Universität Bielefeld

Bachelorarbeit

Entity Linking through Indexing : Implementierung und Evaluierung

Author:
Tim Pontzen

Supervisor:
Prof. Dr. Philipp Cimiano

A thesis submitted in fulfilment of the requirements for the degree of Bachelor of Science

in the

Semantic Computing Group
Universität Bielefeld

26. Mai 2014

Declaration of Authorship

Ich, Tim Pontzen, bestätige hiermit, dass die Bachelorarbeit mit dem Titel 'Entity Linking through Indexing : Implementierung und Evaluierung' und das dazugehörige Programm selbstständig von mir geschrieben wurde. Ich bestätige:

- Diese Arbeit wurde komplett während meiner Studienzeit an der Universtität Bielefeld für den Bachelor of Science geschrieben.
- Ich habe die einbezogene Arbeit von anderen immer deutlich kenntlich gemacht.
- Ich habe zu jedem Zitat die Quelle angeben. Außerhalb dieser Zitate habe ich alles selbständig verfasst.

Signed: _____

Date: _____

Abkürzungen

ERU	**E**ntity-**R**ecognition-**U**nit
GUI	**G**raphical **U**ser **I**nterface
IBEL	**I**ndex **B**ased **E**ntity **L**inker
IBELU	**I**ndex **B**ased **E**ntity **L**inker **U**tility
idf	**i**nverse **d**ocument **f**requency
JAR	**J**ava **AR**chieve
NER	**N**amed **E**ntity **R**ecognizer
RDF	**R**esource **D**escription **F**ramework
tf	**t**erm **f**requency
URI	**U**niform **R**esource **I**dentifier

Abbildungsverzeichnis

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der heutigen Zeit spielen Wissensdatenbanken wie Wikipedia[1] eine immer größere Rolle. Sie repräsentieren das gesammelte Wissen der Menschheit und stellen es zur freien Verfügung. Eine wichtige Eigenschaft von Wikipedia ist die Reichhaltigkeit von eingebetteten Links in jedem Artikel, welche die wichtigsten Terme mit anderen Seiten verknüpfen und somit einem Benutzer einen schnellen Weg zu zusäztlichen Informationen zu gewähren[MC07].

Während diese Links in Wikipedia, dank der massiven Anzahl an Beitragenden, manuell erstellt werden können, ist für viele andere Dienste, wie zum Beispiel Web-Personensuche und Informationsextraktion, eine automatisierte Lösung gewünscht. Die dafür nötige Named-Entity-Disambiguierung ist problematisch, da eine Entität multiple Benennungen hat und diese unter Umständen auch mit anderen Entitäten teilt [WST10].

Die Aufgabe des Entity-Linkings ist es, Entitäten aus einem Webtext mit dem Äquivalent aus einer Wissensdatenbank zu verbinden. Dies ist nicht zu verwechseln mit dem named entity recognition Prozess, der sich auf die Identifizierung von Dingen in einem Text und nicht auf die Suche nach der Referenz, die diese Entität darstellt beschränkt. Eine solche Disambiguierung kann zum Beispiel, wie in dieser Arbeit und [MW08] vorstellt, mit der Hilfe von Anchors erfolgen.

[1] wikipedia.org

1.1 Verwandte Arbeiten

1.1.1 DBpedia

Während die meisten Wissensdatenbanken heutzutage auf spezifische Domänen beschränkt sind und von einer kleinen Gruppe von Wissensmodelierern, unter hohen Kosten, erstellt und gewartet werden, hat es Wikipedia geschafft, zu einer der zentralen Wissensquellen der Menschheit zu werden, die von tausenden von Beitragenden erhalten wird [BLK+09]. Jedoch gibt es trotz dieses Erfolges einige Probleme und ungenutztes Potential [MLAH12]:

- Suche ist in Wikipedia auf Keywordmatching begrenzt
- Mögliche Inkonsistenzen durch Datenduplikation auf verschiedenen Seiten und in unterschiedlichen Wikipedia-Sprachausgaben.

Das DBpedia Projekt extrahiert strukturierte Informationen von Wikipedia und macht diese für das Web verfügbar. Um die Daten stets aktuell zu halten, wird über den "Wikipedia live article feed" jede Änderung in Wikipedia registriert und eine entsprechende Aktualisierung in DBpedia übernommen. Alle DBpedia-Entitäten besitzen einen einzigartigen globalen Identifier, der nach den Linked Data Grundsätzen[2] dereferenziert werden kann [BLK+09]. Dies und die breitgefächerte Domänenabdeckung hat dazu geführt, dass viele Data-Publisher ihre Datenquellen mit RDF-Links zu DBpedia versehen. Dies macht DBpedia zu einem der zentralen Linked Open Data Netzknoten (Hub)[3].

1.1.2 DBpedia Spotlight

DBpedia Spotlight[4] ist ein open source Projekt zur automatischen Annotation von Entitäten aus DBpedia[DMP13]. Ursprünglich nur für die englische Sprache entwickelt, ist es inzwischen um 9 weitere Sprachen erweitert worden. Während viele semantische Annotationssysteme auf geringe Gruppen von Entitätstypen wie zum Beispiel Personen, Organisation oder Orte beschränkt sind, versucht DBpedia Spotlight bis zu 320 verschiedene Klassen in der DBpedia Ontology zu annotieren[PJMC11]. Um unterschiedlichen

[2] http://www.w3.org/DesignIssues/LinkedData.html
[3] http://webknox.com/p/linked-open-data-visualization
[4] spotlight.dbpedia.org

Anforderungen für verschiedene Anwendungsgebiete gerecht zu werden, ist es dem Benutzer möglich die Suchdomänen, sowie die Fehlertoleranz anzupassen.

1.2 Konzeption

Das Ziel dieser Arbeit ist ein ähnliches, aber effizienteres Programm wie DBpedia Spotlight zu entwickeln. Während die meisten Annotationsprogramme einen Graphen für die Entitätenmenge aufbauen und diesen dann durchsuchen, wird in dem Index Based Entity Linker (IBEL) ein indexbasierter Ansatz verfolgt. Mit Hilfe des Anchor-Indexes der Semantic Computing Group Bielefeld[5] und DBpedia-Datendumps wird ein Nachbarschafts-Anchor-Index erstellt, der jede Entität mit den Anchors seiner DBpedia-Graphnachbarn verknüpft. Über diesen Index kann dann jede Entität sehr schnell und effizient mit ihren potentiellen DBpediaäquivalenten in Verbindung gebracht und dann über die Nachbar-Anchor disambiguiert werden. Dazu werden alle weiteren Entitäten in der Textquelle mit den Nachbar-Anchors verglichen, um den semantischen Kontext in die Suche mit einzubeziehen. Daraus resultiert dann ein Score, der zur Auswahl der verschiedenen Suchergebnisse dient.

[5] http://www.sc.cit-ec.uni-bielefeld.de/

Kapitel 2

Systemübersicht

In diesem Kapitel wird die Systemarchitektur und dessen Informationsfluss behandelt. Insgesamt besteht das System aus 4 Teilen:

- Graphical User Interface (GUI)
- Entity-Reconition-Unit (ERU)
- Linking Unit
- Index

Das nachfolgende Bild bietet einen Überblick über die Architektur und das Zusammenspiel der einzelnen Komponenten. Deren Funktion wird im Anschluss näher erläutert.

