Untersuchungen zur Kombination von klassischer und semantischer Textindizierung

Diplomarbeit

von

Torben Rebhan
Mat.-Nr.: 22793

Abgabe am 09. Juni 2009

Erstprüfer: Prof. Dr. rer.-nat. habil. Ralf Möller
Zweitprüfer: Prof. Dr.-Ing. Dr. rer. nat. habil. Karl-Heinz Zimmermann
Betreuer: M.Sc. Atila Kaya

Technische Universität Hamburg-Harburg
Institut für Softwaresysteme

Eidesstattliche Versicherung

Ich versichere, daß ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen oder anderen Quellen entnommen sind, sind als solche eindeutig kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht veröffentlicht und noch keiner Prüfungsbehörde vorgelegt worden.

Hamburg, den 09. Juni 2009

_____

Torben Rebhan

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Das Suchen von Informationen in einer Menge von Daten ist eine häufig auftretende Problemstellung. Dabei entscheidet die Form der vorliegenden Daten grundlegend über die möglichen Lösungen. Man kann die Formen grob in zwei Kategorien einteilen:

1. Strukturiert vorliegend (zum Beispiel in einer Datenbank)
2. Unstrukturiert vorliegend (zum Beispiel Textdateien mit Fließtext oder HTML-Dateien)

Im Weiteren gehe ich auf den zweiten Fall ein, welcher zum Beispiel Anwendungen in digitalen Nachschlagewerken, Hilfesystemen oder im Paradefall Suchmaschinen findet. Dabei bedeutet „unstrukturiert“, dass es sich um Texte mit dem Schwerpunkt der Interpretierbarkeit durch Menschen handelt (das HTML-Format besitzt auch eine Struktur, welche aber zum größten Teil nur zur Darstellung der Inhalte verwendet wird).

Suchen in unstrukturierten Daten hat besonders durch das World Wide Web rasant an Bedeutung gewonnen. So hatte die Firma Google Inc. im Jahre 2004 laut eigenen Angaben mehr als acht Milliarden Webseiten indiziert (siehe [8]) und ist damit zu dem populärsten Anlaufpunkt für das Suchen von Informationen im WWW geworden.

Die Techniken und Algorithmen zur Indizierung von unstrukturierten Daten sind schon seit einiger Zeit Forschungsgegenstand, und durch immer effizientere Algorithmen hochgradig optimiert worden. Parallel dazu ist die Leistung der Computerhardware gestiegen, so dass immer größere Mengen an Daten schnell durchsucht werden können.

Das Problem der Textindizierung ist, dass sich die Suche auf rein syntaktische Vergleiche beschränkt. Die Information, welche gesucht wird, muss demnach wörtlich im Text vorkommen, die Bedeutung spielt keine Rolle. So können implizite Informationen nicht berücksichtigt werden. Weitere Probleme treten zum Beispiel bei Homonymen und Synonymen auf, welche nicht ohne Weiteres behoben werden können. Ein Beispiel ist das Wort „Tau“, welches entweder ein Seil oder durch Temperaturunterschiede erzeugter Niederschlag bezeichnet. Da der jeweilige Kontext nicht berücksichtigt werden kann, werden bei einer entsprechenden Suche eventuell irrelevante Dokumente zurück gegeben oder relevante Dokumente nicht gefunden.

Die semantische Indizierung ist in der Lage, in einem bestimmten Umfang Informationen aus den vorliegenden Daten zu generieren, welche auf den Inhalt und nicht auf den Syntax abzielen (die Grundlagen werden in Abschnitt 2.2 beschrieben). Dazu ist allerdings Vorwissen über den Kontext der zu indizierenden Texte notwendig.

Ziel dieser Arbeit ist, zu untersuchen, ob sich semantische und klassische Textindizierung sinnvoll kombinieren lassen, um die Vorteile von Beiden zu vereinen.

1.2 Aufgabenstellung

In dieser Arbeit sollen Untersuchungen darüber angestellt werden, ob sich Ergebnisse von Suchanfragen durch eine Kombination von semantischer und klassischer Textindizierung verbessern lassen. Dazu soll ein Methode entwickelt werden, die zu indizierenden Texte mit vorhandenen, semantischen Information anzureichern. Diese Methode soll in einem Programm, welches indizieren und suchen kann, umgesetzt werden. Zur Verbesserung der Suchen soll Stemming[1] verwendet werden.

Um eine Veränderung der Suchen zu erkennen, muss eine Menge von Testsuchanfragen definiert werden, die entsprechenden Ergebnisse müssen dann ausgewertet werden. Für diese Auswertung sollen Kriterien erarbeitet werden.

1.3 Struktur der Arbeit

Der folgende Teil der Arbeit gliedert sich in vier Teile:

1. Theoretische Grundlagen der semantischen und klassischen Textindizierung, des Stemmings und der Bewertung von Ergebnissen (Kapitel 2)
2. Praktische Durchführung mit Konzeption des Programms, Beschreibung der wichtigsten Programmteile und Durchführung der Testsuchen (Kapitel 3)
3. Auswertung und Diskussion der Ergebnisse (Kapitel 4)
4. Zusammenfassung der Arbeit und Vorstellung möglicher weiterer Themen (Kapitel 5)

Zusätzlich befindet sich im Anhang die Liste der verwendeten Testsuchanfragen sowie eine Statistik mit den Ergebnissen.

[1] Rückführung von Wörtern auf ihre Grundformen, siehe Abschnitt 2.3

2 Theoretische Grundlagen

2.1 Textindizierung

In diesem Abschnitt möchte ich auf zwei grundlegenden Mechanismen und Vorgehensweisen bei der Textindizierung eingehen:

1. Boolesches Retrieval
2. Vektorraum-Retrieval

Dazu gehört die Erstellung des Suchindexes und das eigentliche Suchen darin. Eine detailreichere Beschreibung findet man unter anderem in [9].

Zur Beschreibung der unten genannten Verfahren werden folgende Begriffe verwendet:

Term: Ein Term ist die kleinste, suchbare Einheit. In vielen Fällen ist ein Term ein Wort im Ursprungstext.

Dokument: Ein Dokument ist die Einheit, in der die zu indizierende Information vorliegt (beispielsweise Dateien).

Token: Ein Text wird in Token zerlegt, ein Token wird in der Regel zu einem Term.

2.1.1 Boolesches Retrieval

Boolesches Retrieval zielt darauf ab, Anfragen mit booleschen Operatoren (AND, OR, NOT) zu beantworten. Dazu wird ein sogenannter invertierter Index generiert, welcher einen Bezug zwischen Termen und Dokumenten herstellt.

Um aus den gegebenen Dokumenten Terme zu extrahieren, wird der Text in Token aufgeteilt, eine einfache Möglichkeit besteht dabei darin, Leer- und Satzzeichen als Trennzeichen zu verwenden. Als Beispiel soll nun folgender Textausschnitt aus einer Nachrichtenmeldung der International Association of Athletics Federations (IAAF) dienen:

```
Franka Dietzsch, the World champion in the women's Discus Throw,
provided the only German win of the day with a second round release
of 64.47m, enough to defeat surprise European champion Darya
Pishchalnikova of Russia (62.17m).
```

Eine mögliche Aufteilung in Token sieht mit der oben genannten Methode folgendermaßen aus:

```
|Franka|Dietzsch|the|World|champion|in|the|women|s|Discus|Throw
|provided|the|only|German|win|of|the|day|with|a|second|round|release
|of|64|47m|enough|to|defeat|surprise|European|champion|Darya
|Pishchalnikova|of|Russia|62|17m|
```

Hier werden auch gleich einige Probleme deutlich, die dieses Verfahren mit sich bringt, so bleiben Zahlen mit Dezimaltrennzeichen nicht erhalten und es entstehen sinnlose Token wie das „s“ nach „women“. Diese Probleme können nur gelöst werden, wenn Hintergrundwissen über die verwendete Sprache vorhanden ist.

Zusätzlich werden häufig vorkommende Wörter in der Regel entfernt, da diese den Index unnötig vergrößern würden, ohne zu sinnvollen Suchergebnissen beizutragen. Diese Stopwortlisten müssen sprachabhängig angelegt werden. Mit Berücksichtigung der genannten Punkte (inklusive Wissen über Zahlen und Apostrophe) ergibt sich folgende Aufteilung:

```
|Franka|Dietzsch|World|champion|women's|Discus|Throw|provided|
only|German|win|day|second|round|release|64,47m|enough|defeat|
surprise|European|champion|Darya|Pishchalnikova|Russia|62,17m|
```

Mit diesem und weiteren in Token zerlegte Dokumente lässt sich eine sogenannte Term-Dokument-Matrix aufstellen, in der das Vorkommen eines Terms in einem Dokument hinterlegt wird. Mit dem Beispiel in Tabelle 2.1 (mit fiktiven Dokumenten) können

	Dokumente			
Terme	News2004-1	News2004-2	News2004-3	News2004-4
`World`	1	1	0	0
`champion`	1	0	0	1
`Discus`	1	0	0	1
`throw`	1	1	0	1
`German`	1	1	1	0

Tabelle 2.1: Ein Beispiel für eine Term-Dokument-Matrix mit den Einträgen für „kommt vor“ (1) und „kommt nicht vor“ (0)

jetzt boolesche Anfragen wie zum Beispiel „`Discus AND throw AND NOT German`“ auf folgende Art beantwortet werden:

1. Die Vektoren für die einzelnen Terme in der Anfrage der Term-Dokument-Matrix entnehmen (`Discus:` 1001, `throw:` 1101 und `German:` 1110).
2. Den Vektor für `German` negieren: 0001.
3. Alle Termvektoren mit einem bitweisen UND verknüpfen: $1001 \wedge 1101 \wedge 0001 = 0001$.

Damit passt nur das Dokument „News2004-4“ zu der Anfrage und wird als Treffer zurückgegeben.

Die Term-Dokument-Matrix ist in der Regel dünn besetzt[1], so wird bei der Speicherung viel Platz verschwendet. Die Speicherung als invertierte Index löst das Problem, in dem die Verknüpfung zwischen Term und Dokument in einer wörterbuchähnlichen Struktur abgelegt wird. Zu diesem Zweck wird für jedes Dokument eine eindeutige Dokumentnummer vergeben (DocID), ein Eintrag besteht so aus dem Term und einer Liste von DocIDs. Zusätzlich können noch weitere Informationen abgespeichert werden, wie zum Beispiel:

Dokumentenfrequenz: Die Anzahl der Dokumente, in denen der Term vorkommt.

Termfrequenz: Die Anzahl der Vorkommen eines Term im Dokument.

Termpositionen: Die Positionen, an denen der Term im Dokument vorkommt.

Terme	Dok.-Freq.	DocID-Liste
`champion`	2	$1 \rightarrow 4$
`Discus`	2	$1 \rightarrow 4$
`German`	3	$1 \rightarrow 2 \rightarrow 3$
`throw`	3	$1 \rightarrow 2 \rightarrow 4$
`World`	2	$1 \rightarrow 2$

Tabelle 2.2: Der invertierte Index zu der in 2.1 dargestellten Term-Dokument-Matrix, die Terme sind hier alphabetisch sortiert.

In Tabelle 2.2 ist der zu 2.1 korrespondierende invertierte Index dargestellt. Einfache Anfragen wie „`Discus AND throw AND NOT German`“ können damit auf folgende Art und Weise beantwortet werden:

1. Die Liste der DocIDs im Wörterbuch nachschlagen: (`Discus:` $1 \rightarrow 4$, `throw:` $1 \rightarrow 2 \rightarrow 4$, `German:` $1 \rightarrow 2 \rightarrow 3$).

2. Die Liste für `German` invertieren: (4).

3. Die Schnittmenge aus allen Listen bilden: (4).

Damit bildet das Dokument mit der DocID vier das einzige Resultat, welches selbstverständlich das selbe wie oben ist. Allerdings ist die Methode zum Finden der Schnittmenge der Listen bei beliebigen Anfragen nicht trivial. Besonders die Invertierung bei einem logischen NOT ist bei einer Vielzahl von Dokumenten im Index kostenintensiv. Deswegen kommen meistens speziell optimierte Algorithmen zum Einsatz (für Beispiele siehe [9]).

[1] Sie enthält viele Nullen und wenig Einsen.

2.1.2 Vektorraum-Retrieval

Das Vektorraum-Retrieval verfolgt einen ganz anderen Ansatz als das boolesche Retrieval. Hier wird versucht, möglichst relevante Ergebnisse zu frei formulierbaren Suchanfragen zu finden. Das Aufteilen der Dokumente in Token und das Extrahieren der Terme wird wie in Abschnitt 2.1.1 durchgeführt. Dann wird jeder in einem Dokument auftretender Term gewichtet. Dieses Gewicht ist im einfachsten Fall die Termfrequenz (tf), dabei spielt die Reihenfolge, in der die Terme auftreten keine Rolle. Diese Art der Gewichtung hat allerdings das Problem, dass alle Terme gleich behandelt werden, was bei überproportional häufig auftretenden Termen zu einer Übergewichtung führen kann. Durch das beschriebene Verwenden von Stopwortlisten können generell häufig auftretende Wörter ausgeschlossen werden, das Problem bleibt bei thematisch häufig auftretenden Termen bestehen (zum Beispiel kommt in Dokumenten der Sportartikelhersteller das Wort „Ball“ sehr häufig vor). Deswegen müssen die Gewichte dieser Terme in Abhängigkeit ihres Auftretens verringert werden, dazu gibt es grundlegend zwei Möglichkeiten:

1. Terme mit hoher Sammlungsfrequenz (Summe der Termfrequenzen über alle Dokumente) schwächer gewichten.

2. Terme mit hoher Dokumentenfrequenz (df) schwächer gewichten.

Möglichkeit zwei wird am häufigsten verwendet und wird hier weiter beschrieben. Damit wird die inverse Dokumentenfrequenz (idf) eines Terms t definiert als:

$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}.$$

Dabei ist N die Anzahl aller Dokumente im Index.

Durch Kombinierung der Term- und der inversen Dokumentenfrequenz ergibt sich die sogenannte tf-idf-Gewichtung eines Terms t in einem Dokument d folgendermaßen:

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \cdot \mathrm{idf}_t.$$

Mit dieser Gewichtung wird ein Term genau dann stark gewichtet, wenn er sehr häufig in einer kleinen Zahl von Dokumenten vorkommt. Je größer die Zahl der Dokumente wird, desto schwächer wird der Term gewichtet.

Betrachtet man nun die Dokumente als Vektoren mit den Gewichten aller Terme im Index als Elemente, bilden diese einen Vektorraum, welcher die Grundlage des Vektorraum-Retrievals ist. Dabei wird für alle Terme, welche nicht im entsprechenden Dokument vorkommen, null als Gewicht festgelegt.

