Bachelorarbeit

Semantische Suche in Zeitungstexten

Niklas Meinzer

Albert-Ludwigs-Universität Freiburg im Breisgau
Technische Fakultät
Institut für Informatik
Lehrstuhl für Algorithmen und Datenstrukturen

Bearbeitungszeitraum

31. 05. 2011 – 31. 08. 2011

Gutachterin

Prof. Dr. Hannah Bast

Betreuer

Prof. Dr. Hannah Bast
Björn Buchold

Danksagung

Ich möchte mich bei einigen Personen bedanken, die bei mich bei dieser Arbeit unterstützt haben.
Großer Dank gilt zunächst meiner Betreuerin und Gutachterin Prof. Dr. Hannah Bast sowie meinem Betreuer Björn Buchold, die beide außerordentlich viel Zeit für mich hatten und mir bei vielen großen und kleinen Problemen geholfen haben.
Vielen Dank auch an Ina Baumgarten, die mit mir zusammen an diesem Projekt gearbeitet hat und ebenfalls darüber ihre Bachelorarbeit schreiben wird. Die Zusammenarbeit gelang stets problemlos und hat viel Spaß gemacht.
Wer etwas von der Syntax von Programmiersprachen versteht, hat noch lange keine Ahnung von natürlichen Sprachen. Zu dieser Erkenntnis verhalfen mir Eva Höfflin und Antony Neu, die sich die Zeit nahmen diese Arbeit korrekturzulesen.

Erklärung

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum	Unterschrift

Das in dieser Arbeit beschriebene Projekt wurde in Zusammenarbeit mit Ina Baumgarten entwickelt, die darüber ebenfalls ihre Bachelorarbeit verfasst hat. Es wurden jedoch unterschiedliche Teilbereiche jeweils selbstständig bearbeitet. Alle in dieser Arbeit beschriebenen Themen wurden von mir bearbeitet. Sollten der Vollständigkeit halber von Ina Baumgarten bearbeitete Themen erwähnt werden, wird dies im Text explizit gekennzeichnet.

Inhaltsverzeichnis

Abstract

Im Gegensatz zur klassischen Volltextsuche, bei der wörtliche Vorkommen der Suchanfrage in den durchsuchten Dokumenten gefunden werden, ist die Semantische Suche ein Verfahren, bei dem die Suchanfrage zunächst auf ihre Bedeutung untersucht wird. Ziel ist es, dem Benutzer zu ermöglichen, Anfragen in einer natürlicheren Form zu verfassen und abstraktere Formulierungen zu wählen. Broccoli ist eine existierende semantische Suchmaschine, die zum Beispiel die online Enzyklopädie Wikipedia durchsucht. Wir präsentieren eine semantische Suchmaschine für deutsche Nachrichtentexte, die auf Broccoli aufbaut. Das zentrale Problem hierbei ist die Entitätserkennung in deutschen Texten. Weitere Teilprobleme sind die automatisierte Datenbeschaffung von Online-Ressourcen und das Erstellen der Eingabedaten für Broccoli. In der anschließenden Evaluation der Entitätserkennung erreichen wir konstant sehr gute Werte.

1 Einleitung und Motivation

Mit zunehmender Bedeutung des Internets im alltäglichen Leben und stetig steigender Informationsmenge ist es für die Benutzer immer wichtiger, effektiv navigieren zu können. Suchmaschinen spielen dabei seit jeher eine entscheidende Rolle und sind aus der Online-Welt nicht mehr wegzudenken.
Die meist verbreitetste Form der Suche ist dabei die klassische Volltextsuche, die eine Liste von Suchbegriffen entgegennimmt und Resultate liefert, in denen diese Begriffe vorkommen. Die Bedeutung der Suchbegriffe hat also keinen Einfluss auf die Suche. Ein alternativer Ansatz ist die so genannte Semantische Suche.

1.1 Semantische Suche

Bei der Semantischen Suche wird versucht, die Bedeutung der Suchanfrage zu verstehen. Dies kann auf verschiedene Weise geschehen: So kann zum Beispiel versucht werden, ganze Sätze zu interpretieren, um natürlich gestellte Fragen wie "Wie oft war Bayern München deutscher Meister?" beantworten zu können. Einen solchen Ansatz verfolgt Wolfram Alpha[1].
Eine andere Möglichkeit besteht darin, einzelne Begriffe, die zum Beispiel eine Kategorie von Personen beschreiben, aufzuschlüsseln, um dann auch nach Personen aus dieser Kategorie zu suchen.

Möchte man etwa Informationen über Verletzungen US-amerikanischer Sportler im allgemeinen bekommen, gestaltet sich das mit Volltextsuche schwierig, denn "US-Sportler" ist eine Kategorie von Personen und wird so in den durchsuchten Texten selten vorkommen. Diese Arbeit baut auf der semantischen Suchmaschine "Broccoli" auf. Als semantische Suchmaschine kann Broccoli mit solchen Kategorien umgehen und sucht dann zum Beispiel auch nach "Kobe Bryant", der unter die Kategorie "US-Sportler" fällt (siehe **??**).

[1] www.wolframalpha.com/

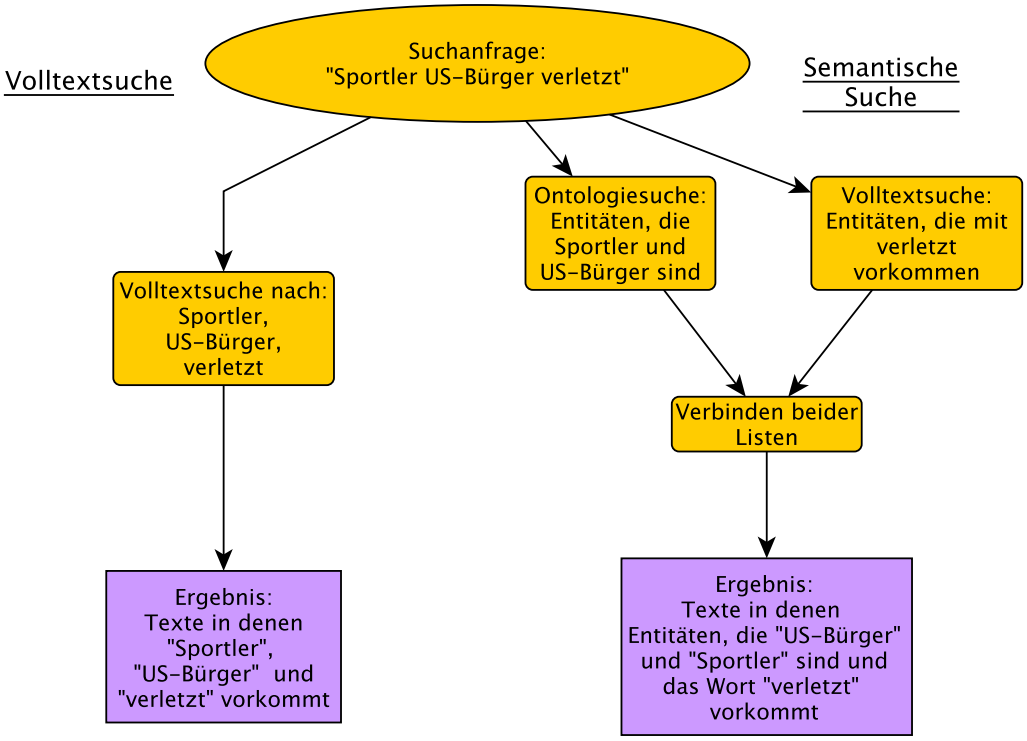

Abbildung 1.1: Während bei Volltextsuche die Suchbegriffe direkt im Suchindex gesucht werden, werden sie bei semantischer Suche nach Bedeutung aufgeschlüsselt.

Für diese Kategorisierung von Begriffen benötigt man eine möglichst umfangreiche Sammlung von Hintergrundwissen, eine Ontologie, die Fakten über Entitäten enthält. Also zum Beispiel “Kobe Bryant ist ein US-Bürger und Sportler” oder “Angela Merkel lebt in Berlin”.
Um dieses Wissen bei der Suche einsetzen zu können, müssen in einem Vorverarbeitungsschritt die Einträge dieser Ontologie in den zu durchsuchenden Texten erkannt und für die Suchmaschine möglichst effizient nutzbar markiert werden.

1.2 Beitrag der Arbeit

Ziel dieser Arbeit ist es, eine semantische Suchmaschine für Nachrichtentexte in deutscher Sprache zu entwickeln, die auf das bereits vorhandene System Broccoli (siehe **??**) aufsetzt, welches derzeit nur Wikipedia durchsuchen kann.
Dabei werden frei verfügbare Nachrichtentexte aus dem Internet als Suchraum genutzt.