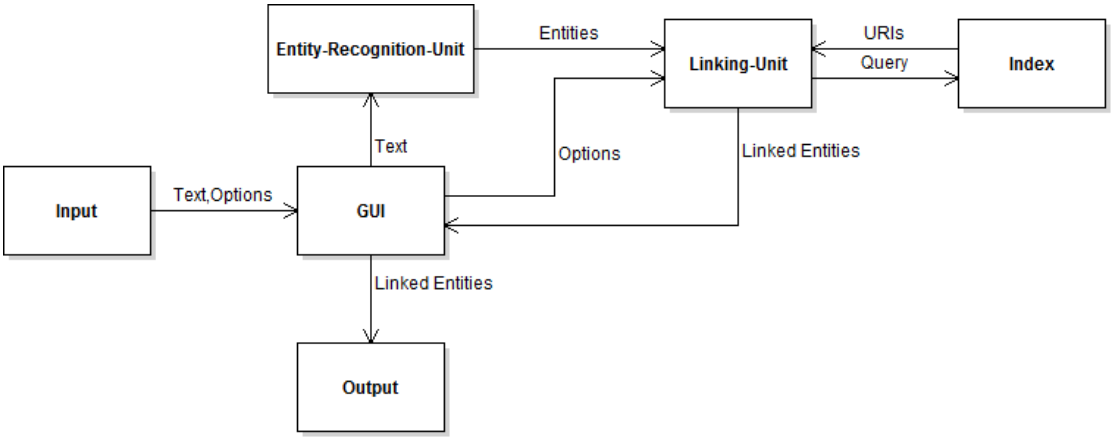

Abbildung 2.1: Systemübersicht

2.1 Graphical User Interface

Die GUI stellt die Schnittstelle zwischen Benutzer und Programm dar. Diese erlaubt ihm, seinen zu anotierenden Text zu übergeben und zwischen verschiedenen Optionen zu wählen. Der Input wird entgegenommen und dessen Text an die Entity-Recognition-Unit weitergeleitet. Am Ende bekommt es die verlinkten Entities von der Linking-Unit zurürck und gibt diese an den Benutzer weiter.

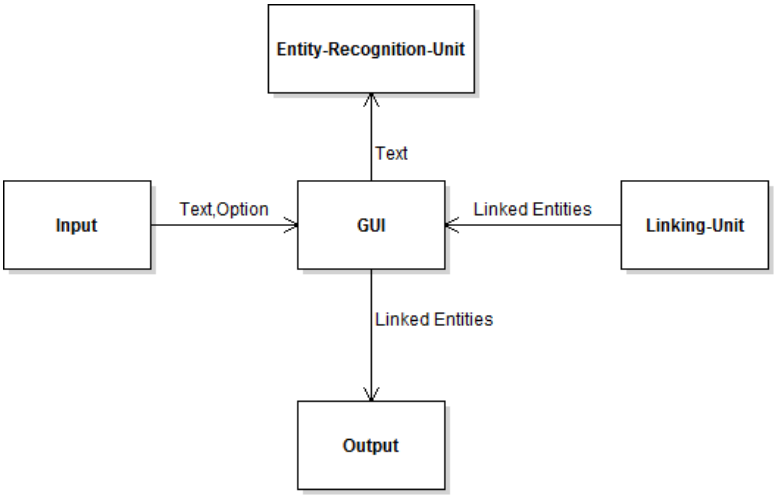

Abbildung 2.2: GUI-Verbindungen

2.2 Enitity-Recogniction-Unit

Diese Einheit ist für die Erkennung von Named Entities verantwortlich. Dies bedeutet, dass sie den ihr übergebenen Text analysiert und alle sich darin befindlichen Entitäten extrahiert (z.B. Personen oder Orte). Das Ergebnis dieses Prozesses wird dann an die Linking-Unit weitergeleitet.

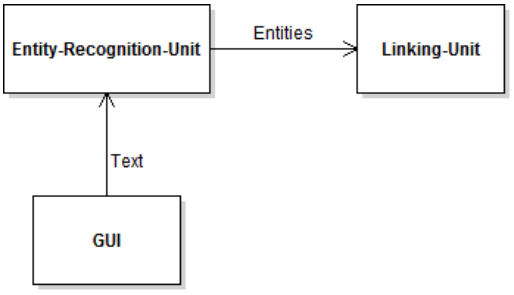

Abbildung 2.3: Enitity-Recognition-Unit

2.3 Linking-Unit und Index

Die Linking-Unit versucht mit Hilfe des Indexes für jede an sie weitergeleitete Entität einen passenden uniform resource Identifier (URI) zu finden. Dazu erstellt sie, unter Berücksichtigung der vom Benutzer eingestellten Optionen, eine Query und lässt den Index diese ausführen. Dessen Rückgabewert wird dann mit den Named Entities verknüpft und an die GUI weitergereicht.

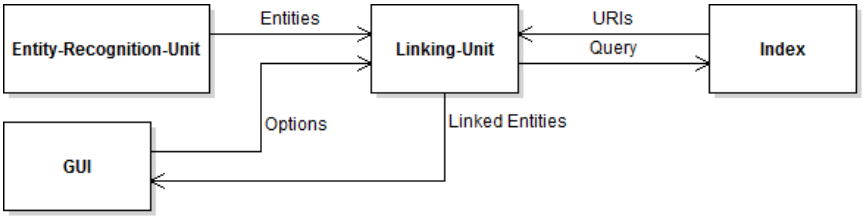

Abbildung 2.4: Linking-Unit

2.4 Informationsfluss

Um die Informationsverarbeitung noch einmal in einer Gesamtübersicht zu betrachten, sind alle Abläufe in dem folgenden Sequenzdiagramm verdeutlicht. Folgende Szenarien sind abgedeckt :