Folgende kurze Sätze sollen als Beispiel dienen:

1. `Maryam Jamal Yussuf of Bahrain ran the fastest time in the world this year.`

2. `The World Record was set by Maryam Jamal Yussuf in the year 2005.`

3. `Maryam Jamal Yussuf of Bahrain was the fastest female runner of the year 2005.`

Wie in Abschnitt 2.1.1 beschrieben werden Stopwörter entfernt und die Sätze in Token aufgeteilt. Zusätzlich werden noch alle Buchstaben in Kleinbuchstaben umgewandelt. Man erhält folgende Terme:

1. `maryam, jamal, yussuf, bahrain, ran, fastest, time, world, year`
2. `world, record, set, maryam, jamal, yussuf, year, 2005`
3. `maryam, jamal, yussuf, bahrain, fastest, female, runner, year, 2005`

Term	df_t	idf_t
2005	2	0,18
bahrain	2	0,18
fastest	2	0,18
female	1	0,47
jamal	3	0
maryam	3	0
ran	1	0,47
record	1	0,47
runner	1	0,47
set	1	0,47
time	1	0,47
world	2	0,18
year	3	0
yussuf	3	0

Tabelle 2.3: Werte für die Dokumentenfrequenz (df_t) und die inverse Dokumentenfrequenz (idf_t)

Tabelle 2.3 zeigt die Werte für df_t und idf_t für alle Terme, da die Termfrequenz für alle Terme in allen Dokumenten eins ist, gilt hier $\text{tf-idf}_{t,d} = idf_t$. Mit diesen Werten sind die Vektoren für die drei Dokumente wie folgt:

1. $\begin{pmatrix} 0 & 0,18 & 0,18 & 0 & 0 & 0 & 0,47 & 0 & 0 & 0 & 0,47 & 0,18 & 0 & 0 \end{pmatrix}^T$
2. $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0,47 & 0 & 0,47 & 0 & 0,18 & 0 & 0 \end{pmatrix}^T$
3. $\begin{pmatrix} 0,18 & 0,18 & 0,18 & 0,47 & 0 & 0 & 0 & 0 & 0,47 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}^T$

Um eine Suche durchzuführen, wird die Suchanfrage analog zu den Dokumenten in Terme aufgeteilt und die entsprechende Vektorrepräsentation berechnet. Sei $\vec{V}(x)$ eine

Vektorrepräsentation von x, q eine Suchanfrage und d ein Dokument, dann wird die sogenannte Kosinus-Score definiert als:

$$\text{score}(q,d) = \frac{\overrightarrow{V}(q) \cdot \overrightarrow{V}(d)}{|\overrightarrow{V}(q)||\overrightarrow{V}(d)|}.$$

Der verwendete Operator im Zähler ist das Skalarprodukt der Vektoren.

Die Kosinus-Score ist eine Möglichkeit, über die Winkel der Vektoren ein Maß für die Ähnlichkeit der Dokumente zur Suchanfrage zu berechnen. Als Beispiel soll in den gegebenen Dokumenten mit der Anfrage:

```
Which runner has set a record this year?
```

gesucht werden. Die Terme dieses Textes sind: „`runner`“, „`set`“, „`record`“, „`year`“. Mit den gegebenen Dokumentenfrequenzen ergibt sich die Vektorrepräsentation zu:

$$\overrightarrow{V}(q) = \begin{pmatrix} 0 & \dots & 0 & 0,47 & 0,47 & 0,47 & 0 & \dots & 0 \end{pmatrix}^T.$$

In Tabelle 2.4 sind die berechneten Ergebnisse aufgelistet. Im Gegensatz zu dem

Position	Dokument	Score
1	`The World Record was set by Maryam Jamal Yussuf in the year 2005.`	0,79
2	`Maryam Jamal Yussuf of Bahrain was the fastest female runner of the year 2005.`	0,37
3	`Maryam Jamal Yussuf of Bahrain ran the fastest time in the world this year.`	0

Tabelle 2.4: Ergebnisse der Beispielanfrage

Booleschen Retrieval ergibt sich für die Liste der Ergebnisse durch die Score eine Ordnung.

2.2 Semantische Indizierung

Dieser Abschnitt beinhaltet eine kurz gehaltene Einführung der für diese Arbeit wichtigen Teile der Beschreibungslogik, eine Beschreibung des BOEMIE-Projektes[2] und die

[2] Bootstrapping Ontology Evolution with Multimedia Information Extraction

theoretische Ausarbeitung des Anreicherungsprozesses, welcher die von BOEMIE generierten semantischen Daten sukzessive in den normalen Indizierungsvorgang einbringen soll. Die Implementierung wird dann in Kapitel 3 beschrieben. Eine ausführliche Beschreibung von Beschreibungslogiken findet man in [1].

2.2.1 Kurze Einführung in die Beschreibungslogik

Die Beschreibungslogik (Description Logics, DL) ist eine Familie von formalen Sprachen zur Repräsentation von Wissen. Die grundlegenden Elemente von DLs sind Konzepte, Rollen und Individuen. Die Konzepte und Rollen bilden zusammen ein Modell der gegebenen Domäne (die Terminologie der Domäne), diese Menge wird „terminological box“ (TBox) genannt. Darin können die Konzepte eine hierarchische Struktur aufweisen (Subsumption). Die Menge der Individuen und ihre konkreten Beziehungen repräsentiert den Zustand der modellierten Welt und wird „assertional box“ (ABox) genannt. Zusätzlich wurde die Möglichkeit vorgesehen, Individuen über Rollen konkrete Werte zuweisen zu können (sogenannte konkrete Domänen). Eine TBox und eine korrespondierende ABox bilden zusammen die sogenannte „knowledge base“ (KB).

Beispielsweise könnte es in einer Sport-Domäne die Konzepte „`SportsEvent`“ für eine Sportveranstaltung und „`SportsEventName`“ für den Namen einer Sportveranstaltung geben. Eine Rolle zur Zuweisung eines Namens zu einer Sportveranstaltung hieße dann eventuell „`hasSportsEventName`“, und den konkreten Namen der Sportveranstaltung würde die Eigenschaft „`hasSportsEventNameValue`“ beinhalten.

Die verschiedenen DLs bauen auf einander auf und sind nach Ausdrucksmächtigkeit sortiert auszugsweise hier dargestellt:

$\mathcal{AL}$ (Attribute Language), beinhaltet die Negation von atomaren Konzepten, Konjunktion, Universalquantifikation und unqualifizierende Existenzquantifikation.

$\mathcal{ALC}$ mit der zusätzlichen Möglichkeit, komplexe Konzepte zu negieren.

$\mathcal{ALCQ}$ mit der zusätzlichen Möglichkeit, zahlenmäßig einschränkbare Existenzquantoren zu verwenden.

$\mathcal{ALCQ}(\mathcal{D})$ mit der zusätzlichen Möglichkeit, konkrete Domänen zu verwenden.

Im Folgenden wird auf $\mathcal{ALCQ}(\mathcal{D})$ weiter eingegangen. Die Grammatik zum Definieren von komplexen Konzepten wird in Tabelle 2.5 dargestellt, damit können Ausdrücke für Konzepte in Form von terminologischen Axiomen definiert werden. Terminologische Axiome haben im allgemeinen Fall die Form:

$$C \sqsubseteq D \ (R \sqsubseteq S) \text{ oder } C \equiv D \ (R \equiv S).$$

Damit können symbolische Namen für komplexere Ausdrücke definiert werden. Zum Beispiel könnte das Symbol für einen Athleten folgendermaßen aussehen:

$$\text{Athlete} \equiv \text{Person} \sqcap \exists \text{participatesIn.SportsEvent}.$$

Syntax	Bedeutung
$C,\ D \longrightarrow A$	Atomares Konzept
$\top$	Universelles Konzept (Top)
$\bot$	Bottom Konzept
$\neg A$	Atomare Negation
$C \sqcap D$	Schnittmenge
$\forall R.C$	Wertebeschränkung
$\exists R.C$	Qualifizierte Existenzbeschränkung
$\exists_{\leq n} R.C$	Qualifizierte Min.-Beschränkung
$\exists_{\geq n} R.C$	Qualifizierte Max.-Beschränkung

Tabelle 2.5: $\mathcal{ALCQ}(\mathcal{D})$-Grammatik

Die Semantik der DL wird durch eine Interpretation $\mathcal{I}_\mathcal{D}$ gegeben, welche aus den zwei Mengen $\Delta^\mathcal{I}$, $\Delta^\mathcal{D}$ und einer Funktion $\cdot^\mathcal{I}$ besteht. $\Delta^\mathcal{I}$ ist die Menge aller Individuen, $\Delta^\mathcal{D}$ ist die Menge aller konkreten Objekte. Beide Mengen sind nichtleer und disjunkt. Damit lässt sich $\mathcal{I}_\mathcal{D}$ schreiben als:

$$\mathcal{I}_\mathcal{D} = \left(\Delta^\mathcal{I}, \Delta^\mathcal{D}, \cdot^\mathcal{I}\right).$$

$\cdot^\mathcal{I}$ ist die sog. Interpretationsfunktion und weist jedem atomaren Konzept C die Menge $C^\mathcal{I} \subseteq \Delta^\mathcal{I}$ und jeder Rolle R die Tupelmenge $R^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$ zu. Weiterhin gilt:

$$\begin{aligned}
\top^\mathcal{I} &= \Delta^\mathcal{I} \\
\bot^\mathcal{I} &= \emptyset \\
(C \sqcap D)^\mathcal{I} &= C^\mathcal{I} \cap D^\mathcal{I} \\
(C \sqcup D)^\mathcal{I} &= C^\mathcal{I} \cup D^\mathcal{I} \\
(\neg C)^\mathcal{I} &= \Delta^\mathcal{I} \setminus C^\mathcal{I} \\
(\exists R.C)^\mathcal{I} &= \left\{x | \text{Es existiert ein } y \in \Delta^\mathcal{I}, \text{ für das gilt } (x,y) \in R^\mathcal{I} \text{ und } y \in C^\mathcal{I}\right\} \\
(\forall R.C)^\mathcal{I} &= \left\{x | \text{Für alle } y \in \Delta^\mathcal{I} \text{ gilt, wenn } (x,y) \in R^\mathcal{I} \text{ dann } y \in C^\mathcal{I}\right\} \\
(\exists_{\leq n} R.C)^\mathcal{I} &= \left\{x | \sharp\left\{y | (x,y) \in R^\mathcal{I} \text{ und } y \in C^\mathcal{I}\right\} \leq n\right\} \\
(\exists_{\geq n} R.C)^\mathcal{I} &= \left\{x | \sharp\left\{y | (x,y) \in R^\mathcal{I} \text{ und } y \in C^\mathcal{I}\right\} \geq n\right\}
\end{aligned}$$

In einer ABox können nun Individuen Konzepten und Rollen durch sogenannte Concept Assertions zugewiesen werden. Zum Beispiel werden durch:

$$\begin{aligned}
&\text{Person}(\text{person}_1) \\
&\text{PersonName}(\text{name}_1) \\
&\text{hasPersonName}(\text{person}_1, \text{name}_1) \\
&\text{hasPersonNameValue}(\text{name}_1, \text{'Meseret Defar'})
\end{aligned}$$

die Individuen „person_1“ als Person und „name_1“ als Personenname deklariert. Beide Individuen stehen durch die Rolle „hasPersonName“ in Beziehung, und „name_1“

wird der String „Meseret Defar“ als konkreter Wert (sogenannter Datatype Filler) zugewiesen.

Systeme, welche DL verwenden, bieten in der Regel Dienste für Anfragen über die gegebene KB (Reasoning) an und implementieren die entsprechenden Algorithmen. Folgende Funktionen gehören unter anderem zu den Standard Inference Services:

- Test auf Konsistenz (existiert ein gemeinsames Modell für die TBox und die ABox der gegebenen KB.)
- Subsumptionstest (stammen zwei gegebene Konzepte von einander ab.)
- Erfüllbarkeitstest (kann ein gegebenes Konzept überhaupt instanziert werden.)
- Instanzüberprüfung (ist ein gegebenes Individuum eine Instanz eines gegebenen Konzeptes.)

Zusätzlich zu diesen formalen Entscheidungsproblemen dienen die Retrieval Inference Services dazu, durch komplexe Anfragen Informationen aus der KB zu gewinnen. Dazu gibt es die konjunktive Anfragen (Conjunctive Queries), welche die generelle Form:

$$\{(\text{Kopf})|\text{Körper}\}$$

haben. Der Kopf besteht dabei aus Variablen, an welche die Antworten (soweit existent) des DL-Systems gebunden werden. Jede Variable muss mindestens einmal im Körper vorkommen, welcher ansonsten durch die sogenannten Anfrageatome (Query Atoms) gebildet wird. Mögliche Anfrageatome sind:

- Konzeptanfragen $C(X)$
- Rollenanfragen $R(X, Y)$
- Festlegung von Gleichheit $X = Y$

Beispielsweise würde folgende Anfrage:

$$\{(X, Y)|\text{Person}(X), \text{PersonName}(Y), \text{hasPersonName}(X, Y)\}$$

alle Paare von Personen und Personenname zurückgeben, welche einander zugeordnet sind.

2.2.2 Überblick über das BOEMIE-Projekt

BOEMIE ist ein von der Europäischen Union gefördertes Forschungs- und Entwicklungsprojekt, die Abkürzung BOEMIE steht für Bootstrapping Ontology Evolution with Multimedia Information Extraction (siehe [5]). Das Ziel von BOEMIE ist, automatisiert Wissen aus Multimedia-Inhalten zu gewinnen. Als Grundlage dient dabei ein Repository, welches HTML-Seiten, Bilder und Videos beinhaltet. Diese Dateien wurden von verschiedenen Webseiten über Leichtathletik (unter anderem von der Seite der IAAF[3] und der USATF[4]) gesammelt.

Das Hintergrundwissen besteht aus den folgenden drei Ontologien:

[3] International Association of Athletics Federations
[4] USA Track & Field

- Leichtathletik (Athletics Event Ontology, AEO)
- Multimedia (Multimedia Content Ontology, MCO)
- Geographie (Geographic Information Ontology, GIO)

Die Leichtathletik-Ontologie enthält das Hintergrundwissen der entsprechenden Sportdomäne mit Sportlern, Sportarten und so weiter. Die Multimedia-Ontologie bildet die einzelnen Medien und deren Zusammenhänge ab. Wissen über geographische Zusammenhänge befinden sich in der Geographie-Ontologie. Alle drei Ontologien sind TBoxen. In Abbildung 2.1 ist ein Teil der aus den drei Einzelontologien kombinierten Ontologie abgebildet.

Die Verarbeitung der Multimediainhalte erfolgt zyklisch. In jedem Zyklus werden folgende drei Schritte durchgeführt:

1. Extraktion von Informationen aus den Multimediadaten mit Hilfe von Textanalyse, Bild- und Videoerkennung, die Ergebnisse sind Analyse-ABoxen.
2. Interpretation der in Schritt eins generierten Analyse-ABoxen, generiert werden Interpretations-ABoxen.
3. Evolution der vorhandenen Ontologien mit Hilfe der Interpretations-ABoxen und Techniken des maschinellen Lernens.

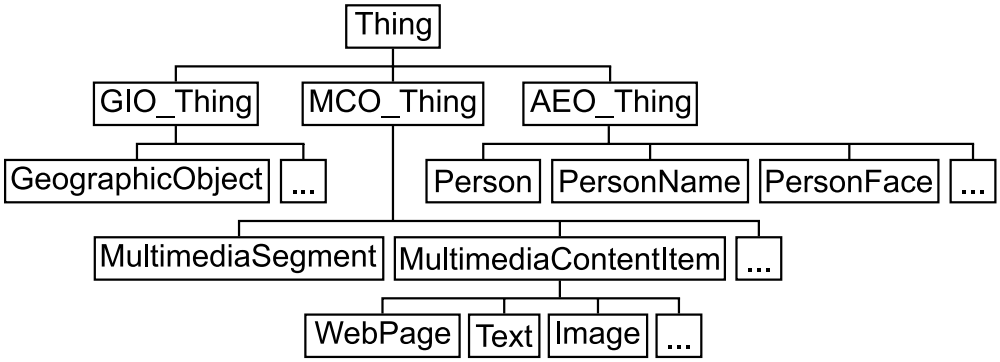


Abbildung 2.1: Ein Auszug aus den Ontologien

In Schritt eins fließen zusätzlich geographische Informationen eines externen Geoinformationssystems (GIS) mit ein. Der zweite Schritt nutzt ein im Rahmen von BOEMIE entwickeltes Programm namens Semantic Interpretation Engine, welches basierend auf den in Schritt eins generierten Low-Level-Annotationen Deep-Level-Annotationen generiert und dann die einzelnen Analyseergebnisse der verschiedenen Modalitäten in einer HTML-Seite in eine einzige Interpretations-ABox verschmilzt. Low-Level-Annotationen sind zum Beispiel ein im Text erkannter Name einer Person oder ein in einem Bild erkanntes Gesicht. Die korrespondierende Deep-Level-Annotation ist dann das Individuum, welches die Person repräsentiert.