Das zentrale Problem ist die Entitätserkennung in den deutschen Texten.
Es wurden folgende Teilbereiche bearbeitet:

- **Datenbeschaffung** Die als Eingabe genutzten Daten stammen aus frei zugänglichen Nachrichtenportalen im Internet. Das automatische Durchsuchen dieser Portale und das Herunterladen der Daten wird von einem eigens dafür geschriebenen Webcrawler durchgeführt. Des Weiteren müssen die so gewonnenen Daten vorverarbeitet werden. Insbesondere wird der Artikelinhalt aus den HTML-Dokumenten extrahiert und in einem günstigen Format abgespeichert.
- **Entitätserkennung** Um in den heruntergeladenen Daten suchen zu können, müssen die der zugrundeliegenden Onthologie bekannten Begriffe in den Texten wiedergefunden werden. Dabei sucht zunächst ein mehrstufiger Algorithmus nach Entitäten, die dann anschließend mit der Ontologie verbunden werden. Da die Ontologie englischsprachig ist, wird dafür ein Mapping der Form <deutsche Entität> $\leftrightarrow$ <englische Entität> benötigt, das wir aus Wikipedia gewinnen.
- **Evaluation** Um die Qualität des System einschätzen zu können, bedarf es einer umfangreichen Evaluation. Das Hauptaugenmerk liegt dabei auf der Entitätserkennung.

1.3 Vorhandene Systeme

1.3.1 YAGO

YAGO ist eine Ontologie und Wissensdatenbank, die aus Wikipedia, sowie der Wortdatenbank "WordNet" abgeleitet ist. YAGO wird am Max-Planck-Institut für Informatik in Saarbrücken entwickelt und in dieser Arbeit als Ontologie für die semantische Suche verwendet.[**?**]

1.3.2 Broccoli

Unter dem Namen "Broccoli" wird an der Uni Freiburg eine semantische Suchmaschine entwickelt.[**?**] Derzeit existieren mehrere Versionen, die Wikipedia durchsuchen. Teile des Gesamtsystems sind dabei:

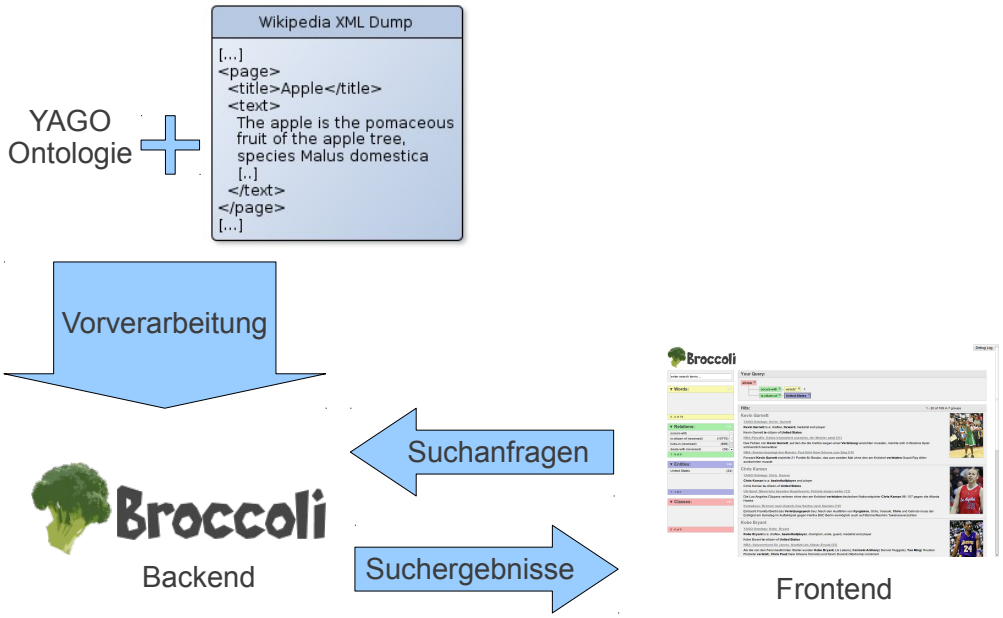

Abbildung 1.2: Broccoli verwendet die von Wikimedia bereitgestellten Wikipedia XML Dump Dateien als Eingabe, bereitet diese in einem Vorverarbeitungsschritt auf und stellt schließlich ein Webinterface zur Onlinesuche bereit.

- Das Vorverarbeitungssystem, das die von Wikipedia bereitgestellten XML-Dumps, also große XML-Dateien, die sämtliche Wikipediaartikel enthalten, verarbeitet und dabei die Entitäten aus der Yago Ontologie annotiert. Anschließend wird ein Suchindex erstellt, der dann von der Suchmaschine (dem

Backend) verwendet wird.

- Das Webfrontend oder User Interface, über das der Benutzer Suchanfragen zusammenstellen und abschicken kann. Die Suchergebnisse werden ebenfalls dort angezeigt.
- Das Backend, die eigentliche Suchmaschine, die Suchanfragen vom Frontend entgegennimmt, diese auf den Suchindex anwendet und die Resultate wieder an das Frontend zurückgibt.

Für die Entitätserkennung wird bei Broccoli die Tatsache ausgenutzt, dass sämtliche in Wikipedia (und damit mit großer Wahrscheinlichkeit auch in Yago) bekannte Entitäten innerhalb der Artikel als Verweis auf den dazugehörigen Artikel gekennzeichnet sind. Diese Annahme vereinfacht die Entitätserkennung enorm. Es müssen lediglich weitere Vorkommen der selben Entität innerhalb eines Artikels annotiert werden, da jeweils nur das erste Vorkommen pro Artikel als Verweis gekennzeichnet ist. Des Weiteren wird durch ein experimentelles Verfahren versucht, Pronomen den entsprechenden Entitäten zuzuordnen.

Nach der Entitätserkennung werden die Daten in ein maschinell schneller lesbares Binärformat gespeichert und komprimiert.

1.3.2.1 Broccoli User Interface

Über das Broccoli User Interface oder Webfrontend können Benutzer Suchanfragen erstellen, nachträglich bearbeiten sowie Suchergebnisse einsehen.[**?**] Da das in dieser Arbeit vorgestellte System das selbe interface benutzt, wird es im Folgenden kurz vorgestellt: In der oberen linken Ecke findet sich die Eingabemaske, über die Suchbegriffe eingegeben werden können. Dabei werden in den vier Boxen darunter Vorschläge angezeigt. Die Suchanfrage wird dann durch Eingeben aller gewünschten Begriffe zusammengesetzt. Die aktuelle Anfrage wird stets in der Box oben rechts durch eine Art Baumstruktur repräsentiert. Sowohl in den Vorschlagsboxen als auch in der Baumrepräsentation werden die vier verschiedenen Funktionen der Wörter farblich unterschieden:

- **Gelb - Words:** Gelbe Wörter werden gesucht ohne ihre Semantik aufzulösen. Eine Suche, die nur aus gelben Wörtern besteht, entspricht also einer normalen Volltextsuche.

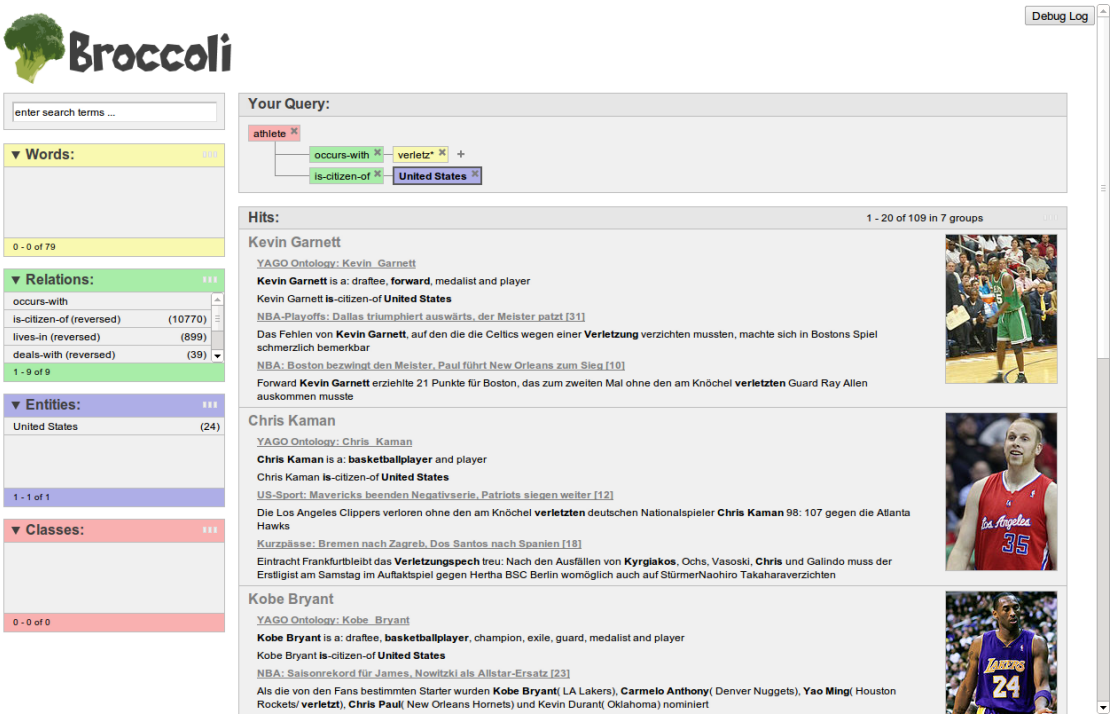

Abbildung 1.3: Eine Beispielsuchanfrage, bei der nach Artikeln über Sportler gesucht wird, die US-Bürger sind und im Kontext mit Wörtern stehen, die mit "verletz" anfangen.