- Texteingabe
- Optionsauswahl
- Annotierung

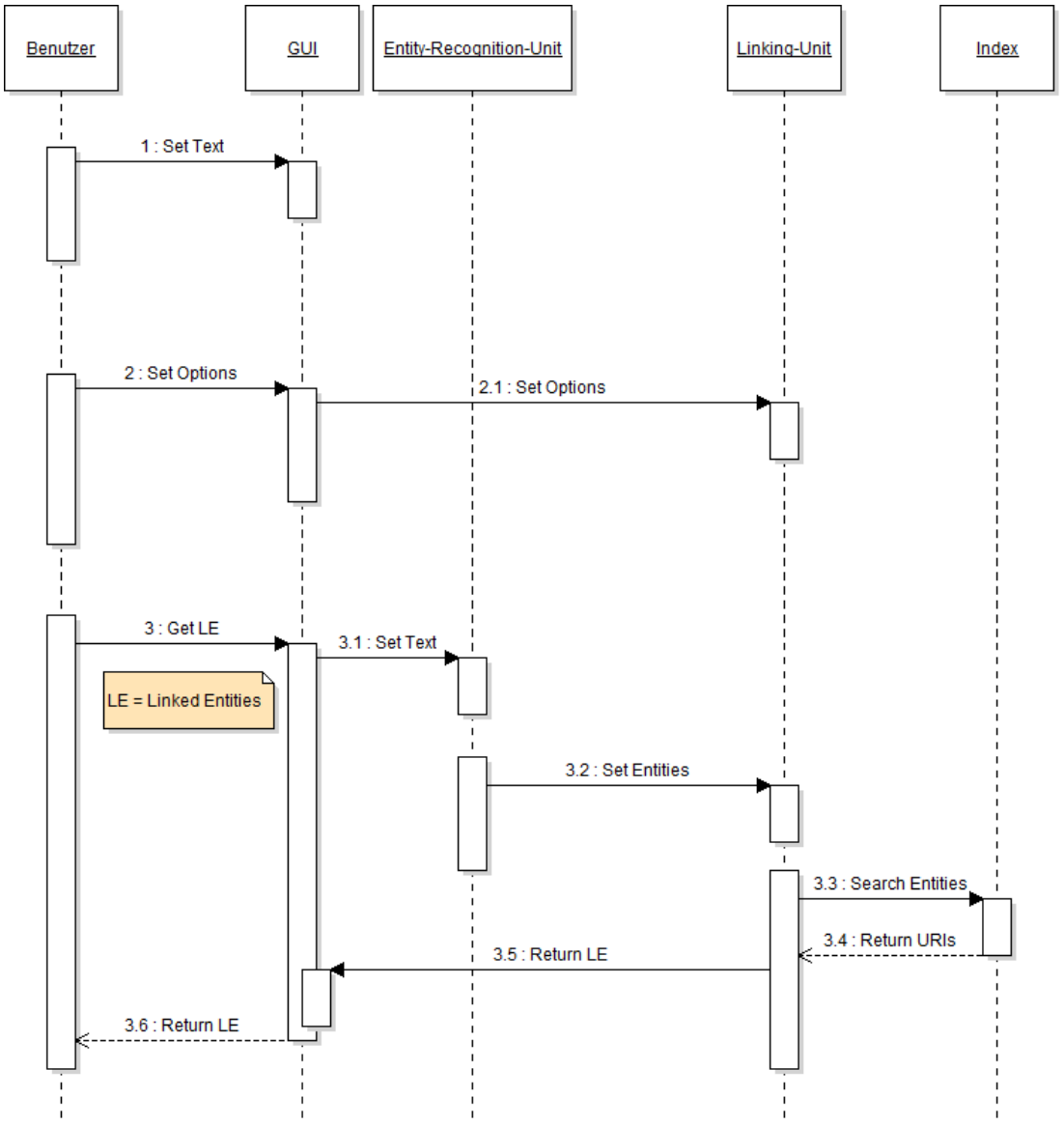

Abbildung 2.5: Programmablauf

Kapitel 3

Implementierung

Der Source-Code für den Index Based Entity Linker(IBEL) wird als ein Maven[1]-Projekt für Java verwaltet. Dies ermöglicht ein komfortables Warten und Weiterentwickeln des Codes, da alle benötigten Bibliotheken automatisch über das zentrale Maven-Repository bezogen werden. Zusätzlich ist sichergestellt, dass diese die richtige Version besitzen, um Kompatibilitätsprobleme zu vermeiden.
Als Entwicklungsumgebung wurde Netbeans 8.0 verwendet und als Java-Version Java 7.
Im Folgenden wird nun die Implementierung der einzelnen Funktions-Einheiten, welche in Kapitel 2 vorgestellt wurden, behandelt.

3.1 GUI und Controller

Um die einzelnen Bestandteile des IBEL zu kapseln und somit leichter zu Warten, wurde eine Controller implementiert, dessen alleinige Aufgabe es ist, die Kommunikation zwischen den Klassen zu handhaben.
Die Gui nimmt weiterhin die Eingaben des Benutzers entgegen, jedoch wird die Auswertung der Interaktion nun von dem Controller übernommen. Hierfür implementiert dieser das ActionListener-Interface des java.awt.event-Packages.

[1] http://maven.apache.org/

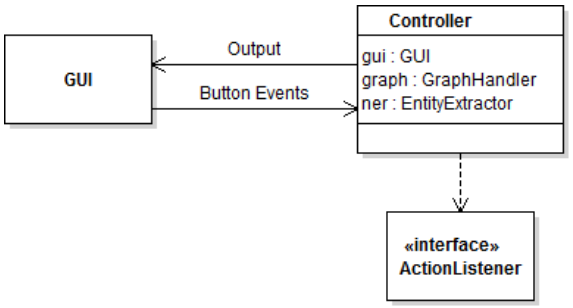

Abbildung 3.1: GUI and Controller

3.2 Enitity-Recognition-Unit

Um den vom Benutzer eingegeben Text untersuchen zu können, benutzt IBEL den Stanford-NER, einen von der Universität Stanford entwickelten Named Entity Recognizer, der Wortsequenzen in einem Text kennzeichnet, bei denen es sich um Namen für Dinge handelt, wie zum Beispiel Personen oder Firmennamen [2].

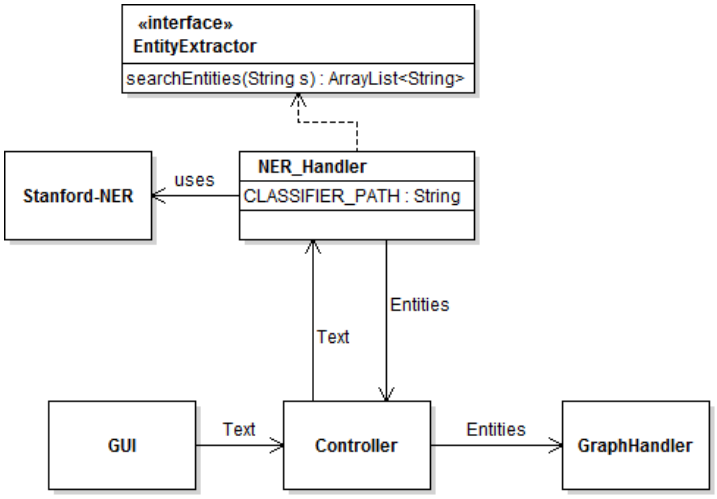

Abbildung 3.2: Entity-Recognition-Unit Implementation

Die Schnittstelle zwischen dem Controller und der Enitity-Recognition-Unit (ERU) bildet das EntityExtractor-Interface. Jede Klasse, die dieses implementiert, kann vom Controller genutzt werden, um Entitäten in Texten erkennen zu lassen. Der NER_Handler, welcher dieses Interface implementiert, nutzt die Classifier des Stanford-NER, um eine Liste mit Entitäten in dem ihm übergebenen Text zu extrahieren.

[2] http://nlp.stanford.edu/software/CRF-NER.shtml

3.3 Linking-Unit

Nachdem die Entity-Recognition-Unit alle gefundenen Entitäten an den Controller zurückgeben hat, werden diese an den GraphHandler zum Entity Linking weitergeleitet. Dieser erstellt Querys für Lucene, einer performanten Open Source Java-Bibliothek für Textsuche[3], und lässt diese dann ausführen. Die ermittelten URIs werden mit den Entitäten verknüpft und an den Controller zurückgegeben.

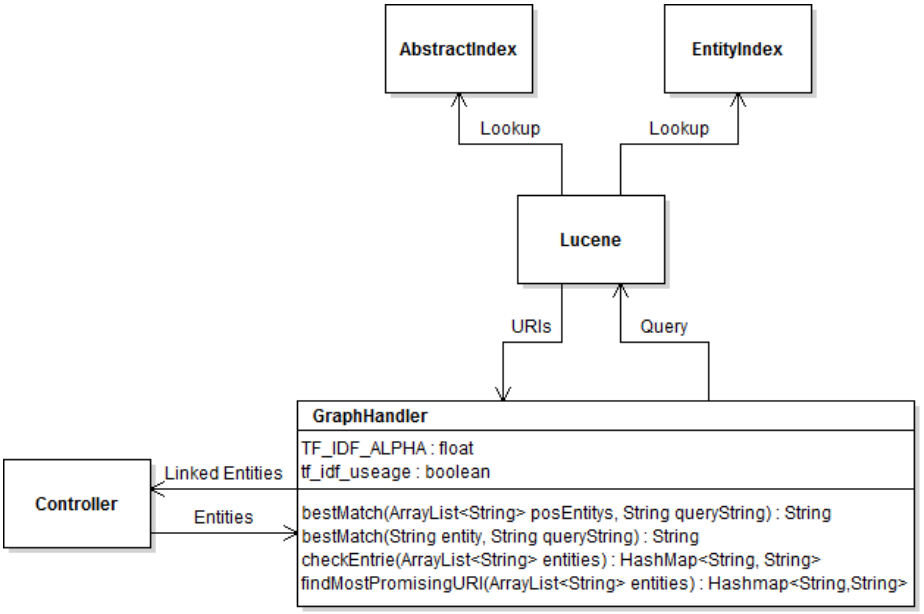

Abbildung 3.3: Entity Linking Unit

Ablauf des Linkings

Der Entity-Linking-Algorithmus erhält eine Liste von Entitäten und durchläuft dann drei Phasen:

1. Titelvergleich: Es wird geschaut, ob die Entität ein Titel für eine DBpedia Seite ist. Zum Beispiel wäre die Entität „United States“ der Titel der DBpedia Seite mit der URI „http://dbpedia.org/resource/United_States“ .