Die KB von BOEMIE wird im BOEMIE Ontology Repository abgelegt. Das Format für die TBoxen und ABoxen ist die Web Ontology Language (OWL).

Um die Ergebnisse für den Benutzer anschaulich darzustellen, wurde der BOEMIE Semantic Browser entwickelt. Diese Software stellt interaktiv die zugrunde liegenden Daten und die generierten Low- und Deep-Level-Annotationen dar und bietet die Möglichkeit, darin zu navigieren.

2.2.3 Anreicherung mit semantischen Metadaten

Gegeben sind nun die im vorherigen Abschnitt beschriebenen Interpretations-ABoxen. Diese enthalten die Low- und Deep-Level-Annotationen, die schrittweise extrahiert werden müssen, um sie dem Textindizierungsprozess zuführen zu können. Dazu wurde von mir ein Algorithmus basierend auf der Breitensuche in Graphen entwickelt, dieser wird hier vorgestellt.

Ein Graph ist das Paar $G = (V, E)$ mit $E \subseteq [V]^2$, V und E sind disjunkt. V ist die Menge der Knoten und E ist die Menge der Kanten des Graphen (siehe [6]). Ein gerichteter Graph hat zusätzlich zwei Funktionen init: $E \to V$ und ter: $E \to V$, welche jeder Kante $e \in E$ einen Knoten $v_1 \in V$ als Startknoten $\text{init}(e) = v_1$ und einen Knoten $v_2 \in V$ als Endknoten $\text{ter}(e) = v_2$ zuweisen. Ist $v_1 = v_2$, spricht man von einer Schlinge. In Abbildung 2.2 wird als Beispiel der Graph mit $V = \{1, 2, 3, 4, 5\}$ und $E = \{\{1, 2\}, \{2, 5\}, \{1, 3\}, \{3, 5\}, \{3, 4\}, \{4, 3\}\}$ dargestellt.

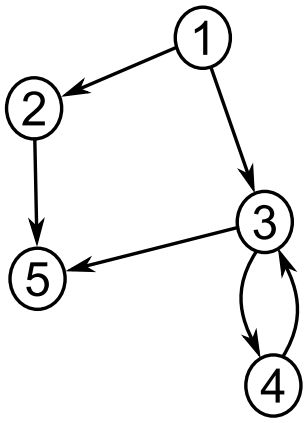


Abbildung 2.2: Ein einfacher, gerichteter Graph mit fünf Knoten und sechs Kanten

Um in Graphen nach Knoten zu suchen, existieren diverse Suchverfahren, welche auf unterschiedliche Art und Weise agieren. Die Breitensuche gehört zu den uninformierten Suchverfahren, das heißt, es werden keine Hintergrundinformationen zum Suchen verwendet. Die Suche liefert alle Knoten, welche von dem übergebenden Startknoten erreichbar sind und arbeitet dabei die Knoten der Entfernung zum Startknoten nach ab.

Der Algorithmus 2.1 aus [4] stellt eine generische Breitensuche dar. Diese Variante benutzt eine Warteschlange Q zum Speichern der zu behandelnden Knoten, einen den Knoten zugeordneten Zustand (hier die Farben weiß, grau und schwarz), um besuchte und unbesuchte Knoten zu unterscheiden (in *color* abgelegt) und der Array d wird zu Speichern der Entfernung zum Startknoten s verwendet.

```
1:  for each vertex u ∈ V[G] − {s} do
2:      color[u] ← WHITE
3:      d[u] ← ∞
4:      π[u] ← NIL
5:  color[s] ← GRAY
6:  d[s] ← 0
7:  π[s] ← NIL
8:  Q ← ∅
9:  ENQUEUE(Q, s)
10: while Q ≠ ∅ do
11:     u ← DEQUEUE(Q)
12:     for each v ∈ Adj[u] do
13:         if color[v] = WHITE then
14:             color[v] ← GRAY
15:             d[v] ← d[u] + 1
16:             π[v] ← u
17:             ENQUEUE(Q, v)
18:     color[u] ← BLACK
```

Algorithmus 2.1: Der Breitensuchalgorithmus aus [4]

In der Initialisierungsphase ab Zeile 1 bis 9 werden alle Knoten (außer dem Startknoten s) als weiß (also unbesucht), als unendlich weit von s entfernt und als vorgängerlos markiert. Der Startknoten wird als aktuell zu verarbeiten gekennzeichnet (grau) und in Q eingereiht. Die Verarbeitung beginnt ab Zeile 10, dort wird der aktuelle Knoten u aus der Warteschlange entfernt, und jeder verbundene Knoten auf seine Farbe überprüft. Ist sie weiß, wurde er bisher noch nicht behandelt. Dann wird er grau markiert und in Q aufgenommen.

Abbildung 2.3 zeigt die Breitensuche anhand des Graphs aus Abbildung 2.2. In den Knoten stehen nun die Entfernungen zum Startknoten, die gestrichelten Kanten zeigen die benutzten Wege der Suche. Der in 2.3(a) grau markierte Knoten wurde als Startknoten festgelegt.

Betrachtet man nun die Individuen einer ABox als Knoten und die Rollen als Kanten, kann man die ABox als gerichteten Graphen darstellen. Sei die Funktion „related“ ein Service eines DL-Systems, welche alle Individuen zurückgibt, welche mit dem gegebenen Individuum in Relation stehen. Sei weiter d eine maximale Tiefe für die Suche und I eine Menge von Startindividuen, so lässt sich die Breitensuche wie in Algorithmus 2.2 modifizieren.

Eine Markierung der bereits verarbeiteten Knoten entfällt hier zu Gunsten eines

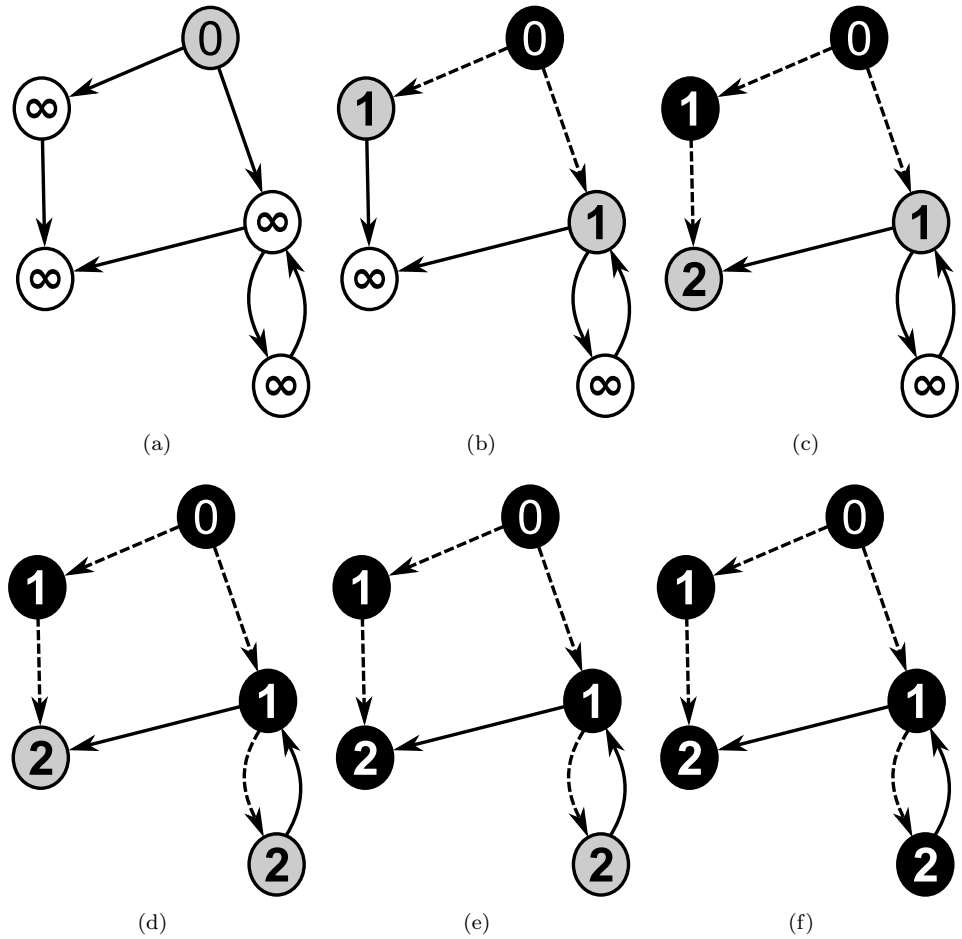


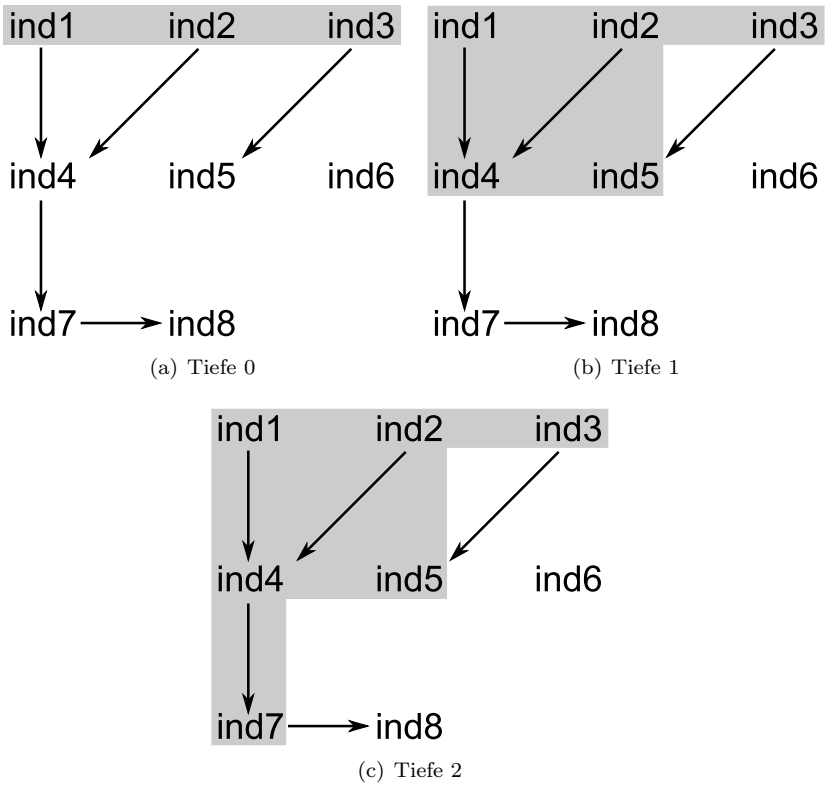
Abbildung 2.3: Darstellung der Breitensuche

1: $n \leftarrow 0$
2: **while** $d > 0$ and $n \neq |I|$ **do**
3: $\quad n \leftarrow |I|$
4: $\quad S \leftarrow \emptyset$
5: $\quad$ **for** each $i \in I$ **do**
6: $\quad\quad S \leftarrow S \cap \text{related}(i)$
7: $\quad I \leftarrow I \cap S$
8: $\quad d \leftarrow d - 1$

Algorithmus 2.2: Die modifizierte Breitensuche

einfacheren Algorithmus. Der Algorithmus terminiert, wenn die angegebene Tiefe erreicht wurde oder wenn keine neuen Individuen gefunden werden. Die Schnittmengen-Operatoren in den Zeilen 6 und 7 sorgen dafür, dass pro Schritt jedes Individuum nur einmal verarbeitet wird.

In Abbildung 2.4 wird der Algorithmus auf eine kleine ABox mit acht Individuen ausgeführt (Rollennamen sind nicht von Bedeutung und wurden weggelassen). Als Startmenge wurde die Menge $I = \{\text{ind1}, \text{ind2}, \text{ind3}\}$ gewählt. Die maximale Tiefe ist zwei, dadurch wird das Individuum „ind8“ nicht erreicht, außerdem ist „ind6“ weder in der Startmenge noch mit den anderen Individuen verbunden und wird deswegen auch nicht erreicht.



(a) Tiefe 0 (b) Tiefe 1

(c) Tiefe 2

Abbildung 2.4: Modifizierte Suche in der ABox

Um die tatsächlichen Namen der Individuen zu erhalten, müssen noch weitere Schritte unternommen werden, diese und die konkrete Implementation werden in 3.4 beschrieben.

2.3 Stemming

Stemming ist das Zurückführen von Wörtern auf ihre Wortstämme. Beispielsweise ist „throw“ der Wortstamm des englischen Wortes „throwing“. Dieses Verfahren wird eingesetzt, um die Suchergebnisse zu verbessert. Die Verbesserung tritt dadurch ein, dass verschiedene Wörter mit ein und demselben Wortstamm in der Regel die selbe oder eine ähnliche Bedeutung haben. So findet jemand mit einer Suche nach „throwing“ in einem Index mit Stemming auch Dokumente mit „throws“, „thrown“ und weitere Formen von „throw“. Stemming reduziert die Anzahl der Terme im Index und erlaubt dadurch schnelleres Suchen. Im folgenden Abschnitt wird der Stemming-Algorithmus von M. F. Porter vorgestellt (siehe [13]). Dieser Algorithmus wird heutzutage in dieser Form nicht oder kaum noch verwendet, eignet sich aber dazu, das Prinzip zu verdeutlichen.

2.3.1 Der Porter-Stemming-Algorithmus

Der Porter-Stemming-Algorithmus basiert auf Regeln, welche nacheinander in mehreren Schritten auf ein Wort angewendet werden. Dabei wird das Wort bis auf eine Grundform verkürzt.

Man unterscheidet zwischen Vokalen (a, e, i, o, u und y hinter einem Konsonanten) und Konsonanten (jedes Zeichen, was kein Vokal ist). Sei nun V eine Aneinanderreihung von Vokalen und C eine Aneinanderreihung von Konsonanten jeweils mit der Mindestlänge eins, so kann man jedes englische Wort folgendermaßen beschreiben:

$$[\mathrm{C}]m \cdot (\mathrm{VC})[\mathrm{V}].$$

Die Ausdrücke in den eckigen Klammern sind optional, der Multiplikator m wird das Maß des Wortes genannt. Beispielsweise besitzt das Wort „creativity“ die Aufteilung `|CR|EAT|IV|IT|Y|` und damit das Maß drei. Um Endungen zu entfernen, sind folgende Bedingungen aufgestellt worden:

m > x: Das Maß des Wortes ist größer als x.