- **Grün - Relations:** Relationen werden in Broccoli grün dargestellt. Mit Relationen können verschiedene Suchbegriffe semantisch miteinander verbunden werden. Beispiele für Relationen sind "is-citicen-of", wenn die Suche auf Bürger eines bestimmten Staates eingeschränkt werden soll oder "occurs-with", um zu erzwingen, dass zwei Wörter gemeinsam auftreten müssen.
- **Blau - Entities:** Entitäten werden blau dargestellt. Es werden nur noch Ergebnisse gefunden, in denen diese Entität vorkommt - wörtlich, in einer anderen Form oder als Pronomen.
- **Rot - Klassen:** Um nach ganzen Klassen von Entitäten zu suchen benötigt man die roten Einträge. Diese Einträge repräsentieren jeweils eine ganze Reihe von Entitäten, die dieser Klasse angehören.

Die Ergebnisse erscheinen in der großen zentralen Box, sortiert nach gefundenen Entitäten. Darunter erscheinen die Sätze aus den Dokumenten, in denen die Treffer gefunden wurden.

1.4 Beispielanfragen

Um den praktischen Einsatz der fertigen Suche zu demonstrieren werden im Folgenden ein paar Beispielanfragen demonstriert. Das User Interface steht unter http://stromboli.informatik.uni-freiburg.de:6222/BroccoliNewsSearch/ bereit.

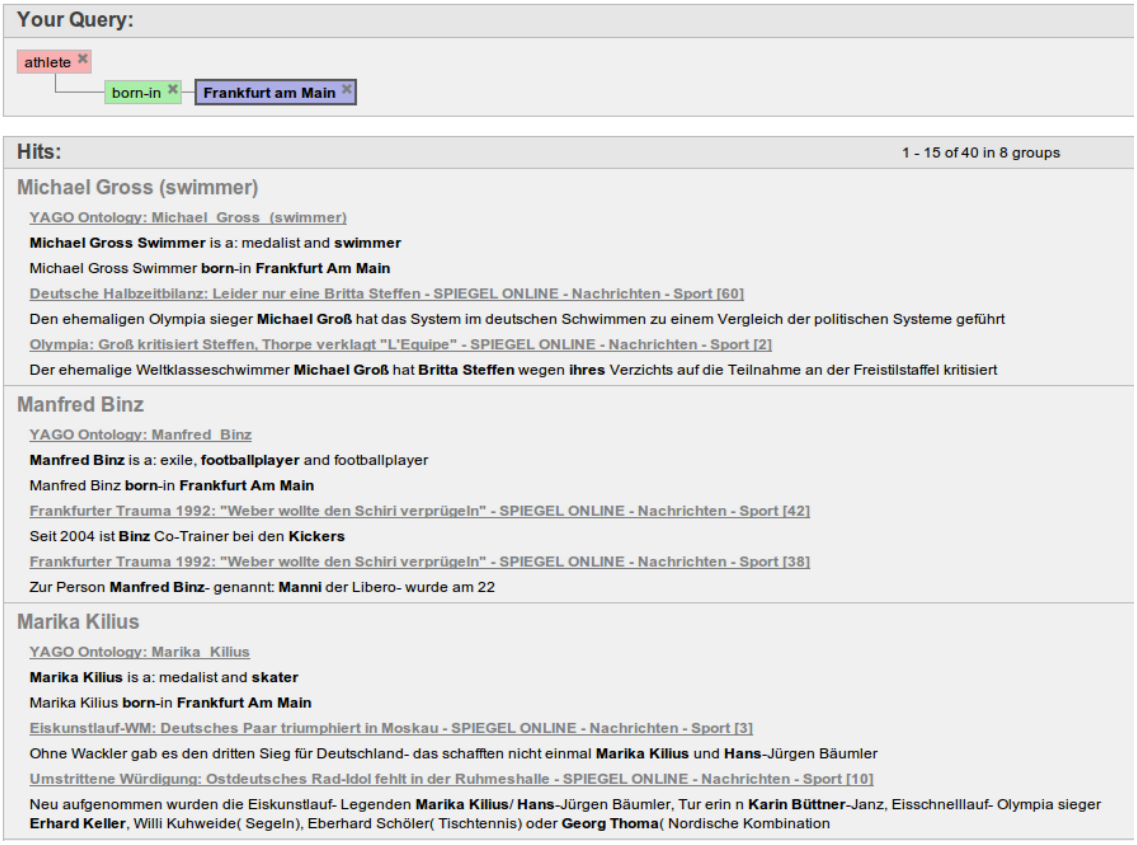

Abbildung 1.4: Artikel über Sportler, die in Frankfurt am Main geboren sind.

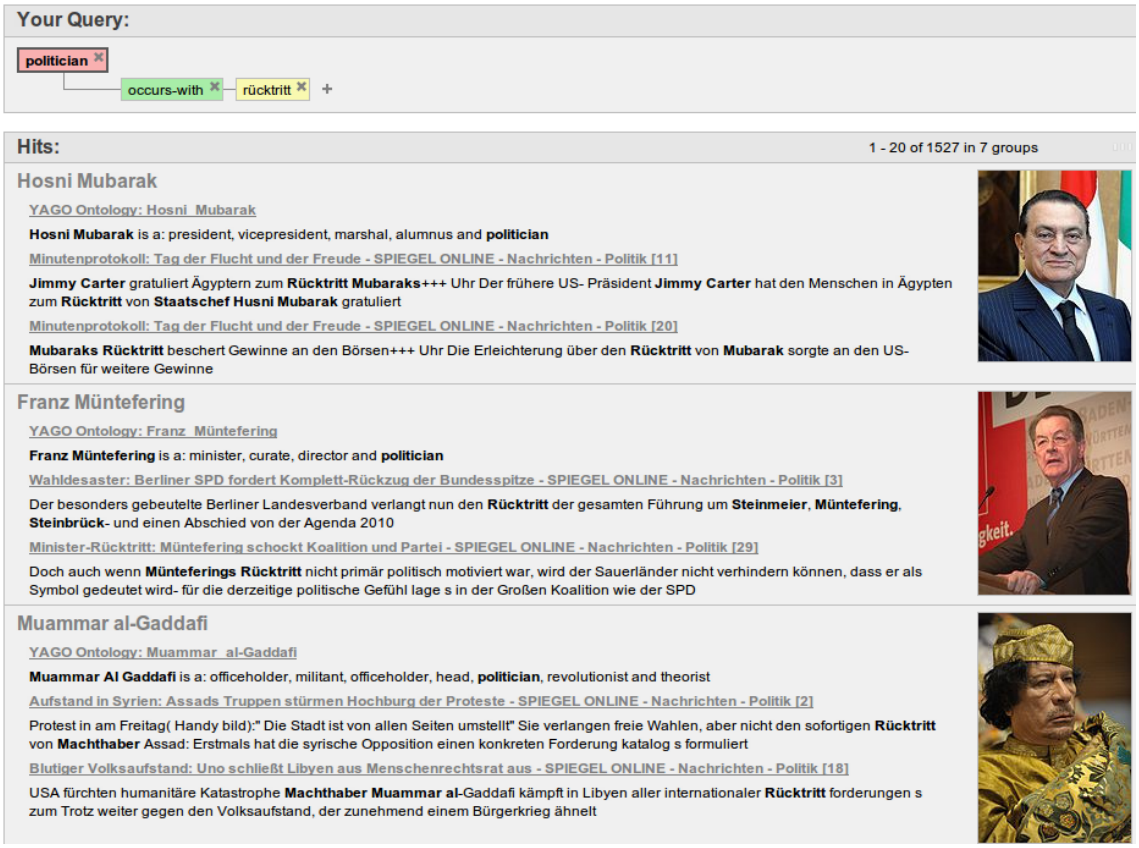

Abbildung 1.5: Politiker, die mit “Rücktritt” in Verbindung stehen.