2. Anchorsuche: Falls die Titelsuche kein Ergebnis liefert, so wird überprüft, ob die Entität ein verzeichneter Anchor einer URI ist. In dem United States Beispiel

[3] http://lucene.apache.org/core/

wäre „USA“ einer dieser Anchors. Da ein Anchor disambig sein kann, wird versucht, über den semantischen Kontext eine Entscheidung zu treffen. Dazu werden die Nachbaranchors der möglichen Kandidaten mit allen anderen gefunden Entitäten verglichen. Der Kandidat mit den meisten Übereinstimmungen hat die größte Wahrscheinlichkeit, die gesuchte URI zu sein und wird deshalb mit der Entität verknüpft.

3. tf-idf-Suche: Wurde in den ersten beiden Schritten keine passende URI gefunden und der Benutzer hat die entsprechende Option angegeben, so wird versucht über eine tf-idf-Suche eine URI zu ermitteln. Dazu wird in den Abstract-Texten aller DBpedia Seiten nach dem Vorkommen der Entität gesucht. Der “term frequencey“-Wert (tf) gibt dabei an, wie häufig die Entität gefunden wurde und die “inverse document frequencey“ (idf) wertet die Bedeutung im Kontext mit allen Dokumenten (in diesem Falle DBpedia Seiten). Die 20 erfolgreichsten Suchtreffer werden nun behandelt als wären sie Kandidaten für einen disambiguen Anchor. Mit ihnen wird wie in der vorherigen Phase verfahren.

Pseudocode :

```
function findURIs(List entities)
        map<k,v> result;
        URI tmp;
        forall entity in entities
                tmp=searchTitle(entity)                          \\titelsuche
                if(tmp!=null)
                        result.put(entity,tmp);
                else
                        tmp=anchorSearch(entity)                 \\Anchorsuche
                        if(tmp!=null)
                                result.put(entity,tmp);
                        else if(tf_idf_Search==true)
                                list posURIs=tf_idf_Lookup(entity)\\Abstractsuche
                                tmp=anchorSearch(posURIs)\\Anchorsuche mit
                                                        \\Abstractsuchergebnissen
                                result.put(entity,tmp)
                        endif
                endif
        endfor
        return result;
```

In dem Folgenden Aktivitätsdiagramm wird noch einmal der Linking-Algorithmus verdeutlicht.

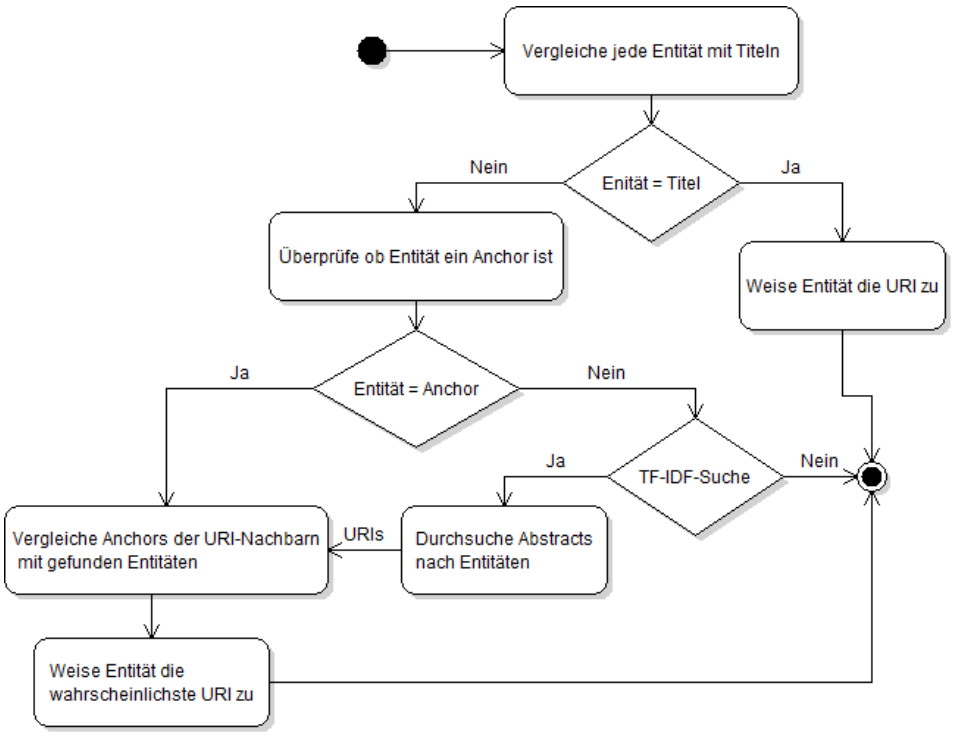

Abbildung 3.4: Entity Linking

3.4 Index

Ein Lucene-Index ist eine indexierte Ansammlung von Dokumenten. Ein Dokument besitzt eine beliebige Anzahl von Feldern, in denen Informationen gespeichert werden können. Der Index aus der Systemübersicht wurde in der Implementierung in 2 Indexe unterteilt. Dies liegt hauptsächlich daran, dass sie unterschiedliche Formate besitzen, um die aufgabenspezifischen Querys effizienter zu machen.

3.4.1 EntityIndex

Der EntityIndex wurde als BlockIndex angelegt, d.h. er besitzt zwei unterschiedliche Dokumenttypen:

1. Kinddokument: Ein Kinddokument ist ein gewöhnliches Dokument.

2. Elterndokument: Elterndokumente unterscheiden sich von Kinddokumenten durch ein Indikatorfeld, das für alle Elterndokumente den gleichen Wert hat.

Um Eltern und deren Kinder bei der Queryauswertung richtig zuordnen zu können, muss bei der Indexerstellung eine bestimmte Reihenfolge eingehalten werden: Erst werden alle Kinddokumente eines gleichen Elter hinzugefügt, gefolgt von dem Elterndokument.
Dieses besondere Format ermöglicht `ToParentBlockJoinQuerys`. Diese Querys selektieren erst Elterndokumente und führen dann eine weitere Query auf den Kindern dieser Eltern aus. Dadurch kann sehr gezielt und somit effizient gesucht werden.
In der Implementation repräsentieren Elterndokumente die einzelnen Seiten aus DBpedia. Sie speichern den Titel, die URI und alle Anchors einer Seite. Die Kinddokumente besitzen nur ein Feld mit einem Anchor eines Nachbarn der URI aus dem Elterndokument.
Zur Veranschaulichung wird hier noch einmal das Format des EntityIndexes gezeigt:

```
<anchorN_{1_1}> (Kind 1 von URI_1)
...
<anchorN_{n_1}> (Kind n von URI_1)
<URI_1,anchor_{1_1} ... anchor_{i_1},title_1,type> (Elter der Kinder 1 - n)
```

3.4.2 AbstractIndex

Der AbstractIndex ist ein Standart-Index, dessen Dokumente zwei Felder enthalten:

1. URI
2. Abstract der URI

Sein Format sieht wie Folgt aus:

```
<URI_1,abstract_1>
...
<URI_n,abstract_n>
```

Da Lucene für sein Query-Scoring tf-idf-Werte verwendet und auch genau diese bei der Abstract Suche erwünscht sind, liefert eine normale Query bereits das gewünschte Ergebnis.