***S:** Das Wort endet mit dem Buchstaben s (und analog zu anderen Buchstaben).

***v*:** Das Wort endet mit einem Vokal.

***d:** Das Wort endet mit einem doppelten Konsonanten.

***o:** Das Wort endet mit der Kombination Konsonant-Vokal-Konsonant, wobei der zweite Konsonant kein w, x oder y ist.

Boolesche Operatoren sind zur Verknüpfung der Bedingungen zugelassen.

Mit diesen Bedingungen sind Regeln der Form:

$$(\text{Bedingung})\ S_1 \rightarrow S_2$$

definiert worden, wobei S_1 und S_2 jeweils Suffixe des Wortes sind. Beispielsweise wird das Wort „creativity“ durch die Regel (Bedingung) ITY → E zu „creative“. Folgende acht Schritte werden nun unternommen, um die Wortstämme zu erhalten:

Schritt 1a-1c: Entfernung von Pluralendungen und Endungen der „past participle“-Form.

Schritt 2-4: Transformation von spezifischen, längeren Endungen in Kürzere.

Schritt 5a,b: Entfernung von übrig gebliebenen und doppelten Buchstaben.

Durch das Maß wird sichergestellt, dass ein zu kurzes Wort nicht weiter verkürzt wird. In Tabelle 2.6 werden die Regeln auszugsweise dargestellt. In jedem Schritt wird genau die Regel mit dem längsten, passenden S_1 angewendet.

Schritt	Regel			Beispiel		
	SSES	→	SS	caresses	→	caress
1a	IES	→	I	ponies	→	poni
	...			...		
	$(m > 0)$ EED	→	EE	caresses	→	caress
1b	IES	→	I	ponies	→	poni
	...			...		
1c	(*v*) Y	→	I	happy	→	happi
	$(m > 0)$ ATIONAL	→	ATE	relational	→	relate
2	$(m > 0)$ TIONAL	→	TION	conditional	→	condition
	...			...		
	$(m > 0)$ ICATE	→	IC	triplicate	→	triplic
3	$(m > 0)$ ATIVE	→		formative	→	form
	...			...		
	$(m > 1)$ AL	→		revival	→	reviv
4	$(m > 1)$ ANCE	→		allowance	→	allow
	...			...		
...	...			...		

Tabelle 2.6: Auszüge aus den Regeln des Porter-Stemming-Algorithmus

Ein Nachteil dieses Verfahrens ist, dass zwischen normalen und speziellen Wörtern (wie zum Beispiel Eigennamen) nicht differenziert wird. Beispielsweise wird aus dem Nachnamen „Phillips“ der Name „Phillip“. Dieser Umstand kann, wenn beide Namen in den Dokumenten vorhanden sind, zu falschen Ergebnissen führen.

2.4 Bewertung von Suchergebnissen

In diesem Abschnitt werden Kriterien vorgestellt, mit denen Listen von Suchergebnissen bewertet werden können. Danach wird eine Auswahl getroffen, welche Kriterien zur Auswertung der Testanfragen im Rahmen dieser Arbeit benutzt werden.

Um Ergebnislisten bewerten zu können, müssen die einzelnen Ergebnisse vorher als relevant oder nicht relevant markiert worden sein. Dabei wird die Relevanz nicht auf die Suchanfrage, sondern auf eine textuelle Beschreibung des „Suchbedürfnisses“ (im folgenden Information Need genannt) bezogen. Im Folgenden bedeutet der Ausdruck „relevantes Ergebnis“ immer „relevant für den aktuellen Information Need“.

2.4.1 Precision, Recall und F-Measure

Sei D_r die Anzahl der relevanten Ergebnisse und N die Anzahl aller Ergebnisse in einer Ergebnisliste. Damit ist die Precision P definiert als:

$$P = \frac{D_r}{N}. \tag{2.1}$$

Sei weiterhin D_n die Anzahl aller relevanten Dokumente im Index, damit ist der Recall R definiert als:

$$R = \frac{D_r}{D_n}. \tag{2.2}$$

Precision und Recall berücksichtigen die Reihenfolge der Ergebnisse in der Ergebnisliste nicht, können demnach auch auf ungewichtete Resultate angewendet werden. Das Problem des Recalls ist, dass der tatsächliche Wert für D_n in den meisten Fällen durch die in der Regel hohe Zahl von Dokumenten im Index nicht in vertretbarer Zeit ermittelt werden kann.

Durch die Kombination von Precision und Recall erhält man den sogenannten F-Measure (siehe [9]). Sei $\alpha \in [0, 1]$, der F-Measure ist definiert als:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R}, \text{ mit } \beta^2 = \frac{1-\alpha}{\alpha}. \tag{2.3}$$

Der F-Measure ist also das gewichtete, harmonische Mittel von Precision und Recall. Mit $\beta = 1$ werden Precision und Recall gleich gewichtet und die Gleichung vereinfacht sich zu:

$$F_{\beta=1} = \frac{2PR}{P+R}. \tag{2.4}$$

Selbstverständlich muss der Wert für R und damit auch für D_n bekannt sein, um den F-Measure berechnen zu können.

In Tabelle 2.7 sind beispielhaft zwei gleichlange Ergebnislisten angegeben, bei denen zusätzlich auch die Anzahl an relevanten Ergebnissen übereinstimmt. Es ist zusätzlich bekannt, dass $D_n = 15$ ist. Es ergeben sich in beiden Fällen für die Precision $P = 0,4$, für den Recall $R = 0,27$ und für den gleichgewichteten F-Measure:

$$F_{\beta=1} = \frac{2 \cdot 0,4 \cdot 0,27}{0,4 + 0,27} = 0,32.$$

Für ein weiteres Beispiel mit hundert Ergebnissen und davon dreizehn relevanten Ergebnissen ist $P = 0,13$, $R = 0,87$ und $F_{\beta=1} = 0,23$.

	Relevanz	
Pos.	1. Liste	2. Liste
1	✓	
2		
3	✓	
4	✓	✓
5		
6		✓
7	✓	
8		
9		✓
10		✓

Tabelle 2.7: Zwei beispielhafte Ergebnislisten mit als relevant markierten Ergebnissen

2.4.2 Average Precision

Die Average Precision (Avg. Precision, siehe [2]) berücksichtigt im Gegensatz zu der Precision aus Abschnitt 2.4.1 die Reihenfolge, in denen die Ergebnisse in der Ergebnisliste vorkommen.

Sei D_i die Anzahl relevanter Ergebnisse bis Position i (inklusive). Damit ist die relative Precision P_i definiert als:

$$P_i = \frac{D_i}{i}. \tag{2.5}$$

Der relative Recall R_i ist definiert als:

$$R_i = \frac{D_i}{D_r}. \tag{2.6}$$

Mit der relativen Precision und dem relativen Recall lässt sich die Pseudo-Precision $\tilde{P}(x)$ bei Recall-Level x wie folgt definieren:

$$\tilde{P}(x) = \max P_i, \text{ für } x \leq \frac{D_i}{D_r} \text{ und } i = 1, \ldots, n. \tag{2.7}$$

Damit lässt sich die Avg. Precision P_{av} folgendermaßen darstellen:

$$P_{av} = \frac{1}{n} \sum_{i=0}^{n-1} \tilde{P}\left(\frac{i}{n-1}\right). \tag{2.8}$$

In meiner Arbeit verwende ich als Recall-Level zur Berechnung von P_{av} die Werte $k/10$ für $k = 1, \ldots, 10$ und $n = 10$.

Gegeben sei das Beispiel in Tabelle 2.7. In Tabelle 2.8 sind die Werte für P_i und R_i für die Ergebnislisten dargestellt. Damit berechnet sich für das erste Beispiel:

$$P_{av} = \frac{1}{10}(0 + 0 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1) = 0,8.$$

	1. Liste			2. Liste		
Pos.	Relevanz	P_i	R_i	Relevanz	P_i	R_i
1	✓	1	0,25		0	0
2		0,5	0,25		0	0
3	✓	0,67	0,5		0	0
4	✓	0,75	0,75	✓	0,25	0,25
5		0,6	0,75		0,2	0,25
6		0,5	0,75	✓	0,33	0,5
7	✓	0,57	1		0,29	0,25
8		0,5	1		0,25	0,25
9		0,44	1	✓	0,33	0,75
10		0,4	1	✓	0,4	1

Tabelle 2.8: Ein Beispiel zur Berechnung der Avg. Precision

Für das zweite Beispiel hingegen ist:

$$P_{av} = \frac{1}{10}(0 + 0 + 0,25 + 0,25 + 0,33 + 0,33 + 0,33 + 0,33 + 0,33 + 0,4) = 0,26.$$

Daran sieht man deutlich, dass das zweite Beispiel mit den relevanten Ergebnissen auf den hinteren Plätzen wesentlich schlechter als das erste Beispiel bewertet wird.

2.4.3 Mean Average Precision

Sei Q die Menge aller Information Needs und $d_1, \ldots, d_{m_j}$ die Menge aller relevanten Dokumente eines Information Needs $q_j \in Q$. Sei weiterhin R_{jk} die Menge aller Suchergebnisse in der Ergebnisliste von q_j bis Dokument d_k, so ist die Mean Average Precision definiert als:

$$\text{MAP}(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} \text{Precision}(R_{jk}). \tag{2.9}$$

Seien die erste Ergebnisliste aus Tabelle 2.7 dem Information Need $q_{11} \in Q_1$ und die zweite Liste dem Information Need $q_{21} \in Q_2$ zugeordnet. Sei weiterhin $|Q_1| = |Q_2| = 1$ und $m_j = 6$. Mit Gleichung 2.9 ergibt sich dann:

$$\text{MAP}(Q_1) = 0,17 \cdot (1 + 0,67 + 0,75 + 0,57 + 0 + 0) = 0,5$$

$$\text{MAP}(Q_2) = 0,17 \cdot (0,25 + 0,5 + 0,33 + 0,4 + 0 + 0) = 0,25$$

Die MAP berücksichtigt genau wie die Avg. Precision die Reihenfolge der Ergebnisse in der Ergebnisliste, allerdings muss man den Wert von m, also die Anzahl aller relevanten Dokumente für jeden Information Need kennen. Hier besteht also wieder das gleiche Problem wie beim Recall.

2.4.4 Auswahl der Kriterien

Für die Bewertung der Suchergebnisse und damit der Indizes wurden von mir folgende Kriterien ausgewählt:

- Die Anzahl der Ergebnisse in der Ergebnisliste
- Die Anzahl der relevanten Ergebnisse in der Ergebnisliste
- Die Precision
- Die Avg. Precision

Grund für die Auswahl war primär die fehlende Information über alle relevanten Dokumente pro Information Need im Index. Da der Recall, die F-Measure und die MAP diese Information benötigen, können sie nicht zur Auswertung herangezogen werden. Statt des Recalls wird die Anzahl der relevanten Ergebnisse herangezogen.

2.5 Annahme über das Ergebnis der Auswertung

Ohne Kenntnis der Daten würde ich die Annahme treffen, dass sich die Suchergebnisse durch Anreicherung von semantischen Daten um einiges verbessern lassen. Die Anzahl an zurückgegebenen Ergebnissen würde sich erheblich steigern lassen, was selbstverständlich die Precision verschlechtert. Trotzdem sollten genug relevante Ergebnisse dabei sein, um eine signifikante Verbesserung zu erzielen.

Allerdings verschlechtern die vorhandenen Daten diese Prognose erheblich. Da die zu indizierenden Dateien Nachrichtenmeldungen sind, ist zu erwarten, dass die wichtigen Informationen an den meisten Stellen direkt im Text stehen. Zusätzlich ist die Anzahl an Bildern in den Dateien eher klein, so dass kein großer Gewinn durch aus den Bildern extrahierte Informationen zu erwarten ist. Das die Bilderkennung teilweise fehlerhaft war und zu falschen Informationen geführt hat, kommt noch erschwerend hinzu. Deswegen gehe ich davon aus, dass sich die Ergebnisse nur mäßig verbessern lassen.

3 Durchführung

3.1 Konzeption des Programms

Da das Ziel dieser Arbeit ist, semantische Informationen in den normalen Textindizierungsprozess zu übernehmen und die entstehenden Indizes zu bewerten, muss das Programm folgende Mindestanforderungen erfüllen:

- Eine vorgegebene Menge von HTML-Seiten indizieren.
- Beim Indizieren die semantischen Daten berücksichtigen.
- Im Index suchen und die Suchergebnisse zurückgeben.

Um die allgemeine Bedienung und die Auswertung der Suchergebnisse zu erleichtern, soll das Programm noch weitere Funktionen bieten:

- Indizes speichern und laden.
- Anfragen speichern und laden.
- Informationen über den Index anzeigen (Anzahl der Dokumente, Indexgröße, Art der semantischen Informationen).
- Markierung der relevanten Ergebnisse.
- Berechnung der in Abschnitt 2.4.4 gewählten Kriterien.
- Einstellungen aus einer Properties-Datei laden und speichern.

Zusätzlich kommt noch eine Sekundärfunktion dazu, welche ein Mapping zwischen HTML- und OWL-Dateien mit deren entsprechenden Interpretations-ABoxen erstellen und speichern soll. Dieses Mapping wird dann im Indizierungsprozess verwendet.

Aus Gründen der Kapselung und der Wiederverwendbarkeit habe ich mich für eine Aufteilung in zwei Komponenten entschieden:

1. Bibliothek zum Suchen und Indizieren (genannt SemTexSearch)
2. GUI[1] zur Benutzerinteraktion (genannt SemTexSearchGui)

Im Folgenden beziehe ich mich auf die von mir im Rahmen dieser Arbeit erstellte Bibliothek SemTexSearch, die GUI wird in Abschnitt 3.6 beschrieben. Die Kernfunktionalitäten von SemTexSearch sind ein Prozess zur Indizierung von HTML-Dateien inklusive Vorverarbeitung (siehe Abbildung 3.1) und die Suche in den erstellten Indizes. In den folgenden Abschnitten werden die wichtigsten Programmteile beschrieben.

[1] Graphical User Interface

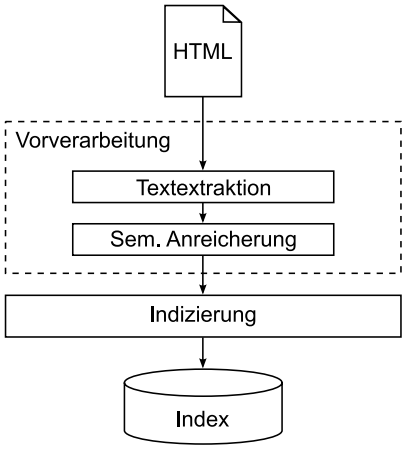


Abbildung 3.1: Indizierungsprozess einer HTML-Datei mit Vorverarbeitung

3.2 Textindizierung und Suche mit Lucene

Apache Lucene ist eine Bibliothek zur Indizierung von Textdateien, welche von der Apache Software Foundation unter der Apache License veröffentlicht wird (siehe [12]). Die Originalbibliothek wurde in Java geschrieben, es existieren allerdings Ports nach C, C++, Perl und weiteren Sprachen. Lucene ist sehr weit verbreitet und wird unter anderem zur Volltextsuche in der Wikipedia oder zur Suche in den Hilfeseiten der Eclipse IDE verwendet.

3.2.1 Indizierung

Zur Indizierung bietet Lucene eine Vielzahl unterschiedlicher Textanalysierungs- und Filterklassen an (siehe [7]), welche die Quelldokumente verarbeiten. Ich habe davon den Snowball-Analysierer verwendet, welcher den Snowball-Stemming-Algorithmus (eine moderne Form des in Abschnitt 2.3.1 vorgestellten Porter-Stemmers für die englische Sprache), einen Stopwortfilter mit einer vorgegebenen Wörterliste und einen Kleinbuchstabenfilter verwendet. Der Kleinbuchstabenfilter wandelt alle Buchstaben in Kleinbuchstaben um. Weiterhin existieren noch Analysierer und Stemmer für verschiedene Sprachen (wie zum Beispiel Deutsch, Spanisch, Französisch).

