (Hrsg.): *IJCAI Workshop on Ontology Learning.* Seattle/ WA, August 2001

[19] Heyer, Gerhard ; Quasthoff, Uwe ; Wittig, Thomas: *Wissensrohstoff Text*. Kap. Kollokationen I. Siehe [21]. – [to appear]. – ISBN 3937137300

[20] Heyer, Gerhard ; Quasthoff, Uwe ; Wittig, Thomas: *Wissensrohstoff Text*. Kap. Text Mining. Siehe [21]. – [to appear]. – ISBN 3937137300

[21] Heyer, Gerhard ; Quasthoff, Uwe ; Wittig, Thomas: *Wissensrohstoff Text*. W3l, August 2004. – [to appear]. – ISBN 3937137300

[22] Heyer, Gerhard ; Quasthoff, Uwe ; Wolff, Christian: Automatic Analysis of Large Text Corpora - A Contribution to Structuring Web Communities. In: Unger, Herwig (Hrsg.) ; Böhme, Thomas (Hrsg.) ; Mikler, Armin (Hrsg.): *Innovative Internet Computing Systems. Proc. Second International Workshop* Bd. 2346. Kühlungsborn : Springer, Juni 2002, S. 15–36. – ISBN 3-540-43790-8

[23] Heyer, Gerhard ; Quasthoff, Uwe ; Wolff, Christian: Information Extraction from Text Corpora: Using Filters on Collocation Sets. In: *Proc. LREC-2002. Third International Conference on Language Resources and Evaluation* Bd. 3. Las Palmas, Mai 2002, S. 241–246. – ISBN 2-9517408-0-8

[24] Heyer, Gerhard ; Quasthoff, Uwe ; Wolff, Christian: Möglichkeiten und Verfahren zur automatischen Gewinnung von Fachbegriffen aus Texten. In: Bullinger, Hans-Jörg (Hrsg.) ; Weisbecker, Anette (Hrsg.): *Content Management - Digitale Inhalte als Bausteine einer vernetzten Welt*. Stuttgart : Fraunhofer IRBN Verlag, Juni 2002, S. 43–49. – ISBN 3-8167-6155-0

[25] Wendel, Ulf: *Integrated Template [Extension] - IT[X].* – URL `http://www.ulf-wendel.de/projekte/itx/`. – Zugriffsdatum: 2004-08-12

[26] Ishikawa, Hiroshi ; Ohta, Manabu ; Watanabe, Takuya ; Yokoyama, Shohei ; Katayama, Kaoru: Toward Active Web Usage Mining for Page Recommendation and Restructuring. In: *Proceedings of I-KNOW '03*. Graz, Juli 2003, S. 492–499

[27] *JDBC Overview.* – URL `http://java.sun.com/products/jdbc/overview.html`. – Zugriffsdatum: 2004-08-10

[28] Kaiser, Alexander: *Computer-unterstütztes Indexieren in Intelligenten Information Retrieval Systemen. Ein Relevanz-Feedback orientierter Ansatz zur Informationserschließung in unformatierten Datenbanken*, Wirtschaftsuniversität Wien, Dissertation, 1993

[29] Kubicek, Herbert ; Rizvi, Sylvia (Hrsg.) ; Klaeren, Herbert (Hrsg.): *Möglichkeiten und Gefahren der „Informationsgesellschaft“*. Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, 1999

[30] Lachowicz, Dom ; McNamara, Caolán: *wvWare.* 2000. – URL `http://wvware.sourceforge.net/`. – Zugriffsdatum: 2004-02-26

[31] Lezius, Wolfgang: Automatische Extrahierung idiomatischer Bigramme aus Textkorpora. In: Rapp, R. (Hrsg.): *Tagungsband des Linguistischen Kolloquiums.* Germersheim, Germany, 1999

[32] *Logging Services.* – URL `http://logging.apache.org/`. – Zugriffsdatum: 2004-08-11

[33] Läuter, Martin ; Quasthoff, Uwe: Linguistisches Wissen zur Textaufbereitung. In: *GLDV-Workshop „Werkzeuge zur automatischen Analyse und Verarbeitung von Texten“*. Trier, September 2001

[34] Montulli, Lou ; Grobe, Michael ; Rezac, Charles: *Lynx source distribution*. – URL `http://lynx.isc.org/`. – Zugriffsdatum: 2004-02-26

[35] *MySQL Reference Manual*. – URL `http://dev.mysql.com/doc/mysql/`. – Zugriffsdatum: 2004-08-02

[36] *PostgreSQL Documentation*. – URL `http://www.postgresql.org/docs/`. – Zugriffsdatum: 2004-08-10

[37] Quasthoff, Uwe: Projekt Der Deutsche Wortschatz. In: Heyer, Gerhard (Hrsg.) ; Wolff, Christian (Hrsg.): *Linguistik und neue Medien*, Deutscher Universitätsverlag, 1998, S. 93 – 99

[38] Quasthoff, Uwe ; Richter, Matthias ; Wolff, Christian: Wörter des Tages - tagesaktuelle wissensbasierte Analyse und Visualisierung von Zeitungen und Newsdiensten. In: [16], S. 369–372. – ISBN 3-89669-759-5

[39] Quasthoff, Uwe ; Wolff, Christian: Aiding Web Searches by Statistical Classification Tools / Institut für Informatik der Universität Leipzig. 2000. – Forschungsbericht

[40] Quasthoff, Uwe ; Wolff, Christian: The Poisson Collocation Measure and its Applications. In: *Second International Workshop on Computational Approaches to Collocations*. Wien, Juli 2002. – [to appear]

[41] Schmalenbach, K.: *Begriffe der Terminologie*. 2002. – URL `http://www.dit-online.com/termi/main.htm`. – Zugriffsdatum: 2004-08-03

[42] Schmidt, Fabian: *Automatische Ermittlung semantischer Zusammenhange lexikalischer Einheiten und deren graphische Darstellung.* Kap. 1.2. Kollokationen. Siehe [44]

[43] Schmidt, Fabian: *Automatische Ermittlung semantischer Zusammenhange lexikalischer Einheiten und deren graphische Darstellung.* Kap. 1.3. Projekt Deutscher Wortschatz. Siehe [44]

[44] Schmidt, Fabian: *Automatische Ermittlung semantischer Zusammenhange lexikalischer Einheiten und deren graphische Darstellung*, Institut für Informatik der Universität Leipzig, Diplomarbeit, April 1999

[45] *The Soundex Indexing System.* – URL `http://www.archives.gov/research_room/genealogy/census/soundex.html`. – Zugriffsdatum: 2004-08-05

[46] Witschel, Hans F.: *Text, Wörter, Morpheme - Möglichkeiten einer automatischen Terminologie-Extraktion.* Kap. 1.4.2 Mehrwortterme. Siehe [47]

[47] Witschel, Hans F.: *Text, Wörter, Morpheme - Möglichkeiten einer automatischen Terminologie-Extraktion*, Institut für Informatik der Universität Leipzig, Diplomarbeit, März 2004

[48] Zipf, G. K.: *Relative Frequency as a Determinant of Phonetic Change.* 1929. – Wiederabdruck in den Harvard Studies in Classical Philology, Volume XL

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, 2004-08-12

Patrick Mairif