Kapitel 4

Evaluierung

Dieses Kapitel beschäftigt sich intensiv mit der Analyse des Linking-Algorithmus. Insbesondere wird auf die Aspekte Geschwindigkeit und Genauigkeit eingegangen. Als Basis für die Auswertung wurden Texte aus Wikipedia[1] genommen und für die Semantik-Analyse Datensätze des Max Plank Institutes.
Die Tabellen für die Graphen finden sich im Anhang.

4.1 Geschwindigkeit

Der maßgeblichste Faktor der Geschwindigkeit ist die Benutzungsdauer des Programms. Man spricht hierbei von einem "warmed up Index", was bedeutet, dass aufgrund von Cashe-Einträgen Daten schneller ermittelt werden können.

4.1.1 Unwarmed Mode

Da der "warmed up"-Modus sehr inkonsistente Ergebnisse für wiederholte Versuche mit gleichen Texten liefert, wird der "unwarmed mode" betrachtet, in dem das Programm noch über keinerlei Cashe-Einträge für Querys verfügt. Dies ist auch gleichzeitig eines der Worst-Case-Szenarien des Linking-Algorithmus.

[1] www.wikipedia.org

In einem durchschnittlichen Text lassen sich fast 80% aller , durch den Stanford-NER, gefundenen Entitäten entweder mit einem Titel oder einem Anchor verbinden. Der Rest muss mit Hilfe von tf_idf-Werten bestimmt werden.

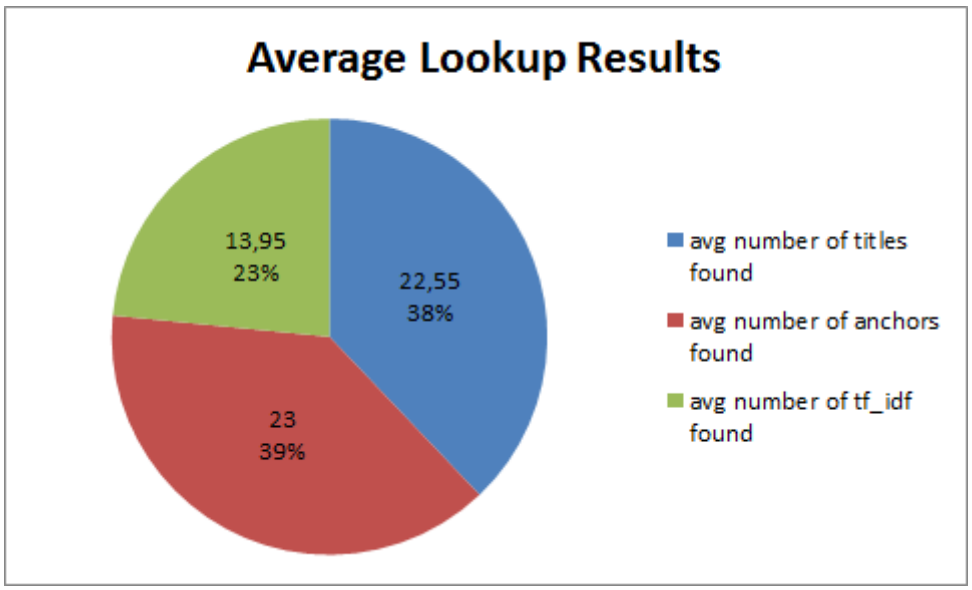

Abbildung 4.1: Average Lookup Results

Um die einzelnen Operationen in einem zeitlichen Zusammenhang betrachten zu können, wird als nächstes die durchschnittlich benötigte Zeit für jede dieser Operationen betrachtet.

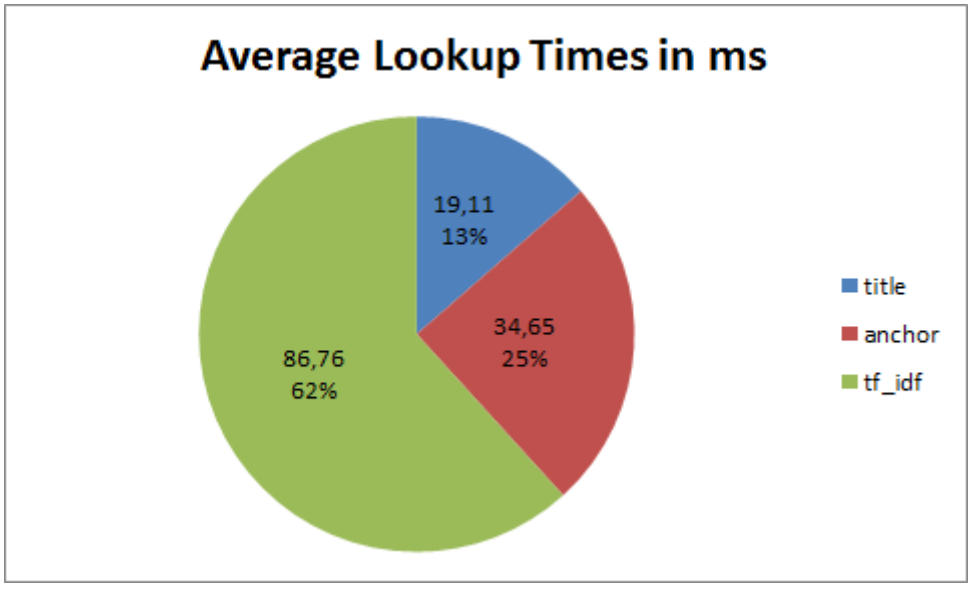

Abbildung 4.2: Average Lookup Times

Wie man in der Abbildung gut sehen kann, sind Titel- und Anchorsuche günstige Operationen im Vergleich zu der tf-idf-Suche, wobei die Titelsuche fast doppelt so schnell veräuft wie die Anchorsuche. Im Idealfall sollten die zu linkenden Entitäten entweder ein Titel oder ein Anchor sein.

Anmerkung: Bei der Abbildung handelt es sich nur um die benötigte Zeit der verschiedenen Operationen. Da diese aber sequentiell ablaufen, bedeutet dies, dass die Linking-Zeit der Summe der benutzten Operationen entspricht (z.B. beträgt die Zeit für Anchor-Linking Titel-Lookup + Anchor-Lookup).

4.1.2 Warmed Up

Zum Vergleich mit einem “warmed up Index “ wurde eine Hälfte der Testtexte benutzt, um Cashe-Einträge anzulegen, und die andere Hälfte wurde normal annotiert. Das Ergebnis ist ein deutlicher Anstieg in Geschwindigkeit.