Lucene bietet weiterhin die Möglichkeit, die Informationen in sogenannten Fields abzulegen. Diese Fields können separat indiziert und der Inhalt im Index gespeichert werden. Der eigentliche Inhalt wird standardmäßig im Field „content“ abgelegt. Es ist allerdings nicht ohne zusätzlichen Aufwand möglich, mehrere Fields gleichzeitig zu durchsuchen.

Um die zu indizierenden Dateien effektiv und flexibel vorverarbeiten zu können, habe ich die abstrakte Klasse DocumentPreprocessor definiert, welche den Inhalt einer Datei und den Dateipfad als Eigenschaften hat und den verarbeiteten Text samt einer

beliebigen Menge an Fields zurück gibt. Entsprechende Unterklassen können dann im Indizierungsprozess beliebig kaskadiert werden (siehe Abbildung 3.2). Solch eine Vorverarbeitung ist notwendig, weil Lucene von sich aus kein HTML sinnvoll verarbeiten kann und die semantischen Annotationen vor dem Indizieren im Text untergebracht werden sollen. Weiteres zur HTML-Verarbeitung beschreibe ich in Abschnitt 3.3, der semantische Präprozessor wird in 3.4 beschrieben.

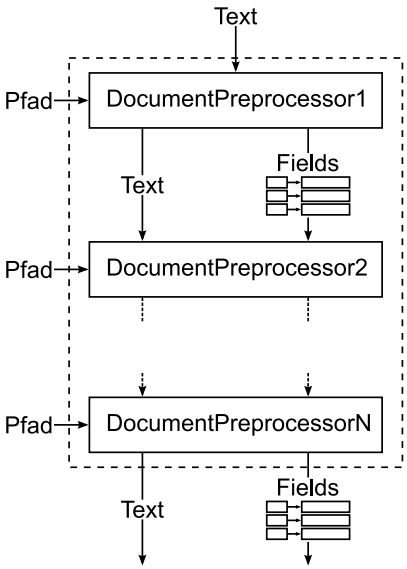


Abbildung 3.2: Kaskadierung der Präprozessoren zur Vorverarbeitung

Die Indizierungsroutine erhält folgende Parameter:

- Der Pfad der zu indizierenden Dateien
- Das Verzeichnis des neuen Index
- Den Pfad zur Indexdatei
- Die Liste der Präprozessoren

Der zu indizierende Pfad kann entweder eine HTML-Datei oder ein Verzeichnis sein. Wenn ein Verzeichnis angegeben wird, werden alle Dateien und Unterverzeichnisse rekursiv durchlaufen und alle vorhandenen HTML-Dateien indiziert. Die eigentliche Indizierung einer Datei läuft folgendermaßen ab:

1. Datei einlesen.
2. Eventuell vorhandene Präprozessoren der Reihenfolge nach auf die Datei anwenden, dabei:

- Die Ausgabe eines Präprozessors an den nächsten weiter reichen.
- Die extrahierten Fields in eine Field-Map einfügen.

3. Ein neues Lucene-Dokument erzeugen.
4. Die extrahierten Fields hinzufügen.
5. Den vorverarbeitete Dateiinhalt als Content-Field hinzufügen.
6. Das neue Dokument an den Lucene-Indizierer übergeben.

In Listing 3.1 wird der entsprechende Ausschnitt aus der Indexer-Klasse dargestellt. Zum Indizieren wird die Lucene-Klasse IndexWriter zusammen mit dem oben genannten SnowballStemmer verwendet (siehe Zeile 2). Ab Zeile 21 werden die Fields zum Dokument hinzugefügt, dabei wird Lucene durch folgende Flags signalisiert, wie die Fields beim Indizieren zu verarbeiten sind:

Store.YES: Der Inhalt wird im Index gespeichert.

Store.NO: Der Inhalt wird nicht im Index gespeichert.

Index.ANALYSED: Der Inhalt wird analysiert und kann durchsucht werden.

Index.NOT_ANALYSED: Der Inhalt wird nicht analysiert.

Hier wird der Dateipfad nur zur Information gespeichert und der Inhalt der Datei wird aus Platzgründen zwar analysiert, aber nicht gespeichert. Alle anderen Fields werden analysiert und gespeichert.

Um Extrainformationen über den entstandenen Index zu speichern, habe ich eine Index-Klasse definiert, welche mit Hilfe der JAXB[2]-Schnittstelle in eine XML-Datei geschrieben (marshalling) und von dort auch wieder gelesen (unmarshalling) werden kann. Diese Klasse hält Informationen über die Anzahl von Dokumenten im Index, den Pfad zu den Indexdateien und über die Ausprägung der semantischen Informationen. In Listing 3.2 ist solch eine XML-Datei für einen Index als Beispiel dargestellt.

3.2.2 Suche

Die Suche ist unter Lucene sehr einfach implementiert. Da es sich in diesem Anwendungsfall um von Anwendern eingegebene Anfragen handelt, müssen diese auf die gleiche Art und Weise wie die Dokumente analysiert werden. In Listing 3.3 ist die Methode, welche die Suche implementiert, dargestellt. Die Klasse QueryParser in Zeile 7 sorgt dafür, dass der Suchstring in eine von Lucene interpretierbare Query transformiert wird. Zur Formulierung der Anfrage stellt Lucene eine Vielzahl von Schlüsselwörtern bereit, welche die Suche beeinflussen. Zum Beispiel sind boolesche Operatoren zugelassen, ein Plus vor einem Wort erzwingt das Vorkommen in einem Dokument und mit einer Tilde (~) kann eine Unschärfe zugelassen werden. So liefert die Suche nach „`text~`“ auch Ergebnisse mit ähnlichen Wörtern (zum Beispiel

[2] Java Architecture for XML Binding

```
1  /*...*/
2  IndexWriter writer = new IndexWriter(indexPath, new
       SnowballAnalyzer("English", StandardAnalyzer.STOP_WORDS),
       true, IndexWriter.MaxFieldLength.LIMITED);
3  /*...*/
4  private void indexDocument(File f) /*throws ...*/ {
5    /*...*/
6    Map<String, String> fields = new HashMap<String, String>();
7    if (preprocessors != null) {
8      for (DocumentPreprocessor processor : preprocessors) {
9        processor.setDocument(content);
10       processor.setDocPath(f.getAbsolutePath());
11       content = processor.process();
12       if(processor.getFields() != null) {
13               for(String field : processor.getFields().keySet
                     ()) {
14                 if(fields.containsKey(field)) {
15                fields.put(field, fields.get(field) +processor.
                     getFields().get(field));
16             } else {
17                fields.put(field ,processor.getFields().get(
                     field));
18             }
19   /*..*/
20   Document doc = new Document();
21   doc.add(new Field("path", f.getPath(), Field.Store.YES,
         Field.Index.NOT_ANALYZED));
22   for(String field : fields.keySet()) {
23     doc.add(new Field(field, fields.get(field), Field.Store.
         YES, Field.Index.ANALYZED));
24   }
25   doc.add(new Field("content", content, Field.Store.NO, Field.
         Index.ANALYZED));
26   writer.addDocument(doc);
27   /*...*/
```

Listing 3.1: Indizierung eines Dokuments

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<index>
    <docCount>606</docCount>
    <fullPath>/path/to/the/index/</fullPath>
    <semanticDepth>0</semanticDepth>
    <semanticInfo>false</semanticInfo>
    <semanticInfoFromTextOnly>true</semanticInfoFromTextOnly>
</index>
```

Listing 3.2: Beispielindex im XML-Format

„`test`“). Mit „`"personal best"~5`“ werden Dokumente gefunden, welche die Terme „`personal`“ und „`best`“ im Abstand von maximal fünf Termen enthalten. Durch Angabe eines Fields wie in „`title:olympics`“ kann in einem anderen als im Standard-Field gesucht werden. Weitere Möglichkeiten zur Formulierung von Anfragen findet man in [7].

Die eigentliche Suche wird in Zeile 9 durchgeführt, dabei werden maximal so viele Dokumente wie in Variable „maxHits“ definiert zurückgegeben.

Um die Ergebnisse weiter verarbeiten zu können, habe ich eine Datenstruktur bestehend aus den Klassen InformationNeed und QueryResult definiert. Eine Instanz der Klasse InformationNeed hält:

- die textuelle Beschreibung der Suche
- den Suchstring
- den Pfad zum verwendeten Index
- Auswertungsdaten (siehe 3.5)
- eine Liste von QueryResult-Objekten mit den Ergebnissen

Ein QueryResult-Objekt besteht aus:

- dem Dateipfad zum gefundenen Dokument
- der von Lucene ermittelten Score
- einem vom Benutzer zu setzendem Relevant-Flag
- den Fields des Dokuments

Sei Q die Menge der Terme einer Anfrage, $t \in Q$, d das Dokument und f das durchsuchte Field. Die Score des Ergebnisses wird von Lucene mit Hilfe der in Abschnitt 2.1 beschriebenen tf-idf-Gewichtung folgendermaßen berechnet (siehe [7]):

$$score(Q) = \sum_{t \in Q} \text{tf-idf}_{t,d} \cdot \text{boost}(f) \cdot \text{norm}(f).$$

```
1  /*...*/
2  public static List<QueryResult> search(Index index,
       InformationNeed need, int maxHits) {
3      List<QueryResult> res = new ArrayList<QueryResult>();
4      IndexReader reader = IndexReader.open(index.getFullPath());
5      Searcher searcher = new org.apache.lucene.search.
           IndexSearcher(reader);
6      Analyzer analyzer = new SnowballAnalyzer("English",
           StandardAnalyzer.STOP_WORDS);
7      QueryParser parser = new QueryParser("content", analyzer);
8      Query query = parser.parse(need.getQueryString());
9      TopDocs docs = searcher.search(query, maxHits);
10     for (int i = 0; i < Math.min(docs.totalHits, maxHits); i++)
           {
11         QueryResult r = new QueryResult(searcher.doc(docs.
               scoreDocs[i].doc).get("path"), docs.scoreDocs[i].
               score);
12         for(Object o : searcher.doc(docs.scoreDocs[i].doc).
               getFields()) {
13             Field f = (Field)o;
14             if(!f.name().equals("path")) {
15                 r.getFields().put(f.name(), f.stringValue());
16             }
17         }
18         res.add(r);
19     }
20     return res;
21 }
```

Listing 3.3: Suche im Index

Dabei ist boost(f) ein frei wählbarer Faktor und norm(f) normiert die Score auf die Länge des Fields. Sollte die Score des am besten bewerteten Dokuments größer als eins sein, werden alle weiteren Ergebnisse erneut normiert, um eine Score kleiner gleich eins zu gewährleisten.

Um die Suchergebnisse zusammen mit der verwendeten Anfrage speichern und laden zu können, wird wie im vorherigen Abschnitt beschrieben JAXB verwendet. Ein Auszug aus einer Anfrage findet man in Listing 3.4.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<informationNeed>
    <avgPrecision>0.4</avgPrecision>
    <description>Ergebnisse der 4 x 400m Staffellaeufe der
        Herren.</description>
    <indexPath>/path/to/the/index</indexPath>
    <precision>0.44444445</precision>
    <queryString>+results +"4x400m relay" +men</queryString>
    <relevantResults>4</relevantResults>
   <!-- ... -->
    <results>
        <fields>
            <entry>
                <key>title</key>
                <value>IAAF International Association of
                    Athletics Federations - IAAF.org - News</
                    value>
            </entry>
        </fields>
        <path>/path/to/the/documents/doc.html</path>
        <relevant>false</relevant>
        <score>0.3056562</score>
    </results>
    <!-- ... -->
</informationNeed>
```

Listing 3.4: Beispielanfrage im XML-Format

3.3 Textextraktion aus HTML-Dateien mit NekoHTML

Wie in 3.2.1 schon erwähnt wurde, kann Lucene nur Textdateien verarbeiten. Würde man HTML-Dateien ohne Vorverarbeitung indizieren, würden alle HTML-Tags mit indiziert werden und den Index unnötig vergrößern. Um das zu verhindern, muss der Text aus HTML-Dateien extrahiert werden, für diese Aufgabe habe ich die freie Bibliothek NekoHTML verwendet (siehe [3]).

NekoHTML ist ein HTML-Parser, welcher in der Lage ist, auch ungültige HTML-Dateien einzulesen und zu reparieren. Eingelesene Dateien können mit Standard-Techniken für den Zugriff auf XML-Dokumente manipuliert werden. NekoHTML basiert auf dem Xerces Native Interface (XNI) der Apache Foundation, welches ein Framework für generische Parser darstellt. Damit kann NekoHTML von jedem Programm ohne spezielle Anpassungen verwendet werden, welches von Haus aus das XNI ansprechen kann.

Um HTML-Dateien zu manipulieren, bietet NekoHTML eine sogenannte Filter Chain an, welche eine beliebige Anzahl von Filtern kaskadieren kann. Da der Titel der HTML-Seite als Field extra indiziert werden soll, muss der Inhalt des entsprechenden HTML-Tags als erstes gesondert extrahiert werden. Für diese Aufgabe habe ich einen eigenen Filter basierend auf der DefaultFilter-Klasse von NekoHTML geschrieben. Als zweites werden mit einer Instanz der Klasse ElementRemover alle HTML-Tags entfernt. Als letzter Schritt wird das Dokument serialisiert und in einen String geschrieben. Die von mir geschriebene Klasse, welche diese Filter Chain und damit NekoHTML verwendet, ist gleichzeitig eine Unterklasse von DocumentPreprocessor, damit sie in die Vorverarbeitung wie in Abschnitt 3.2.1 beschrieben einbezogen werden kann.

Als Beispiel soll hier der Auszug aus einer Web-Seite in Listing 3.5 dienen. Das Produkt der Verarbeitung durch die Filter Chain ist zum einen der Titel („IAAF International Association of Athletics Federations - World Indoor Championships 2003 - News“) und zum anderen der reine Text:

> Tuesday 11 March 2003 Monte-Carlo - The women's Pole Vault, which pits USA's Stacy Dragila against Russia's Svetlana Feofanova, is one of the most eagerly awaited events of this weekend's 9th IAAF World Indoor Championships in Athletics, which takes place in Birmingham's National Indoor Arena (14-16 March). The women's world record has already been improved three times this season.

3.4 Textanreicherung durch semantische Daten mit RacerPro

Als zweiter Schritt in der in 3.2.1 beschriebenen Kette von Präprozessoren sollen die Dokumente mit semantischen Informationen erweitert werden. Die theoretischen Grundlagen wurden im Abschnitt 2.2 erörtert. Hier soll speziell die Implementation des in 2.2.3 beschriebenen Algorithmus mit Hilfe von RacerPro dargestellt werden.