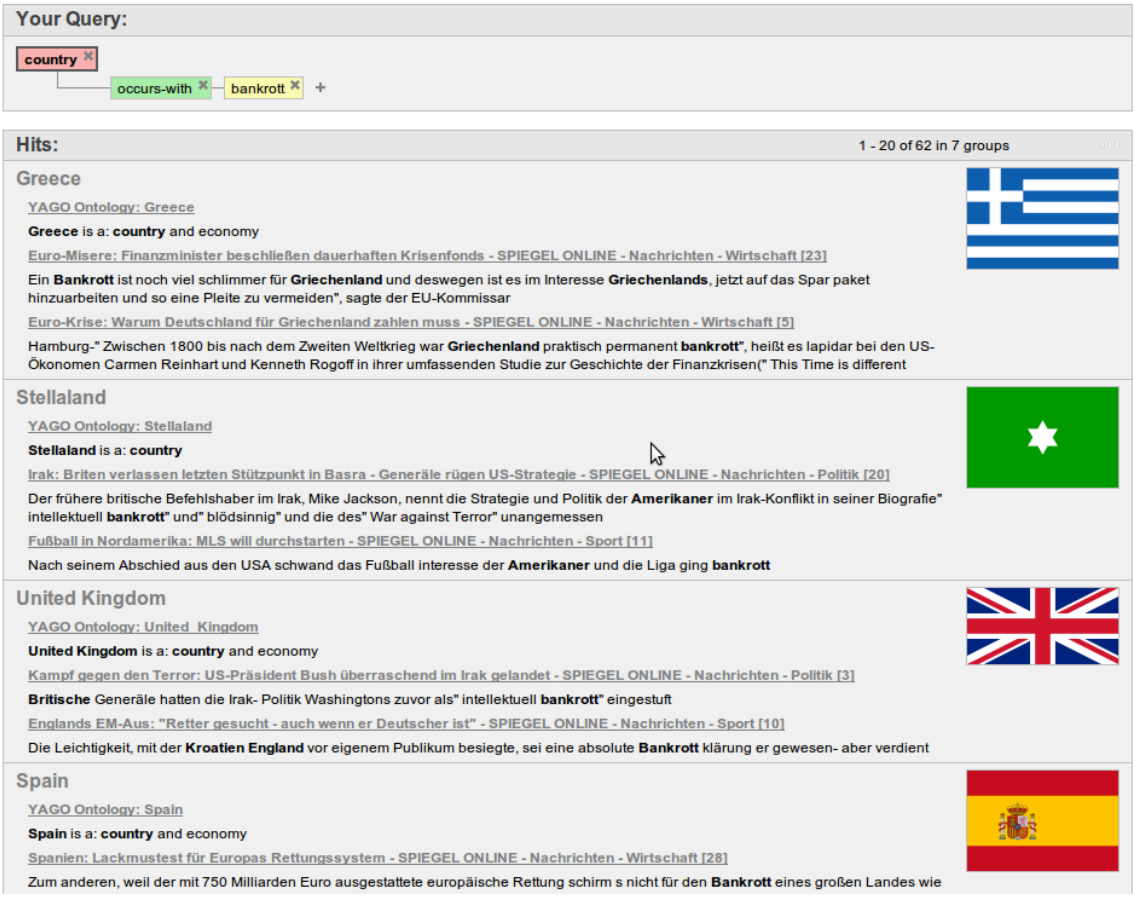

Abbildung 1.6: Artikel über Länder, die mit “Bankrott” in Verbindung stehen.

2 Eingabedaten

Als Suchraum sollen Nachrichtentexte in deutscher Sprache verwendet werden. Im deutschsprachigen Internet finden sich zahlreiche Onlineportale deutscher Tages- und Wochenzeitungen, auf denen solche Texte in großer Zahl frei zur Verfügung stehen.
Für die effiziente Aufbereitung müssen lokale Kopien der Daten zur Verfügung stehen.

2.1 Datenbeschaffung

Die einfachste Möglichkeit, einen Artikel von einem Newsportal herunterzuladen, besteht darin, die entsprechende Seite einfach per HTTP herunterzuladen und dann die gewünschten Daten zu extrahieren.
Für diesen Zweck benutzen wir einen eigens dafür in Python geschriebenen Webcrawler, also ein Programm, dass automatisiert im Web nach bestimmten Seiten sucht und diese verarbeitet.

2.1.1 Funktionsweise des Crawlers

Der von uns geschriebene Crawler soll bestimmte HTML Dokumente von den News-Portalen herunterladen, diese nach neuen interessanten Links durchsuchen, den Artikeltext extrahieren und in einem einfachen Format abspeichern.
Dabei gibt es zwei wichtige Aspekte:

Interessante Links identifizieren

Die meisten HTML Dokumente enthalten Links jeglicher Art, wie zum Beispiel solche zum Nachladen externer Ressourcen, Verweise auf externe Seiten und Verweise

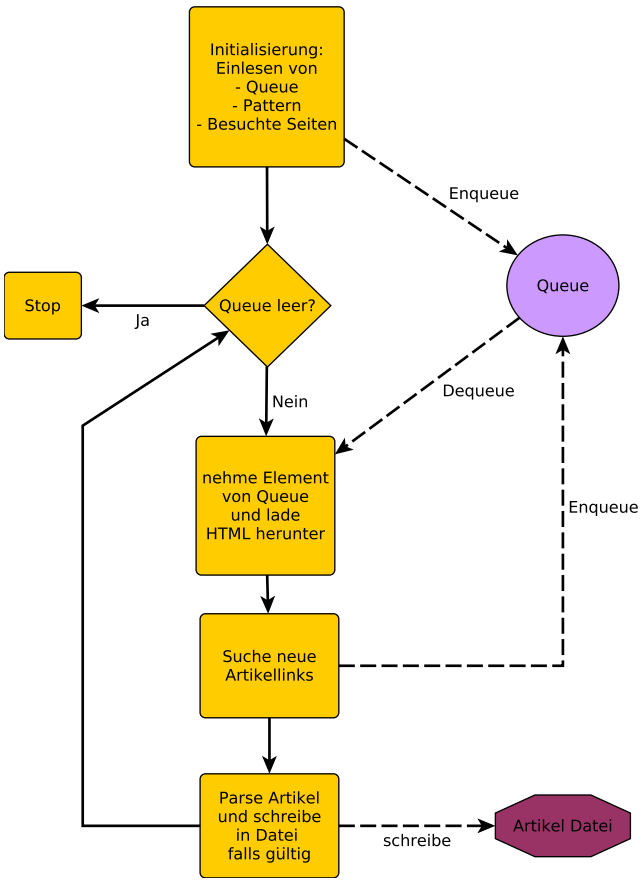

Abbildung 2.1: Die Funktionsweise des Crawlers skizziert in einem Flowchart.

auf interne Seiten. Letztere sind dabei die einzigen für den Crawler interessanten, denn möglicherweise verweisen sie auf einen weiteren Artikel.
Die Entscheidung, ob ein Link in die Warteschlange aufgenommen wird, hängt davon ab, ob er auf eines der fest einprogrammierten Mustern passt. Die von uns genutzten News-Portale haben für die Artikel spezielle Präfixe. Alle Artikel von Spiegel Online beginnen zum Beispiel mit *http://www.spiegel.de/<kategorie>/*, wobei *<kategorie>* die jeweiligen Ressorts sind (Politik, Wirtschaft, etc,...). Darüber hinaus lassen sich keine weiteren Informationen aus den Links ablesen. Interessant wäre zum Beispiel das Erscheinungsdatum, um zu alte Artikel vor dem Herunterladen ausschließen zu können.

Verarbeiten der heruntergeladenen Seiten

Bevor ein Artikel heruntergeladen wird, muss überprüft werden, ob er bereits besucht wurde. Der Crawler führt dazu eine Liste von bereits besuchten Seiten mit und vergleicht jeden neuen Link mit dieser.
Der heruntergeladene Artikel liegt zunächst als komplette HTML Seite vor. Für unsere weitere Arbeit sind wir lediglich an dem Artikeltext interessiert, wie ihn auch ein Webbrowser darstellen würde.
Ebenfalls muss entschieden werden, ob es sich überhaupt um einen "normalen" Artikel handelt, denn wie bereits erwähnt, können einige Spezialseiten nicht über den Link identifiziert werden. Dazu gehören unter anderem:

- Fotostrecken
- Übersichtsseiten
- Quizes

Es wird versucht diese unerwünschten Seiten anhand des Vorkommens von bestimmten Schlüsselphrasen zu identifizieren. Auf allen Spiegel Online Artikelseiten befindet sich ein Abschnitt, in dem Links zum Posten des Artikels in verschiedenen sozialen Netzwerken zur Verfügung gestellt werden. Die Spezialseiten enthalten keinen solchen Abschnitt. Der Satz *"Auf anderen Social Networks posten:"* ist fester Bestandteil dieser Abschnitte und kann somit dazu verwendet werden Artikel von Spezialseiten zu unterscheiden.