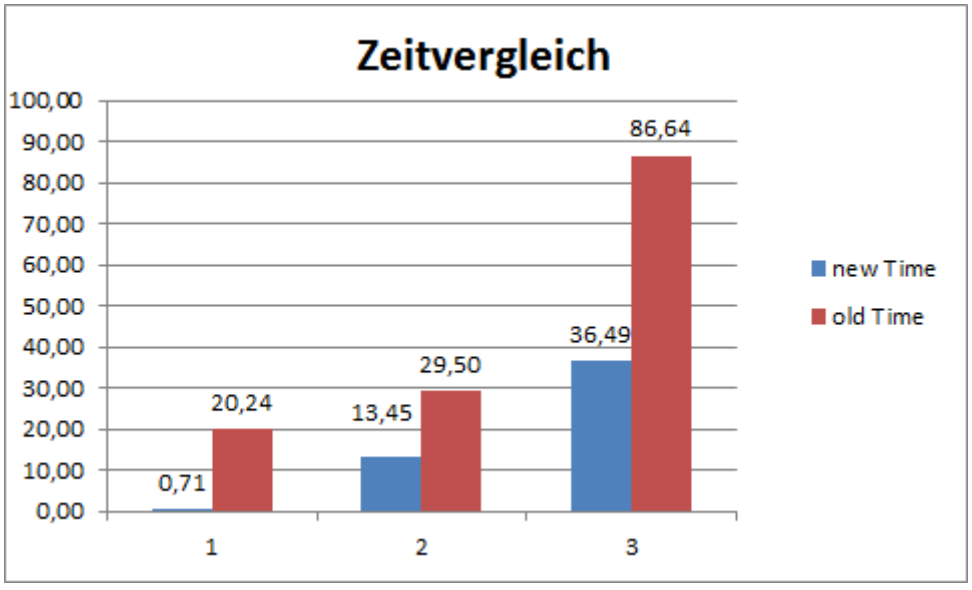

Abbildung 4.3: Zeitvergleich

Um diesen Effekt zu maximieren, sollten thematisch verwandte Texte sequentiell annotiert werden.

4.2 Genauigkeit

Nach der Betrachtung der Geschwindigkeit ist nun die Ergebnis-Qualität der einzelnen Operationen interessant. Dazu wurden die Ergebnisse des IBEL mit den Ergebnissen von DBpedia Spotlight vergleichen.

4.2.1 Titelsuche

Da die Titelsuche nur für exakte Übereinstimmungen ein Ergebnis liefert, werden die Entitäten nahezu immer richtig verlinkt. Eine Fehlverlinkung tritt nur auf, wenn eine Entität mit einem Titel übereinstimmt, aber dieser im semantischen Kontext eine andere Bedeutung hat.

Beispiel:

Tiger was lost in the woods when he got divorced from Elin.[2]

Der Stanford-NER liefert für diesen Satz die zwei Entitäten Tiger und Elin zurück. Die Titelsuche findet dann für Tiger die URI `http://dbpedia.org/resource/Tiger`. Richtig wäre in diesem Falle aber `http://dbpedia.org/resource/Tiger_Woods`.

4.2.2 Anchorsuche

Die Anchorsuche besitzt die selben Eigenschaften wie die Titelsuche. Ihre Ergebnisse sind sehr Präziese und nur falsch, wenn durch den semantische Kontext eine andere Bedeutung entsteht.

Anmerkung: Jeder Titel einer DBpedia-Seite ist auch gleichzeitig ein einzigartiger Anchor. Dies bedeutet das die Titelsuche ebenfalls eine Anchorsuche ist, aber nur auf der Menge der eindeutigen Anchors. Würde die Titelsuche deaktiviert werden, so würde IBEL trotzdem das gleiche Ergebnis zurück liefern, aber mehr Zeit in Anspruch nehmen.

4.2.3 Tf-Idf-Suche

Die tf-idf-Suche ist im Vergleich zu den ersten beiden Suchenmethoden sehr unzuverlässig. Sie ermittelt zwar eine URI für eine Seite, die zu dem semantischen Kontext

[2] Auszug des KORE50-Datensatzes des Max Planck Instituts

und der Entität passt, ist aber meistens eine falsche Verlinkung. Folgende Ursachen können dafür verantwortlich sein:

- Die Entität besitzt keine passende DBpedia-Seite.
- Der indexierte Abstract-Text der richtigen DBpedia-Seite ist zu kurz um eine Disambiguirung zu ermöglichen.
- Eine thematisch nahe verwandte DBpedia-Seite besitzt einen größeren Abstract-Text mit mehr Referenzen auf die Entität.
- Der Stanford-NER hat ein falsches Ergebnis geliefert.

Der letzte Fall ist besonders von Interesse, da die tf-idf-Suche eine sehr teure, optionale Operation ist. Bei der Verlinkung der Test-Texte wurde diese Suche für 220 Kandidaten verwendet. Circa 56% davon waren falsche Entitäten, wie zum Beispiel die Adjektive german, russian und jewish .

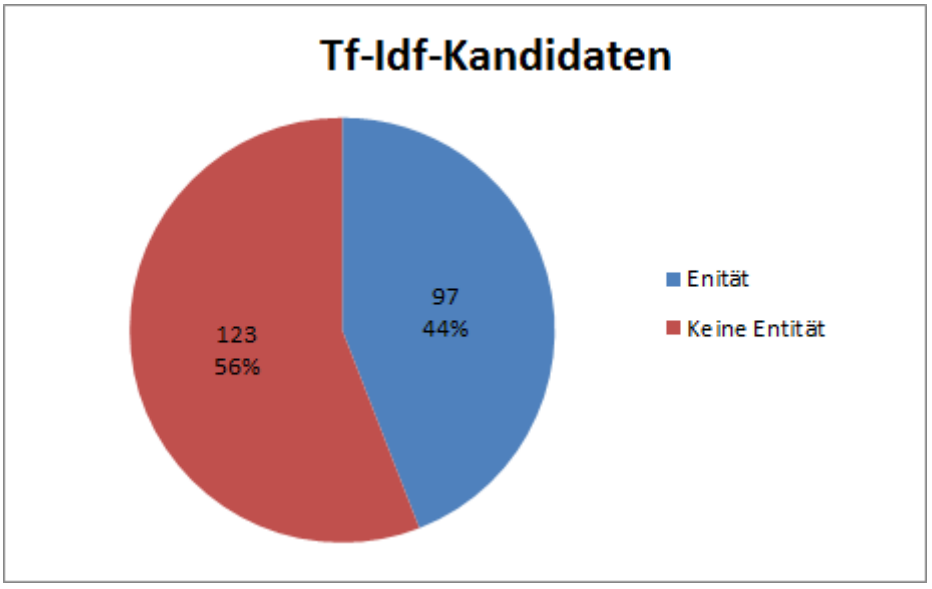

Abbildung 4.4: Tf-Idf-Kandidaten

4.3 Semantikanalyse mit KORE

Das Max Planck Institut für Informatik[3] bietet von Hand gefertigte Datensätze zum Testen von Disambiguirungsalgorithmen. Diese Keyphrase Overlap Relatedness for Entity Dismabiguation (KORE)[4] Testdaten sind so erstellt, dass sie besonders schwer zu disambiguieren sind.

Da nahezu alle vom Stanford-NER erkannten Entitäten in diesen Texten ihre Bedeutung nur über den semantischen Kontext erhalten und dies genau die Schwäche der Titel- und Anchorsuche sind, schneidet der Index Based Entity Linker (IBEL) schlecht ab.

Entäten ges.	Entäten gefunden	Anchors erkannt	richtig Verlinkt
148	130 (87,8 %)	75 (50,6 %)	26 (17,6 %)

Insgesamt wurden also nur 17,6% aller Entitäten richtig verlinkt. Zwar können über 50% mit einem Anchor verbunden werden, aber viele dieser Anchors sind eindeutig und bieten somit keine Möglichkeit zur semantischen Interpretation.