3.4.1 RacerPro

RacerPro ist die Abkürzung von *Renamed ABox and Concept Expression Reasoner Professional* (siehe [11]). RacerPro ist ein DL-System, implementiert $\mathcal{ALCQ}(\mathcal{D})$ und bietet die in 2.2 beschriebenen Dienste an. Weiterhin ist RacerPro in der Lage, Ontologien im OWL-Format zu verarbeiten und bietet sich damit an, die Anfragen nach den benötigten Daten aus der KB zu beantworten.

```
<html>
<head>
<title>IAAF International Association of Athletics Federations
    – World Indoor Championships 2003&amp;nbsp;–&amp;nbsp;News</title
   >
<!– ... –>
<SPAN class="NewsDate">Tuesday 11 March 2003</SPAN>
<table cellpadding="0" cellspacing="5" border="0" width="100%"
    xmlns:msxsl="urn:schemas-microsoft-com:xslt" xmlns:user="
   http://mycompany.com/mynamespace">
   <tr>
      <td class="NewsBody">
         <P>Monte-Carlo – The women's Pole Vault, which pits
            USA's Stacy Dragila against Russia's Svetlana
            Feofanova, is one of the most eagerly awaited
            events of this weekend's 9th IAAF World Indoor
            Championships in Athletics, which takes place in
            Birmingham's National Indoor Arena (14-16 March).
            </P>
         <P>The women's world record has already been improved
             three times this season.</P>
<!– ... –>
```

Listing 3.5: Auszug aus einer HTML-Datei (Quelltext)

Als Anfragesprache zur Kommunikation mit RacerPro dient nRQL[3], eine Sprache mit Lisp-ähnlichem Syntax. Mit nRQL können konjunktive und andere Anfragen und Befehle ausgeführt werden. Seien beispielsweise die Individuen „`ALICE`“, „`BOB`“ und „`CAROL`“ Instanzen des Konzeptes „`person`“, so würde folgende nRQL-Anfrage:

```
(retrieve (?x) (?x person))
```

als Resultat die folgende Liste zurück geben:

```
(((?X ALICE)) ((?X BOB)) ((?X CAROL))).
```

In Tabelle 3.1 sind auszugsweise ein Paar von mir verwendete RacerPro-Befehle aufgeführt, weitere Befehle werden im nächsten Abschnitt bei Bedarf beschrieben (eine vollständige Auflistung aller Befehle findet man in [10]).

Befehl	Beschreibung
`(owl-read-file` *Pfad* ...`)`	Liest eine OWL-Datei ein und erstellt eine KB oder fügt die neuen Daten zu einer bestehenden hinzu.
`(current-abox)`	Gibt den Namen der aktuellen ABox zurück.
`(delete-abox` *ABoxname*`)`	Verwirft die genannte ABox.
`(individual-direct-types` *Ind.*`)`	Gibt die spezifischsten Konzepte zurück, welche das Individuum instanziert.
`(role?` *Rolle*`)`	Prüft, ob der gegebene Parameter eine bekannte Rolle in der TBox ist.
`(concept?` *Konzept*`)`	Prüft, ob der gegebene Parameter ein bekanntes Konzept in der TBox ist.
`(full-reset)`	Stellt den Startzustand von RacerPro wieder her.

Tabelle 3.1: Einige RacerPro-Befehle

RacerPro bietet eine TCP-Schnittstelle (standardmäßig über Port 8088) an, für verschiedene Sprachen stehen Clientbibliotheken für die Kommunikation bereit. Ich verwendete die JRacer-Bibliothek für Java.

[3] new Racer Query Language

3.4.2 Die SemanticAdder-Klasse

Zu Implementierung wurde von mir die Klasse SemanticAdder erstellt, welche von DocumentPreprocessor erbt, um in die Verarbeitungskette aufgenommen werden zu können. Um die semantischen Daten verarbeiten zu können, habe ich folgende Annahmen basierend auf der Struktur der TBox und der Testdaten getroffen:

1. Es existieren „WebPage“-Individuen, welche ein Datatype Property namens „hasURL“ mit der zugeordneten Webseite besitzen.
2. Diese Individuen sind mit Individuen des Konzeptes „MultiMediaContentItem“ über die Rolle „contains“ verbunden.
3. Entitäten von Deep-Level-Konzepten sind über die Rolle „depicts“ mit den Individuen der Klasse „MultiMediaContentItem“ verbunden.

Diese Einschränkungen ermöglichen der SemanticAdder-Klasse einen festen Einstiegspunkt in die Menge der Individuen in der ABox.

Weitere Einschränkungen beziehen sich auf die Namen und Werte von Individuen. Um den Namen oder den Wert eines Individuums herausfinden zu können, muss einer der folgenden Fälle erfüllt sein:

1. Das Konzept X hat das Datatype Property „hasXValue“
2. Das Konzept X hat das Datatype Property „hasXNameValue“
3. Das Konzept X steht in über „hasXName“ in Relation zu Konzept „XName“, welches das Datatype Property „‘hasXNameValue“ besitzt.

Als Hilfsklasse habe ich die Klasse RacerQueries definiert, welche eine Anzahl von statischen Methoden anbietet, die eine oder mehrere RacerPro-Befehle absetzen und die Ergebnisse parsen. Dazu gehören Methoden zum Laden von OWL-Dateien, Löschen der aktuellen ABox und zum Resetten von RacerPro. Weitere Methoden werden im Folgenden kurz vorgestellt:

getConceptOfIndividual(Individuum): Ermittelt durch `(individual-direct-types` *Individual*`)` spezifischsten Konzepte vom gegebenen Individuum.

isDatatypeProperty(Rolle): Entscheidet mit Hilfe von `(role?` *Rolle*`)` und `(role-used-as-datatype-property-p` *Rolle*`)` ob der Parameter ein Datatype Property ist.

getPropertyFromIndividual(Individuum, Rolle): Gibt den Wert der als Datatype Property genutzten Rolle zurück und verwendet dazu `(individual-told-datatype-fillers` *Individuum Rolle*`)`.

getIndividualsFromWebPage(URL, textOnly): Diese Methode nutzt die oben aufgeführten Einschränkungen, um alle Individuen zurückzugeben, welche der gegebenen URL zugeordnet sind. Benutzt wird folgende konjunktive Anfrage:

```
(retrieve (?x) (and
  (?y WebPage)
  (?y (string= hasURL "URL"))
  (?z Text)
  (?y ?z contains)
  (?z ?x depicts)))
```

Der Parameter „textOnly“ bestimmt, ob die Einschränkung auf Texte mit `(?z Text)` erfolgen soll oder nicht. Wenn nicht, dann wird dieser Teil der Anfrage weggelassen.

getRelatedIndividuals(Individuum): Erhält alle mit dem gegebenen Individuum in Beziehung stehende Individuen, in denen das gegebene Individuum in der Domäne ist durch `(all-role-assertions-for-individual-in-domain` *Individuum* `)`.

getNameOfIndividual(Individuum): Verwendet die oben genannten Annahmen, um den Namen oder den Wert des gegebenen Individuums zu ermitteln. Ob die entsprechenden Datatype Properties existieren, wird jeweils mit der Methode *isDatatypeProperty* ermittelt und im jeweiligen Erfolgsfall der Wert mit *getPropertyFromIndividual* erfragt. Trifft weder der erste noch der zweite Fall zu, wird versucht, durch:

```
(retrieve (?x (datatype-fillers (hasXNameValue ?x)))
  (and (?x XName)
  (same-as ?y Individuum)
  (?y ?x hasXName)))
```

den Namen zu erhalten. Hierbei steht X für die Bezeichnung des Konzepts von dem gegebenen Individuum, welche durch *getConceptOfIndividual* vorher ermittelt wurde.

In Listing 3.6 wird der entscheidende Teil der SemanticAdder-Klasse dargestellt. In Zeile 2 wird die OWL-Datei zu der aktuellen HTML-Datei geladen, in Zeile 4 werden alle direkt mit den Multimediadaten verbundenen Individuen in die Ergebnismenge mit aufgenommen. Dann beginnt die Suche nach den weiteren erreichbaren Individuen gemäß des von mir entwickelten Algorithmus aus Abschnitt 2.2.3. Die Suche bricht ab, wenn die angegebene Tiefe erreicht wurde, oder wenn in einem Schritt keine unbekannten Individuen gefunden werden konnten (siehe Zeile 13). In Zeile 18 werden alle Namen der gefundenen Individuen in eine weitere Menge eingefügt, um doppelte Namen zu verhindern. Diese Namen werden dann später vor das eigentliche Dokument geschrieben und zusätzlich noch als Field zurück gegeben. Ein extra Field lässt sich gesondert durchsuchen und bei der Suche darstellen.

In Abbildung 3.3 wird eine Beispielsuche mit dem beschriebenen Algorithmus (maximalen Tiefe gleich eins) durchgeführt. Ausgehend vom Individuum „WebPage1“ werden über „Text1“ die Individuen „Person1“ und „SportsEvent1“ als Startmenge ausgewählt. Im ersten Schritt wird nicht in die Breite expandiert, deswegen werden die

```
1  /*...*/
2  RacerQueries.loadOwlFile(connection, abox, false, null);
3  Set<String> individuals = new HashSet<String>();
4  individuals.addAll(RacerQueries.getIndividualsFromWebPage(
       connection, textOnly, docUrl));
5  for (int i = 0; i < depth; i++) {
6      int size = individuals.size();
7      Set<String> newIndividuals = new HashSet<String>();
8      for (String ind : individuals) {
9          newIndividuals.addAll(RacerQueries.getRelatedIndividuals
               (connection, ind));
10     }
11     individuals.addAll(newIndividuals);
12     if (individuals.size() == size) {
13         break;
14     }
15 }
16 Set<String> names = new HashSet<String>();
17 for (String ind : individuals) {
18     String res = RacerQueries.getNameOfIndividual(connection,
           ind);
19     if(res != null) {
20         names.add(res.toLowerCase());
21     }
22 }
23 /*...*/
```

Listing 3.6: Ausschnitt aus der SemanticAdder-Klasse

Individuen „Country1“ und „Date1“ nicht gefunden, die Menge der gefundenen Individuen bleibt gleich. Dann werden die Namen der Individuen („James Beckford“ und „dkb istaf berlin“) ermittelt und der Menge der gefundenen Werte hinzugefügt. Im zweiten Schritt werden nun „Country1“ und „Date1“ gefunden und deren Namen ebenfalls ermittelt („jamaica“ und „2006-09-05“).

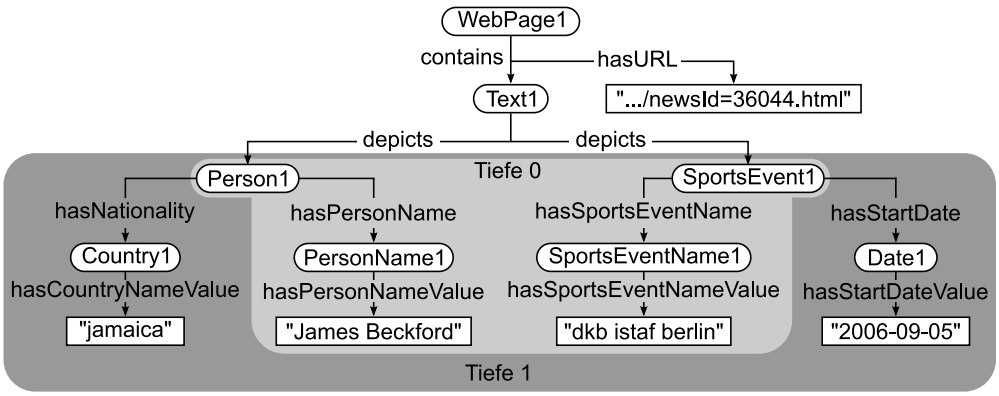


Abbildung 3.3: Beispielsuche mit Tiefe eins nach Annotationen in einer ABox

Die Verbindung zu RacerPro wird in einem Objekt der Klasse RacerConnection gehalten und den entsprechenden Methoden übergeben. Die Klasse benutzt JRacer, um die Verbindung aufzubauen, abzubauen und Befehle zu senden. Zusätzlich wird zur Steigerung der Performance und zur Entlastung von RacerPro eine Möglichkeit bereit gestellt, Anfragen zu cachen. Der Cache besteht dabei aus einer Zuordnung aus Suchanfrage zu Ergebnis in Form einer Map. Der Cache ist für die Dauer der Verbindung gültig, kann aber auch manuell geleert werden.

3.5 Berechnung der Ergebnisbewertungen

Für die Berechnungen der in Abschnitt 2.4.4 ausgewählten Kriterien zur Bewertung habe ich eine Klasse namens Formulas erstellt. Diese Klasse bietet drei statische Methoden zur Berechnung an, welche alle als Parameter eine Liste von QueryResult-Objekten bekommen. Folgende Methoden wurden definiert:

countRelevantResults: Zählt alle relevanten Ergebnisse in der Liste.

calcPrecision: Berechnet die precision (siehe 2.4.1).

calcAveragePrecision: Berechnet die average precision (siehe 2.4.2).

Die zweite und dritte Methode benutzen die Erste zur Berechnung.

3.6 Eine grafische Oberfläche zur Durchführung der Suchen

Das von mir entwickelte Programm SemTexSearchGui stellt eine einfache grafische Oberfläche zur Benutzung der Funktionen von SemTexSearch dar. Die Ziele waren hier, die Funktion der API zu demonstrieren und bei der Auswertung der Suchergebnisse zu unterstützen.

Die GUI wurde komplett in Java programmiert, zum Design der Oberflächen setzte ich den Swing GUI Builder der Netbeans IDE ein, welcher eine grafische Manipulation der Swingkomponenten ermöglicht und den entsprechenden Code automatisch generiert.

Abbildung 3.4 ist ein Screenshot des Hauptfensters mit einer durchgeführten Suche. Die Ergebnistabelle enthält die Position des Suchergebnisses, den Dateinamen, die zugehörigen Fields und die von Lucene ermittelte Score. Per Doppelklick auf die Suchergebnisse kann die Seite im voreingestellten HTML-Betrachter geöffnet werden. Durch einen Befehl im Menü kann man zwischen der normalen Ansicht und der Auswertungsansicht mit Eingabemöglichkeit für die Relevanz der Ergebnisse umschalten (siehe Abb. 3.5). In der Statusleiste am unteren Rand werden dann zur Auswertung die in 2.4.4 ausgewählten Kriterien angezeigt und bei jeder Änderung der relevanten Ergebnisse automatisch neu berechnet.