Für das eigentliche Parsen des HTML Dokuments verwenden wir "BeautifulSoup"[1], einen HTML Parser für Python. BeautifulSoup konvertiert HTML Dokumente zunächst in XML Dokumente, dabei werden zum Beispiel fehlende schließende Tags ergänzt. Anschließend kann auf das Dokument per DOM zugegriffen werden.
Um den Textinhalt herauszufiltern, werden zunächst alle <p>-Tags im DOM gesucht und dann rekursiv deren Inhalt, sowie der Inhalte der Kinder extrahiert. Bei manchen Portalen ist es nötig, nach einer bestimmten Schlüsselphrase aufzuhören, um Kommentare etc. zu ignorieren.

Die mit dieser Methode gewonnenen Daten können immer Fehler enthalten, da man aufgrund der Effizienz den Regelsatz möglichst klein und einfach halten möchte. Für die Zwecke dieser Arbeit ist es aber nicht entscheidend, ob sich einige fehlerhafte Artikel im Datensatz befinden.

[1] www.crummy.com/software/BeautifulSoup/

2.2 Deutsch-English Mapping

Die vorverarbeiteten Texte werden anschließend vom Broccoli weiterverarbeitet. Dieses System arbeitet mit der YAGO Onthologie[**?**], einer englischsprachigen Wissensdatenbank. Um es mit deutschsprachigen Texten nutzen zu können, ist es nötig, Vorkommen der englischsprachigen Einträge aus YAGO in den deutschen Texten zu erkennen. Beispielsweise muss "Deutschland" als Instanz der Entität "Germany" erkannt werden.

Für diese Aufgabe wird ein Wörterbuch benötigt. Allerdings enthält ein klassisches Wörterbuch in diesem Fall zu viele Informationen: Für die Entitätensuche sind Verben, Adjektive, Konjunktionen, etc. uninteressant, denn Entitäten sind ausschließlich Substantive und Eigennamen. Daher kommt in dieser Arbeit ein aus Wikipedia gewonnenes deutsch-englisch Mapping zum Einsatz.
Jede deutsche Wikipediaseite, die ein englisches Pendant hat, enthält einen Link auf diesen englischen Artikel innerhalb des Wikipediacodes.
Die Seite "Deutschland" enthält also den Link "[[en:Germany]]".
Anhand dieser Informationen wird der komplette Inhalt der Wikipedia durch ein Perlscript durchsucht und anschließend in einer separaten Datei gespeichert. Als Eingabe für dieses Script wird ein XML-Dokument, dass den kompletten Textinhalt der deutschen Wikipedia enthält, verwendet. Die Ausgabe sieht wie folgt aus:

Deutschland	Germany	250056
Deutsches Hygiene-Museum	German Hygiene Museum	8515
Deuce	Deuce	768

Die Zahlen in der dritten Spalte repräsentieren die Länge des dazugehörigen deutschen Artikels, die später in der Entitätserkennung verwendet werden.

Spezialfälle

Beim Parsen des Wikipedia Dumpfiles gibt es einige Sonderfälle zu beachten:

- **Redirects:** Da es für viele Dinge mehrere Begriffe und Schreibweisen gibt, existieren in Wikipedia sogenannte Redirects, also Einträge die auf einen bereits vorhandenen Eintrag verweisen. "USA" leitet zum Beispiel auf "Vereinigte Staaten" weiter.

Diese Redirect Einträge enthalten keine Links auf die englischen Pendants und müssen daher gesondert behandelt werden: Neben der Entitätenliste erstellt der Wikipedia-Parser eine zweite Datei, die die Redirecteinträge der Form
`USA Vereinigte Staaten`
enthält, die dann vom Entitätserkennungssystem gesondert eingelesen werden.

- **Begriffsklärung:** Es gibt einige Begriffe, insbesondere Abkürzungen, die verschiedene Dinge beschreiben, "UNO" zum Beispiel steht für "United Nations Organization" und das Kartenspiel "UNO".
  Um diese Mehrfachbedeutungen aufzulösen, gibt es in Wikipedia sogenannte Begriffsklärungsseiten, die für einen Begriff die verschiedenen Bedeutungen auflisten.
  Diese Seiten sind auf verschiedene Arten eingebaut:

  Eine Suche nach dem mehrdeutigen Begriff führt auf eine Begriffsklärungsseite. Diese Vorgehensweise wird in der deutschen Wikipedia bevorzugt.

  Eine Suche nach dem mehrdeutigen Begriff führt direkt auf den geläufigsten Artikel, auf dem die Begriffsklärung dann verlinkt ist. So wird es in der englischen Wikipedia meist gehandhabt.

  Einen einheitlichen Standard gibt es aber nicht. Daher erstellt der Wikipediaparser auch für die Begriffsklärungsseiten eine extra Datei, um diesen Spezialfall im Entitätserkennungsschritt gesondert zu behandeln.

2.3 Datenformate

Alle Dateien, die im Rahmen dieser Arbeit auf Festplatte geschrieben werden, stehen in einem CSV-Format. Bei diesem Format steht jeder Eintrag in einer Zeile und die Attribute eines Eintrags sind durch Tabulatoren getrennt. Auf diese Weise lassen sich die Dateien leicht von Mensch und Maschine lesen und bearbeiten, ohne dass ein kompliziertes Format eingehalten werden muss. Außerdem ist ein solches Format leicht erweiterbar, da man, falls weitere Attribute gespeichert werden sollen, einfach eine Spalte hinzufügen kann.

3 Entitätserkennung

Wikipedia beschreibt eine Entität folgendermaßen:

> Entität [...] ist in der Philosophie ein ontologischer Sammelbegriff für alles Existierende bzw. Seiende. So gehören Gegenstände, Eigenschaften, Prozesse usw. zur Klasse der Entitäten. [...]

In diesem speziellen Fall beschreibt Entität etwas, woüber die zugrundeliegende YAGO Ontologie Wissen besitzt. Die zentrale Aufgabe, mit der sich diese Arbeit beschäftigt, ist das Erkennen von Entitäten in deutschsprachigen Texten. Das Augenmerk liegt dabei auf Effektivität und Genauigkeit.

3.1 Übersicht

Die Entitätserkennung ist in verschiedene Module aufgeteilt, die sequentiell versuchen, Entitäten zu erkennen. Zunächst wird der Artikel Wort für Wort eingelesen und für jedes Wort dann die im Folgenden beschriebenen Module aufgerufen.

3.1.1 Schritt 1: Eindeutige Entitätserkennung

Im ersten Schritt wird der Text Wort für Wort durchgegangen und versucht, Entitäten, die aus einem oder mehreren Wörten bestehen eindeutig zu erkennen. Kann die Bedeutung eines Wortes nicht eindeutig festgestellt werden, wird es in eine Liste mehrdeutiger Entitäten eingetragen, die anschließend an Schritt 2 übergeben werden. Für eine detaillierte Beschreibung der eindeutigen Entitätserkennung siehe **??**.

3.1.2 Schritt 2: Disambiguierung

Nachdem der gesamte Artikel durchsucht wurde, wird das Disambiguierungsmodul aufgerufen. Dieses Modul erhält eine Liste mehrdeutiger Entitäten. Jeder Eintrag

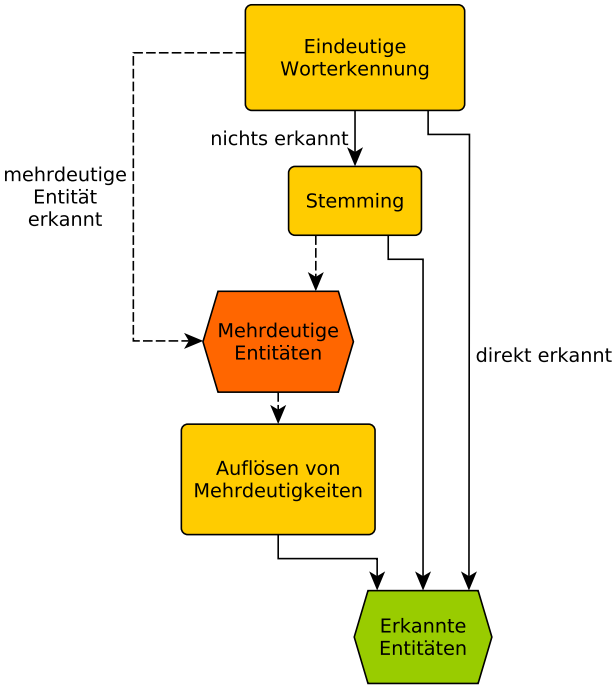

Abbildung 3.1: Übersicht über den modularen Ablauf der Entitätserkennung

dieser Liste enthält außerdem eine Liste von in Frage kommenden Kandidaten, bestehend aus den gefundenen Suffixen. Dieses Modul wird es an dieser Stelle nur schematisch beschrieben und für eine detaillierte Beschreibung auf [**?**] verwiesen.