Beispiel:

David and Victoria added spice to their marriage.[5]

Gesucht werden die Entitäten David und Victoria Beckham. Diese besitzen aber nur Variationen der Schreibweise ihrer vollen Namen (z.B. David Beckam, David Beckham und David_Beckham) oder Dinge, die sie identifizieren (z.B Posh Spice, ein Album von Victoria Beckham) als Anchor, jedoch nicht David oder Victoria. Diese beiden sind nämlich für die URIs `http://dbpedia.org/resource/David` und `http://dbpedia.org/resource/Victoria` eindeutig.

Dies führt dazu, dass viele Entitäten falsch verlinkt werden.

[3] http://www.mpi-inf.mpg.de/index_d.php
[4] http://www.mpi-inf.mpg.de/yago-naga/aida/download/KORE50.tar.gz
[5] Auszug des KORE50-Datensatzes des Max Planck Instituts

4.4 Vergleich mit DBpedia

Der Index Based Entity Linker (IBEL) liefert für alle getesteten Texte ähnlich gute Ergebnisse. Jedoch ist DBpedia Spotlight IBEL im Bezug auf Semantikanalyse noch überlegen.

Entäten ges.	Entäten gefunden	richtig Verlinkt
148	46 (31 %)	30 (20,3 %)

Zwar schneidet DBpedia Spotlight mit 20% richtig verlinkten Entitäten ebenfalls nicht gut bei den KORE-Datensätzen ab, aber etwas besser als IBEL. Beide Programme haben also ähnliche Schwächen.

Kapitel 5

Deployment und Erweiterung auf andere Sprachen

5.1 Deployment

Um das Programm zu deployen muss lediglich die JAR mit compilierten Dependencies(falls das Programm aus dem Sourcecode compiliert wurde, ist dies die "IndexBasedEntityLinker-1.0-jar-with-dependencies.jar"), sowie ein passender Entity- und Abstract-Index in ein gleiches Verzeichnis kopiert werden.

5.1.1 Generierung der Indexe mit Utility-1.0

Sollten keine gültigen Indexe vorhanden sein, so können diese entweder manuell oder über die IndexBasedEntityLinker_Utility(IBELU) generiert werden.

Abbildung 5.1: Index Based Entity Linker Utility

Dafür werden folgende Dateien benötigt:

- Der DBpedia Anchor-Index der Computer Semantic Group der Universität Bielefeld
- Mapping-based Properties (en) von DBpedia[1]
- Extended Abstracts (en) von DBpedia[2]

Die Abfolge der auszführenden Operationen ist von links nach rechts sortiert und sollte auch in dieser Reihenfolge abgewickelt werden.
Zur Erstellung des Entity-Indexes werden der Anchor-Index und die Mapping-based Properties benötigt. Der Ablauf ist wie folgt:

1. Clean Properties: In dem zugehörigen Textfeld oberhalb des Buttons muss der Dateipfad (lokal oder absolut) zu der Mapping-Based Properties Datei angegeben werden. Diese wird dann von allen irrelevanten Daten gesäubert.
   Output: cleaned_properties.txt, cleaned_properties_neigborToEntity.txt und entities.txt.
2. Extract Anchors: Liest den Anchor-Index aus und schreibt alle <URI,anchor>-Paare in eine Datei.
   Output: anchors.txt
3. Pre BlockIndex: Erstellt für die Schnittmenge der URIs aus entities.txt und anchors.txt Dateien zur BlockIndex Generierung.
   Output:combined.txt, entity_anchors.txt, entity_neighbor_anchorsN.txt, neighbor_entity_anchorsE.txt
4. Create BlockIndex Erzeugt den BlockIndex(EntityIndex).
   Output : Blockindex

Zu Generierung des Abstract-Indexes werden die entities.txt aus dem ersten Teil und die Long-Abstracts benötigt:

[1] http://downloads.dbpedia.org/3.9/en/mappingbased_properties_en.nt.bz2
[2] http://downloads.dbpedia.org/3.9/en/long_abstracts_en.nt.bz2

1. Clean Abstracts: Nach Angabe des Dateipfades (lokal oder absolut) der long-abstracts-Datei werden für alle URIs die ein Abstract besitzen und in entities.txt vorkommen <URI, abstract>-Paare erstellt und gespeichert.
   Output: abstract_clean.txt

2. Create Abstract Index: Erstellt den Abstract-Index.
   Output:Abstract Index

Über Erfolg oder Misserfolg (Fehlermeldungen) der ausgeführten Operationen wird der Benutzer der IBELU über ein Konsolen-Feld (siehe Abbildung 5.1) informiert.

Anmerkung: Diese Operationen sind teilweise sehr speicherintensiv. Es sollten mindestens 12GB RAM zur Verfügung gestellt werden und sichergestellt sein, dass die JVM diesen auch nutzen darf (VM Parameter: -Xmx12g).

5.1.2 Eigenständige Generierung von Dateien

Die IBELU kann auch nur zur reinen Indexerstellung genutzt werden, während die dafür benötigten Dateien anderweitig erstellt werden. Im Nachfolgenden werden die Dateien und ihre Formate näher erläutert.

Dateien zur Erstellung des Entity-Indexes

Um den Index über die IBELU zu erstellen, sind zwei Dateien erforderlich:

- combined.txt: In dieser Datei werden alle URIs auf ihre Nachbarn und deren Anchors gemappt. Ein Nachbar einer URI A ist in diesem Fall eine URI B, deren Graphknoten eine Kante zu dem Knoten von A besitzt.
  Format:

$\texttt{URI}_1\texttt{|Neighbor}_1\texttt{|anchor}_{1,1}\texttt{;...;anchor}_{1,i}$
...
$\texttt{URI}_1\texttt{|Neighbor}_{m_1}\texttt{|anchor}_{m_1,1}\texttt{;...;anchor}_{m_1,j}$
...
...
$\texttt{URI}_n\texttt{|Neighbor}_{m_n}\texttt{|anchor}_{m_n,1}\texttt{;...;anchor}_{m_n,k}$

- entity_anchors.txt: Diese Datei mappt alle URIs auf ihre Anchors.

  Format:

  $\texttt{URI}_1\,|\,\texttt{anchor}_{1,1};\ldots;\texttt{anchor}_{1,i}$
  ...
  ...
  ...
  $\texttt{URI}_n\,|\,\texttt{anchor}_{n,1};\ldots;\texttt{anchor}_{n,j}$

Diese Dateien sollten lexikographisch sortiert sein, um den resultierenden Index performanter zu machen.

Anmerkung: Natürlich können auch die Indexe eigenständig erstellt werden. Aufbau und Funktionsweise werden in dem Kapitel „Implementierung “ausführlich behandelt.

5.2 Erweiterung um andere Sprachen

Um IBEL mit anderen Sprachen nutzen zu können, müssen neue Indexe erstellt und die existierende Entity-Recognition-Unit angepasst beziehungsweise ersetzt werden.

5.2.1 Indexe

Für jede zu unterstüzende Sprache muss ein neuer Entity-Index und ein neuer Abstract-Index erstellt werden. In Kapitel 5.1 wurde dies bereits ausführlich behandelt. Die dafür benötigten Mapping-Based Properties sowie die long-Abstracts werden von DBpedia in 119 verschieden Sprachen zur Verfügung gestellt.
Die entsprechenden Anchors für diese Sprache müssen allerdings entweder eigenständig generiert oder aus einer anderweitigen Quelle bezogen werden.