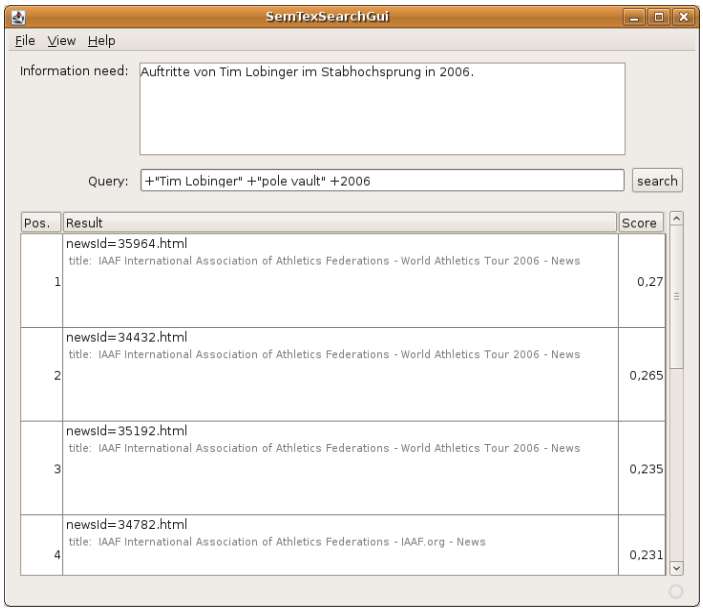


Abbildung 3.4: Das Hauptfenster der GUI

Im Menü befinden sich weiterhin Punkte zum:

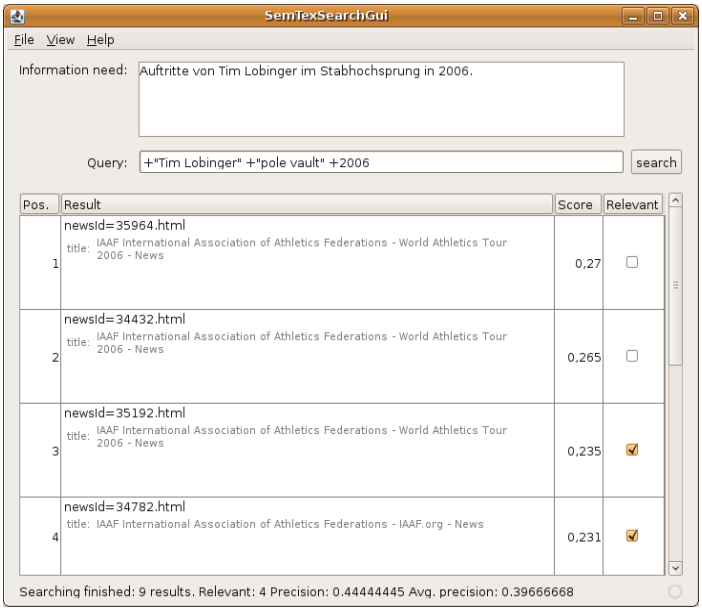


Abbildung 3.5: Das Hauptfenster mit Anzeige der Relevanz

- Laden und Speichern von Anfragen
- Laden von Indizes
- Generierung von Indizes
- Anzeige von Indexdetails
- Mappen von OWL- und HTML-Dateien

Der Dialog zum Erstellen eines neuen Indexes wird in Abb. 3.6 gezeigt. Einstellbar sind die nötigen Pfade und die Optionen für die semantische Anreicherung.

3.7 Erstellung der Indizes und Suchanfragen zur Auswertung

Um später in der Auswertung eine Aussage über eine etwaige Verbesserung der Suchergebnisse treffen zu können, habe ich fünf Indizes mit unterschiedlicher semantischer Beigabe und einen Testanfragenpool mit vierzig Anfragen (siehe Anhang A) erstellt.

In Tabelle 3.2 sind die Indizes mit den Merkmalen dargestellt. Darin kann man erkennen, dass auf Grund der Größe der Indexdateien zwischen den Indizes zwei und drei kein allzu großer Unterschied bestehen kann. Das gleiche gilt für vier und fünf. Dieser Umstand wird sich höchstwahrscheinlich in der Auswertung bemerkbar machen.

Abbildung 3.6: Der Dialog zum Erstellen eines neuen Index

Nr.	Mit sem. Inf.	Nur Text	Tiefe	Indexgröße in MB
1				1,7
2	✓	✓	0	2,2
3	✓		0	2,2
4	✓		1	2,8
5	✓		2	2,8

Tabelle 3.2: Übersicht der erstellten Indizes

4 Auswertung und Schlussfolgerung

4.1 Bewertung der Indizes

Zur Auswertung habe ich alle Anfragen des Testanfragenpools auf jeden der in Abschnitt 3.7 beschriebenen Indizes ausgeführt und alle Ergebnisse auf ihre Relevanz hin überprüft. Die relevanten Ergebnisse wurden entsprechend markiert. In Anhang B befindet sich eine Tabelle mit den berechneten Ergebnissen. Darin ist die Anzahl der Ergebnisse, die Anzahl der relevanten Ergebnisse, die Precision und die Avg. Precision pro Anfrage und Index enthalten.

Um nun eine Aussage treffen zu können, sind in Tabelle 4.1 die Mittelwerte der Ergebnisse dargestellt. Zusätzlich zeigen die Abbildungen 4.1, 4.2, 4.3 und 4.4 die Werte

	Mittelwert			
Index	Ergebnisse	Rel. Ergebnisse	Precision	Avg. Precision
I_1	13,3	4,45	0,4	0,56
I_2	13,4	4,53	0,4	0,57
I_3	13,4	4,53	0,4	0,58
I_4	14,03	4,65	0,4	0,57
I_5	14,03	4,65	0,4	0,57

Tabelle 4.1: Mittelwerte der Ergebnisse aus Anhang B

als Balkendiagramme. In Abbildung 4.1 sieht man, dass sich die Anzahl der gelieferten Suchergebnisse nur um zirka ein Ergebnis im Mittel erhöht hat. Dieser Umstand deutet darauf hin, dass auch die anderen Größen pro Index nicht signifikant unterschiedlich sein werden. Außerdem kann man erkennen, dass zwischen Index zwei und drei kein Unterschied besteht, diese Beobachtung deckt sich mit der vergleichbaren Indexgröße aus Abschnitt 3.7. Das Gleiche gilt für Nummer vier und Nummer fünf. Das Diagramm in Abbildung 4.2 zeigt auch, dass der Unterschied zwischen zwei und drei und vier und fünf vernachlässigbar gering und ansonsten sehr klein ist. Selbstverständlich kann die Anzahl der relevanten Ergebnisse nicht größer als die Zahl aller Ergebnisse sein, deswegen war eine überproportionale Steigerung auf Grund der bisherigen Erkenntnisse auch nicht zu erwarten. Positiv kann vermerkt werden, dass immerhin eine Steigerung bezüglich des Indexes ohne semantische Informationen vorhanden ist.

Die Diagramme 4.3 und 4.4 zeigen, dass es keine maßgebliche Veränderung bezüglich der Precision und der Avg. Precision gegeben hat.

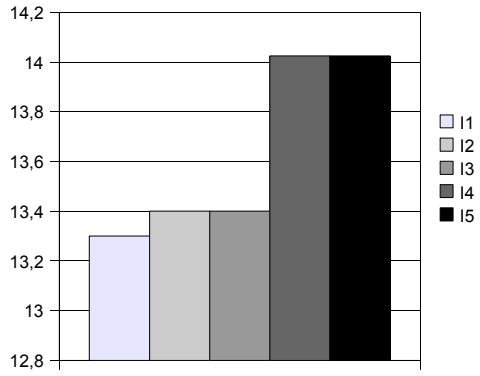


Abbildung 4.1: Anzahl der Suchergebnisse pro Index (Y-Achse, Mittelwert)

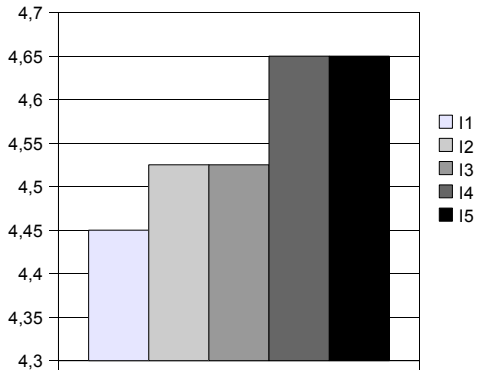


Abbildung 4.2: Anzahl relevanter Suchergebnisse pro Index (Y-Achse, Mittelwert)

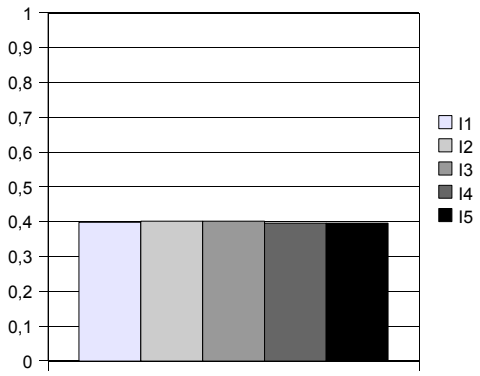


Abbildung 4.3: Precision pro Index (Y-Achse, Mittelwert)

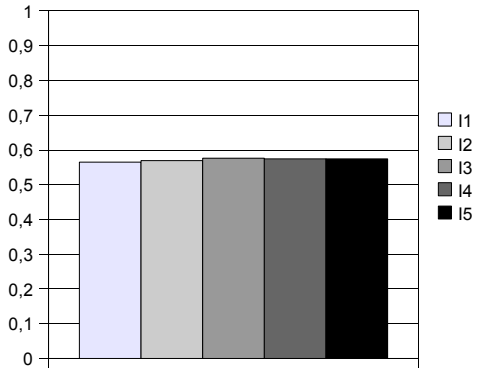


Abbildung 4.4: Avg. Precision pro Index (Y-Achse, Mittelwert)

4.2 Diskussion der Ergebnisse

Wie schon in Abschnitt 2.5 beschrieben, beeinflussen die vorhandenen Daten das Ergebnis sehr stark. Meine aufgestellte Annahme war, dass sich auf Grund der Art der indizierten Webseiten die Suchergebnisse nur mäßig verbessern lassen, und die im vorherigen Abschnitt gezeigten Ergebnisse der Auswertung bestätigen dies.

Als Beispiel für eine Verbesserung der Suchergebnisse soll hier Anfrage Nummer drei aus dem Pool dienen. Der Suchstring lautet:

```
+results +"4x400m relay" +men
```

Die Auswertung für Anfrage drei ist in Tabelle 4.2 dargestellt. Darin sieht man, dass die Verbesserung durch die Verwendung von Index vier erzielt wird, weil darin die Ergebnisliste um zwei Positionen erweitert wurde und ein Ergebnis davon sogar als relevant markiert wurde. In Tabelle 4.3 sind die Ergebnisse bezüglich des vierten (und

Index	Ergebnisse	Relevante Erg.	Precision	Avg. precision
I_1	9	4	0,44	0,4
I_2	9	4	0,44	0,43
I_3	9	4	0,44	0,43
I_4	11	5	0,45	0,5
I_5	11	5	0,45	0,5

Tabelle 4.2: Auswertung der Testanfrage Nr. 3.

fünften) Indizes aufgelistet. Die hinzugekommenen Ergebnisse sind fett markiert. Das neunte Ergebnis lautet auszugsweise:

> ... Simpson said: "That was a great result in this type of field. The wind made it difficult but I was focussed on the race so it wasn't an issue"...
> ... But there was final event mishap for the men in the 4 x 400m relay...

Der Grund für das zusätzlich gewonnene, relevante Ergebnis ist, dass in den semantischen Annotationen die Sportart „viermal vierhundert Meter Staffellauf" mit „4x400m relay" und nicht wie im Dokument selbst mit „4 x 400m relay" bezeichnet wird, und nach genau der Version ohne Leerzeichen gesucht wurde. Alle anderen Suchterme sind entweder genau so („men") oder in einer durch Stemming reduzierten Form („results" wird zu „result") direkt im Dokument enthalten.

Dieses Beispiel zeigt, dass die gesuchten Informationen in der Regel durch den Nachrichtencharakter der Webseiten wörtlich im Text stehen und nur in wenigen Fällen die semantischen Extrainformationen deutlichen Mehrwert erzielen können. Außerdem zeigt das Beispiel, dass Stemming zur Verbesserung beitragen kann, weil sonst das exemplarisch betrachtete Beispiel durch den Unterschied von „results" zu „result" nicht als Ergebnis zurück geliefert worden wäre.

Position	Dokument	Relevant
1	newsId=31981.html	
2	newsId=34989.html	✓
3	newsId=26475.html	
4	3197341.stm.html	✓
5	3540411.stm.html	✓
6	newsId=25609.html	
7	newsId=22260.html	
8	USATF_2006_09_17_11_55_46.html	✓
9	article-41/index.html	✓
10	newsId=36001.html	
11	newsId=34796.html	

Tabelle 4.3: Ergebnisse der Testanfrage Nr. 3.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung der Arbeit

Das Ziel dieser Arbeit war, zu untersuchen, ob sich Suchergebnisse durch die Beigabe von semantischen Informationen signifikant verbessern lassen. Dazu wurden in Kapitel 2 die theoretischen Grundlagen folgender Themen erörtert:

- Klassische Textindizierung
- Semantische Indizierung
- Stemming
- Die Bewertungsmöglichkeiten von Suchergebnissen

Es wurde ein Algorithmus basierend auf der Breitensuche in (gerichteten) Graphen entworfen, welcher im Zuge der Implementierung umgesetzt wurde. Außerdem mussten aus Gründen der Umsetzbarkeit aus den möglichen Bewertungsverfahren eine Menge ausgewählt werden, welche zur Auswertung der Ergebnisse dienen sollte.

Die Durchführung wurde in Kapitel 3 beschrieben. Dazu gehörten die grobe Planung des zu schreibenden Programms, und eine detailreichere Beschreibung der Kernelemente:

- Die Textindizierung mit Lucene
- Die Textextraktion aus den gegebenen HTML-Dateien mit NekoHTML
- Die Benutzung von RacerPro zur Anreicherung der Dokumente mit den semantischen Annotationen
- Die Implementierung der Bewertungsmethoden

Zusätzlich wurde kurz die erstellte grafische Oberfläche zur Indexerstellung, Suche und Auswertung gezeigt.

Die Ergebnisse der Arbeit wurde in Kapitel 4 präsentiert und diskutiert. Dabei kam heraus, dass sich die Suchergebnisse nicht signifikant durch das entwickelte Verfahren verbessern oder verschlechtern lassen. Als ein Grund wurde der Nachrichtencharakter der gegebenen Dokumente genannt.

5.2 Ausblick

Wie in 4.2 beschrieben ist eine Erklärung der Ergebnisse der Auswertung die Art der zugrundeliegenden Daten. Um diese Aussage zu überprüfen und um eventuell eine Verbesserung der Suchergebnisse zu erzielen, würde sich ein Versuch mit einer anderen Datenbasis anbieten. Das in dieser Arbeit vorgestellte Programm wäre dafür geeignet, solange die in Abschnitt 3.4.2 genannten Einschränkungen gelten.

Zusätzlich zu der Auswertung mit einer festen Menge an Testanfragen und einem oder mehreren designierten Testern könnte man das A/B-Testverfahren einsetzen (siehe [9]). Dieses Verfahren basiert darauf, dass eine größere Nutzerbasis existiert, welche das System verwendet. Um das A/B-Testverfahren umzusetzen, lässt man zwei Systeme mit gleicher Oberfläche, aber unterschiedlichem Verhalten, gleichzeitig laufen und verteilt die Benutzer nach einem festgelegten Verhältnis. Weiterhin muss eine Möglichkeit gefunden werden, die Benutzerinteraktionen in Bewertungen für die Systeme umzusetzen (zum Beispiel die Verweildauer oder die Anzahl Klicks auf bestimmte Steuerelemente). So kann man nun nach einem zu bestimmenden Zeitraum feststellen, welches System effektiver ist. Als Beispiel könnte man die verschiedenen Systeme mit zwei unterschiedlichen Indizes ausstatten, um hinterher eine Aussage treffen zu können. Das Problem hierbei ist der nötige Zeitaufwand für die Datensammlungsphase und die hinreichend große Nutzergruppe.

Ein weiteres Thema wäre die Verwendung von einer Relevanz mit Stufencharakter. In dieser Arbeit wurde die Relevanz immer nur als binär angesehen. Ein Dokument konnte nur „relevant“ oder „nicht relevant“ sein. Ein alternativer Ansatz könnte die Relevanz eines Dokuments zum Beispiel als Prozentsatz ausdrücken. Dieses Vorgehen würde dann andere Methoden zur Auswertung der Suchergebnisse erfordern und könnte zu differenzierteren Ergebnissen führen.