3.1.3 Schritt 3: Stemming

Ein groß geschriebenes Wort, dass in Schritt 1 nicht erkannt wird und auch keine Kandidaten für eine Mehrfachbedeutung hat, kann entweder ein Wort sein, dass nicht in der Entitätenliste enthalten ist, oder eines, das nicht in der Grundform steht. In letzterem Fall kann versucht werden, durch so genanntes Stemming das Wort auf seine Grundform zu bringen. Für eine detaillierte Beschreibung des von uns verwendeten Stemming-Algorithmus siehe **??**

3.2 Trie Datenstruktur

Die häufigste Operation bei der Entitätserkennung ist das Überprüfen, ob ein bestimmter String (ein Wort aus dem Text) in einer Menge Strings (die Menge der

bekannten Entitäten) enthalten ist. Dabei sollen aufwändige Stringvergleiche möglichst vermieden werden.
Für diesen Zweck bedienen wir uns einer Trie Datenstruktur. Trie ist eine Abkürzung für "Retrieval Tree". Ein Trie ist ein spezieller Baum, in dem Elemente mit String-Schlüsseln gespeichert werden können. Die Knoten des Baumes sind die einzelnen Buchstaben der Schlüssel.

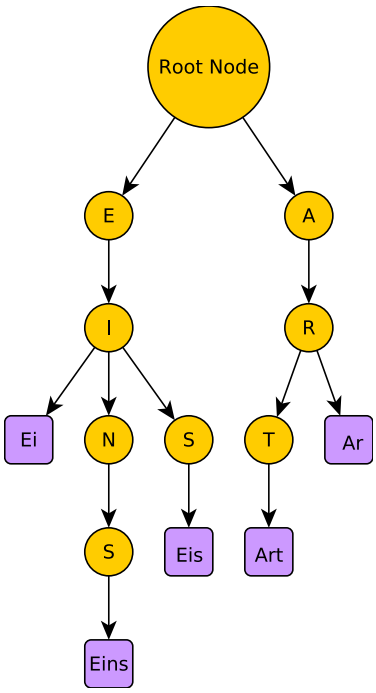

Abbildung 3.2: Ein Trie mit 5 Elementen

3.2.1 Implementierung

Innerhalb des Projektes gibt es mehrere Stellen, an denen ein Trie eine günstige Datenstruktur ist, um unterschiedliche Daten zu speichern. Daher implementierten wir eine Template Klasse, die mit verschiedene Datentypen speichern kann.Die einzige Voraussetzung an die Daten ist, dass sie einen eindeutigen String-Schlüssel generieren können.

Die beiden verwendeten Klassen sind *Trie* und *TrieNode* (siehe **??**), die jeweils ein Template Argument enthalten. Für die Entitätserkennung verwenden wir einen *Trie<Entity>*, der *Entity*-structs speichert. Diese enthalten alle während des Pro-

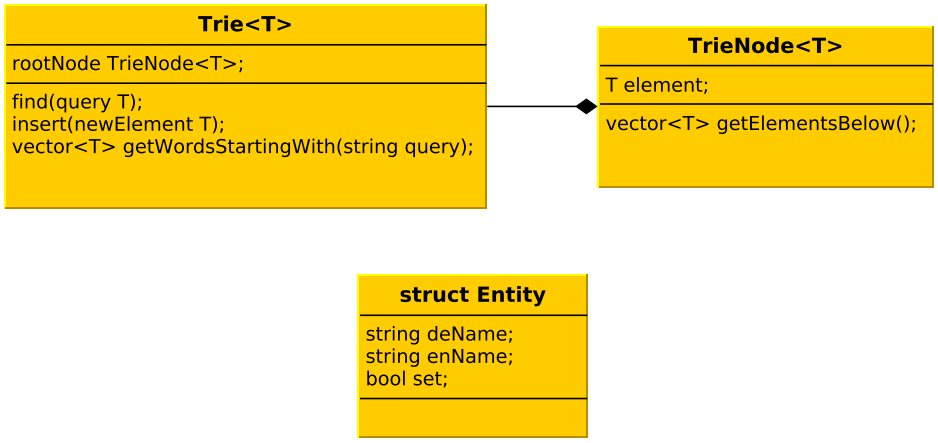

Abbildung 3.3: Vereinfachtes Klassendiagramm der verwendeten Trieklassen.

zesses benötigten Daten, wie z.B. deutscher Name, englischer Name, Länge des dazugehörigen Artikels (wird währende der Disambiguierung verwendet), etc.

3.2.2 Laufzeit

Einer der Vorteile eines Tries ist, dass die Operationen Einfügen und Suchen schnell und von der Größe des Tries unabhängig sind. Diese Operationen werden bei der Entitätserkennung am häufigsten verwendet. Das Einfügen beim initialen Aufbauen des Tries und das Suchen beim eigentlich Erkennen der Entitäten. Eine weiter wichtige Funktion ist das Suffixfinden. Im Folgenden werden diese drei Operationen genauer beschrieben sowie eine Laufzeitanalyse skizziert.

- **Einfügen:** Beim Einfügen eines neuen Elements in den Trie wird zunächst der Schlüsselstring in seine einzelnen Buchstaben zerlegt. Vom Wurzelknoten aus wird dann Buchstabe für Buchstabe den Baum hinabgestiegen. Wenn kein passendes Kind existiert, wird es neu erzeugt. Zuletzt wird das neue Element in den Elementbehälter des letzten Knotens eingesetzt.
  Die maximale Laufzeit ist dabei $n\cdot$ Knoten erzeugen + Element setzten. Da die für "Knoten erzeugen" und "Element setzten" benötigte Zeit konstant ist, liegt die gesamte Operation in $\mathcal{O}(n)$.
- **Suchen:** Die Suchoperation ist der Einfügeoperation sehr ähnlich. Der Schlüssel des zu suchenden Elements wird in die einzelnen Buchstaben zerlegt, anhand derer dann der Baum herabtraversiert wird. Wenn ein gesuchter Kindknoten nicht existiert, kann abgebrochen werde, denn das Element ist offenbar

nicht enthalten. Wenn die Buchstaben komplett abgelaufen werden können, muss überprüft werden, ob der Elementbehälter des aktuellen Knotens belegt ist.

- **Suffixe finden:** Für die Entitätserkennung ist es nötig zu wissen, ob es für einen Suchbegriff gültige Entitäten gibt, die mit diesem Suchbegriff anfangen, also Suffixe des Suchbegriffs sind.
  Die Suffixe eines Suchbegriffs sind innerhalb eines Tries genau die Elemente der Knoten unterhalb des letzten Knotens des Suchbegriffs. Um eine Suffixliste zu erhalten muss der Baum ab diesem Knoten rekursiv komplett durchlaufen werden. Somit hängt die Laufzeit von der Anzahl der Suffixe ab. Im worst-case (Suffixe des Wurzelknotens) müssen alle Knoten des Tries besucht werden.
  Dieses worst-case Szenario wird bei der Entitätserkennung nie eintreten und mit ansteigender Länge des Suchbegriffs sinkt die Zahl der Suffixe schnell, da der Entity-Trie einen hohen Verzweigungsgrad hat (theoretisch kann jeder Knoten für jedes Unicode-Zeichen ein Kind haben). Jedoch können gerade kurze Entitäten oder häufige Vornamen sehr viele Suffixe haben.
  Aus diesem Grund wird die ”Suffix finden“-Operation für jede Entität maximal einmal aufgerufen und die Ergenbnisliste dann im entsprechenden *Entity*-struct gespeichert.

3.3 Eindeutige Entitiätserkennung

Der Idealfall der Entitätserkennung tritt ein, wenn ein im Text vorkommendes Wort wörtlich in der Entitätsliste enthalten ist und der entsprechende Eintrag in der Entitätsliste keine gültigen Suffixe hat, zum Beispiel ”Taschenlampe“. Wird eine solche Entität entdeckt, wird sie in die Liste gefundener Entitäten eingetragen und direkt zum nächsten Wort übergegangen.
Wenn eine Entität E gültige Suffixe hat, also Entitäten die mit E Anfangen, kann das verschiedene Bedeutungen haben:

- Das gefundene Wort hat mehrere Bedeutungen. Zum Beispiel ”Neustadt“ hat die Suffixe ”Neustadt (Weinstr)“, ”Neustadt (Donau)“ und viele mehr. In diesem Fall wird in einem späteren Schritt versucht die Mehrfachbedeutung aufzulösen.
- Das gefundene Wort ist Teil einer Entität, die aus mehreren Wörtern besteht.