5.2.2 Entity Recognition

Für die Entity-Recognition-Unit kann, durch einen Austausch der Klassifizierer, weiterhin der Stanford-NER verwendet werden. Auf der Downloadseite des Stanford-NER[3] finden sich Klassifizierer für die deutsche und chinesische Sprache. Für andere Sprachen müssen eigene Klassifzierer trainiert werden. Das nötige Vorgehen dafür ist auf der FAQ-Seite[4] beschrieben. Um den neuen Klassifizierer nutzen zu können, muss der Ladepfad in der CLASSIFIER_PATH-Variable in der Klasse NER-Handler entsprechend angepasst werden.

Es wird auch jede andere Implementation von Named-Entity-Regocnition unterstütz. Dafür muss lediglich ein Adapter[5] geschrieben werden, welcher das EntityExtractor-Interface implementiert. Dieser muss dann in der Klasse App.java der GUI anstelle des NER_Handlers im Konsturktoraufruf übergeben werden.

Anmerkung: Da das Programm im Nachhinein in eine Serverapplikation für die Semantic Computer Group umgebaut wird, wird die GUI voraussichtlich nicht mehr verwendet werden. Daher muss der Adapter dann den NER_Hanlder an einer anderen Stelle ersetzen.

[3] http://nlp.stanford.edu/software/CRF-NER.shtml
[4] http://nlp.stanford.edu/software/crf-faq.shtml#a
[5] http://de.wikipedia.org/wiki/Adapter_(Entwurfsmuster)

Kapitel 6

Zusammenfassung und Ausblick

Zusammenfassend kann man sagen, dass der Index Based Entity Linker (IBEL) eine gute Alternative zu DBpedia Spotlight ist, aber in einigen Fällen noch ein wenig unpräzieser. Insgesamt ist es ein Programm, das schnell und zuverlässig Enititäten, die einen Anchor besitzen und durch den semantischen Kontext keine andere Bedeutung erhalten, annotieren kann. Wenn man thematisch verwandte Texte sequentiell annotiert, ist sogar ein deutlicher Anstieg in der Geschwindigkeit messbar, da IBEL über Cache-Einträge Daten schnell erneut Abrufen kann.
Im Bereich der fortgeschrittenen Semantikanalyse, wie sie die KORE-Datensätze fordern, besteht noch Verbesserungsbedarf, da die tf-idf-Suche noch sehr ungenau ist und eindeutige Anchors keinen Raum für semantische Interpretation lassen, obwohl es für diese Texte notwendig wäre. IBEL bietet an diesen Stellen noch viel Platz für Optimierung und Erweiterung. Unter anderem können folgende Vorschläge zur Verbesserung getestet werden :

- Längere und ausführlichere Abstract-Texte.
- Wenn für eine Entität kein Titel oder Anchor gefunden wird, anstelle der tf-idf-Suche einen Ähnlichkeitsabgleich mit anderen Anchors machen (bei der Anchor-Suche wird derzeit nur ein exakter Treffer verwendet) und dann eine.
- die tf-idf-Querygewichte in Abhänigkeit zu dem tf-idf-score dynamisch anpassen.
- Kombinationen der vorherigen Punkte.

Weiterhin sollte der Stanford-NER eventuell gegen einen anderen Name and Entity Recognitioner ausgetauscht werden, da dieser noch fehlerhafte Ergebnisse, wie zum Beispiel Adjektive, zurückliefert. Durch den modularen Aufbau von IBEL ist ein solcher Austausch[1] sehr einfach zu realisieren.

Schlussendlich kann der Index Based Entity Linker, wie in Kapitel 5 beschrieben, um weitere Sprachen erweitert werden. Dafür werden lediglich zwei zusätzliche Indexe pro Sprache und eine Option zum Umschalten zwischen den Indexen benötigt.

[1] siehe Kapitel 3.2

Anhang A

Evaluierungstabellen

Unwarmed Index

Source	number of titles	avg entryCheck	number of anchors	avg anchorTime	number of tf_idf	avg tf_idf_time
hitler	19	18,97	25	35,37	13	111,31
united states	46	12,27	35	30,5	22	93,61
obama	26	18	22	33,73	17	47,62
ww2	31	15,26	27	27,19	14	121,93
ww1	33	16,14	20	22,95	13	93,57
holy roman empire	15	31,71	10	37,58	5	150,99
germany	22	21,54	15	39,9	8	50,86
chirstianty	21	19,95	20	30,59	11	96,47
jews	23	16,78	26	37,75	15	50,05
communism	6	9,09	12	102,53	6	52,43
canda	8	35,46	14	49	8	121,11
nato	18	20,13	22	31,67	11	99,14
origins of cold war	29	13,28	37	17,54	17	107,32
twelfe olypmian	20	13,72	39	19,34	35	62,08
gandhi	15	23,19	21	27,15	12	90,53
american civil war	20	20,37	22	30,76	13	83,89
napoleon	35	11,71	42	18,47	25	87,74
goethe	14	28,27	13	34,95	8	55,05
isaac newton	34	16,86	14	23,51	9	119,3
schiller	16	19,4	24	42,57	17	40,22

Warmed Index

source	number of titles	avg entryCheck	number of anchors	avg anchorTime	number of tf_idf	avg tf_idf_time
canda	8	0,76	14	20,21	8	38,8
nato	18	0,73	22	12,27	11	36,6
origins of cold war	29	0,76	37	15,38	17	37,5
twelfe olypmian	20	0,36	39	13,72	35	29,6
gandhi	15	0,7	21	15,15	12	38,45
american civil war	20	1,3	22	12,98	13	36,46
napoleon	14	0,78	13	10,1	8	33,55
goethe	34	0,61	14	8,3	9	48
isaac newton	9	0,65	17	13,87	8	38,27
schiller	15	0,43	24	12,48	17	27,63

Literaturverzeichnis

[BLK+09] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Chirstian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point fro the web of data. 2009.

[DMP13] J. Daiber, Jakob M., and Mendes P.N. Improving efficiency and accuracy in multilingual entity extraction. 2013.

[MC07] Rada Mihalcea and Andreas Csomai. Wikify! Linking Documents to Encyclopedic Knowledge. 2007.

[MLAH12] Mohamed Morsey, Jens Lehmann, Sören Auer, and Sebastian Hellmann. DBpedia and the live extraction of structured data from Wikipedia. 2012.

[MW08] David Milne and Ian H. Witten. Learning to link with Wikipedia. 2008.

[PJMC11] Mendes P.N., Daiber J., Jakob M, and Bizer C. Evaluating DBpedia Spotlight for the TAC-KBP Entity Linking Task. 2011.

[WST10] Zhang Wei, Jian Su, and Chew Lim Tan. Entity linking leveraging automaticly generated annotation. 2010.

Internetquellen

- http://maven.apache.org/
- http://nlp.stanford.edu/software/CRF-NER.shtml
- http://lucene.apache.org/core/
- www.wikipedia.org

- http://www.mpi-inf.mpg.de/index_d.php
- http://www.mpi-inf.mpg.de/yago-naga/aida/download/KORE50.tar.gz
- http://de.wikipedia.org/wiki/Adapter_(Entwurfsmuster)
- http://nlp.stanford.edu/software/crf-faq.shtml#a
- dbpedia.org
- http://www.w3.org/DesignIssues/LinkedData.html
- spotlight.dbpedia.org
- http://webknox.com/p/linked-open-data-visualization
- http://www.sc.cit-ec.uni-bielefeld.de/