Weiterhin wäre es denkbar, die Klassifizierung der Ergebnisse in „relevant“ und „nicht relevant“ (oder in Abstufungen) einem Programm mit Techniken der künstlichen Intelligenz zu überlassen. Solch ein Programm würde bei einer hinreichend kurzen Bearbeitungszeit zum Beispiel die Berechnung des Recalls für eine große Menge von Dokumenten ermöglichen.

Literaturverzeichnis

[1] Baader, F. ; Calvanese, D. ; McGuinness, D. ; Nardi, D. ; Patel-Schneider, P. : *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003

[2] Berry, M. W. ; Browne, M. : *Understanding search engines: mathematical modeling and text retrieval*. Society for Industrial and Applied Mathematics, 1999. – ISBN 0–89871–437–0

[3] Clark, A. : *CyberNeko HTML Parser*. http://nekohtml.sourceforge.net/index.html. Version: Mai 2009

[4] Cormen, T. H. ; Leiserson, C. E. ; Rivest, R. L. ; Stein, C. : *Introduction to Algorithms*. Second Edition. MIT Press, 2003. – ISBN 0–262–53196–8

[5] Das BOEMIE-Konsortium: *Die Webseite des BOEMIE-Projekts*. http://www.boemie.org. Version: Mai 2009

[6] Diestel, R. : *Graphentheorie*. Springer-Verlag, 2006. – ISBN 3–540–21391–0

[7] Gospodnetić, O. ; Hatcher, E. : *Lucene in Action*. Manning, 2005. – ISBN 1–932394–28–1

[8] Levene, M. : *An Introduction to Search Engines and Web Navigation*. Addison-Wesley, 2006. – ISBN 0–321–30677–5

[9] Manning, C. D. ; Raghavan, P. ; Schütze, H. : *An Introduction to Information Retrieval*. Cambridge University Press, 2008

[10] Racer Systems GmbH & Co. KG (Hrsg.): *RacerPro Reference Manual*. Racer Systems GmbH & Co. KG, Dezember 2005

[11] Racer Systems GmbH & Co. KG (Hrsg.): *RacerPro User's Guide*. Racer Systems GmbH & Co. KG, Dezember 2005

[12] The Apache Software Foundation: *Die Webseite des Apache Lucene-Projekts*. http://lucene.apache.org/. Version: Mai 2009

[13] *Kapitel* 6. In: van Rijsbergen, C. ; Robertson, S. ; Porter, M. : *An Algorithm for Suffix Stripping*. British Library, 1980

Anhang A

Testanfragen

1. **Beschreibung:** Ergebnisse von Usain Bolt im 200m Lauf.
   Anfrage: `+"usain bolt" +results +200m`
2. **Beschreibung:** Ergebnisse im Dreisprung bei den Olympischen Spielen 2004.
   Anfrage: `+"triple jump" +"Olympic Games 2004" +results`
3. **Beschreibung:** Ergebnisse der 4 x 400m Staffelläufe der Herren.
   Anfrage: `+results +"4x400m relay" +men`
4. **Beschreibung:** Alle Informationen über Yulia Pechonkina bei der TDK Golden League.
   Anfrage: `+"yulia pechonkina" +"TDK Golden League"`
5. **Beschreibung:** Gewinner einer Goldmedaille im Hammerwurf der Frauen.
   Anfrage: `+"gold medal" +"hammer throw" +"women"`
6. **Beschreibung:** Welche Medaille hat Dwight Phillips bei den Olympischen Spielen 2004 im Weitsprung gewonnen?
   Anfrage: `+"dwight phillips" +"olympic games" +"long jump" +2004 +medal`
7. **Beschreibung:** Ergebnisse von Ezekiel Kemboi im 3000m Hindernislauf
   Anfrage: `+"ezekiel kemboi" +steeplechase +3000m +results`
8. **Beschreibung:** Welche russische Athletinnen haben am Kugelstoßen während des Grand Prix 2004 teilgenommen?
   Anfrage: `+"shot put" +women +russia +"grand prix" +2004`
9. **Beschreibung:** Ergebnisse im 5000m-Lauf der Herren während der Golden League 2006
   Anfrage: `+5000m +men +results +"golden league" +2006`
10. **Beschreibung:** Goldmedaillengewinner bei den Olympischen Spielen im Gehen
    Anfrage: `+"olympic games" +walk +"gold medal"`
11. **Beschreibung:** Persönliche Bestleistungen von Yuriy Borzakovskiy im 1500m-Lauf
    Anfrage: `+"yuriy borzakovskiy" +1500m +"personal best"`

12. **Beschreibung:** Speerwürfe von Frauen in Weltrekordweite.
    Anfrage: `+"javelin world record" 10 +women`

13. **Beschreibung:** Ergebnisse von Tatyana Lebedeva im Dreisprung in 2006.
    Anfrage: `+"Tatyana Lebedeva" +"triple jump" +2006`

14. **Beschreibung:** Ergebnisse von Tyson Gay während der World Athletics Tour 2006.
    Anfrage: `+"tyson gay" +"World Athletics Tour 2006"`

15. **Beschreibung:** Goldmedaillen im Marathon der Herren während den Olympischen Spielen
    Anfrage: `+marathon +"olympic games" +gold +men`

16. **Beschreibung:** Ergebnisse im 400m Hürdenlauf der Frauen während der World Athletics Tour.
    Anfrage: `+"400m hurdles" +"World Athletics Tour" +women +results`

17. **Beschreibung:** Ergebnisse von Svetlana Feofanova oder Stacy Dragila bei den World Indoor Championships.
    Anfrage: `+("svetlana feofanova" OR "Stacy Dragila")`
    `+"Indoor Championships"`

18. **Beschreibung:** Rekorde im Stabhochsprung
    Anfrage: `+"pole vault world record"~10`

19. **Beschreibung:** Finalergebnisse von Dwight Phillips im Hochsprung.
    Anfrage: `+"Dwight Phillips" +"long jump" +"final" +results`

20. **Beschreibung:** Auftritte von Tim Lobinger im Stabhochsprung in 2006.
    Anfrage: `+"Tim Lobinger" +"pole vault" +2006`

21. **Beschreibung:** Siebenkämpfe mit deutscher Beteiligung
    Anfrage: `+Heptathlon +(german OR "from germany")`

22. **Beschreibung:** Ergebnisse im Siebenkampf der Frauen in 2004
    Anfrage: `+heptathlon +women +results +2004`

23. **Beschreibung:** Siege von Meseret Defar im 5000m-Lauf
    Anfrage: `+"Meseret Defar" +victories +5000m`

24. **Beschreibung:** Ergebnisse von Louis van Zyl im 400m Hürdenlauf
    Anfrage: `+"Louis van Zyl" +400m +hurdles`

25. **Beschreibung:** Ergebnisse von US-amerikanischen Teams im 4x100m Staffellauf.
    Anfrage: `+"4x100m relay" +american +team +results`

26. **Beschreibung:** Medaillen von Yelena Isinbayeva im Stabhochsprung
    Anfrage: `+"Yelena Isinbayeva" +"pole vault" +medal`

27. **Beschreibung:** Ergebnisse von Yipsi Moreno im Hammerwurf
    Anfrage: `+"hammer throw" +"Yipsi Moreno" +results`

28. **Beschreibung:** Ergebnisse von Christian Cantwell im Kugelstoßen in 2004
    Anfrage: `+"Christian Cantwell" +"shot put" +2004 results`

29. **Beschreibung:** Deutsche Beteiligung am Marathonlauf bei den Europäischen Meisterschaften
    Anfrage: `+"European Championships" +marathon +"german"`

30. **Beschreibung:** Welche Gold- oder Silbermedaillen wurden im 200m-Lauf der Frauen in 2006 errungen?
    Anfrage: `+"200m" +women +("gold medal" OR "silver medal") +2006`

31. **Beschreibung:** Auftritte von Allen Johnson im 110m-Hürdenlauf bei den Indoor Championships.
    Anfrage: `+110m +hurdles +"Allen Johnson" +"Indoor Championships"`

32. **Beschreibung:** Ergebnisse von Chen Qi im Speerwurf
    Anfrage: `+"Chen Qi" +javelin +results`

33. **Beschreibung:** Persönliche Bestzeiten im Halbmarathon der Herren.
    Anfrage: `+"half marathon" +"personal best" +men`

34. **Beschreibung:** Zeiten von Tatyana Andrianova im 800m-Lauf.
    Anfrage: `+"Tatyana Andrianova" +800m`

35. **Beschreibung:** Gewonnene Goldmedaillen von Franka Dietzsch im Diskuswurf.
    Anfrage: `+"Franka Dietzsch" +discus +gold medal`

36. **Beschreibung:** Ergebnisse von Yelena Slesarenko im Hochsprung
    Anfrage: `+"Yelena Slesarenko" +"high jump" +results`

37. **Beschreibung:** Teilnahmen von Mark Lewis-Francis im 4x100m Staffelauf
    Anfrage: `+"Mark Lewis-Francis" +4x100m +relay`

38. **Beschreibung:** Ergebnisse im 1500m-Lauf der Frauen bei den Olympischen Spielen
    Anfrage: `+1500m +women +"olympic games" +results`

39. **Beschreibung:** Russische Teilnehmer am Hindernislauf in 2006.
    Anfrage: `+steeplechase +2006 +russia`

40. **Beschreibung:** Persönliche Bestleistungen von Andreas Thorkildsen im Speerwurf
    Anfrage: `+"Andreas Thorkildsen" +"javelin throw" +"personal best"`

Anhang B

Ergebnisstatistik

	Ergebnisse					Relevante Erg.					Precision					Avg. precision				
Nr	I_1	I_2	I_3	I_4	I_5	I_1	I_2	I_3	I_4	I_5	I_1	I_2	I_3	I_4	I_5	I_1	I_2	I_3	I_4	I_5
1	10	10	10	10	10	7	7	7	7	7	0,7	0,7	0,7	0,7	0,7	0,9	0,9	0,9	0,9	0,9
2	3	3	3	3	3	1	1	1	1	1	0,33	0,33	0,33	0,33	0,33	0,05	0,05	0,05	0,05	0,05
3	9	9	9	11	11	4	4	4	5	5	0,44	0,44	0,44	0,45	0,45	0,4	0,43	0,43	0,5	0,5
4	2	2	2	2	2	1	1	1	1	1	0,5	0,5	0,5	0,5	0,5	0,1	0,05	0,05	0,1	0,1
5	15	15	15	15	15	9	9	9	9	9	0,6	0,6	0,6	0,6	0,6	0,9	0,9	0,9	0,9	0,9
6	10	10	10	10	10	2	2	2	2	2	0,2	0,2	0,2	0,2	0,2	0,6	0,6	0,6	0,6	0,6
7	7	7	7	7	7	6	6	6	6	6	0,86	0,86	0,86	0,86	0,86	0,9	0,65	0,9	0,9	0,9
8	9	9	9	9	9	1	1	1	1	1	0,11	0,11	0,11	0,11	0,11	0,05	0,1	0,1	0,1	0,1
9	12	12	12	13	13	4	4	4	4	4	0,33	0,33	0,33	0,31	0,31	0,53	0,53	0,53	0,53	0,53
10	16	17	17	17	17	9	10	10	10	10	0,56	0,59	0,59	0,59	0,59	0,9	1	1	1	1
11	23	23	23	25	25	2	2	2	2	2	0,09	0,09	0,09	0,08	0,08	0,6	0,6	0,6	0,6	0,6
12	19	19	19	19	19	6	6	6	6	6	0,32	0,32	0,32	0,32	0,32	0,9	0,9	0,9	0,9	0,9
13	14	14	14	14	14	9	9	9	9	9	0,64	0,64	0,64	0,64	0,64	0,67	0,61	0,63	0,9	0,9
14	20	20	20	20	20	3	3	3	3	3	0,15	0,15	0,15	0,15	0,15	0,23	0,18	0,18	0,18	0,18
15	23	23	23	23	23	1	1	1	1	1	0,04	0,04	0,04	0,04	0,04	0,01	0,01	0,01	0,01	0,01
16	17	17	17	20	20	6	6	6	8	8	0,35	0,35	0,35	0,4	0,4	0,45	0,38	0,38	0,45	0,45
17	8	9	9	9	9	2	3	3	3	3	0,25	0,33	0,33	0,33	0,33	0,22	0,31	0,31	0,31	0,31
18	23	23	23	23	23	11	11	11	11	11	0,48	0,48	0,48	0,48	0,48	1	1	1	1	1
19	18	18	18	18	18	11	11	11	11	11	0,61	0,61	0,61	0,61	0,61	1	1	1	1	1
20	9	9	9	9	9	4	4	4	4	4	0,44	0,44	0,44	0,44	0,44	0,4	0,8	0,8	0,8	0,8
21	9	9	9	9	9	3	3	3	3	3	0,33	0,33	0,33	0,33	0,33	0,7	0,7	0,7	0,7	0,7
22	11	11	11	12	12	1	1	1	1	1	0,09	0,09	0,09	0,08	0,08	0,1	0,1	0,1	0,1	0,1
23	23	24	24	26	26	9	9	9	9	9	0,39	0,38	0,38	0,35	0,35	0,9	0,9	0,9	0,9	0,9
24	7	7	7	10	10	6	6	6	8	8	0,86	0,86	0,86	0,8	0,8	0,9	0,9	0,9	0,9	0,9
25	16	16	16	16	16	11	11	11	11	11	0,69	0,69	0,69	0,69	0,69	1	1	1	1	1
26	17	18	18	18	18	5	6	6	6	6	0,29	0,33	0,33	0,33	0,33	0,9	0,9	0,9	0,9	0,9
27	7	7	7	7	7	6	6	6	6	6	0,86	0,86	0,86	0,86	0,86	0,9	0,9	0,9	0,9	0,9
28	11	11	11	11	11	1	1	1	1	1	0,09	0,09	0,09	0,09	0,09	0,01	0,01	0,01	0,02	0,02
29	6	6	6	6	6	2	2	2	2	2	0,33	0,33	0,33	0,33	0,33	0,6	0,6	0,6	0,6	0,6
30	13	13	13	14	14	3	3	3	3	3	0,23	0,23	0,23	0,21	0,21	0,7	0,7	0,7	0,7	0,7
31	8	8	8	8	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
32	4	4	4	4	4	4	4	4	4	4	1	1	1	1	1	0,8	0,8	0,8	0,8	0,8
33	37	37	37	37	37	1	1	1	1	1	0,03	0,03	0,03	0,03	0,03	0,02	0,03	0,03	0,01	0,01
34	7	7	7	8	8	6	6	6	6	6	0,86	0,86	0,86	0,75	0,75	0,9	0,9	0,9	0,64	0,64
35	8	8	8	8	8	3	3	3	3	3	0,38	0,38	0,38	0,38	0,38	0,7	0,7	0,7	0,7	0,7
36	10	10	10	10	10	7	7	7	7	7	0,7	0,7	0,7	0,7	0,7	0,9	0,9	0,9	0,9	0,9
37	11	11	11	11	11	6	6	6	6	6	0,55	0,55	0,55	0,55	0,55	0,9	0,9	0,9	0,9	0,9
38	23	23	23	27	27	2	2	2	2	2	0,09	0,09	0,09	0,07	0,07	0,16	0,2	0,2	0,15	0,15
39	21	21	21	23	23	2	2	2	2	2	0,1	0,1	0,1	0,09	0,09	0,6	0,6	0,6	0,32	0,32
40	16	16	16	19	19	1	1	1	1	1	0,06	0,06	0,06	0,05	0,05	0,1	0,05	0,05	0,1	0,1