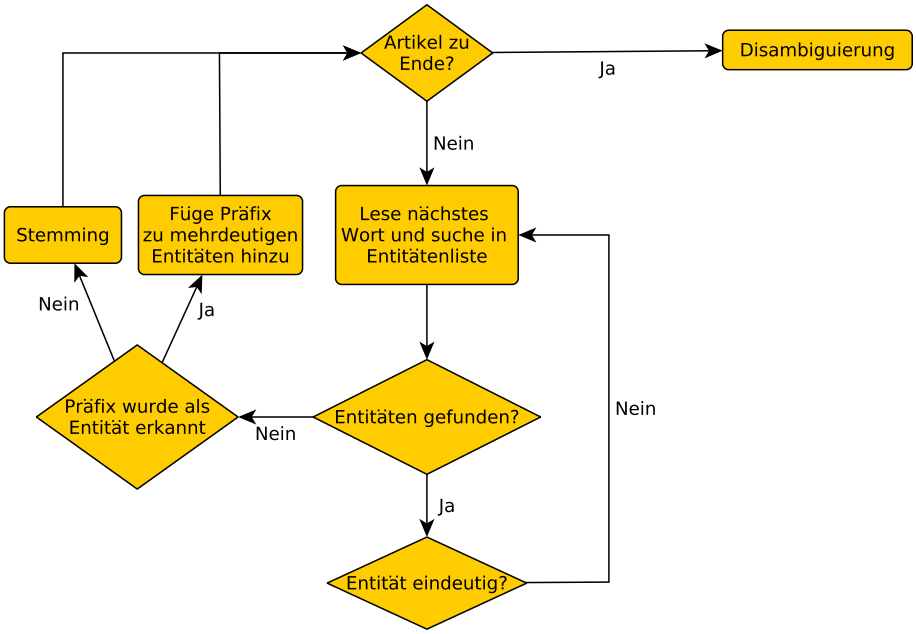

Abbildung 3.4: Ein Flowchart, der die Eindeutige Entitätserkennung beschreibt.

Wenn zum Beispiel das Wort "Michael" gefunden wird, ist es weniger wahrscheinlich, dass der Vorname Michael gemeint ist, als einer der Suffixe wie z.B. "Michael Ballack".

- Eine Kombination der beiden Möglichkeiten.

Es wird zunächst der zweite Fall angenommen und Schritt 1 auf das aktuelle und nächste Wort gemeinsam angewandt. Wenn dabei kein Ergebnis gefunden wird, wird das aktuelle Wort in die Liste mehrdeutiger Entitäten eingetragen, die dann Schritt 2 übergeben wird.

3.4 Stemming

"Stemming" beschreibt eine Gruppe von Verfahren, die ein Wort auf einen gemeinsamen Wortstamm reduzieren. Also zum Beispiel Plural auf Singular, etc. Dies ist in der Regel keine triviale Aufgabe und für eine fehlerfreie Lösung müssen zahlreiche grammatische Regeln der jeweiligen Sprache beachtet werden. Daher kommen meist heuristische Verfahren zum Einsatz, die sehr effizient sind, mit einfachen Regeln auskommen und dafür nicht zu 100% korrekt sind.

Wird während der Entitätserkennung ein Wort nicht erkannt, ist aber groß geschrie-

ben, verwenden wir eine modifizierte Variante des in [**?**] beschriebenen Stemmingalgorithmus für deutsche Wörter:

Algorithm 1 Der verwendete modifizierte Stemming Algorithmus

```
function findEntityUsingStemming(word):
if Entität für word gefunden then
  füge word zu gefundenen Entitäten hinzu
  return TRUE
end if
if |word|< 3 then
  return FALSE // Wenn das Wort kürzer als 3 ist, wird kein Stemming angewandt
end if
if |word|> 6 und word endet auf "nd" then
  entferne "nd" von word
  findEntityUsingStemming(word)
end if
if |word|> 5 und word endet auf "em" oder "er" then
  entferne "em" bzw. "er" von word
  findEntityUsingStemming(word)
end if
if |word|> 4 und word endet auf "e", "s", "n" oder "t" then
  entferne letzten Buchstaben von word
  findEntityUsingStemming(word)
end if
// falls keine der Regeln anwendbar ist, ist das Wort endgültig nicht erkannt
return FALSE
```

Das Ziel des in [**?**] beschriebenen Algorithmus ist es, jede Form eines Wortes auf einen gemeinsamen Stamm zurückzuführen. Dieser Stamm muss nicht unbedingt die Grundform des Wortes sein. "Koalitionen" würde zum Beispiel auf "Koalitio" zurückgeführt werden und nicht auf die Grundform "Koalition". Diesen Nachteil versuchen wir durch erneutes Abfragen nach jedem Stemmmingschritt auszugleichen: Im Beispiel würde zunächst das "n" entfernt und "Koalitione" geprüft. Da dies nicht zu einem Treffer führt wird das "e" entfernt und dann das Wort "Koalition" erneut überprüft, bevor das "n" entfernt wird.

4 Evaluation

Um die Qualität der Entitätserkennung einschätzen zu können und somit auch Rückschlüsse auf die Qualität der Suche ziehen zu können, bedarf es einer Evaluation der Ergebnisse.
Eine hohe Qualität drückt sich hierbei durch eine hohe Zahl korrekt erkannter Entitäten und gleichzeitig eine niedrige Zahl falsch erkannter Entitäten aus. Zu entscheiden, ob eine Entität falsch oder richtig erkannt wurde, ist hierbei keine triviale Aufgabe.

4.1 Rahmenbedingungen

Für die Evaluation sind einige limitierende Eigenschaften der verwendeten Systeme zu beachten:

- **Wikipedia:** Da unsere Entitätsliste aus Wikipedia generiert wurde, ist das Entitätserkennungssystem automatisch auf die dort enthaltenen Entitäten beschränkt: Wenn ein Artikel über den Vorsitzenden eines Fußballvereins aus der siebten Liga berichtet, dieser aber nicht in der Wikipedia beschrieben wird, kann er von der Entitätserkennung nicht gefunden werden.

  Die zweite Einschränkung, die durch die Nutzung von Wikipedia als Entitätsliste zustande kommt, ergibt sich durch das deutsch-englisch Mapping: Die Entitätsliste enthält keine Einträge für Entitäten, die zwar in der deutschen, nicht aber in der englischen Wikipedia vertreten sind. Durch diese Einschränkung gehen vor allem nur im deutschsprachigen Raum bedeutende Entitäten, wie lokale Persönlichkeiten oder kleinere Städte, verloren.

  Wenn es einen passenden englischen Artikel gibt, dieser aber auf dem deutschen nicht verlinkt ist, geht der Eintrag ebenfalls verloren.

- **Stemming:** Der verwendete Stemming Algorithmus ist heuristisch und produziert somit auch einige falsche Ergebnisse, die in falsch erkannten Entitäten resultieren können.

Für die Evaluation der Entitätserkennung verwenden wir ein Ground Truth Verfahren. Bei diesem Verfahren werden in einigen Texten Entitäten manuell markiert. Dann wird der Entitätsfindungsalgorithmus auf den unmarkierten Text angewandt und die gefundenen Ergebnisse mit den manuell annotierten verglichen.

4.2 Ground Truth Erstellung

Bei der Erstellung der Ground Truth muss ein gewisses Hintergrundwissen über das Entitätserkennungssystem vorhanden sein, um eine faire Evaluation zu ermöglichen. Für viele Entitäten gibt es unterschiedliche Schreibweisen, wie zum Beispiel "NATO“ oder "N.A.T.O.“, andere enthalten in der Wikipedia einen erklärenden Zusatz wie "Tor_(Fußball)“. Beim Annotieren der Texte muss also darauf geachtet werden, die Schreibweise zu wählen, die in der Entitätsliste eingetragen ist.

Die Ground Truth Artikel werden jeweils in einer separaten Datei abgespeichert. Jede Zeile enthält entweder nur ein Wort, dass keine Entität ist, oder eine Entität in der Form, in der sie im Text vorkommt und mit einem Tab getrennt den Namen der Entität, die erkannt werden soll. Zum Beispiel:

Heute	
abend	
wird	
Bundeskanzlerin	Bundeskanzler
Angela Merkel	Angela Merkel
in	
Berlin	Berlin
die	
Fußball Weltmeisterschaft	Fußball Weltmeisterschaft
eröffnen.	

Das Erstellen von guten Ground Truth Daten muss komplett von Hand geschehen und erfordert viel Zeit. Daher wurden bisher nur 4 solcher Artikel erstellt.

4.3 Evaluator

Der Ground Truth Evaluator verwendet das selbe EntityRecognition Objekt, das von EntityRecognitionMain verwendet wird und fungiert so quasi als Wrapper um das Entitätserkennungssystem.
Die Eingabe für den Evaluator ist ein Ground Truth Artikel. Dieser wird zunächst eingelesen und die zu findenden Entitäten gespeichert. Dann wird eine nicht annotierte Version des Artikels erstellt und die Entitätserkennung darauf angewandt. Anschließend werden die Ergebnisse verglichen und folgende Werte erhoben:

- **Hits:** Die Anzahl der korrekt erkannten Entitäten.
- **False Positives:** Anzahl der Entitäten die erkannt wurden, aber nicht annotiert waren.
- **False Negatives:** Anzahl der Entitäten, die annotiert waren, aber nicht erkannt wurden.

Ein Wort kann also gleichzeitig in die false positives und in die false negatives zählen.

Aus diesen Werten werden dann mit folgenden Formeln die beiden Werte *Positive Prediction Value (PPV)* und *Sensitivity* errechnet:

$$PPV = \frac{Hits}{Hits + falsePositives} \quad Sensitivity = \frac{Hits}{Hits + falseNegatives}$$

Außerdem wird eine HTML-Seite generiert, auf der die Ergebnisse visuell dargestellt werden. (Siehe **??**) Eine detaillierte Beschreibung findet sich in [**?**].

4.4 Ergebnisse

Die hier präsentierten Ergebnis wurden mit vier präparierten Ground Truth Artikeln erstellt.

Bei der Auswertung der Ergebnisse fällt auf, dass falsch positive Treffer wesentlich häufiger sind, als falsch negative. Daraus resultieren die hohen Sensitivity Werte im Vergleich zu den PPV Werten.
Das häufigste Probleme sind:

- Es werden andere Entitäten als erwartet erkannt: Das häufigste Problem ist das falsch erkennen von Entitäten, also das Erkennen einer Entität in einem

Abbildung 4.1: Die Visualisierung eines Evaluationsergebnisses: Grün sind korrekt erkannte Entitäten, Rot nicht erkannte (false negatives), Gelb falsch erkannte (false negative und false positive) und Blau falsch erkannte an Stellen, die nicht annotiert waren (false positives). Wie in **??** beschrieben, unterscheidet das Broccoli Userinterface die Funktion der Suchbegriffe durch Farbcodes. Diese haben nichts mit den hier gezeigten Farben zu tun.

Tabelle 4.1: Evaluationsergebnisse

	PPV	Sensitivity
Artikel 1	0.77	0.97
Artikel 2	0.78	0.97
Artikel 3	0.65	0.97
Artikel 4	0.73	0.97

Wort, für das eine andere Entität annotiert war. Diese Problem resultiert meist aus falscher Disambiguierung.

Beispiel: In ”Die Protestbewegung Echte Demokratie Jetzt! hatte angekündigt, sich dem Bann widersetzen zu wollen.“ wurde ”Bann” als “Verbannung” erkannt und nicht wie annotiert als “Bann (Recht)”.

- False positives: Eine Entität wurde in einem nicht annotierten Wort erkannt. Diese Fehler resultieren meist daraus, dass die betreffenden Begriffe zwar grundsätzlich eine Entität oder einen Teil repräsentieren können, dies aber im speziellen Fall nicht tun und daher nicht annotiert wurden.

  Beispiel: In dem Satz ”Bevor es zu einem Netzausfall käme“ wurde zum Beispiel das Wort ”Bevor“ als Entität (Das Buch ”Bevor ich sterbe“) gekennzeichnet, ist jedoch nicht annotiert.

  Eine weitere große Gruppe unter den false positives sind Personennamen, die

nicht in der Entitätenliste vorhanden sind und deren Vor- und Nachnamen dadurch als einzelne Entitäten durch Disambiguierung (falsch) erkannt werden.

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine semantische Suchmaschine für deutsche Zeitungstexte vorgestellt.
Dabei wurden folgende Teilaufgaben bearbeitet:

- Für die Datenbeschaffung wurde ein Webcrawler programmiert, der systematisch Nachrichtenseiten durchsucht und Artikel extrahiert und speichert.
- Die so gewonnenen Artikel werden dann von einem Entitätserkennungssystem verarbeitet. Dabei werden Entitäten aus der englischsprachigen Yago-Ontologie in den Texten erkannt. Die Entitätserkennung läuft in drei modularen Schritten ab: wörtliches Wiederfinden von Entitäten, Disambiguierung und Stemming.
- Um die englischen Begriffe in deutschen Texten zu erkennen, wird ein deutsch-englisch Mapping aus Wikipedia erstellt.
- Um das bestehende System "Broccoli", dass normalerweise in Wikipedia-Artikeln sucht, verwenden zu können, werden die verarbeiteten Artikel in einem Wikimedia-XML Format gespeichert.
- Um die Entitätserkennung evaluieren zu können, wurden Ground Truth Daten erstellt, also Artikel, in denen Entitäten manuell annotiert wurden. Weiterhin wurde ein Evaluationsalgorithmus geschrieben und die Entitätserkennung dann auf den Ground Truth Daten getestet.

Ausblick

Während der Bearbeitung dieser Probleme wurden immer wieder neue Teilprobleme entdeckt oder bessere Lösungen für bereits bearbeitete Probleme gefunden. Einige

dieser Verbesserungen wurden bereits implementiert, für andere war im Bearbeitungszeitraum kein Platz mehr.
Im Folgenden werden einige Vorschläge für zukünftige Arbeiten vorgestellt:

Verbesserung der Entitätserkennung

Größten Einfluss auf die Qualität der Suche hat die Entitätserkennung. Die Evaulation hat gezeigt, dass diese bereits recht gut funktioniert. Trotzdem bleibt Raum für Verbesserungen:

- Der verwendete Stemming Algorithmus hat in der Evaluation einige falsch positiven Treffer erzeugt. Durch zusätzliche Regeln, wie zum Beispiel "Abkürzungen dürfen nicht durch Stemming erkannt werden", ließe er sich noch verfeinern. Eventuell gibt es auch bessere Stemming Algorithmen für die deutsche Sprache.
- Die Disambiguierung wies in der Evaluation noch viele Fehler auf. Für eine genauere Beschreibung dieses Aspekts wird auf [**?**] verwiesen.

Bessere Integrierung in Broccoli

Im Rahmen dieser Arbeit wurden erstmals Zeitungsartikel als Eingabedaten für Broccoli verwendet. Dabei wurden einige Features vermisst, die nur speziell für solche Artikel interessant sind:

- Es sollte möglich sein, die Suche auf Artikel einzuschränken, die in einem bestimmten Zeitraum erschienen sind.
- Genau so sollte man die Suche auf eine bestimmte Quelle (Spiegel Online, Zeit Online, etc) beschränken können.
- Erscheinungsdatum und Quelle sollten in den Ergebnissen angezeigt werden.
- Derzeit ist es nicht möglich in Broccoli nach Klassen auf deutsch zu suchen. Statt "Sportler" muss "athlete" eingegeben werden.

Erweiterung des Crawlers

Der News-Crawler ist derzeit beschränkt auf eine Quelle, nämlich Spiegel Online. Er ist allerdings so geschrieben, dass er auf weitere Nachrichtenseiten erweitert werden

kann. Aufgrund der sehr individuellen Struktur der Artikelseiten und Konventionen der Links auf diese Seiten, muss allerdings für jede Seite individueller Code entwickelt werden.

Ontologie

Die größte qualitative Beschränkung der Entitätserkennung ist, dass nur Entitäten erkannt werden können, die in der verwendeten Yago Ontologie vorhanden sind. Dies stellt insbesondere eine Einschränkung in deutschen Texten dar, da Entitäten die ausschließlich im deutschsprachigen Raum bedeutend sind, nicht vorhanden sind.
Die Entwicklung einer eigenen Ontologie könnte daher die Qualität der Entitätserkennung und damit der gesamten Suche deutlich verbessern.

Literaturverzeichnis

[Bau11a] Baumgarten, Ina: *Semantische Suche in Zeitungstexten / Entitätserkennung*. 2011. – Uni Freiburg

[Bäu11b] Bäurle, Florian: *A User Interface for Semantic Full Text Search*, Uni Freiburg, Diplomarbeit, 2011

[BCSW07] Bast, Holger ; Chitea, Alexandru ; Suchanek, Fabian M. ; Weber, Ingmar: ESTER: efficient search on text, entities, and relations. In: [**?**], S. 671–678. – ISBN 978–1–59593–597–7

[Buc10] Buchhold, Björn: *SUSI: Wikipedia Search Using Semantic Index Annotations*, Uni Freiburg, Diplomarbeit, 2010

[Cau] Caumanns, Jörg: A Fast and Simple Stemming Algorithm for German Words / Free University of Berlin, CeDiS. – Forschungsbericht

[Hau11] Haussmann, Elmar: *Contextual Sentence Decomposition with Applications to Semantic Full-Text Search*, Uni Freiburg, Diplomarbeit, 2011

[KVC+07] Kraaij, Wessel (Hrsg.) ; de Vries, Arjen P. (Hrsg.) ; Clarke, Charles L. A. (Hrsg.) ; Fuhr, Norbert (Hrsg.) ; Kando, Noriko (Hrsg.): *SIGIR 2007: Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Amsterdam, The Netherlands, July 23-27, 2007*. ACM, 2007 . – ISBN 978–1–59593–597–7

[MPI] MPI für Informatik: *YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia*. – http://www.mpi-inf.mpg.de/yago-naga/yago/